



COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

D7.4 –
Extraction,
Transformation
& Loading tools
and Model
Checking v2



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 955310

D7.4 – Extraction, Transformation & Loading Tools and Model Checking v2

Dissemination Level:	Public
Deliverable Type:	Demonstrator
Lead Partner:	UCL
Contributing Partners:	UPM
Due date:	31-10-2022
Actual submission date:	01-11-2022

Authors

Name	Beneficiary	Email
Georgios N. Lilis	UCL	g.lilis@ucl.ac.uk
Fang Zigeng	UCL	z.fang@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
Raúl García-Castro	UPM	rgarcia@fi.upm.es

Reviewers

Name	Beneficiary	Email
Apostolia Gounaridou	CERTH	agounaridou@iti.gr
Martin Straka	NT	straka@novitechgroup.sk

Version History

Version	Editors	Date	Comment
0.1	UCL	20/08/2022	ToC
0.2	UCL	15/9/2022	Draft version
0.3	UPM	19/10/2022	Section 4 updates
0.7	UCL	24/10/2022	Consistency check – Review version
0.8-0.9	UCL	30/10/2022	Update to address internal review
1.0	UCL, Hypertech	31/10/2022	v1.0 for Submission to EC

Disclaimer

©COGITO Consortium Partners. All right reserved. COGITO is a Horizon 2020 Project supported by the European Commission under Grant Agreement No. 958310. The document is proprietary to the COGITO consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the property rights owner. The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Communities. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use, which may be made, of the information contained therein.

Executive Summary

The COGITO solution provides an integrated and extensible toolkit to support construction projects in the planning and implementation phases. Three broad application areas have already been identified as core application domains: health and safety, quality control, and workflow modelling and monitoring. Each domain presents its own challenges, distinct information requirements, and differing stakeholder needs and perspectives. The digital twin platform is a data integration middleware, providing a Master Data Management service, and supports data integration workflows, to expose application-specific data views. These views are in the form of the ontologies developed within WP3, and additional data, like the reality capture data or dynamic time-series data captured from sensors in the field. These data are stored, linked, and made available to applications through the DTP's APIs.

The instantiation of these application-specific models is not trivial, as it relies on processing imperfect data that may come in various container formats. What is more egregious is that there are no safeguards that even when the information provided is syntactically correct, it will contain all information required or that the quality of this information will meet baseline quality expectations so that the information will be usable. Input data include structured information like, for example, BIM models or construction schedules in well-defined formats as exported from project management tools. Dynamic information includes time series data that might have quality issues like gaps or out-of-range values.

The work in this task stems partly from the need to identify and, where possible, address data quality issues. Once data are deemed of acceptable quality, Extraction, Transformation, and Loading (ETL) tools transform data in a form usable by the upstream applications. Some of these are application-specific, and others are rather general. The actor-based architecture of the Digital Twin Platform (DTP) allows the deployment and management of multiple data quality checks and ETL components. The DTP supports complex data checking and integration workflows in an extensible and general manner.

A significant part of the information uploaded on the platform is in the form of BIM models. BIM models (e.g., geometry and schedules) are given as input files but can also be the output of applications (e.g., semantic enrichment with health and safety information). When BIM data exchanges take place (e.g., when a BIM model is uploaded to the platform or when a specific ETL tool is called), we can encode the particular information requirements as a specific Model View Definition (MVD). We have implemented a general-purpose completeness checker, the MVD checking tool, which applies data completeness checks on input IFC4x3 files. Given a specific exchange, the MVD checker can investigate whether particular information is available, as encoded in an exchange-specific mvdXML file. This allows performing Model Checking (MC) to ensure that a BIM model has the complete information required for a specific use. Once the model has been imported, further checks can occur; one example is clash detection, implemented as part of the Geometric Clash Checker (GCC) tool.

Data providers should correct errors detected by the MC tools before the DTP can use the BIM information. The B-rep generation tool (ETL) allows the visualisation of detected issues to DTP's GUI so errors or missing data can be corrected. The process might require several iterations, but ETL tools can be applied when an error-free model has been received. One such tool is the IFC Optimizer, which involves a lossless compression algorithm to reduce BIM data sizes. Or the GCC tool enriches the BIM model with containment relationships among construction elements and their construction zones (when these are not present).

The Construction RDF ETL tool instantiates the RDF graph model from the optimized and enriched BIM data. Once the optimised and enriched BIM model has been loaded, along with supplementary information in non-BIM formats (e.g., construction schedules), and all input data quality checks and transformations have been performed, we can instantiate the COGITO ontologies. Other inputs might come in structured (XML/JSON) files and can populate additional information. In particular: the Project RDF ETL for project-related data, the Process RDF ETL for construction schedules, and the Resources RDF ETL for construction project resources.

Once all ETL processes have been applied, the resulting RDF models should be fully instantiated and ready for querying. An additional model checking takes then place. The Shapes Constraint Language (SHACL) can

capture different application requirements. Then the DTP checks the finalised RDF models to determine whether they meet application-specific requirements (as defined by additional SHACL files). If the SHACL tests pass, the queries performed by applications should be able to return results. This comprehensive bottom-up (input checking) and top-bottom (application checking) should help ensure that applications receive good-quality data.

