COGITO

CONSTRUCTION PHASE DIGITAL TWIN MODEL

cogito-project.eu

# D7.3 – Extraction, Transformation & Loading tools and Model Checking v1

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

# D7.3 – Extraction, Transformation & Loading Tools and Model Checking v1

| | |
|---|---|
| Dissemination Level: | Public |
| Deliverable Type: | Demonstrator |
| Lead Partner: | UCL |
| Contributing Partners: | UPM |
| Due date: | 31-05-2022 |
| Actual submission date: | 31-05-2022 |

## Authors



| Name | Beneficiary | Email |
|---|---|---|
| **Georgios N. Lilis** | UCL | g.lilis@ucl.ac.uk |
| **Kyriakos Katsigarakis** | UCL | k.katsigarakis@ucl.ac.uk |
| **Dimitrios Rovas** | UCL | d.rovas@ucl.ac.uk |
| **Salvador González Gerpe** | UPM | salvador.gonzalez.gerpe@upm.es |



## Reviewers

| Name | Beneficiary | Email |
|---|---|---|
| **Giorgos Giannakis** | Hypertech | g.giannakis@hypertech.gr |
| **Jochen Teizer** | AU | teizer@cae.au.dk |

## Version History

| Version | Editors | Date | Comment |
|---|---|---|---|
| **0.1** | UCL | 20.04.2022 | Table of Contents |
| **0.2** | UCL | 30.04.2022 | Sections 1 and 2 |
| **0.3** | UCL, UPM | 05.05.2022 | Section 3 |
| **0.5** | UCL | 20.05.2022 | Sections 4 and 5 |
| **0.6** | UCL | 24.05.2022 | Draft version ready for internal review |
| **0.7** | Hypertech, AU | 27.05.2022 | First draft internal review |
| **0.8** | UCL | 30.05.2022 | Review comments addressed |
| **0.9** | UCL | 30.05.2022 | Final version |
| **1.0** | UCL, Hypertech | 31.05.2022 | Submission to the EC portal |

## Disclaimer

## Executive Summary

This deliverable presents the first version of the Extraction Transformation and Loading (ETL) and Model Checking (MC) tools of the COGITO framework. These tools are individual software components that are located in two layers of DTP's six-layer architecture presented in D7.1: the data ingestion layer and the data post-processing layer, as well as in the COGITO input data sources component. Based on the characteristics of their input data and the three data domains defined in D3.2 (construction, resources, and process) these tools can be classified into (a)construction-, (b)resource- and (c)process-related tools as well as (d) general tools receiving data from all the above domains and dynamic data from IoT devices. Additionally, depending on the type of operation these tools apply to their input data, they can be classified into transformation (ETL) and Model Checking (MC) tools. The ETL tools apply data transformations on COGITO input data, while the MC checking tools check the quality of COGITO's input data and report missing, incomplete, inconsistent, and incorrect data structures if any. In this deliverable domain dependent, ETL and MC tools will be analysed. Tools acting on IoT data will be introduced but discussed extensively in D8.2. The main part of the deliverable is split into two sections depending on the location of the analysed ETL and MC tools.

In the first main section, tools belonging to the Data Ingestion layer of DTP are analysed. These, include one MC tool, the RDF validator which applies SACHL shapes to validate the generated RDF files against the defined COGITO ontology, and four COGITO data domain-dependent RDF generator tools: (1) the Project RDF Generator which transforms project-related data into RDF TTL file format, (2) the Construction RDF Generator which transforms input IFC BIM data into RDF semantically linked data files, (3) the Process RDF generator which transforms schedule input data into RDF TTL files and (4) the Resources RDF Generator which transforms input data related to the construction project resources into RDF TTL files.

In the second main section of this deliverable tools placed at the Data Post-processing layer of DTP are presented. These include: one MC tool: the MVD checking tool which applies data quality checks on input IFC4x3 files and two ETL tools: (1) the B-rep Generator which transforms input IFC BIM files into graphics friendly OBJ files containing triangulated boundary representations for visual purposes and (2) the IFC optimizer which applies lossless compression on input IFC BIM files to reduce their size while maintaining their format.

The technology and dependencies of these tools are described in detail and application examples of these components are also presented in the following sections.

# Table of contents

# List of Figures

# List of Tables

## List of Acronyms

| Term | Description |
|---|---|
| **COGITO** | Construction Phase diGItal Twin mOdel |
| **DTP** | Digital Twin Platform |
| **BIM** | Building Information Model |
| **IFC** | Industry Foundation Class |
| **ETL** | Extraction Transformation & Loading |
| **MC** | Model Checking |
| **ESB** | Enterprise Service Bus |
| **SOA** | Service Oriented Architecture |
| **CSV** | Comma Separated Values |
| **SHACL** | SHapes Constraint Language |
| **BIM** | Building Information Model |
| **XML** | Extensible Markup Language |
| **RDF** | Resource Description Framework |
| **MVD** | Model View Definition |
| **JSON** | JavaScript Object Notation |
| **GUI** | Graphical User Interface |
| **COGITO** | Construction Phase diGItal Twin mOdel |
| **RDF** | Resource Description Framework |
| **TTL** | Terse Triple Language |
| **BRG** | B-Rep Generator |
| **DCC** | Digital Command Centre |
| **DigiTAR** | Digital Twin Visualization with AR |
| **UC** | Use Cases |
| **GPU** | Graphics Processing Unit |
| **OBJ** | Object file |
| **QC** | Quality Control |
| **H&S** | Health and Safety |

# 1 Introduction

The COGITO Deliverable D7.3 "Extraction, Transformation & Loading Tools and Model Checking" describes in detail all the Extraction Transformation, and Loading (ETL), and Model Checking (MC) components of COGITO's infrastructure. These tools are the main checking and processing blocks required to perform the necessary operations on COGITO's input data to support all other COGITO tools and applications and their requirements. These tools can be classified as (i) ETL tools, which include the tools that transform input data to appropriate data structures required for the formation of the COGITO digital twin model and for the data needs of other COGITO applications, and (ii) MC tools, which ensure that the generated data structures conform to the defined COGITO ontology and well-established standards.

## 1.1 Scope and Objectives of the Deliverable

The main scope of this deliverable is to describe in detail and demonstrate the operation of several components of COGITO's infrastructure characterized as Extraction Transformation and Loading (ETL) tools and Model Checking (MC) tools. These tools perform the following operations on COGITO's input data:

1. Checking the quality of COGITO's input data to ensure their suitability for further processing. The tools which belong to this category are characterized as Model Checking (MC) tools;
2. Applying transformations on these input data to generate the necessary new data structures to support the diverse data requirements of all COGITO tools. These tools are characterized as Extraction Transformation and Loading (ETL) tools; and
3. Ensuring that the generated semantically linked data model conforms to the defined COGITO ontology in D3.2.

The characteristics and goals of these components include:

- Conformity to multiple widely used input data formats including IFC, CSV, XML, and JSON from the various input sources such as laser scanners, UAVs, 4D BIM authoring tools, location sensing devices, multisource imagery data, and results of the Quality Control, Workflow Management and Health & Safety services;
- Provision of the necessary quality checking services to ensure that the input data content is complete, correct, and consistent with the COGITO ontology definition in WP3 D3.2.
- When the above two conditions are met, application of the necessary data transformation operations to support the various input data formats of other COGITO software components.

To minimize the overall execution time of their operations, some of the ETL and MC tools are organized in a scalable, containerized environment which enables asynchronous and independent execution. Furthermore, these tools are also diversified concerning their input data content as tools acting on static and dynamic data and as BIM-related and non-BIM-related tools. These tools are treated differently in terms of their required computational resources.

## 1.2 Relation to other Tasks and Deliverables

Deliverable D7.3: Extraction, Transformation & Loading Tools and Model Checking v1 depends on the following COGITO deliverables:

- D7.1: Digital Twin Platform Design & Specification v1, which presents the initial architecture design of COGITO's Digital Twin Platform. D7.3 presents an initial implementation of COGITO's ETL and MC tools. Some of these tools are contained in the DTP architecture described in D7.1.
- D3.2: COGITO Data Model & Ontology Definition and Interoperability Design v1, which describes the initial structure of COGITO's ontology. The operation of COGITO's ETL tools which generate semantic graphs (RDF generation tools) depends on the data classes of COGITO's ontology.
- D3.5: IoT Data Pre-processing Module v1, which describes the operation of the IoT Data Pre-processing module. The IoT Data Pre-processing module contains an ETL tool named Processing engine which implements data transformation operations on COGITO's input IoT data. These tools will be briefly mentioned here. A deeper analysis of these tools will be performed in D3.5.

## 1.3 Structure of the Deliverable

The COGITO's ETL and MC tools are classified depending on the nature of the data they are applied to. With this classification in mind, section 2 introduces the tools and relates them to the architecture of COGITO's Digital Twin Platform (DTP) described in D7.1. Two categories of ETL and MC tools are identified: tools acting on data that remain relatively constant for long periods of time (**static data)** and tools acting on data that change more frequently such as: the automatically updated data from IoT-enabled devices, point cloud data and the data generated by the QC Workflow and H&S applications (**dynamic data)**. "Static data" tools are classified further depending on their location with respect to the DTP architecture as: tools of the Data Ingestion layer and tools of the Data Post-processing layer. This deliverable focuses on tools acting on static data and describes them in sections 3 and 4.

With the above classification in mind, the ETL and MC tools of DTP's data Ingestion layer acting on static data are described in section 3.

In more detail section 3 consists of the following subsections:

- Subsection 3.1 introduces the RDF Validator as an MC tools of **Data Ingestion layer**.
- Subsections 3.2, 3.3, 3.4, 3.5 describe four RDF Generator tools referring to project data and input data from the three COGITO data domains defined in D3.2: Construction domain (CONS), Process domain (PROC) and Resource domain (RESO). These tools are ETL tools of DTP's Data Ingestion layer.

The ETL and MC tools of DTP's **Data Post-processing layer** acting on static BIM data are described in section 4. More specifically, section 4 consists of the following subsections:

- Subsection 4.1 introduces the MVD Checker as an MC tool of Data Post-processing layer.
- Subsection 4.2 describes the B-rep Generator as an ETL tool of the Data Post-processing layer.
- Subsection 4.3 describes the IFC optimizer as an ETL tool of the Data Post-processing layer.

This deliverable concludes in section 5.

# 2 ETL and MC tools of COGITO

Within COGITO's DTP architecture, the first version of which is described extensively in D7.1, there are ETL and MC tools applied on static COGITO input data, which are introduced in subsection 2.1. ETL and MC tools acting on dynamic data, belong to the IoT data Pre-processing component of COGITO and are introduced in subsection 2.2.

## 2.1 ETL and MC tools on static data

As displayed in Figure 1, the ETL and MC tools applied on static data are placed into two groups (displayed in blocks A and B in Figure 1) and in respective DTP layers: (A) the Data Ingestion layer (layer 2) and more specifically in its Knowledge Graph Generator component and (B) the Data Post-Processing layer (layer 6). These tools are briefly presented next and described analytically in section 3 and 4.

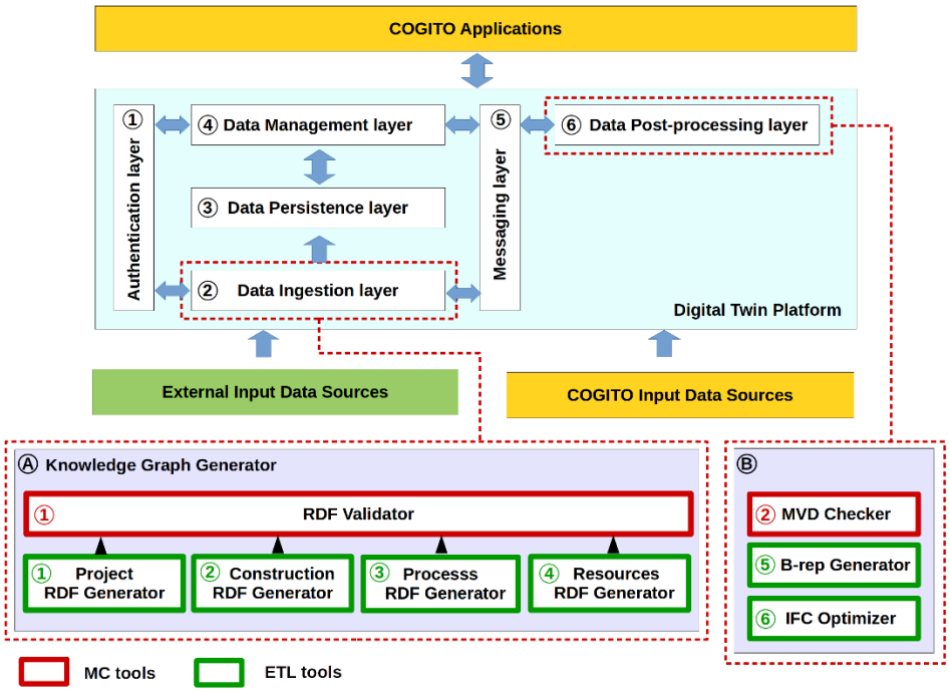



**Figure 1: ETL and MC tools on static COGITO input data.**

### 2.1.1 Data Ingestion layer tools (Block A)

Less computationally intensive ETL and MC tools are placed in the Data Ingestion layer of DTP. These tools include:

One MC tool, displayed with red rectangle 1 in block A in, Figure 1:

1. The **RDF Data Validator** which validates the RDF output files produced by the three RDF Generator tools (tools 1 - 4) using SACHL shapes, against pre-defined checking rules.

Four ETL tools which are displayed with green rectangles (1-4) in block A of Figure 1, and include:

1. The **Project RDF Generator** that converts project-related data such as project id, project name and project description to RDF TTL files
2. The **Construction RDF Generator** which converts input 3D BIM data (in the form of IFC4x3 files) to appropriate RDF files representing the semantic graph of 3D BIM data.

3. The **Process RDF Generator** which transforms the input construction schedule files into RDF files representing the fourth (time) dimension of the BIM data.
4. The **Resource RDF Generator** which converts the input construction resource data into respective RDF data.

### 2.1.2 Data Post-processing layer tools (Block B)

More computationally demanding ETL tools acting on COGITO 3D BIM input files are placed in the Data Post-Processing layer of DTP. To execute these tools in an effective manner, asynchronous and parallelizable connections using the Enterprise Service Bus (ESB) framework, were established between the Data Persistence layer the Data Management layer and the host layer of these tools the Data Post-processing layer, as illustrated in Figure 2.