Some dynamic data quality checks have been delegated to the IoT pre-processing module for dynamic data. Two tools for construction site data logging, the IoT data logger and serving these data to applications, the IoT Data Retriever, are part of the DTP and are described in detail.

This demonstrator deliverable describes the data quality checking and ETL components described above and developed to support the specific COGITO tasks and use cases. The deliverable is the output of Task 7.2. D7.4 is the second version expanding upon D7.3 to include additional work up to M24 of the project.

In the following sections, we describe the input/output format, technology, requirements and dependencies of these tools in detail. The discussion focuses on describing the inputs and outputs of each tool developed, referencing the detailed data structures in other deliverables. Section 1 provides an introduction and details on updates from the previous version (D7.3) of this deliverable. Section 2 discusses the operational sequence of the tools presented. For static data, Section 3 discusses MC and ETL tools that work with BIM data, and Section 4 tools that take as input non-BIM data. For dynamic data, Section 5 discusses related ETL tools. We give examples of the application of these components on data from COGITO pre-validation sites.

Table of contents

Executive Summary	2
Table of contents	4
List of Figures	8
List of Tables	9
List of Acronyms	10
1 Introduction	11
1.1 Objectives of the Deliverable	11
1.2 Relation to other Tasks and Deliverables	12
1.3 Structure of the Deliverable	12
1.4 Updates from the previous version	12
2 ETL and MC tools of COGITO	14
2.1 ETL and MC tools for static data	14
2.1.1 Functional role of the static ETL and MC tools	15
2.1.2 Tools that work with BIM data as input	16
2.1.3 Tools that work with other (non-BIM) data as input	16
2.1.4 The operational sequence of ETL and MC tools on static data	17
2.2 ETL tools for dynamic data	18
2.2.1 The operational sequence of ETL tools on dynamic data	19
3 Tools that work with BIM data as input	20
3.1 Model-View Definition (MVD) Checker	20
3.1.1 Prototype Overview	20
3.1.2 Technology Stack and Implementation Tools	21
3.1.3 Input, Output, and API Documentation	21
3.1.4 Application examples	23
3.1.5 Installation Instructions	24
3.1.6 Development and integration status	24
3.1.7 Requirements Coverage	24
3.1.8 Assumptions and Restrictions	25
3.2 B-rep Generator	25
3.2.1 Prototype Overview	26
3.2.2 Technology Stack and Implementation Tools	27
3.2.3 Input, Output, and API Documentation	27
3.2.4 Application examples	28
3.2.5 Installation Instructions	29
3.2.6 Development and integration status	29
3.2.7 Requirements Coverage	29
3.2.8 Assumptions and Restrictions	30

3.3	IFC Optimizer.....	30
3.3.1	Prototype Overview.....	31
3.3.2	Technology Stack and Implementation Tools.....	31
3.3.3	Input, Output, and API Documentation.....	31
3.3.4	Application example.....	32
3.3.5	Installation Instructions.....	32
3.3.6	Development and integration status.....	32
3.3.7	Requirements Coverage.....	33
3.3.8	Assumptions and Restrictions.....	33
3.4	Geometric Clash Checker.....	33
3.4.1	Prototype Overview.....	33
3.4.2	Technology Stack and Implementation Tools.....	34
3.4.3	Input, Output, and API Documentation.....	34
3.4.4	Application examples.....	35
3.4.5	Installation Instructions.....	35
3.4.6	Development and integration status.....	35
3.4.7	Requirements Coverage.....	36
3.4.8	Assumptions and Restrictions.....	36
3.5	Construction RDF ETL.....	36
3.5.1	Prototype Overview.....	37
3.5.2	Technology Stack and Implementation Tools.....	38
3.5.3	Input, Output, and API Documentation.....	38
3.5.4	Application example.....	39
3.5.5	Licensing.....	40
3.5.6	Installation Instructions.....	40
3.5.7	Development and integration status.....	40
3.5.8	Assumptions and Restrictions.....	40
4	Tools that work with other (non-BIM) data as input.....	42
4.1	Project RDF ETL.....	42
4.1.1	Prototype Overview.....	42
4.1.2	Technology Stack and Implementation Tools.....	43
4.1.3	Input, Output, and API Documentation.....	43
4.1.4	Application example.....	43
4.1.5	Licensing.....	44
4.1.6	Installation Instructions.....	44
4.1.7	Development and integration status.....	44
4.1.8	Requirements Coverage.....	44
4.1.9	Assumptions and Restrictions.....	45
4.2	Process RDF ETL.....	45

4.2.1	Prototype Overview.....	45
4.2.2	Technology Stack and Implementation Tools.....	46
4.2.3	Input, Output and API Documentation.....	46
4.2.4	Application example.....	47
4.2.5	Licensing.....	48
4.2.6	Installation Instructions.....	48
4.2.7	Development and integration status.....	48
4.2.8	Requirements Coverage.....	48
4.2.9	Assumptions and Restrictions.....	49
4.3	Resources RDF ETL.....	49
4.3.1	Prototype Overview.....	49
4.3.2	Technology Stack and Implementation Tools.....	50
4.3.3	Input, Output, and API Documentation.....	50
4.3.4	Application example.....	51
4.3.5	Licensing.....	52
4.3.6	Installation Instructions.....	52
4.3.7	Development and integration status.....	52
4.3.8	Requirement Coverage.....	52
4.3.9	Assumptions and Restrictions.....	52
4.4	RDF Validator.....	53
4.4.1	Prototype Overview.....	53
4.4.2	Technology Stack and Implementation Tools.....	54
4.4.3	Input, Output, and API Documentation.....	54
4.4.4	Application examples.....	55
4.4.5	Licensing.....	56
4.4.6	Installation Instructions.....	56
4.4.7	Development and integration status.....	57
4.4.8	Requirements Coverage.....	57
4.4.9	Assumptions and Restrictions.....	57
5	ETL Tools for dynamic data.....	58
5.1	IoT Data Logger.....	58
5.1.1	Prototype Overview.....	58
5.1.2	Technology Stack and Implementation Tools.....	59
5.1.3	Input, Output, and API Documentation.....	59
5.1.4	Application examples.....	59
5.1.5	Development and integration status.....	60
5.1.6	Requirements Coverage.....	60
5.1.7	Assumptions and Restrictions.....	60
5.2	IoT Data Retriever.....	61

5.2.1	Prototype Overview.....	61
5.2.2	Technology Stack and Implementation Tools.....	62
5.2.3	Input, Output, and API Documentation.....	62
5.2.4	Application examples.....	63
5.2.5	Development and integration status.....	63
5.2.6	Requirements Coverage.....	63
5.2.7	Assumptions and Restrictions.....	64
6	Conclusions	65
	References.....	66

List of Figures

Figure 1: ETL and MC tools for static COGITO input data.....	14
Figure 2: Asynchronous parallel invocation of BIM tools via the message broker.	16
Figure 3: Operational sequence (notional) of ETL and MC tools on static data.....	17
Figure 4: ETL tools on IoT COGITO input data	18
Figure 5: Operational sequence of ETL tools acting on dynamic data	19
Figure 6: Block diagram of MVD Checker's MC operation	21
Figure 7: Input mvdXML file format of MVD Checker	22
Figure 8: JSON output file format of MVD Checker	23
Figure 9: Demonstration of MVD Checker's issue reporting using DTP's GUI.....	24
Figure 10: Block diagram of B-rep Generator's ETL operation.	26
Figure 11: Input and output of B-rep Generator tool.....	27
Figure 12: Boundary representations of construction zones of COGITO's RSRG rail project, generated by the B-rep Generator tool.....	28
Figure 13: Boundary surface extraction (A) and triangulated output (B) of BRG tool applied on COGITO's RSRG demonstration site (1: Wireframe display, 2: Details inside a construction zone)	29
Figure 14: Block diagram of IFC Optimizer's ETL operation.....	31
Figure 15: Input and Output files of IFC optimizer	31
Figure 16: Example of IFC file optimization operations of IFC Optimizer tool.	32
Figure 17: Block diagram of GCC tool's ETL operation.	34
Figure 18: I/O file illustration of GCC tool.....	35
Figure 19: Demonstration example of GCC tool checker (clash checking) using DTP's GUI.....	35
Figure 20: Block diagram of Construction RDF ETL's ETL operation.....	37
Figure 21: Input/Output operation of Construction RDF Generation tool	39
Figure 22: Example of TTL RDF file generation by the Construction RDF generation tool (1: IFC input data, 2: TTL RDF data).....	39
Figure 23: Block diagram of Project RDF ETL's ELT operation.	42
Figure 24: Input/Output of Project RDF ETL.....	43
Figure 25: Published Project triples example.....	44
Figure 26: Block diagram of Process RDF ETL's operation.....	45
Figure 27: O/I file ETL operations of the Process RDF ETL tool	47
Figure 28: Example of triple generation of a Leaf and Parent Tasks by the Process RDF generation tool....	47
Figure 29: Published Task triples example.....	48
Figure 30: Block diagram of Resources RDF ETL's operation.	49
Figure 31: O/I file ETL operations of the Resources RDF ETL tool.....	51
Figure 32: Illustration of the I/O file operations of the Resources RDF ETL tool.....	51
Figure 33: Published Resource Type triples example	52
Figure 34: Block diagram of RDF Validator's MC operation.....	54
Figure 35: Application example of RDF validator.....	56
Figure 36: Block diagram of IoT Data Logger's operation	58
Figure 37: I/O illustration of IoT Data Logger.....	59
Figure 38: Application example of the IoT Data Logger.....	60
Figure 39: Block diagram of IoT Data Retriever's operation.	61
Figure 40: Illustration of ETL process on location tracking data performed by the IoT Data Retriever.....	62
Figure 41: I/O illustration of IoT Data Retriever.	62
Figure 42: Application example of IoT Data Retriever.....	63