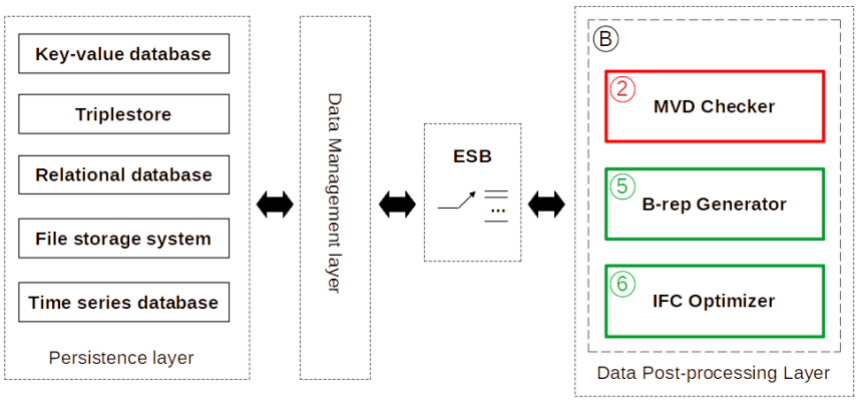



**Figure 2: Asynchronous parallelizable execution of Data Post-processing layer tools via ESB.**

The Data Post-processing layer ETL and MC tools include:

One MC tool, displayed with a red rectangle 2 in block B of Figure 1:

1. The **MVD checker** checks the content of input 3D BIM data files (IFC4x3) against rules defined in an MVD XML file.

Two ETL tools, displayed with a green rectangle (5 and 6) in block B of Figure 1:

1. The **B-rep Generator** transforms subsets of the geometric content of 3D BIM input files to a graphics-friendly format (OBJ format) on-demand, a feature useful for handheld AR devices which have hardware and processing power limitations.
2. The **IFC optimizer** reduces the file size of input 3D BIM files by applying lossless compression.

In the next sections, these tools and their utilized technologies are described in detail, starting with the Data Ingestion layer tools in section 3, followed by the Data Post-processing layer tools in section 4.

## 2.2 ETL and MC tools on dynamic data

Within COGITO infrastructure, there is an ETL tool acting on dynamic data named Processing engine which is displayed with the green rectangle in the IoT Data Pre-processing module block in Figure 2. This tool is a part of COGITO Input Data Sources layer.

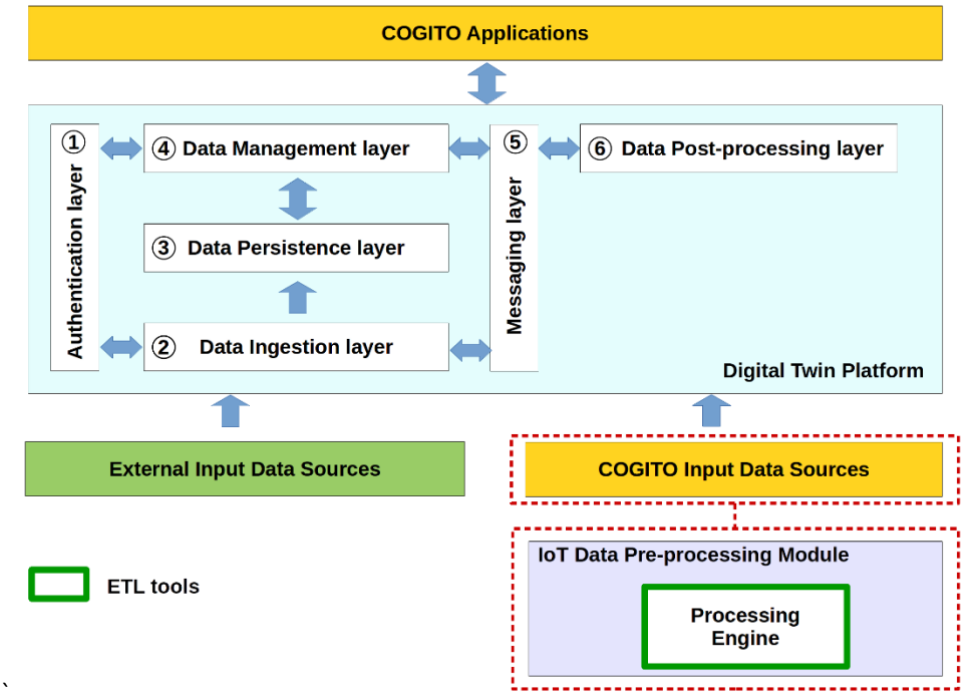



**Figure 3: ETL tools on dynamic COGITO input data**

Processing engine is described analytically in D3.6. In brief, this tool applies the following ETL operations on incoming IoT data:

- **Filtering:** IoT data contain outliers referring to sudden changes in the positions of the sensed construction entities. These outliers are sensed values which are filtered out from the IoT traffic.
- **Aggregating:** The sampled IoT position data are aggregated over constant time intervals, depending on the type of the sensed entity, resulting in smaller total IoT data volumes and data storage requirements.

# 3 ETL and MC tools in Data Ingestion layer of DTP

## 3.1 RDF Validator

A part of the COGITO data model consists of semantic graphs of linked data. These semantic graphs are in the form of inter-linked TTL RDF files using prefixes pointing to semantic ontologies and appropriate URIs. These files are stored in the triplestore data base of Data Persistence layer of DTP. These RDF files follow the TTL file format and are generated using four separate ETL processes described in subsections 3.2, 3.3, 3.4 and 3.5, by the four RDF Generation modules displayed in Figure 1. These files are parts of COGITO's semantic graph. After these parts are generated by the ETL processes are linked together enriching the overall COGITO semantic graph.

Multiple COGITO UC's contain queries to the COGITO semantic graph formed be these inter-linked data. These UCs include:

- UC1.1: PMS tool in transaction 5, requests from the DTP the as-planned resources
- UC2.2: Visual Data Pre-processing tool in transaction 7, requests from DTP building tasks and linked components.
- UC 3.2: Pro-active safety in transaction 1, requests from DTP as-built data.
- UC 3.2: SafeConAI in transaction 13, requests from DTP 4D BIM data and enhances them with safety information.
- UC4.1: DCC in transaction 7, requests task progress from DTP.
- UC4.2: DigiTAR in transaction 8, requests from DTP QC data linked to 3D BIM objects and in transaction 13, DigiTAR requests from DTP H&S data linked to 3D BIM objects.

To support the above use cases the semantic graph of COGITO data model should contain the necessary populated data structures. To ensure this, a checking mechanism should be established which will report possible: data errors, missing content or missing semantic links. These are possible data quality issues that are related to the consistency (missing links), completeness (missing content), and correctness (data errors) of the input data, that should be detected and reported before the formation of the COGITO data model graph. To detect these issues in all produced RDF files an RDF data validator component is implemented called RDF Validator. The operation of the RDF Validator is described in the following subsections.

### 3.1.1 Prototype Overview

The RDF Validator checks the conformity of the generated semantic graphs against a set of predefined checking rules contained, as in the case of the MVD Checker, in an RDF TTL file called the SHACL TTL file, or shapes graph file. This checking operation performed by the RDF validator is illustrated in the block diagram of Figure 4.

As illustrated of Figure 4, the RDF Validator receives two inputs:

- **Input RDF:** this is the file containing the data in TTL RDF form which is going to be checked. This file is also called data graph.
- **Checking rules:** this is a TTL RDF file containing the checking rules which are going to be applied in the input RDF during the checking operations. These rules are formed as TTL triplets according to the SHACL specifications. This file is also called shapes graph.

Finally, as it is also displayed in Figure 4, the RDF Validator outpus the checking results into a JSON text file.

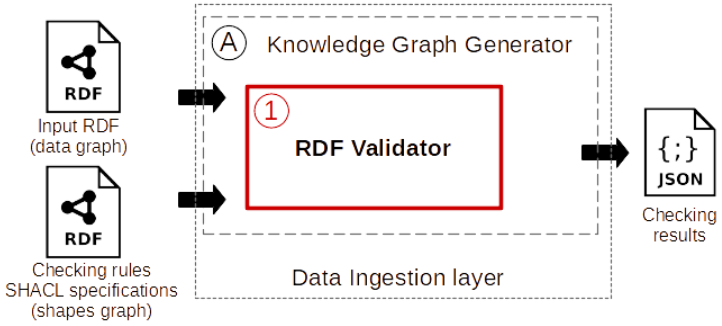



**Figure 4: RDF Validator tool operation**

The operation of the RDF data validator is based on the following operation step sequence:

1. Initially using Apache/Jena or similar RDF processing tools, the triplets contained into the input RDF files (data graph and SHACL shapes graph) are converted into memory objects.
2. For every entity contained in the objects generated from the shapes graph a checking rule is formed.
3. Using every rule formed in step 3 all objects originating from the RDF data file are checked against this rule.
4. The results of the checking procedures in step 3 are gathered and exported in the output JSON file.

### 3.1.2 Technology Stack and Implementation Tools

**Table 1 – Libraries and Technologies used in RDF Validator**

| Library/Technology Name | Version | License |
|---|---|---|
| Rdflib | 6.1.1 | BSD 3-Clause License |
| Pyshacl | 0.19.0 | Apache License 2.0 |

### 3.1.3 Input, Output, and API Documentation

Currently, input TTL RDF files referring to specific construction-related data quality checking operations have not been defined. To define however the input and output file format of MVD Validator, I/O examples from a human resources database are presented Figure 5 which are taken from (W3C, n.d.). The format of the input TTL RDF files (data graph and shapes graph) which are the presented in Figure 5, is the same as the format of construction-related input TTL RDF file used in RDF Validator. In the right part of Figure 5, an example of the validation results referring to the data and space graphs presented in the left part of Figure 5, is presented. As we can see from the data graph of Figure 5, the data to be checked contain three entries referring to three persons: (Alice, Bob, Calvin). For every entry certain properties are defined such as ssn, bith data and company name, by appropriate triplet predicates (ex:ssn, ex:birthdate and ex:worksFor).

Three checking rules are defined using the **sh:nodeShape** predicate in the example shapes graph in Figure 5, targeting entries by the **sh:targetClass**, **sh:property** and **sh:path** predicates. The first rule is related to the format of the **ex:ssn** entry (the ssn must be one entry, which has string format consisting of a set of three groups of three, two and four decimal numbers, separated with the dash character). The second rule declares that an instance of **ex:Person** can have unlimited values for the property **ex:worksFor** and these values are IRIs and SHACL instances of **ex:Company**. The third rule defines that all rdf properties to be ignored.

The checking results are summarized in the example validation results screenshot of Figure 5. All entries of the data graph fail to pass validation for different reasons. Entries ex:Alice and ex:Bob fail in rule 1 (alice not correct ssn format and bob max count of ssn), and ex:Calvin fails in rule 2 (not included company name) and in rule 3 as it uses an rdf type for the birthdate entry.

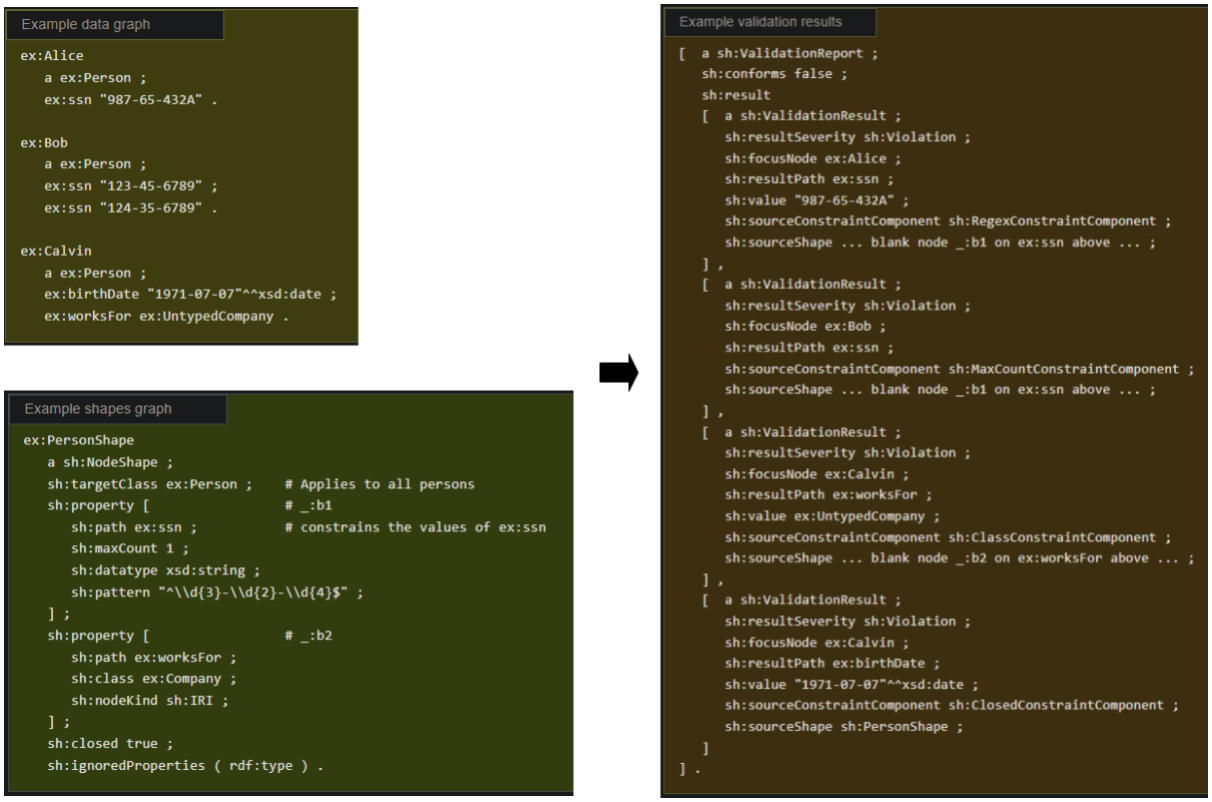



**Figure 5: Illustration of I/O file format of MVD Validator using data entries from an imaginary human resources database.**

Using the previous rationale of these simple examples, all the triplets referring to construction-related data that are contained in the TTL RDF files exported by the RDF Generator tools described in the following subsections, can be validated against rules defined also in TTL RDF files.

### 3.1.4 Application example

Currently, the RDF Validator is under development. SHACL based shape graphs have not been defined as TTL RDF files. Once these knowledge graphs and respective TTL RDF files are created, this tool will perform the validation of the links defined in these TTL RDF files. If the validation is successful, the process will end. In case the validation is not successful, a JSON file with the detected issues, formatted according to the report file of section Figure 5, will be returned.

### 3.1.5 Licensing

The license of the code used for the creation of this tool is available in the GitHub repository (https://github.com/oeg-upm/cogito_validator_module). The Project RDF Generator is a part of the KGG component of DTP. All KGG sub-components are open-source and licenced under the Apache Licence 2.0.