List of Tables

Table 1: Relation to other Tasks and Deliverables.....	12
Table 2: Libraries and Technologies used in MVD Checker.....	21
Table 3: MVD Checker's requirements coverage from D2.5.....	25
Table 4: Libraries and Technologies used in B-rep Generator.....	27
Table 5: BRG tool's requirements coverage from D2.5.....	30
Table 6: Libraries and Technologies used in IFC Optimizer.....	31
Table 7: Libraries and Technologies used in GCC.....	34
Table 8: MVD Checker's requirements coverage from D2.5.....	36
Table 9: Libraries and Technologies used in Construction RDF ETL.....	38
Table 10: Construction RDF ETL requirements coverage from D2.5.....	40
Table 11: Libraries and Technologies used in Project RDF ETL.....	43
Table 12: Project RDF ETL requirements coverage from D2.5.....	44
Table 13: Libraries and Technologies used in Process RDF ETL.....	46
Table 14: Process RDF ETL requirements coverage from D2.5.....	48
Table 15: Libraries and Technologies used in Resources RDF ETL.....	50
Table 16: Resources RDF ETL requirements coverage from D2.5.....	52
Table 17: Libraries and Technologies used in RDF Validator.....	54
Table 18: Examples of involved entities in the RDF validator checking operations.....	55
Table 19: RDF Validator requirements coverage from D2.5.....	57
Table 20: Libraries and Technologies used in B-rep Generator.....	59
Table 21: IoT Data Logger requirements coverage from D2.5.....	60
Table 22: Libraries and Technologies used in IoT Data Retriever.....	62
Table 23: IoT Data Logger requirements coverage from D2.5.....	64

List of Acronyms

Term	Description
API	Application Programming Interface
BIM	Building Information Model
BRG	B-Rep Generator
COGITO	Construction Phase diGital Twin mOdel
CSV	Comma Separated Values
DCC	Digital Command Centre
DigiTAR	Digital Twin Visualization with AR
DTP	Digital Twin Platform
ESB	Enterprise Service Bus
ETL	Extraction, Transformation and Loading
GCC	Geometric Clash Checker
GPU	Graphics Processing Unit
GUI	Graphical User Interface
H&S	Health and Safety
IFC	Industry Foundation Class
JSON	JavaScript Object Notation
MC	Model Checking
MVD	Model View Definition
OBJ	Object file
PMS	Process Modelling and Simulation
QC	Quality Control
RDF	Resource Description Framework
SHACL	SHApes Constraint Language
SOA	Service Oriented Architecture
TTL	Terse Triple Language
UC	Use Cases
WODM	Work Order Definition and Monitoring tool
WOEA	Work Order Execution Assistance tool
XML	Extensible Markup Language

1 Introduction

The COGITO Digital Twin Platform (DTP) ingests data from heterogeneous data sources and in several formats. These data might be of varying quality and not directly usable by COGITO applications. Consequently, a data-quality checking and transformation process is required. In case of missing data, these tools should either impute or correct missing content (if possible) or inform the user about it in any other case. The description and analysis of these tools, which are components of COGITO's DTP, is the main topic of this deliverable D7.4: "Extraction, Transformation & Loading Tools and Model Checking v2".

The tools described fall into two categories:

- (i) **Model (or data quality) Checking (MC)** tools ensure the data quality and that data meet application-specific requirements.
- (ii) **Extraction Transformation and Loading (ETL)** tools include the tools that transform input data into appropriate forms or models as required by the COGITO applications.

The ETL and MC tools are either acting on:

- (i) **Static data** (BIM data, project data, construction schedule data and construction resource data) to instantiate COGITO's data models or;
- (ii) **Dynamic data** from IoT devices (e.g. location tracking) is stored in a time-series database and made available for querying to infer, for example, the location of construction resources.

1.1 Objectives of the Deliverable

This deliverable aims to present the components that comprise the Master Data Management service, consisting of a set of Extraction Transformation and Loading (ETL) tools and Model Checking (MC) components. D7.3 presented preliminary versions of these tools.

The objective is to develop these tools to perform the following operations on COGITO's data:

1. Bottom-up Model Checking (MC) to ensure the quality of input data and determine their suitability for further processing;
2. Extraction, Transformation, and Loading to instantiate COGITO's data models; and
3. Top-bottom Model Checking (MC) to ensure the resulting models meet application data requirements.

Considerations when developing these components include:

- Ability to support a wide range of inputs for structured data formats in STEP, CSV, XML and JSON serialisations, generated from 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 (which can be processed in the DTP);
- Provision 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, the necessary data transformations are applied to instantiate the COGITO data models.

Each tool has different computational requirements depending on the data type and the operations' complexity. Some ETL and MC tools are deployed as containers in the run-time environment, enabling asynchronous and independent execution. The DTP understands the sequentiality of operations and manages the resources to ensure complex data integration tasks are performed effectively and external end-points are notified accordingly. The containerisation approach allows for horizontal scaling when additional computational resources are required. Given that some of these tools might require significant execution time, the DTP implements a fully asynchronous execution pipeline.

1.2 Relation to other Tasks and Deliverables

Deliverable D7.4: Extraction, Transformation & Loading Tools and Model Checking v2 depends upon the following COGITO deliverables:

Table 1: Relation to other Tasks and Deliverables

Del. Number	Deliverable Title	Relations and Contribution
D3.3	COGITO Data Model and Ontology Definition and Interoperability Design 2	D3.3 describes the final structure of COGITO's ontology. The operation of COGITO's ETL tools which generate semantic graphs (RDF generation tools), is specific to the data classes of the COGITO ontology.
D7.2	Digital Twin Platform Design & Specification v2	D7.2 presents the final architecture design of COGITO's Digital Twin Platform, and within this context, D7.4 details the ETL and MC tools in the DTP. We mention some of these tools in D7.2 but without more information.
D7.3	Extraction, Transformation & Loading Tools and Model Checking v1	D7.3 discussed the first version of DTP's ETL and MC tools. D7.4 extends the work presented in D7.3 and discusses updates to the ETL and MC components carried over the second year of the project implementation.

1.3 Structure of the Deliverable

This deliverable discusses COGITO's ETL and MC tools. Section 2 introduces the tools presented below and discusses their functional role. In addition, Section 2 provides the link to COGITO's Digital Twin Platform (DTP) described in D7.2, which hosts these tools.

Section 3 discusses MC and ETL tools that work with BIM data as input. In particular:

- Section 3.1 MC tool: MVD (Model View Definition) Checker;
- Section 3.2 ETL tool: B-rep (Boundary Representation) Generator;
- Section 3.3 MC/ETL tool: GCC (Geometric Clash Checker) tool;
- Section 3.4 ETL tool: IFC optimizer;
- Section 3.5 ETL tool: Construction RDF ETL.

Section 4 discusses MC and ETL tools that work with other (non-BIM) data as input. In particular:

- Section 4.1 ETL tool: Project RDF ETL;
- Section 4.2 ETL tool: Process RDF ETL;
- Section 4.3 ETL tool: Project RDF ETL;
- Section 4.4 MC tool: RDF validator.

Section 5 discusses ETL tools that work with dynamic data. In particular:

- Section 5.1 ETL tool: IoT data logger;
- Section 5.2 ETL tool: IoT data retriever.

We give examples of the application of these components on data from the pre-validation sites. As we move to the validation sites, further updates might be required to the data models and, therefore, some of the data integration and model checking procedures, especially those that transform to the specific COGITO instances.

This deliverable concludes in Section 6 with a discussion of synthesising the individual components.

1.4 Updates from the previous version

We submitted Deliverable D7.3 in M18 to document the status of the various MC and ETL tools developed within COGITO up to May 22. D7.4 extends the description to reflect the additional effort and components developed over the six months since D7.3 was issued.

The bulk of the work focused on further developing each of the individual components to meet the evolving understanding of the data requirements. The use-case-focused working groups have led to more precise data requirement specifications, influencing application-specific ETL tools. This iterative (agile) approach has resulted in iterative refinement of the tools and adaptations. This is relevant for all the RDF ETL tools related to Construction, Project, Process and Project transformations.

Working with IFC4x3 data (given the infrastructure projects within scope) remains challenging. Many existing solutions provide no/partial support in handling the new classes. Yet, our tools can now fully support IFC4x3 data for serialisation, deserialisation and model completeness checking. The BIM Management SDK, also developed in WP7, enables working with such models.

In the last months, specific tools received minor or no updates. Point in case is the IFC optimiser and the MVD checker. In the latter, the focus was on defining relevant exchanges (in the form of various mvdXML files) and testing whether these worked with the data available. This testing helped develop the confidence that these components are production ready.

In some instances, the need for additional supporting additional checking led to the introduction of other MC procedures, like the Geometric Clash Checker (GCC), to perform clash detection or semantic enrichment when relationships between construction elements and spatial construction zones are missing. When such information is missing, the GCC tool can infer the relationship between the construction zones and the construction elements these zones contain using pure geometric means..

The B-rep Generator tool received both functional and application-related updates. As far as its operations are concerned, B-rep Generator (BRG) was updated to support various parametric descriptions referring to curved solid geometries, such as solids generated by revolution operations on surfaces and swept operations along three-dimensional curves. Such parametric geometric descriptions are common in tunnel and rail structures. Additionally, BRG was tested successfully on COGITO's RSRG demonstration site data in its application domain.

The RDF Generator tools in D7.3 are now called RDF ETL tools to highlight their functional purpose. The role of the RDF validator in top-bottom model checking – complementing the bottom-up model checking of the MVD and GCC tools – has been further developed. The SHACL shape files are being refined in response to application requirements – these will be further refined and extended until the end of the project. These shapes (also called application profiles) are the checking rules for the COGITO data model to ensure specific queries will return valid responses.

In addition, two new ETL tools acting on IoT data are introduced and analysed for dynamic data. The first tool, the IoT data logger, receives data from COGITO's IoT Data Pre-processing tool and populates appropriate IoT database structures in DTP's persistence layer. The second tool is the IoT data retriever. Its primary role is to query the structured IoT data in the database and respond to IoT data queries of other COGITO tools.

To reflect the improved understanding of what is required and what the various ETL and MC tools provide, we have restructured this deliverable, improving the presentation clarity. Given that this is a demonstrator deliverable, the focus is on the I/O and describing the functional purpose of these components.