### 3.1.6 Installation Instructions

Currently the RDF Validator will be used internally in DTP and it will not be connected with an external UI, allowing external user interventions. Therefore, installation instructions are not applicable at this point.

### 3.1.7 Development and integration status

Currently, the RDF validator's development and integration into the system is in progress. The local operation of the tool works correctly, but there are still several functionalities to be included, such as the validation of more links between graphs generated by means of SHACL files.

### 3.1.8 Requirements Coverage

All input files to RDF Validator should follow the TTL RDF file format. Furthermore, for the correct operation of the tool the following two conditions should be satisfied:

1. The structure of the input data graph file should follow the COGITO ontology defined in D3.3.
2. The shapes graph should be formatted according to the SHACL shapes language described in (W3C, n.d.).

Any violation of the previous two requirements will result to improper tool execution without the desired checking results reported in the output file of the tool.

### 3.1.9 Assumptions and Restrictions

The operation of the RDF Data Validator tool relies on the following assumption:

- Knowledge graphs that have the links between them must be stored before this process of validation is made.
- The shapes defined in the SHACL TTL file should cover all required checking operations.

## 3.2 Project RDF Generator

Data traffic in COGITO among data sources or providers and data consumers is so diverse in nature, that without any data structures acting as a common point of reference, the related data exchanges will be left unorganized leading finally to confusion. The data which act as a common reference point, essentially bind together all data structures referring to a single construction project. These data are defined as project data.

These project data are generated when the project is created after the COGITO user is being authenticated by the DTP's Authentication layer applications. According to D2.5, within COGITO, a list of projects and project-related data are requested from the DTP by other COGITO tools in all UCs, to refine their data requests. More specifically:

- UC1.1: DTP in transaction 3, returns to PMS UI the list of projects.
- UC1.2: DTP in transaction 3, returns to WOEA the list of projects.
- UC2.1: DTP in transaction 4, returns to Visual Data Pre-processing the list of projects.
- UC2.2: DTP in transaction 4, returns to Visual Data Pre-processing UI the list of projects.
- UC3.1: DTP in transaction 3, returns to SafeConAI UI the list of projects.
- UC3.2: DTP in transaction 11, returns to SafeConAI UI the list of projects.
- UC4.1: DTP in transaction 3, returns to DCC the list of projects.
- UC4.2: DTP in transaction 3, returns to DigiTAR the list of projects.

These project-related data include: the project name, the project description, and the project ID. These data should be included in the semantic graph model of any COGITO project. These data are provided in text JSON form by the authentication layer of DTP. To link these text JSON data to the TTL RDF graph models produced by other RDF Generator tools this project-related information should be converted to TTL file format. This conversion is performed by the Project RDF Generator tool, which is the first ETL tool in the series of four ETL tools of block A, presented in Figure 1. The operation of the Project RDF Generator is described next.

### 3.2.1 Prototype Overview

Project RDF Generator tool essentially is a file translation component. It transforms a JSON text file in the input, containing project-related data, to an TTL RDF file in the output, as summarized in Figure 6.

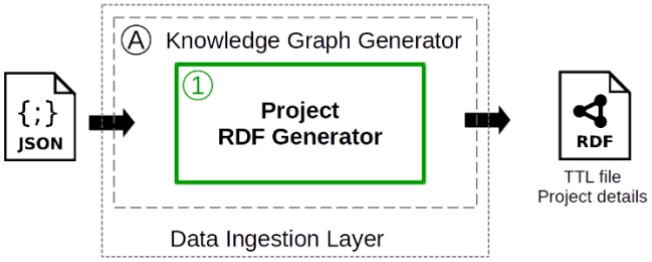



**Figure 6: Project RDF Generator operations**

The output TTL RDF file of Project RDF Generator, contains the triplets of project-related data classes of the input JSON file.

The operation of Project RDF generator is based the following series of processing steps:

1. Initially the input JSON file is parsed, and appropriate memory objects are created.
2. Based on facility domain ontology published at http://cogito.iot.linkeddata.es/def/facility an appropriate memory object is initiated.
3. Using the data from the created memory objects from step 1, the memory objects initiate in step 2 are populated.
4. The populated memory objects from step 3 are used to export the final TTL RDF file in the output of the tool.

### 3.2.2 Technology Stack and Implementation Tools

**Table 2 – Libraries and Technologies used in Project RDF Validator**

| Library/Technology Name | Version | License |
|---|---|---|
| Stellar-base-sseclient | 0.0.21 | MIT License |
| Wheezy.template | 3.1.0 | MIT License |

### 3.2.3 Input, Output, and API Documentation

The input of Project RDF Generator is a JSON file containing the project-related information obtained from the authentication layer of DTP. This information is organized inside the JSON file, using the following classes which are highlighted with different colours in part 1 of Figure 7:

- **project_id:** a string representing the project identifier.
- **project_name**: a string representing the project name.
- **project_description:** a string containing the paragraph of the project description.

The output produced by the Project RDF Generator is an TTL RDF file which is displayed part 2 of Figure 7. This output file contains three triplets which are referring to the classes of the JSON input file. This RDF output file together with the TTL RDF output files of the other three RDF Generator tools described in the following subsections, will be linked together forming the COGITO semantic graph model for every construction project. This model will be stored later in a triple store, where it can be queried by SPARQL queries originated from other COGITO applications.


**Figure 7: Input/Output of Project RDF Generator**

### 3.2.4 Application example

The use of this tool starts with the tool itself reading from a queue of server sent events, where events are published to perform certain actions. The project RDF generator, in particular, is in charge of collecting the events related to the generation of knowledge graphs of the projects to be registered in the platform. Once the event pertaining to the creation of the project is collected, it extracts the information encapsulated in JSON format, to generate the graph arranged by means of the Helio[1] translation tool, together with previously defined mappings. When the translation process is finished, the data generated in the knowledge graph is validated in RDF format and if the validation is positive, the RDF is sent to the system that has published the event to later save the knowledge graph in a Triple-Store.

### 3.2.5 Licensing

The Project RDF Generator is a part of the KGG component of DTP. All KGG sub-components are open-source and licenced under the Apache Licence 2.0.

### 3.2.6 Installation Instructions

The installation instructions are simply to deploy a docker container along with a configuration file indicating the type of event to pick up from the server sent events queue, and the endpoints of the Helio tool and the server providing the events.

### 3.2.7 Development and integration status

Currently, the status of the development and integration of the tool into the system is in progress. The local operation of the tool works correctly, but there are still several functionalities to be included, such as the validation of the graphs generated by means of SHACL files.

At the moment, the tools are tested individually, without considering the output of other RDF Generator components of block A in Figure 1. As integration of COGITO components progresses and as data from the COGITO demonstration construction sites become available, the tool is going to be tested on real data and its output will be linked to other TTL RDF data forming a unified semantic graph for every construction project. Updated results considering these later developments will be presented in the second version of this deliverable D7.4 in M24.

### 3.2.8 Requirements Coverage

The coverage of requirements includes the generation and validation of knowledge graphs in RDF format for the projects to be registered in the Digital Twin Platform.

### 3.2.9 Assumptions and Restrictions

The operation of the Project RDF Generator tool relies on the following three assumptions and restrictions:

[1] Cimmino, A., & García-Castro, R. Helio: a framework for implementing the life cycle of knowledge graphs.

- The JSON obtained from the event queue should only contain the information regarding the project identifier, name and description in a correct string format.
- The translation will be performed asynchronously to the retrieval of the data received from the Server Sent Events queue.
- The identifier of the project must be a UUID.

## 3.3 Construction RDF Generator

According to the use cases defined in D2.4 and updated in D2.5 there are several COGITO applications requiring access to the information related to the elements of COGITO's input 3D BIM files. These tools require not only the geometric descriptions of these construction elements but also properties associated to them such as tasks, material properties, point clouds and others. Within the input 3D BIM files these elements together with their geometry and properties, are organized in a hierarchical tree-like structure formed by containment relations. In this structure the overall construction site is placed at the root of the tree followed by the construction facilities located at the intermediate branches of the tree and finally reaching the elementary facility parts (or construction components which cannot subdivided further) placed at the end leaves of the tree. Inside this tree-like structure of the input 3D BIM files, the semantic links of the construction elements are intertwined with the geometry and properties of these elements, resulting to inefficient query times.

Consequently, the semantic links of this tree structure of the construction elements, should be separated from the geometry and properties of these elements. There are many COGITO's use cases and respective tools which perform separate queries to the semantic links of this tree-like structure of the construction elements of the project, include:

- UC2.1: Geometric QC tool in transaction 16, requests from DTP point cloud data and as planned 4D BIM data including geometry related to specific construction elements.
- UC2.2: Visual Data Pre-processing tool in transaction 7, requests from DTP construction components and tasks.
- UC4.1: DCC tool in transaction 5, requests from DTP as planned 4D BIM data including geometry.
- UC4.2: DigiTAR tool in transaction 5, requests from DTP QC/3DBIM with geometry data and in transaction 10, requests H&S/3DBIM with geometry data.

To support the generation of a tree-like structure of semantically connected construction elements of a construction project within COGITO framework, the construction element related data structures contained in COGITO's 3D BIM input files, excluding geometric data, are translated into a semantic graph in the form of an RDF file. This data translation process is performed by an ETL tool called Construction RDF Generator which is analyzed in the following subsections.

### 3.3.1 Prototype Overview

Briefly, the Construction RDF generator produces a semantically linked RDF file in TTL format, according to the construction domain ontology defined in D3.2, that contains only the semantic links of the construction elements (without geometry). These links are extracted from the 3D BIM data contained in the tool's input IFC file. The operation of the Construction RDF Generator tool is outlined in block diagram of the following Figure 8.

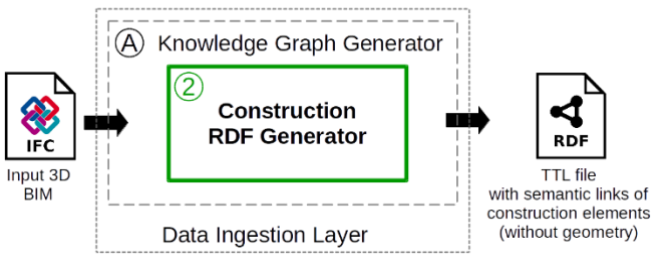



**Figure 8: Construction RDF Generator operation**

The operation of Construction RDF generator is based on the following series of processing steps:

1. The schema version of the input IFC file is detected.
2. Based on the detected schema version appropriate classes are used for the IFC data deserialization performed by the IFC Parser component of the data ingestion layer of DTP. After the deserialization finishes depending on the version all existing IFC classes containing data from the input IFC file are populated on memory.
3. The set of the construction elements is extracted from the loaded IFC classes on memory including their properties such as (area, volume, related construction task and possible containment relations to other construction elements).
4. Based on the COGITO ontology described in D3.4 appropriate memory objects related to the construction domain ontology are initiated.
5. Using the data extracted from step 3, the memory objects created in step 4, are populated.
6. The populated memory objects from step 5 are used to export the final TTL RDF file in the output of the tool.

### 3.3.2 Technology Stack and Implementation Tools

**Table 3 – Libraries and Technologies used in Construction RDF Generator**

| Library/Technology Name | Version | License |
|---|---|---|
| **BIM Library** | 1.0.0 | Open source |
| Stellar-base-sseclient | 0.0.21 | MIT License |
| Wheezy.template | 3.1.0 | MIT License |

### 3.3.3 Input, Output, and API Documentation

The construction RDF Generator tool receives as input an IFC BIM file in IFC4 format (ISO 16739, 2016) (including the latest IFC4x3 schema) as illustrated in the screenshot in part 1 of Figure 9 that refers to an IFC4 file exported from Autodesk's Revit 2021. IFC is an openBIM ISO standard produced by buildingSMART international. The IFC files conforming to this standard follow the EXPRESS data format as displayed in the screenshot of part 1 of Figure 9. These files can act as a common data repository of construction related projects, since the latest IFC4x3 schema has extended the data coverage from the building domain to the rail, road and bridge data domains. In this sense the IFC4 files used as input to the Construction RDF Generator contain geometric as well as non-geometric information of all the elements which are related to a specific construction project.

The geometric content of the IFC file which includes the parametric as well as the non-parametric geometric representations of all of the construction elements are processed by the BIM management component of DTP to be converted to a graphics-friendly format for visual purposes by the B-rep generator tool of the Data Post-processing layer of DTP analysed in section 4.2.

The non-geometric content of the IFC file includes properties of the construction objects some of which are used by the Construction RDF Generator tool, such as area and volume properties. Additionally, the non-geometric content of the IFC file contains relations among the construction elements which are also used by the Construction RDF Generator tool, such as the containment relation.

The output of the Construction RDF Generator tool a TTL RDF file a screenshot of which, is presented in part 2 of Figure 9. This TTL RDF file contains the construction elements of the input IFC file described as subjects which are modelled using appropriate bot or beo elements according to COGITO ontology described in D3.3. The non-geometric properties of respective IFC construction elements related to these subjects are described as triplets of these subjects in this output TTL RDF file. As we can also see in part 2 of Figure 9, and to facilitate the addition of the fourth (time) dimension these IFC construction elements and their respective TTL subjects, are also linked to construction tasks via the triplet **const:involvesTask**. Furthermore, the elements are organized in a tree-like

structure based on containment relations (one element is placed inside another) modelled as triplets using the **bot:containsElement** and **brick:isPartOf** predicates. These containment relations define the overall tree-like structure of COGITO's 3D BIM data.

To reduce verbosity appropriate prefixes are added in the header of the output TTL RDF file referring to the used ontologies.


**Figure 9: Input/Output operation of Construction RDF Generation tool**

The elements contained in the TTL RDF output of the Construction RDF Generator have the following triplets:

- **brick:isPartOf:** Points to the element containing the current element. For example, it can be a building storey containing the current element.
- **bot:containsElement:** Points to the element(s) which are contained in the current element.
- **const:involvesTask:** Points to the construction task related to the current element.
- **props:hasCompressedGuid:** Points to the string of the IFC GUID of the current element.

- **props:hasName:** Points to the string of the name of current element in the IFC input file.
- **props:hasVolume:** Points to a double number which is equal to the volume of the current element.
- **props:hasArea:** Points to a double number which is equal to the area of the current element, if the element is aligned to a horizontal plane (for example floor slab).