Together with the other developments of the DTP platform, all components and tools contribute to a first functional version of the platform, which can be tested (and will no doubt be further refined) in the validation sites. Some specifics (e.g. specific exchanges in the form of mvdXML and SHACL shapes) will be further detailed in WP8.

2 ETL and MC tools of COGITO

The actor-based architecture of the Digital Twin Platform (DTP) allows the deployment and management of multiple data quality checks and ETL components. The DTP supports complex data checking and integration workflows in an extensible and general manner. Within COGITO's DTP architecture, described in D7.2, ETL and MC tools operate on static and dynamic data. These tools are part of the Data Ingestion and Data Post-processing layers. This section maps the provided functionalities of each component against the high-level architecture presented in D7.2.

2.1 ETL and MC tools for static data

As shown in Figure 1, the ETL and MC tools applied to static data are divided into two categories (denoted as A and B in Figure 1). We use red colour for MC tools and green colour blocks for ETL tools.

These two tool categories are:

- **Category A** covers tools that take BIM data as input.
 - Tools that belong to the Data Post-processing layer (layer 6) of DTP include MC tool A1.1 and ETL tools: A2.1, A2.2, and A2.3 in Figure 1.
 - One ETL tool, A2.4, belongs to DTP's Data Ingestion layer (layer 2).
- **Category B** covers tools that work with other (non-BIM) data as input.
 - ETL tools B1.1, B1.2 and B1.3 in Figure 1 and MC B2.2 in Figure 1. These tools use non-BIM data and are part of DTP's Data Ingestion layer (layer 2).

We briefly present these tools next and in more detail in Sections 3 and 4.

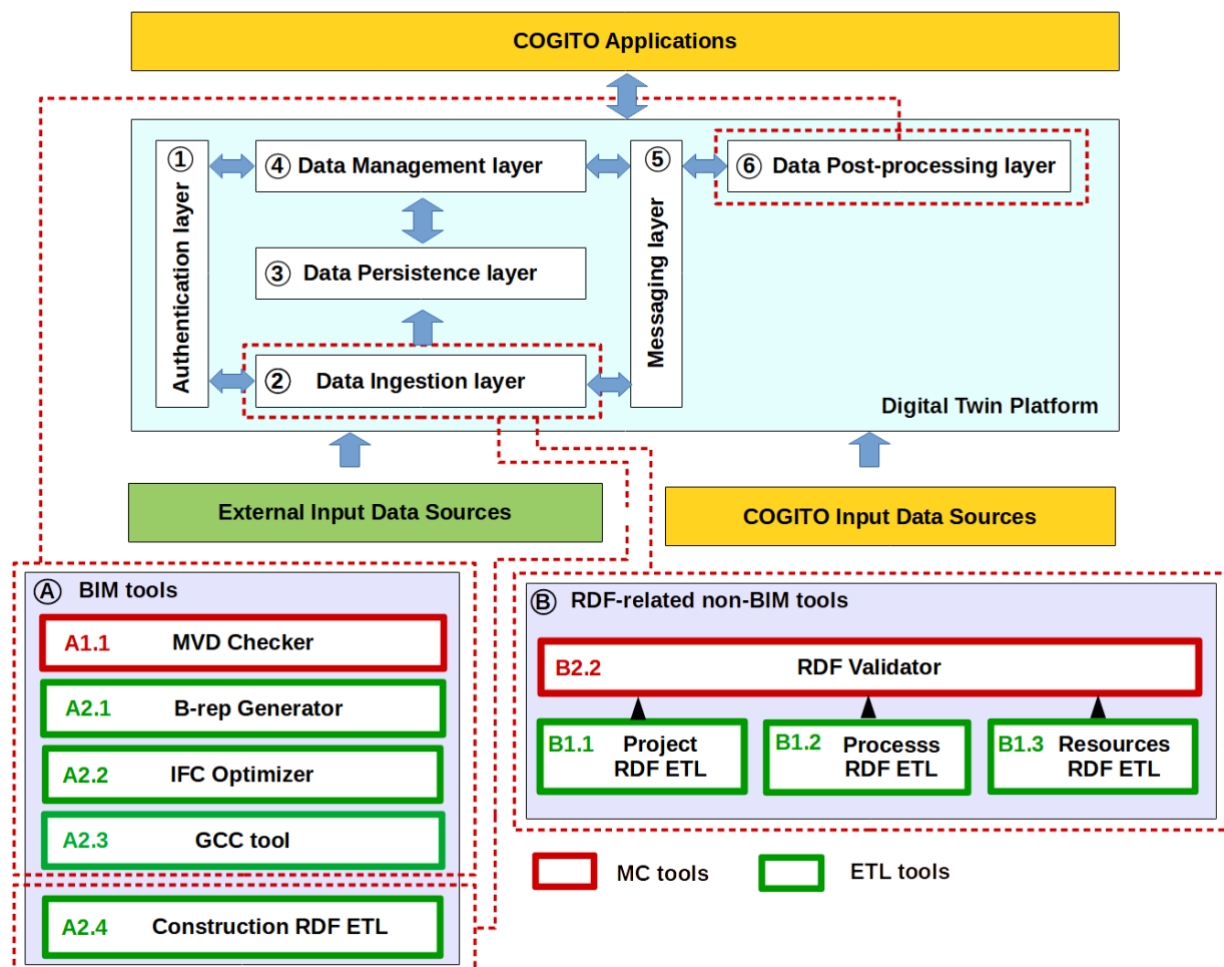


Figure 1: ETL and MC tools for static COGITO input data.

2.1.1 Functional role of the static ETL and MC tools

The COGITO solution provides an integrated and extensible toolkit to support construction projects in the planning and implementation phases. The digital twin platform is a data integration middleware, providing a Master Data Management service, and supports data integration workflows, to expose application-specific data models. These models are the ontologies developed within WP3. The static data are stored, linked, and made available to applications through the DTP's APIs.

Input data include structured information like, for example, BIM models or construction schedules in well-defined formats as exported from project management tools. The instantiation of these application-specific models is not trivial, as it relies on processing imperfect data that may come in various container formats. What is more egregious is that there are no safeguards that even when the information provided is syntactically correct, it will contain all information required or that the quality of this information will meet baseline quality expectations so that the information will be usable. The actor-based architecture of the Digital Twin Platform (DTP) allows the deployment and management of multiple data quality checks and ETL components. The DTP supports complex data checking and integration workflows in an extensible and general manner.

A significant part of the information uploaded on the platform is in the form of BIM models. BIM models (e.g., geometry and schedules) are given as input files but can also be the output of applications (e.g., semantic enrichment with health and safety information). When BIM data exchanges occur (e.g., when we upload a BIM model to the platform or call a specific ETL tool), we can encode the particular information requirements as a specific Model View Definition (MVD). We have implemented a general-purpose completeness checker, the **MVD checking tool (A1.1)**¹, which applies data completeness checks on input IFC4x3 files. Given a specific exchange, the MVD checker can investigate whether particular information is available, as encoded in an exchange-specific mvdXML file. This allows performing Model Checking (MC) to ensure that a BIM model has the complete information required for a specific use. Once the model has been imported into the DTP, further checks can occur; one example is clash detection, implemented as part of the **Geometric Clash Checker (GCC) tool (A2.3)**.

Data providers should correct errors detected by the MC tools before the DTP can use the BIM information. The **B-rep generation tool (A2.1)** allows the visualisation of detected issues to DTP's GUI to correct errors or missing data. The process might require several iterations before ETL tools can be applied that require an error-free model. One ETL tool is the **IFC Optimizer (A2.2)**, which involves a lossless compression algorithm to reduce BIM data sizes. Or the **GCC tool (A2.3)** enriches the BIM model with containment relationships among construction elements and their construction zones (when these are not present).

The **Construction RDF ETL (A2.4)** tool instantiates the RDF graph model from the optimized and enriched BIM data. Once the optimised and enriched BIM model has been loaded, along with supplementary information in non-BIM formats (e.g., construction schedules), and all input data quality checks and transformations have been performed, we can instantiate the COGITO ontologies. Other inputs might come in structured (XML/JSON) files and can populate additional information. In particular: the **Project RDF ETL (B1.1)** for project-related data, the **Process RDF ETL (B1.2)** for construction schedules, and the **Resources RDF ETL (B1.3)** for construction project resources.

Once all ETL processes have been completed, the resulting RDF models should be fully instantiated and ready for querying. An additional model checking takes then place. The Shapes Constraint Language (SHACL) can capture different application requirements. Then the **RDF Validator (B2.2)** checks the finalised RDF models to determine whether they meet application-specific requirements (as defined by additional SHACL shape files). If these tests pass, the queries performed by applications should be able to return results. This comprehensive bottom-up (input checking) and top-bottom (application checking) should help ensure that applications receive good-quality data. It should be noted that this process is transparent to the end users but should result in data usable by upstream applications that can be dynamically updated whenever new information is provided to the DTP.

¹ Nb. These letter references refer to the numbering of tools shown in Figure 1.

2.1.2 Tools that work with BIM data as input

More computationally demanding ETL tools acting on COGITO open BIM input files are placed in the Data Post-Processing layer of the DTP. As the execution of these tools can be time-consuming, especially for larger models, invocation of these components should be done asynchronously, utilising the message broker (ref D7.2 and D7.10) to notify when a particular execution has been completed. The message broker propagates the calculation outputs for further processing (e.g. if the particular tasks are part of a sequence following the actor-based architecture) or for storage in the Data Persistence layer. The message broker oversees internal notifications and messages. Following the micro-services design pattern, all communication and interactions between individual tools occur through the message broker. See Figure 2 for a conceptual schematic, and refer to D7.2 and D7.10 for more details.