### 3.3.4 Application example

An example of a TTL RDF file produced by the Construction RDF generator is displayed in Figure 10. In this example the BIM IFC4 file of a demo building construction pictured in part B of Figure 23, is used as input to the Construction RDF Generator. To check the construction elements TTL RDF triplets produced by the tool and the hierarchical tree-like structure of the construction elements, we examine two elements in random. These are: an IfcColumn element with GUID: 2UD3D7uxP8kecbbBCRtzC2 and an IfcBuildingStorey element with GUID: 0hozoFnxj9leOc81mNbdSw. In this example the IfcBuildingStorey element contains the IfcColumn element. The definition of these elements in the IFC4 text file is displayed in part 1 of Figure 10.


**Figure 10: Example of TTL RDF file generation by the Construction RDF generation tool (1: IFC input data, 2: TTL RDF data).**

The output of the Construction RDF Generator tool for these examined elements is displayed in part 2 of Figure 10. As it is displayed in the TTL RDF screenshots of part 2 of Figure 10, the Construction RDF Generator not only generates the examined elements as elements of beo and bot ontologies, but also creates triplets referring to their IFC GUID via the **props:hasCompressedGuid** and their containment relation (the BuildingStorey element has a triplet: **bot:containsElement** which points to the column element).

### 3.3.5 Licensing

The Construction RDF Generator is a part of the KGG component of DTP. All KGG sub-components are open-source and licenced under the Apache Licence 2.0.

### 3.3.6 Installation Instructions

The installation instructions are simply to deploy a docker container along with a configuration file indicating the type of event to pick up from the server sent events queue, and the endpoints of the Helio tool and the server providing the events.

### 3.3.7 Development and integration status

Currently the Construction RDF Generator has been tested on a BIM IFC files related to a building construction site the geometric content of which is displayed in part B of Figure 23: Examples of triangulated BRG's output:

(A) Road network, (B) Building construction site. In a later stage in the project and as BIM data from COGITO demonstration sites become available the tool is going to be tested on the respective IFC data files of these sites.

Up to now, the tool is tested as an isolated component and its output is verified using imaginary and artificial demo IFC files as input. As the integration of COGITO platform components progresses, the output of this tool will be semantically linked to the output of the other RDF generation tools of block A of Figure 1.

The output TTL RDF data files of the tool when applied on IFC BIM files from COGITO demonstration sites and their semantic links to other TTL RDF data produced by the other RDF generator components, is expected to be presented in the second version of this deliverable D7.4 in M24.

### 3.3.8 Requirements Coverage

The correct operation of the Construction RDF generator requires the existence of an BIM IFC file according to the IFC4 schema with the IFC4x3 extension in case the construction project contains non-building components such as roads, rails and bridges. Currently these specifications cover the tool's requirements in terms of modelling of the construction elements.

To form the hierarchy tree of construction elements in the TTL RDF output file of the tool, necessary element containment relations are required to be present in the input IFC BIM file of the tool. To ensure the existence of these relationships, the following two proactive actions can be performed:

- Issuing of appropriate design guidelines to the person responsible for developing the IFC BIM file, that guide him to define these relations using an appropriate BIM authoring software. For example, the developer of the IFC BIM file can define using a BIM authoring tool, such as Revit, zones containing construction elements.
- Checking whether the IFC exporter of the BIM authoring tool exports these containment relations in its output IFC file.

### 3.3.9 Assumptions and Restrictions

The operation of the Construction RDF Generator tool relies on the following assumptions and restrictions applied on the input IFC BIM file:

- The input IFC BIM file should conform to the IFC4 standard (ISO 16739, 2016).
- The input file IFC file should contain at least one construction element.
- To form a hierarchy tree of construction elements in the output TTL RDF file of the tool, spatial elements such as construction spaces and zones, should be connected with non-spatial construction elements with appropriate containment relationships, in the input IFC file.
- All construction elements contained inside the input IFC file should be associated with a construction task id.
- Material types of all construction elements contained inside the input IFC file, should be present e.g. cement, steel.
- All required property values of the construction elements in the input IFC should be included in the appropriate format e.g. integer, string.

## 3.4 Process RDF Generator

The overall work of a construction project is split into segments defined as construction tasks. The work-related to each construction task is expected to be realized within a predefined time interval defined by start and end time instances. All construction tasks are organized in a tree-like structure following parent-child relation scheme. Tasks are distributed horizontally (modelled as children of the same parent at the same branch level of the tree) according to a time execution sequence (one after another). Tasks are also distributed vertically where one task (parent) has multiple sub-tasks (children). At a higher-than-task level, is the workflow process which contains a set of tasks. The workflows of the construction projects are defined for managerial purposes, as some of them are independent of each other and can be executed in a parallel fashion. The workflow is the parent

which has multiple tasks as children. At the highest (root) level the overall construction project can be defined as a single workflow. This scheduling information (workflows and sets of tasks and their horizontal and vertical relations) is contained in a schedule file which is provided as input to the COGITO framework. This schedule file is provided in either XML or CSV file formats.

Within COGITO framework there are several use cases and respective tools which require access to this workflow and task and subtask tree structure. These include:

- UC1.1: PMS tool in transaction 5, requests 4D BIM data from DTP. These data contain this construction schedule information.
- UC2.2: Visual Data Pre-processing tool in transaction 7, requests from DTP building components and tasks.
- UC4.1: DCC in transaction 5, requests from DTP as planned 4D BIM data.
- UC4.1: DCC in transaction 7, requests from DTP workflow progress.

To support these task-related queries in the above use cases in an efficient and less time-consuming manner, the construction schedule information contained in the input XLM or CSV files of COGITO should be translated into a semantic graph format and linked to respective construction elements contained in the semantic graph output of the Construction RDF Generator (TTL file containing the semantic links of the construction elements without geometry), presented in the previous subsection. The tool that performs this translation is defined as the Process RDF Generator and is described in the following subsections.

### 3.4.1 Prototype Overview

The operation of Process RDF Generator is summarized in the block diagram of Figure 11. Essentially, Process RDF Generator takes as input the construction schedule input file (in XML or CSV format) and populates the data classes according to the COGITO's process domain ontology, forming an output RDF file in TTL file format.

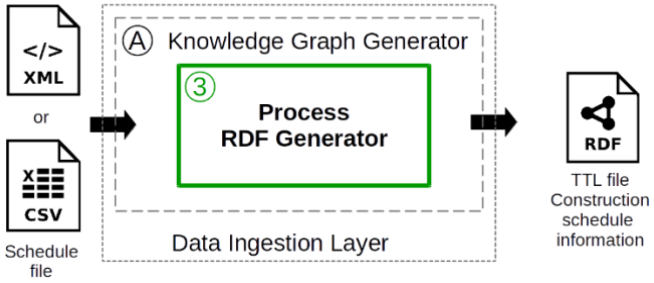



**Figure 11: Process RDF Generator operation**

The operation of Process RDF generator is based on the following series of processing steps:

1. Depending on the type of the input file an appropriate parser software module is used to create populated memory objects for every construction task.
2. Based on the COGITO ontology described in D3.4 appropriate memory objects related to the process domain ontology are initiated.
3. Using the data from the populated memory objects from step 1, the memory objects created in step 2 are populated.
4. The populated memory objects from step 3 are used to export the final TTL RDF file in the output of the tool.

### 3.4.2 Technology Stack and Implementation Tools

**Table 4 – Libraries and Technologies used in Process RDF Generator**

| Library/Technology Name | Version | License |
|---|---|---|
| Stellar-base-sseclient | 0.0.21 | MIT License |

| | | |
|---|---|---|
| Wheezy.template | 3.1.0 | MIT License |

### 3.4.3 Input, Output and API Documentation

Process RDF Generator input XML or CSV file contains the information related to the construction process (construction schedule and related tasks), which is going to be translated into RDF format in the output of the tool.

The input information in the CSV or XML file, is organized in a tree-like structure, containing the project identifier, the URL of the file containing the information to be translated, and more metadata related to construction tasks. Tasks are arranged horizontally in terms of time scale (one task is executed after another task), as well as vertically (each task consists of subtasks). These arrangements are modelled by the the Work Breakdown Structure (WBS), Predecessor and the SubTask task properties. Part 1 of Figure 12 indicates, the properties of each task which are contained in the input to the Process RDF Generator tool, XML file:

- **ID:** Is an integer used for identification of each task.
- **Status:** A string indicating the status of the task such as “Active”.
- **Task_Name:** Is a string describing the name of a task.
- **Duration:** Is a value indicating the duration of the task in time units.
- **SubTasks:** Is a list of the subtasks of the task.
- **Predecessor:** Is a task which is before the current task in the time sequence.
- **Outline_Level:** Is the level of inheritance of a task, i.e. how many parent tasks are above the current task.
- **Is_related_to:** A list of BIM element IDs related to this task.
- **Start_Date**: A string indicating the start time of the task based on the time ontology.
- **Finish_Date**: A string indicating the end time of the task based on the time ontology.
- **WBS:** Is the Work Breakdown Structure of the task containing the higher w.r.t the present task, task IDs in the task hierarchy.

The contents of the output file produced by the RDF Generator is shown in part 2 of Figure 12. This output file is a TTL RDF file, which contains the triples referring to the previous properties of each Task. This RDF file instance generated by the Process RDF Generator tool, will be stored later in a triple store where it can be queried by SPARQL queries.

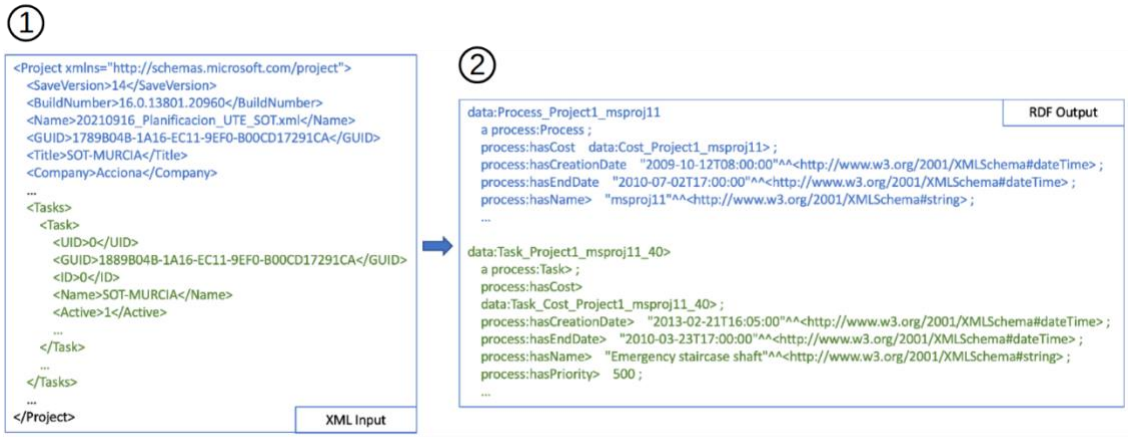



**Figure 12: O/I file ETL operations of the Process RDF Generator tool**

### 3.4.4 Application example

An example of triple generation by the Process RDF generator component, in a leaf and parent task elements the is presented in parts 1 and 2 of Figure 13 respectively. What differentiates the triplets of a leaf task from the triplets of a parent task, as it is indicated by the dashed rectangles in parts 1 and 2 of Figure 13, is the existence of a hasPredecessor triplet in the leaf task and the existence of the hasSubTask triplet in the parent task. In case both hasPredecessor and hasSubTask exist in a task element, the task element is an intermediate task node.

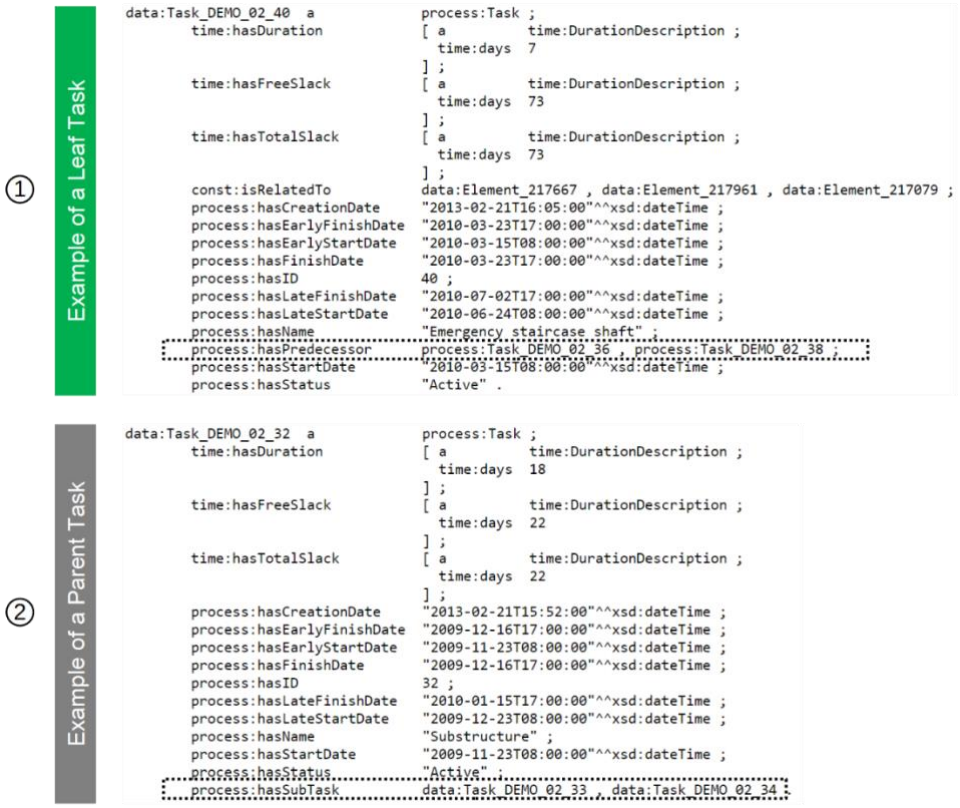



**Figure 13: Example of triple generation of a Leaf and Parent Tasks by the Process RDF generation tool**

### 3.4.5 Licensing

The Process RDF Generator is a part of the KGG component of DTP. All KGG sub-components are open-source and licenced under the Apache Licence 2.0.

### 3.4.6 Installation Instructions

The installation instructions are simply to deploy a docker container along with a configuration file indicating the type of event to pick up from the server sent events queue, and the endpoints of the Helio tool and the server providing the events.

### 3.4.7 Development and integration status

Currently, the status of the development and integration of the tool into the system is in progress. The local operation of the tool works correctly, but there are still several functionalities to be included, such as the validation of the graphs generated by means of SHACL files.

In addition, at the moment only the pre-processing and translation of schedule files in XML format is performed, in the future, the translation of new file formats, also belonging to the field of process-related files, will be carried out.

Additionally, the Process RDF Generator has been tested using artificially created construction schedule CSV files as input. In a later stage in the project and as time schedule data from COGITO demonstration sites become available, the tool is going to be tested on these data from the real construction sites.

Furthermore, the tool is tested as an isolated component and its output is examined without considering the output of other components. As the integration of COGITO platform components progresses, the output of this tool will be semantically linked to the output of the other RDF generation tools of block A of Figure 1. The results of the tool when applied on time schedule data from the real construction sites and their semantic links to other TTL RDF data produced by the other RDF generator components, is expected to be presented in the second version of this deliverable D7.4 in M24.

### 3.4.8 Requirements Coverage

The coverage of requirements includes the pre-processing, generation and validation of knowledge graphs in RDF format for the process-related files to be registered in the DTP.

### 3.4.9 Assumptions and Restrictions

The operation of the Resources RDF Generator tool relies on the following assumptions and restrictions applied on the input CSV or XML files:

- The input file should contain the data related to the construction tasks. At least one task should be present at the root level that should be related to at least one construction element, with its property values completed.
- In the input file, for every construction task all the properties should have values in the appropriate format (integer, string…).

## 3.5 Resources RDF Generator

Every construction project requires a certain number of human and non-human resources to be completed. Within COGITO this resources-related information is described in an input CSV or XML file together with the description of the construction schedule's tasks. This information contains resource types and not instances of the actual resources. For example, it determines that a specific task is going to be executed by a crane without describing a specific crane instance. The binding of the construction tasks with the available resource instances is performed by the WODM tool, after the optimization operations carried out by the PMS tool have been completed.

Within COGITO the tools and respective use cases requiring access to construction resource-related information are:

- UC1.1: PMS in transaction 5, requests 4D BIM data and respective resources from DTP.
- UC1.1: WODM in transaction 24, requests as-planned resource data from the DTP.
- UC1.2: WODM in transaction 25 sends updated workorder data containing resource allocation information to DTP.

To support the resource-related data queries within COGITO and their associations with construction schedule tasks and construction elements, this resource-related data should be translated in to a TTL RDF form and linked to other TTL RDF files produced by the previous RDF Generator tools (Project, Construction and Process RDF Generators), to form the COGITO semantic graph model. This TTL RDF translation of the project resource-related information, from the native XML or CSV format is performed by the Resources RDF Generator tool, as analysed in the following subsections.

### 3.5.1 Prototype Overview

The Resources RDF Generator, as the RDF generator tools analysed in the previous subsections, is a file translation tool which transforms construction resources-related data in XML or CSV format, to TTL RDF file format, as summarized in Figure 14.

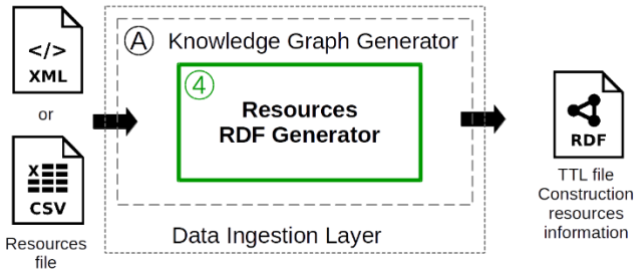



**Figure 14: Resources RDF Generator operation**

The operation of Resources RDF generator is based on the following series of processing steps:

1. Depending on the type of the input file an appropriate parser software module is used to create populated memory objects for every construction resource type.
2. Based on the COGITO ontology described in D3.4 appropriate memory objects related to the resource domain ontology are initiated.
3. Using the data from the populated memory objects from step 1, the memory objects created in step 2 are populated.
4. The populated memory objects from step 3 are used to export the final TTL RDF file in the output of the tool.

### 3.5.2 Technology Stack and Implementation Tools

**Table 5 – Libraries and Technologies used in Resources RDF Generator**

| Library/Technology Name | Version | License |
|---|---|---|
| Stellar-base-sseclient | 0.0.21 | MIT License |
| Wheezy.template | 3.1.0 | MIT License |

### 3.5.3 Input, Output, and API Documentation

Resources RDF Generator input XML or CSV file contains information related to types of construction resources (human and non-human) which are going to be translated into RDF format in the output of the tool. File instances of the tool's input and output files are presented in Figure 15.

As presented in part 1 of Figure 15, the input CSV or XML file instances contain information in the following fields:

- **ID:** Is an integer used for identification of the specific resource type.
- **Resource_Name:** Is a string describing the name of the resource type.
- **Type:** Is a string describing the type of the resource (equipment or people). This is used to differentiate human and non-human resources.
- **Initials:** Is an integer used for the identifying a specific resource within its type.
- **Max_Units**: Is an integer identifying the maximum allowable number of instances of a specific resource type.
- **Cost_Per_Use:** It is a numerical value identifying the cost of using the specific resource type. If the resource type is a worker for example, this is the amount of currency required per hour for the use of the specific worker.

The output of produced by the Resources RDF generator is shown in part 2 of Figure 15, where we can see the triples reflecting the information in RDF format associated to the input process CSV or XML file and the related resource type instances. This RDF file generated will be stored later in a triple store where it can be queried by SPARQL queries.

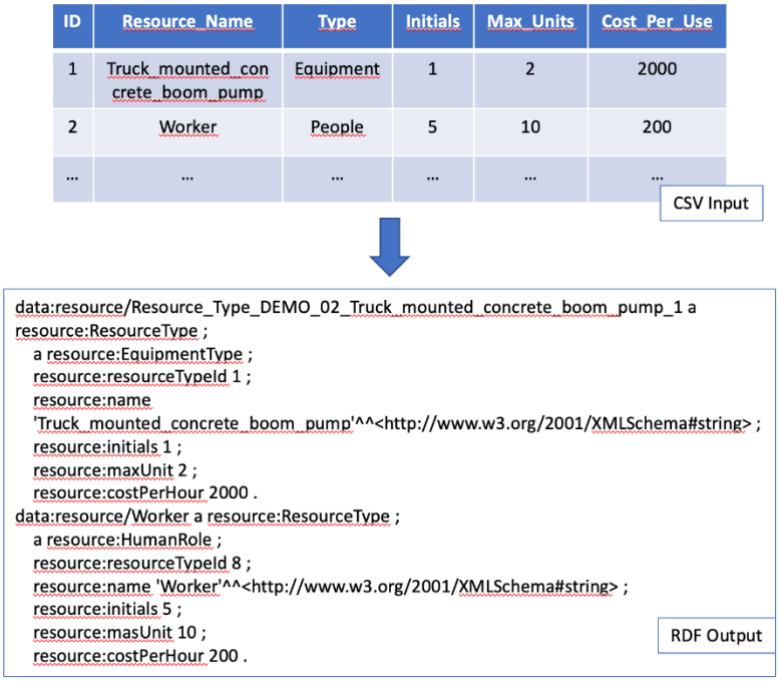



**Figure 15: O/I file ETL operations of the Resources RDF Generator tool**

### 3.5.4 Application example

An application example of the Resources RDF Generator's ETL file operations is illustrated in Figure 16. In part 1 of Figure 16, an example of a CSV file instance, which is used as input to the Resources RDF generator is presented. In this CSV file a row is highlighted which refers to an excavator resource type. This excavator resource type is translated in TTL RDF file format in the output of the tool as presented in part 2 of Figure 16. The portion of the output TTL RDF file referring to this excavator resource instance contains multiple triplets. Each triple refers to each of the properties (columns in the input CSV file) of this resource type. The five translated properties as triplets, for this resource type, are: hasCostPerUse:1500, hasID: 4, hasInitials:1, hasMaxUnits:2 and hasName: "Excavator".

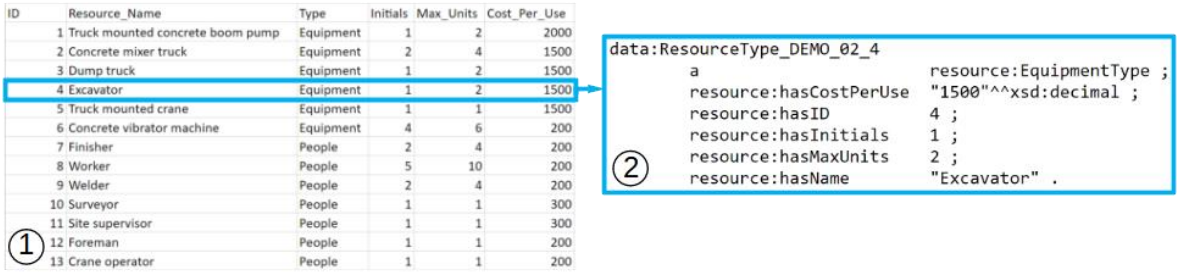



**Figure 16: Illustration of the I/O file operations of the Resources RDF Generator tool.**

### 3.5.5 Licensing

The Resources RDF Generator is a part of the KGG component of DTP. All KGG sub-components are open-source and licenced under the Apache Licence 2.0.

### 3.5.6 Installation Instructions

The installation instructions are simply to deploy a docker container along with a configuration file indicating the type of event to pick up from the server sent events queue, and the endpoints of the Helio tool and the server providing the events.

### 3.5.7 Development and integration status

Currently, the status of the development and integration of the tool into the system is in progress. The local operation of the tool works correctly, but there are still several functionalities to be included, such as the validation of the graphs generated by means of SHACL files.

In addition, only the pre-processing and translation of resources files in CSV format is currently performed, in the future, the translation of new file formats, also belonging to the field of process-related files, will be carried out.

The Resources RDF Generator has been tested using artificially created construction resource-related CSV files as input. In a later stage in the project and as resource type data from COGITO demonstration sites become available, the tool is going to be tested on these real data.

Furthermore, the tool is tested as an isolated component and its output is examined without taking into account the output of other RDF Generator components. As the integration of COGITO platform components progresses, the output of this tool will be semantically linked to the output of the other RDF generation tools of block A of Figure 1. The results of the tool when applied on resource-related data from the real construction sites and their semantic links to other TTL RDF data produced by the other RDF generator components, is expected to be presented in the second version of this deliverable D7.4 in M24.

### 3.5.8 Requirements Coverage

The coverage of requirements includes the pre-processing, generation and validation of knowledge graphs in RDF format for the resources-related files to be registered in the Digital Twin Platform.

### 3.5.9 Assumptions and Restrictions

The operation of the Resources RDF Generator tool relies on the following assumptions and restrictions applied on the input CSV or XML files:

- The input file should contain the data related to the construction resource types. At least one resource type should be present with its property values completed.
- In the input file, for every construction resource type all the properties should have values in the appropriate format (integer, string,…).
- The process identifier must be sent as a parameter to the tool to be added to the existing data in the resource file.

# 4 ETL and MC tools in Data Post-processing layer of DTP

As presented in Figure 2, three tools of the Data Post-processing layer (block B) apply ETL and MC operation on COGITO input BIM IFC files. These operations are more hardware resource demanding than simple ETL operations performed by the RDF Generation tools. To support this requirement, the Data Post-Processing Layer uses Docker Swarm to orchestrate the multiple running instances of the deployed components. The current version of this layer includes three tools: (i) the B-rep Generator component, which produces mesh-based representations of BIM data that conform to the IFC4x3 standard, (ii) the MVD Checker, which validates 4D BIM models by checking the existence of the task identifier of the construction schedule in the property sets for each building, and (iii) the IFC Optimiser which performs lossless compression and is capable to significantly reduce the original file size of an IFC to speed up the data transformation processes. For fast and efficient execution, as it is also illustrated in Figure 2, these tools communicate asynchronously with the Data Persistence layer of DTP via the Data Management layer of DTP and are executed in a containerized and parallel fashion using the ESB. These three tools are described analytically in the following subsections.

## 4.1 MVD Checker

COGITO's input IFC BIM files contain diverse data ranging from pure geometric to non-geometric data that refer to construction elements and their related properties and associations. These data, although supported by the IFC schemas, are not always present in the input IFC files, due to several reasons. These reasons include human errors of the BIM designer, who might omit to enter them when developing the BIM, and non-human flaws due to bugs in BIM exporting software programs. Consequently, a checking mechanism is required to ensure that COGITO input BIMs have the necessary information for every COGITO application using BIM files as input.

This data quality checking mechanism is required in many use cases of COGITO since several COGITO applications use data originated from IFC BIM files. These applications and respective use cases are:

- UC1.1: PMS tool requires the 4D BIM file an especially the links between tasks and 3D BIM elements.
- UC2.1: Geometric QC requires 3D BIM element data containing material data to perform regulation and rule conformity checks.
- UC3.1: SafeCon AI requires access to BIM data to enrich them with H&S information.
- UC3.2: Proactive Safety uses 4D BIM data during the construction phase to prevent potential H&S hazards.
- UC4.1: DCC requires the 4D BIM data for 4D visualization purposes.
- UC4.2: DigiTAR requires access to BIM data for BIM element inspection and issue reporting.

This checking mechanism is offered within COGITO framework by an MC tool called MVD Checker which was also introduced in D7.1. MVD Checker is described in more detailed in the following subsections.

### 4.1.1 Prototype Overview

In a nutshell, MVD Checker ensures that a predefined subset of the populated IFC classes in the input IFC BIM files of COGITO are correct and complete data.

Essentially, MVD Checker, follows the MVD specification defined buildingSMART, which enables the view of only a subset of the overall IFC schema, to facilitate the data transactions among IFC data providers and consumers. This "subset view" of the overall IFC schema is based on a set of predefined Concept Templates and Rules. These Concept Templates and Rules are defined in an mvdXML file produced by buildingSMART's IfcDoc tool. The mvdXML file only defines the IFC data classes, their properties and connecting relations that are going to be checked. The checking rules defined inside the mvdXML file as ConceptTemplates, are essentially sub-graphs of the IFC schema which contain the necessary Concepts, Rules, and Constraints on those.

In a nutshell, to perform its model checking operations, MVD Checker requires two files as input:

- An input IFC BIM file.
- An mvdXML file according to the MVD specification which contains the checking rules (an IFC subset containing the required IFC classes, properties and inter-relations to be checked).

The output of MVD Checker is a JSON file containing the results of the checking process. These input/ output file operations and the position of the MVD Checker within the COGITO framework are illustrated in the following Figure 17.

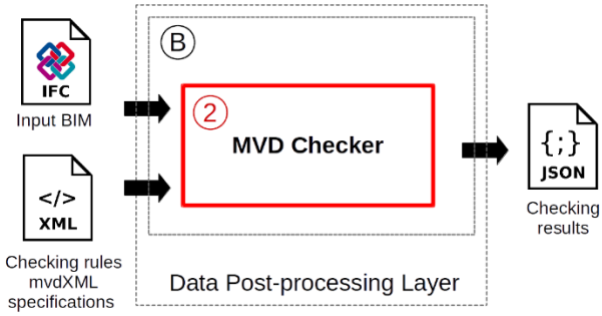



**Figure 17: MVD Checker operation**

To analyze further the operation of the MVD checker one has to look further in to the tool's architecture. The tool as a software block consists of 2 subcomponents: (a) the **MVD parser** and (b) the **MVD algorithm**. The MVD parser parses the input mvdXML as well as the into IFC file and generates respective memory objects. After the parsing of the input files is completed two steps follow:

1. Based on the values of the objects generated from the mvdXML file parsing, IFC search paths inside the IFC schema and checking rules are defined and isolated. These search graphs and rules define a set of IFC classes and attributes to be checked. Using these IFC classes and attributes of the search paths, certain memory objects derived from the IFC file referring to the IFC classes and attributes related are collected. Additionally, based on the mvdXML file parsing the checking rules referring to these isolated memory objects from the IFC file, are also collected.
2. Using these isolated memory objects originated from the IFC file, as well as the checking rules from the first step, the required checking operations are performed by the MVD algorithm component.

After the above both steps are completed for all the collected search paths and respective checking rules, the obtained results from the checking operations are written in the output JSON file.

### 4.1.2 Technology Stack and Implementation Tools

**Table 6 – Libraries and Technologies used in MVD Checker**

| Library/Technology Name | Version | License |
|---|---|---|
| BIM Library | 1.0.0 | Open source |

### 4.1.3 Input, Output, and API Documentation

The input to the MVD Checker tool consists of two files: (a)a BIM file according to IFC schema specification which was also input to Construction RDF Generator as described in subsection 3.3.3, which is going to be checked and (b) an mvdXML text file which contains information related to the checking rules to be used to perform the data quality checks. Input (a) was documented extensively in subsection 3.3.3. As illustrated in Figure 18, input (b) is an XML file containing two basic data items:

- **Concept Templates:** This item contains entity and attribute rules which are nested on one another and define a searchable path within the IFC interconnected classes and their attributes. Two of these concept templates are illustrated with numbered (1 and 2) dashed rectangles Figure 18.
- **Views:** This item defines the checking rules. Every view uses two additional items:
    - **Applicability:** Uses the Templates from the defined Concept Templates to isolate the IFC elements on which the checking rule is to be applied.
    - **Concept:** Contains the checking rules of the view defined by Template Rules applied on Templates obtained from the Concept Templates

  The views are illustrated in the last bottom screenshot of Figure 18.

The output of the MVD checker contains the results of the checking operations performed on the elements of the IFC file located using the routing paths defined in the applicability items inside the views section of the mvdXML input. These results are formed in JSON format as displayed in Figure 19. In the JSON output all entities defined in the input mvdXML document are checked. For every checked entity the JSON file reports the following values:

- **valid**: a boolean value (true/false) representing the checking results of the checked entity.
- **name:** Is a string describing the name of the checked element
- **global**: Is a 22-character string describing the IFC global unique identifier (GUID) of the checked entity.
- **express**: is an integer defining the express id of the checked entity.
- **rules**: a complex entity containing the rules that are checked for the current entity and the checking results for every rule individually. A rule further contains the following entities:
    - **valid**: a boolean value (true/false) representing the checking results of the current rule.
    - **description**: a string describing the current rule.
    - **parameters**: a complex entity containing the parameters of the current rule. Every parameter further contains:
        - **valid**: a boolean value (true/false) representing the checking results of the checked parameter.
        - **name**: a string describing the checked parameter.
        - **value**: a string describing the value of the current checked parameter.

The MVD checking results are communicated back to the COGITO application which initiated the checking process in order for appropriate corrective actions to be performed.

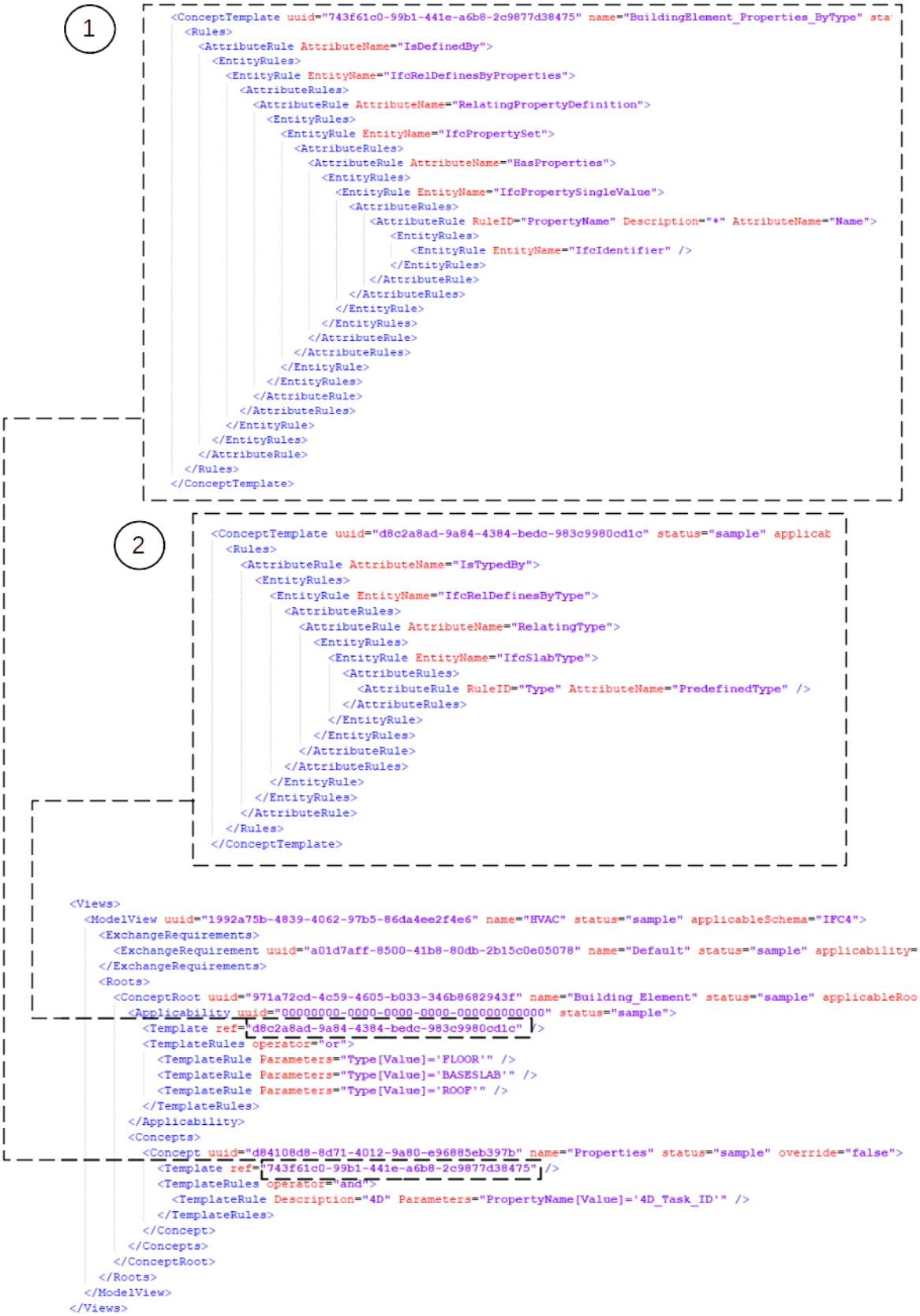



Figure 18: Input mvdXML file format of MVD Checker

```
[{
"entities":
    [{
    "valid":false,
    "name":"Floor:Generic Concrete 300mm:133345",
    "global":"2E3cuzY0LCc9ztHo7GNGl4",
    "express":151151,
    "rules":[{
            "valid":false,
            "description":"4D",
            "parameters":[{
                        "valid":false,
                        "name":"PropertyName",
                        "value":"4D_Task_ID"
                        }]
            }]
    },...]

}]
```


Figure 19: JSON output file format of MVD Checker

### 4.1.4 Application examples

The MVD checker tool is integrated in the UI of DTP allowing the user to visualize elements of the BIM IFC file that have issues in their related IFC classes (incomplete data classes or not semantically connected data classes). An example of such visualization is presented in Figure 20, related to a detected issue in a column element of an IFC file of a demonstration building which is highlighted with green colour.

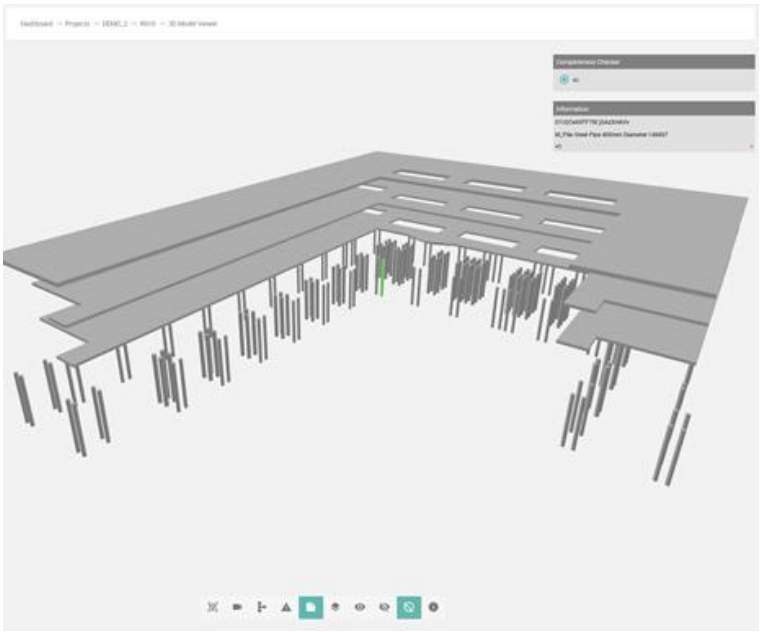


**Figure 20: Example of MVD Checker's issue reporting using DTP's UI**

To understand better how the MVD Checker works, the checking results of the demo construction IFC BIM file referring to the building pictured in part B of Figure 23, are presented in Figure 19 using the checking rules defined in the mvdXML file presented in Figure 18. As we see in the concept template displayed in part 2 of Figure 18, in this example, the three types of the IfcSlab elements which are checked are: FLOOR, BASESLAB and ROOF. Additionally, from the concept template displayed in part 1 of Figure 18, a certain property with description "4D" of these slab types, is checked whether it obtains the value "4D_Task_ID". To put this checking mechanism in fewer words, the IfcSlab elements of the IFC file of type FLOOR, BASESLAB and ROOF are checked whether they

are connected to construction task elements. In other words, the IFC file is checked whether it has a fourth (time) dimension for these slab elements.

### 4.1.5 Installation Instructions

The MVD Checker is an internal component of data post-processing layer of DTP. Consequently, there is no API involved and respective installation procedure.

### 4.1.6 Development and integration status

At present the MVD checker is successfully applied on the IFC BIM file related to demonstration building construction project which is pictured in part B of Figure 23. For this demo case the link between the 3D BIM data and the fourth (time) dimension for specific types of construction elements is successfully checked. The checking results of the MVD checker are also integrated with DTP's 3D Viewer for visualization purposes. This integration offers the user of DTP the ability to locate in 3D space problematic elements of the construction project that violate the checking rules.

Currently the MVD Checker is tested on demo IFC BIM files with artificially created checking rules. As the data requirements of COGITO tools become more solidified new data checking requirements will be added and new respective checking rules will be defined accordingly. With this plan in mind, the MVD checker will be tested with these new checking rules originated from COGITO tools, defined in mvdXML format. These new rules will define new separate checking procedures for the MVD Checker which will become even more complex and more diverse in nature, supporting the variable data needs of COGITO tools. This extension of the MVD checker to support the variable data needs of COGITO tools, will contribute to the integration and seamless and unified operation of the whole COGITO framework.

### 4.1.7 Requirements Coverage

Currently, the MVD Checker receives as input a openBIM file conforming to the IFC4 standard (ISO 16739, 2016) and an mvdXML file according to the MVD specification defined by buildingSMART (buildingSMART, 2018). It is required for the correct operation of the tool both input files to have a structure according to these specifications. Any violation of this well-defined structure of the input files, will result to termination of the tool's execution without any output file generation and any checking results reported.

### 4.1.8 Assumptions and Restrictions

For the correct operation of the tool, and especially if the mvdXML file is generated from a tool other than the IfcDoc tool, then it is assumed that the following conditions are satisfied:

- It should be ensured that the search path of the IFC elements or attributes inside the Rules of Concept Template definitions of the mvdXML file, should be according to the IFC4 schema (ISO 16739, 2016).
- The checking rule definitions inside the Concepts as well as the Applicability elements in the Views section of mvdXML should have the correct references to the defined Concept Templates.
- Finally the input IFC file should conform to the IFC4 standard (ISO 16739, 2016).

## 4.2 B-rep Generator

To avoid text verbosity, the geometric content of the 3D BIM IFC files is in an intricate and highly complex form that contains cyclic references among data classes conforming to the EXPRESS G specifications. Oftentimes, the IFC geometric content contains parametric geometrical descriptions which cannot be used directly for viewing purposes and require geometric transformations to become graphics friendly. The bigger the IFC file, the more the time and processing resources are required to produce files that can be used by 3D viewing software. This fact limits the use of whole IFC files as direct input to hand-held devices (oculus), due to size, hardware resources, and processing power limitations of these devices.

Within COGITO architecture, defined in D2.4 and updated in D2.5, certain use cases and respective tools require portions of the 4D BIM file, referring to specific time intervals, to be converted into a graphics friendly format. These use cases are:

- UC 4.1: DCC in transaction 5, requests from the DTP certain construction elements from the 4D BIM model to be displayed in computer displays.
- UC 4.2: DigiTAR in transactions 5 and 10, requests from the DTP certain construction elements from the 4D BIM model to be displayed in hand-held devices.

To support this on-demand graphics compatible file generation from 4D BIM data, an ETL tool called B-rep generator (BRG) is introduced, as a part of the Data Post-processing layer of DTP, displayed in block 5 in Figure 1. BRG is described in more details in the following subsections.

It is important to point out that although DCC and DigiTAR tools in UC4.1 and UC4.2 respectively, have implemented their own IFC to OBJ converters for development purposes using the ifcoOpenShell (IFCtoMesh Generator reported in D7.5) their final implementation might use the BRG tool.

### 4.2.1 Prototype Overview

The B-rep generator receives as input the static 3D BIM geometric content of COGITO's input openBIM IFC files and produces graphics-friendly files following the open OBJ file format. For a better viewing experience and to be compatible with graphics-related GPU hardware, the produced OBJ files, contain triangulated boundary representations of the 3D BIM elements augmented with the normal vectors of the generated triangle vertices. This representation can be used directly by the GPU hardware as it is in the simplest polygon form (triangle) and can be displayed in a parallel manner by the multiple GPU cores.

To produce these triangulated OBJ files, the B-rep Generator parses an input XML file produced by the IFC Geometry Exporter of the BIM management component of the Data Ingestion layer of DTP, containing the deserialized geometric descriptions of the 3D BIM elements (parametric and non-parametric). This OBJ generation process from an input IFC file is illustrated in Figure 21.

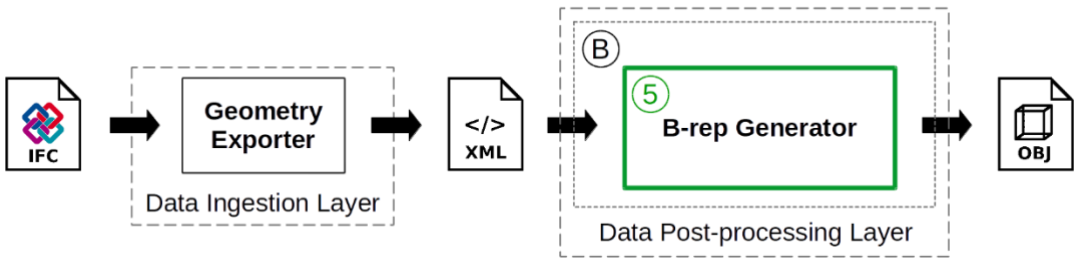



**Figure 21: OBJ generation process from input IFC files of B-rep Generator.**

The operation of B-rep generator is based on the following series of processing steps:

1. Initially the construction elements described inside the input xml file are separated depending on their geometric descriptions into parametric (having parametric geometric descriptions) and non-parametric elements (having non-parametric geometric descriptions).
2. The parametric elements are then passed to appropriate (internal to B-rep Generator ) conversion software components depending on the kind of parametric description of the element (extrusion, CSG Boolean operation, …). These conversion software components convert the parametric geometric description of the input element to a non-parametric geometric description, consisting of a set of surfaces (polygons with holes) in 3D space.
3. The non-parametric elements need no conversion as their geometric descriptions can be expressed as a set of surfaces (polygons with holes) in 3D space.
4. After the geometric descriptions of all elements (parametric and not) are expressed as sets of polygonal surfaces in 3D space, the resulting sets of these polygons are collected (one set per construction element).
5. Finally, every polygon from the formed sets in step 4, is converted into a set of triangles in 3D space using a fast ear-clipping polygon triangulation algorithm, which is implemented using the clipper library (Johnson, 2014), (a library based on Vatti's algorithm (Vatti, 1992)). Ear-clipping-based polygon triangulation works for polygons without holes. In case any polygon from the sets obtain from step 4 contains holes, it is converted to a degenerate polygon without holes according to (Mei, Tipper, & Xu, 2015) and then is triangulated. All the resulting triangles in 3D are gathered in separate triangle sets (a triangle set is formed from the triangles obtained from the polygons of a polygons set generated in step 4).
6. The sets of triangles obtained from the polygonal sets from step 5, are exported in the OBJ text output of the BRG tool (every triangle in a triangle set is written as a faces of the same object in the output OBJ of BRG tool).

### 4.2.2 Technology Stack and Implementation Tools

**Table 7 – Libraries and Technologies used in B-rep Generator**

| Library/Technology Name | Version | License |
|---|---|---|
| Clipper | 6.1.3 | Boost Software Licence |
| RapidXML | 1.13 | Boost Software Licence |

### 4.2.3 Input, Output, and API Documentation

B-rep Generator's input XML file contains deserialized geometric descriptions (IfcDefinitionShapes) of the elements of COGITO's input IFC4x3 file. These descriptions are organized in a tree-like structure as presented in blue colour, in the left part A of Figure 22. At the root level, this structure contains the description of the projects' district (if exists), followed by the building subtree description (if any), which is considered also a construction facility, and non-building facility subtrees (these subtree descriptions include roads, rails, bridges). Every facility subtree consists of facility parts that have certain definition shapes translated to their local coordinate systems according to location, direction, and axes vectors described in the tree structure as facility part end leaves.

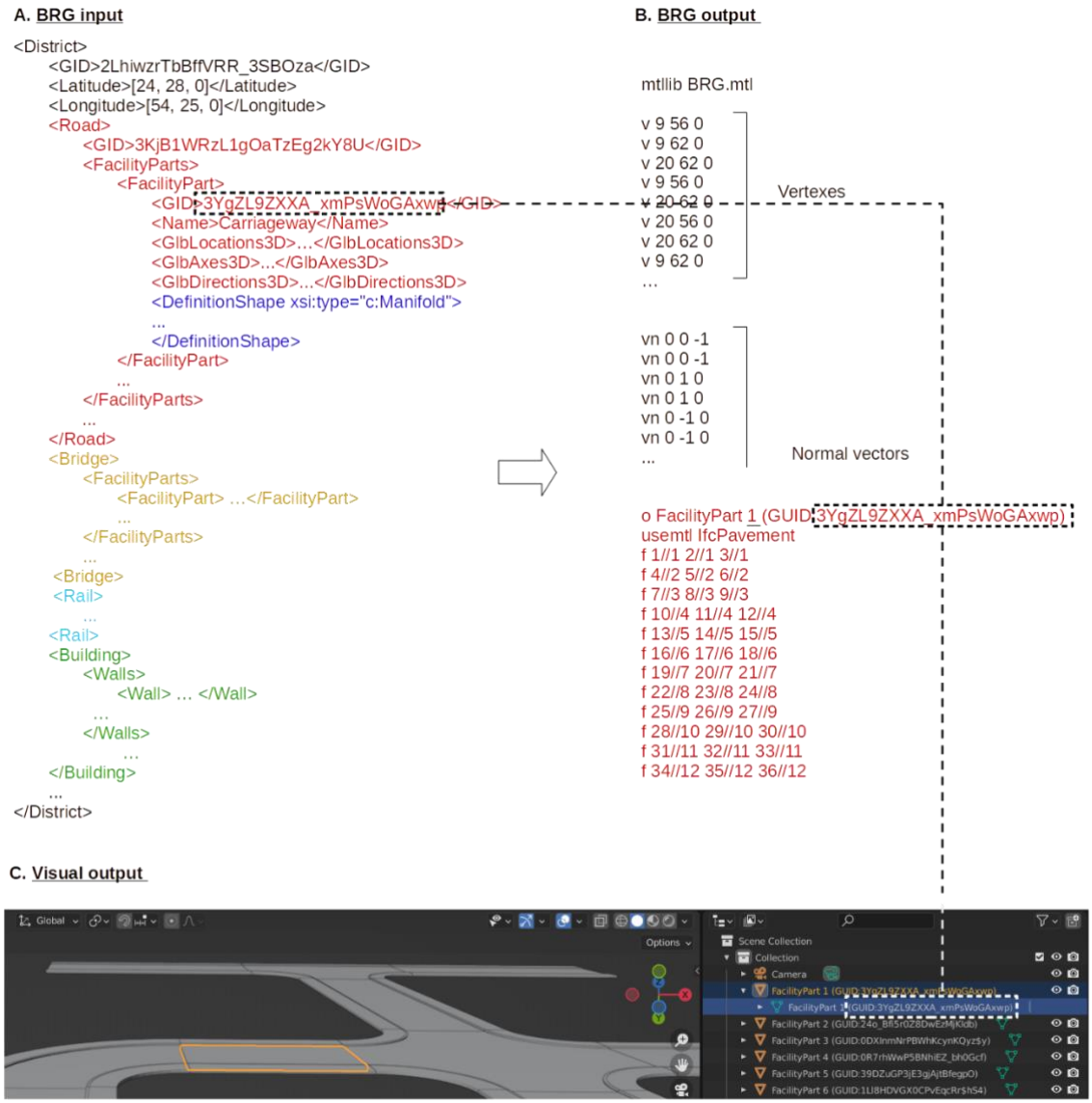



**Figure 22: Input and output of B-rep Generator tool.**

Based on the previous XML input, B-rep Generator produces a triangulated OBJ file at the output containing three fields: (a) the triangle faces of the boundary presentations of the facility parts or building elements (if any) of the input, (b) the vertexes of these faces, (b) the normal vectors of these faces, as displayed in the right part B of Figure 22 for a specific road segment. The objects at B-rep Generator's output, are linked to respective facility parts or building elements (if any) of the B-rep Generator's input, by the IFC GUIDs. This IFC GUID link is illustrated for a facility part related to a road segment with IFC GUID:3YgZL9ZXXA_xmPsWoGAxwp, with linked dashed rectangles across the parts of Figure 22. Finally, the output OBJ file and its elements can then be visualized using geometric mesh viewers such as Blender, as illustrated for a specific road segment in part C of Figure 22.

### 4.2.4 Application examples

Examples of BRG's triangulated OBJ file output are displayed in Figure 23. In the left, part A, of Figure 23, a set of triangulated boundary representations of a road network is displayed, which includes curb, pavement, and road segments exported in IFC4x3 format as facility parts. In the right, part B, of Figure 23, a set of triangulated boundary representations of a building construction site is presented which refer to the following structural elements: slabs walls, columns, and beams.

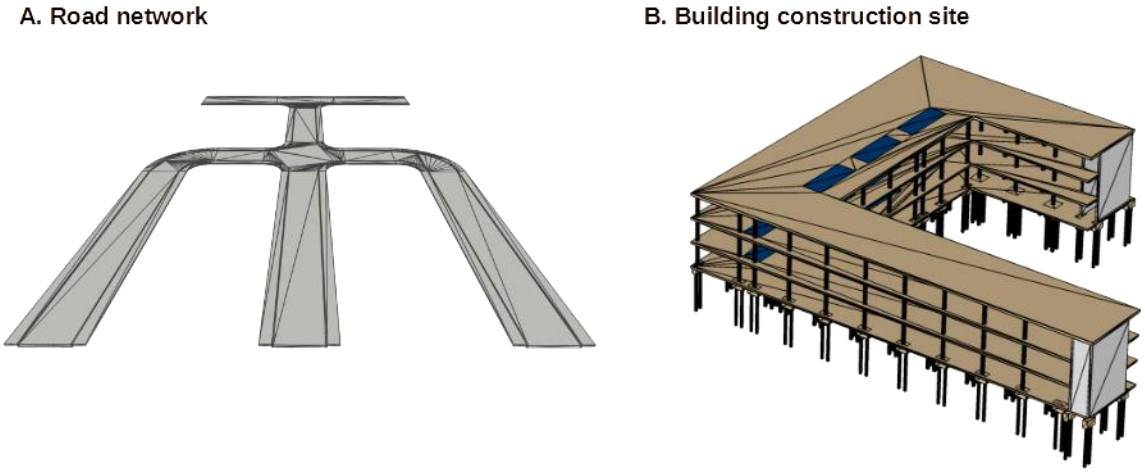



**Figure 23: Examples of triangulated BRG's output: (A) Road network, (B) Building construction site**

### 4.2.5 Installation Instructions

The B-rep Generator is an internal component of data post-processing layer of DTP. Consequently, there is no API involved and respective installation procedure.

### 4.2.6 Development and integration status

The initial version of BRG tool was designed for only building-related elements such as walls, curtain walls, slabs, coverings, roofs, beams, columns, and proxy elements and their respective Opening volumes with their included components (windows, doors, and plates). Aligned to the IFC4x3 specification, the tool is updated in the second version to support the added three facilities from the respective domains which are the road, the bridge, and the rail. For these facilities, the nested facility part abstraction IFC4x3 rationale was adopted to simplify the building domain complexity. Under the facility part at the leaf level, the tool's operations remained unchanged in the second updated version.

### 4.2.7 Requirements Coverage

Currently, the B-rep Generator covers the COGITO's user's 3D visualization requirements as it supports fast, accurate and on-demand, graphics-friendly BIM geometry file generation operations extended to non-building domains such as roads, rails, and bridges. The element of non-building domains is supported by the IFC4x3 schema as facility parts and their geometries are exported in OBJ by BRG.

### 4.2.8 Assumptions and Restrictions

The operation of the B-rep Generator tool relies on the following two assumptions:

- Only polyhedral geometric representations (solid geometric representations with no curved surfaces) in the definition shapes of the facility parts of the building elements (if any), contained in the tool's XML input file, are translated with no approximation in the tool's OBJ output.
- If curved geometric representations exist in the definition shapes of the facility parts and building elements (if any), are approximated linearly by polyhedral boundary representations in the OBJ file output.

## 4.3 IFC Optimizer

The IFC exporters of BIM authoring tools often generate multiple instances of the same object, increasing IFC text verbosity. This “same-object” repetition becomes even more frequent when the geometric content of IFC files is described. In this content, the same, points, lines and whole surfaces are repeated inside a single IFC file. The simple removal of these repeated instances is not possible as it will result to broken links of other entities referring to these removed instances, destroying the IFC file structure. Consequently, a more complex compression mechanism is required that removes not only the repeated objects but also their references from other FC entities.

Such IFC lossless compression operations are required within COGITO framework as many applications process large IFC files. Based on D2.4 and D2.5 the following use cases and respective tools require access to large IFC files:

- UC1.1: PMS tool requests from DTP as planned 4D BIM data in transaction 5.
- UC2.1: Geometric QC requests from DTP as planned 4D BIM data in transaction 16.
- UC2.2: DTP returns 3D BIM data to visual processing UI in transaction 5.
- UC3.1: SafeConAI tool requests from DTP as planned 4D BIM data in transaction 5 and sends also 4D BIM enhanced with safety info in transaction 10.
- UC3.2: Proactive safety requests as built 4D BIM data from DTP in transaction 1. SafeConAI requests as built data enhanced with safety info from DTP in transaction 13. SafeCon AI returns 4D BIM with updated safety info to DTP and SafeCon AI UI in transaction 16.
- UC4.1: DCC requests as planned 4D BIM data from DTP in transaction 5.
- UC4.2: DigiTAR requests from DTP 3D BIM data in transaction 5. DTP returns the requested 3D BIM data in transaction 6.

In all of these transactions the respective IFC files should be optimized by reducing their sizes without loosing content. Reducing the IFC file size without losing content will expedite the operations of these applications, in cases large IFC files are required to be processed. To remove the excess lines in IFC files while maintaining their content, a lossless compression tool is introduced as a BIM management component in COGITO’s post-processing layer (see Figure 1), called IFC Optimizer. The operational structure of IFC Optimizer is described next.

### 4.3.1 Prototype Overview

The IFC Optimizer performs a reduction in the total IFC file size, to accelerate the loading and execution processes of other BIM-related tools. The IFC Optimizer uses the IFC Java Library to compare the hash values of the IFC objects and to merge them in case they are equal. Furthermore, it updates the express identifiers of the deleted objects to maintain the original connections. This optimization operation is applied to an input $IFC_0$, to produce an output $IFC_1$ file as is illustrated in Figure 1.

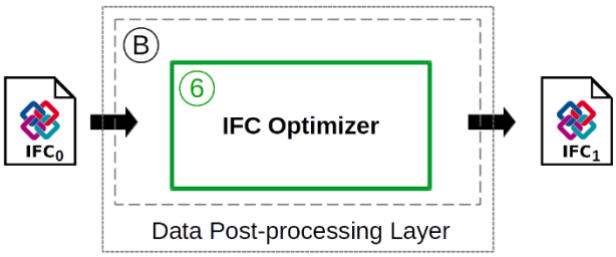



**Figure 24: IFC Optimizer operation**

The operation of IFC Optimizer is based on hashing techniques and relies on the following series of processing steps:

1. The IFC file’s Express IDs are converted into hash keys into a hash table.

2. All IFC text lines which are the same by string comparison and their Express IDs (respective hash keys), are isolated into sets of identical IFC text lines and respective hash key values. For every one of these sets one representative hash key value and respective IFC text line entry is kept.
3. For every set isolated in step 2, all references to its hash key values in the IFC text are replaced by the single representative hash key-value entry of the set. This is done to maintain the IFC structure intact.
4. Finally, after all the replacements of step 3 are completed, for every set extracted in step 2, all the elements (hash key entries and respective IFC text lines) except from the representative ones are removed.

### 4.3.2 Technology Stack and Implementation Tools

**Table 8 – Libraries and Technologies used in IFC Optimizer**

| Library/Technology Name | Version | License |
|---|---|---|
| BIM Library | 1.0.0 | Open Source |

### 4.3.3 Input, Output, and API Documentation

The IFC optimizer uses as input any BIM file that is generated by the exporters of openBIM authoring tools such as Autodesk's Revit or Graphisoft's Archicad, according to the IFC specifications, as described in subsection 3.3.3. These files often contain repeated segments of text due to exportation inefficient operations. The IFC optimizer removes these repeated text segments and keeps only one main segment as a point of reference. Additionally, the tool adds appropriate references to the maintained single text segments, keeping the structure of the whole IFC file unaffected. After these text operations, the final IFC file at the output of the tool is significantly reduced in size. The input and output files are illustrated in Figure 25. Part 1 of Figure 25 is a screenshot of the input IFC file that has a total of 147087 lines and part 2 of Figure 25 is a screenshot of the output file the IFC Optimizer using the input file, that has a total of 97525 lines.

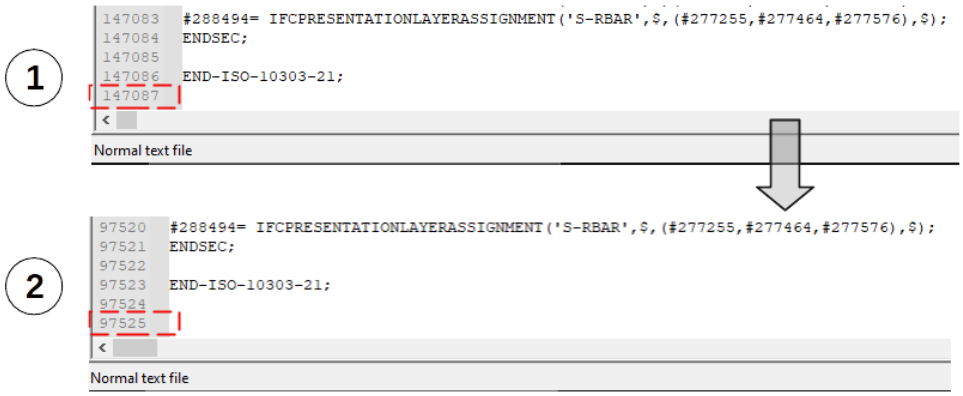



**Figure 25: Input and Output files of IFC optimizer**

### 4.3.4 Application example

These text reduction operations in the input IFC files and the production of optimized output IFC Files by the IFC optimizer, is demonstrated in Figure 26, for the IFC file of the building construction project displayed in part B of Figure 23.

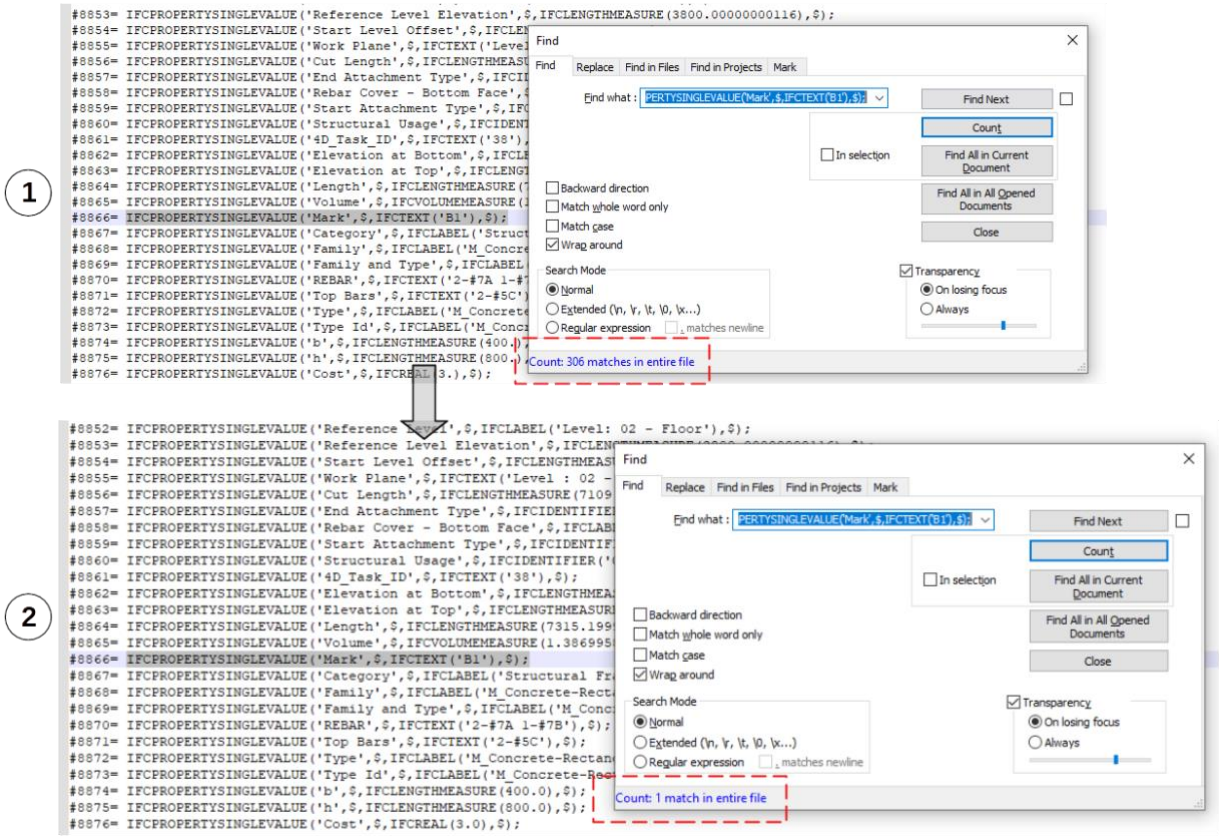



**Figure 26: Example of IFC file optimization operations of IFC Optimizer tool.**

. As we can see in part 1 of Figure 26 the following line:

'' IFCPROPERTYSINGLEVALUE('Mark',$,IFCTEXT('B1'),$); ''

is selected in the input IFC file. The line is counted using Notepad++, 306 times in the text. After the text operations finish and as part 2 of Figure 26 demonstrates, the same line is counted only one time in the optimized output IFC file of the tool. With these text reduction operations, the file size of the IFC for this example is reduced by 35.59% (9,183,232 bytes to 5,914,624 bytes), yielding a compression ratio of 1.55. Additionally, the references to line #8866 have increased from 14 in the input file to 319 in the optimized file. This proves that the necessary references have been added by the IFC Optimizer, to maintain the IFC schema structure.

### 4.3.5 Installation Instructions

The IFC Optimizer is an internal component of data post-processing layer of DTP. Consequently, there is no API involved and respective installation procedure.

### 4.3.6 Development and integration status

Currently the IFC optimizer is tested on a demo IFC BIM file referring to a building construction model displayed in Figure 23. When more complex, non-building BIM construction data are generated representing real entities COGITO's demonstration sites, the BIM optimizer will be tested on bigger IFC files. The respective achieved compression ratios of IFC Optimizer for these new files, is expected to be increased.

In the future, geometric criteria will be added to the IFC Optimizer operations, which will include geometric tolerances to achieve even higher compression ratios and simplification of the IFC geometric content. Using these tolerances multiple lines referring to the same geometric entities will be compressed as the example presented at the end of subsection 4.3.8.

### 4.3.7 Requirements Coverage

Currently the IFC optimizer receives as input openBIM files conforming to the IFC4 standard (ISO 16739, 2016). It is required for the correct operation of the tool the input file to have a structure according to the standard specification. Any violation of this structure in the input file will result to termination of the tool's execution without any output file generation and any compression achieved.

### 4.3.8 Assumptions and Restrictions

The IFC optimizer checks the lines of the IFC files and performs string comparisons to identify similar lines, remove them and place appropriate references without breaking down the IFC structure. Currently the similarity criteria used to detect the similar lines in the IFC text is based on character- by- character checking. These strict criteria are assumed for all text line checks performed by the tool. Geometric criteria for identifying similar lines in geometric context, such as the Euclidean distance, are not considered.

For example, the current version of IFC Optimizer treats the lines:

#51484= IFCCARTESIANPOINT((2.16493489801906E-15,0.));

and

#51582= IFCCARTESIANPOINT((0.,0.));

as different lines.

From a geometric perspective, these two lines are the same referring to point (0,0), if we consider any geometric tolerance smaller than 10^(-14) meters. In the future, by adding the geometric tolerances the previous two lines will be compressed to a single "IFCCARTESIANPOINT((0.,0.))" line.

# 5 Conclusions

In this demonstrator deliverable, eight components or tools of Digital Twin Platform were presented, and their operation was demonstrated on example data referring to one building construction site and one road network construction example. These tools apply extraction transformation and loading (ETL) and model checking (MC) operations on static BIM-related data. Tools applying ETL operations on dynamic data, such as IoT data, are briefly introduced.

Following the architecture presented in D7.1 these eight tools were split into two groups: (A) five tools belonging to the data Ingestion layer of DTP which included one MC tool and four ETL tools applying necessary data operations on COGITO input to form COGITO semantic graph data model and (B) three tools belonging to the data post-processing layer of DTP which include one MC tool and two ETL applying operations on COGITO input BIM files. The operations performed in the tools of group B are demanding in terms of hardware resources which is why they are organized in a parallel and highly containerized fashion to be executed asynchronously via the ESB framework. With this tool-classification in mind, the deliverable was divided into two main sections (3 and 4), where the tools of group A and B were presented in respective subsections. All tools presented in this deliverable are designed to support specific COGITO use cases reported in the tools' respective subsections.

Many of the presented tools can be executed as standalone services in a containerized environment according to a SOA principle. Consequently, simultaneous, parallel and asynchronous executions of these tools are possible, which reduces significantly the overall query response time of COGITO tools, requesting these ETL and MC tool executions as separate services. Furthermore, the coordination of the execution of these tools was designed to be performed via an appropriate messaging scheme issued by the Messaging Layer components, which will facilitate further the data traffic and the required data transformation operations between the data providers (external data sources or stored data) and the data consumers (COGITO tools).

An initial version of these tools was presented and demonstrated in this deliverable. As the project evolves and when data from the project's demo sites are ready the tools will be demonstrated on these data and the results will be presented in the second version of this deliverable (D7.4) which is expected to be released on M24. This second version will include more detailed information related to the detailed technology stack used for the implementation of the tools, as well as updates based on additional tests of these tools, on construction site data arriving from the demonstration sites. Finally, when additional data structures related to reports regarding detected quality control and health and safety issues on construction elements and data originating from construction workflow applications, become available, new RDF Generation tools will be implemented that take as input these new data structures and produce semantic graph outputs in TTL RDF format. These new tools will be incorporated into the Knowledge Graph generator component of DTP and reported in the second version of this deliverable.

## References

ISO 16739. (2016). *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries (ISO 16739:2013).* CEN.

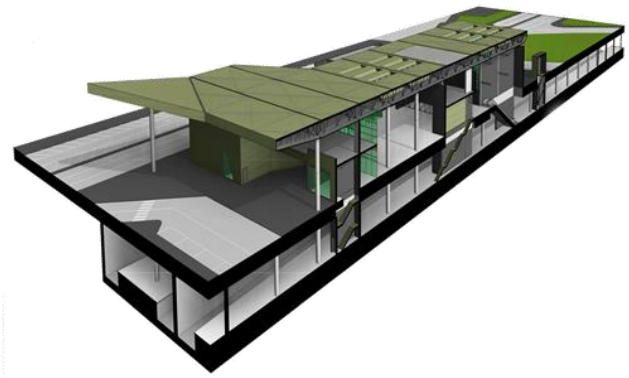
buildingSMART. (2018). *Model View Definition*. Retrieved from https://www.buildingsmart.org/standards/bsi-standards/model-view-definitions-mvd/

Vatti, B. R. (1992). A generic solution to polygon clipping. *Communications of the ACM, 35*(7), 55-36.

Mei, G., Tipper, J. C., & Xu, N. (2015). Ear-Clipping Based Algorithms of Generating High-Quality Polygon Triangulation. *International Conference on Information Technology and Software Engineering* .

Johnson, A. (2014). *Clipper - an open source freeware library for clipping and offsetting lines and polygons*. Retrieved from http://www.angusj.com/delphi/clipper.php

W3C. (n.d.). *Shapes Constraint Language (SHACL)*. Retrieved from https://www.w3.org/TR/shacl/



# COGITO

## CONSTRUCTION PHASE DIGITAL TWIN MODEL

cogito-project.eu

HYPERTECH energy labs

UCL

AARHUS UNIVERSITET

THE UNIVERSITY of EDINBURGH

BoC www.boc-group.com

iti

POLITÉCNICA

QUE TECHNOLOGIES

Novitech New Information Technologies

ASM

ferrovial construction

ΟΛΥΜΠΙΑ ΟΔΟΣ

RHOMBERG SERSA RAIL GROUP

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