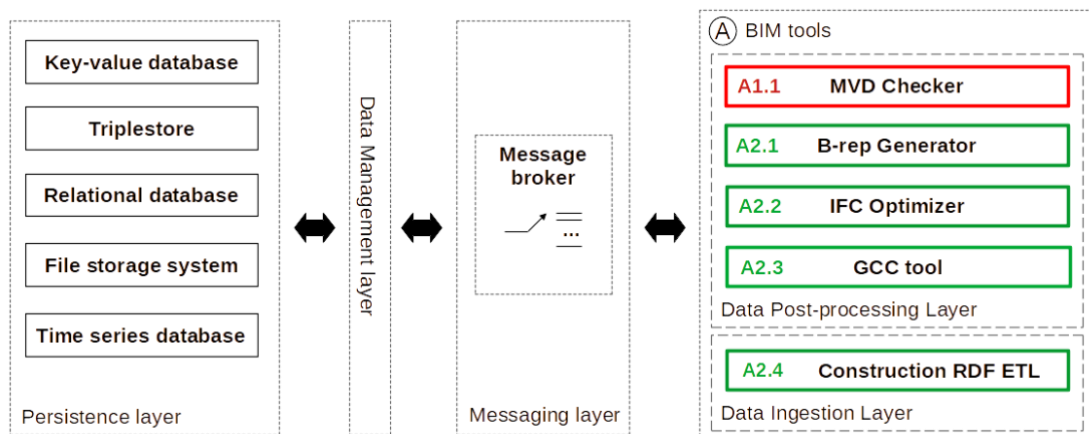


Figure 2: Asynchronous parallel invocation of BIM tools via the message broker.

These ETL and MC tools that work with BIM data as input include:

1. The **MVD checker** performs completeness checks on the content of input 3D BIM data files (IFC4x3) against the rules defined in one or more mvdXML files.
2. The **B-rep Generator (BRG)** transforms subsets of the geometric content of 3D BIM input files to a graphics-friendly format (OBJ format) on-demand, useful for visualisation. The BRG outputs can also be used by handheld Augmented Reality devices with hardware and processing power limitations.
3. The **Geometric Clash Checker (GCC)** checks if certain geometric relations (containment and clash) exist among construction element and construction spatial zone pairs in the geometric content of 3D BIM input data files and enriches the files with the detected containment relationships. Technically this is both an MC (clash detection) and ETL tool (semantic enrichment).
4. The **IFC optimizer** reduces the file size of input 3D BIM files by applying lossless compression.
5. According to the COGITO ontologies, the **Construction RDF ETL** converts 3D BIM (IFC4x3) models to appropriate RDF files.

2.1.3 Tools that work with other (non-BIM) data as input

Less computationally intensive are the three RDF ETLs and the RDF validator, placed in the Data Ingestion layer of DTP. These tools include:

These ETL and MC tools that work with other (non-BIM) data as input include:

1. The **RDF Data Validator** validates the RDF output files produced by the RDF ETL tools using SHACL shapes encoding pre-defined checking rules (application profiles).
2. The **Project RDF ETL** that converts project-related data such as project id, name, and description to RDF TTL files.
3. The **Process RDF ETL** transforms the input construction schedule files into RDF files representing the fourth (time) dimension of the BIM data.

4. The **Resource RDF ETL** converts the input construction resource data into respective RDF data.

2.1.4 The operational sequence of ETL and MC tools on static data

The ultimate goal of the ETL and MC tools acting on static data presented in this deliverable is the formation of a semantically connected and validated graph of elements (in the form of an RDF TTL file) according to the COGITO ontology presented in D3.2. This graph represents the 4D BIM data of a construction project used within the COGITO framework. This graph should be formed initially from the 3D BIM data from the input IFC files and enriched with data from other input files containing schedule data (4D) and construction resource data (5D). This enriched graph should be validated using a set of predefined validation rules. The final enriched and validated graph can be enriched further, with additional H&S and QC data, by the respective COGITO H&S and QC tools.

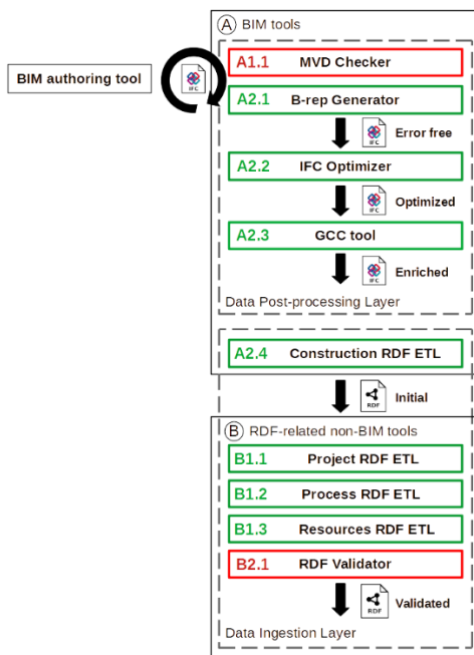


Figure 3: Operational sequence (notional) of ETL and MC tools on static data

The operational sequence in Figure 3 shows COGITO's main data integration pathway. Other data ingestion procedures exist (e.g. for the ingestion of project and process data) but are markedly simpler. The source of information, in this case, is an IFC file exported by a BIM authoring tool. Once this model gets uploaded to the DTP platform, this file passes through a series of checks. The first is a completeness check to ensure all required information is available. Given that the target output (in this contrived example) is the Construction RDF model, the data exchange requirements (primarily geometry) contained in an exchange-specific mvdXML file – we can determine additional mvdXML files for other exchanges.

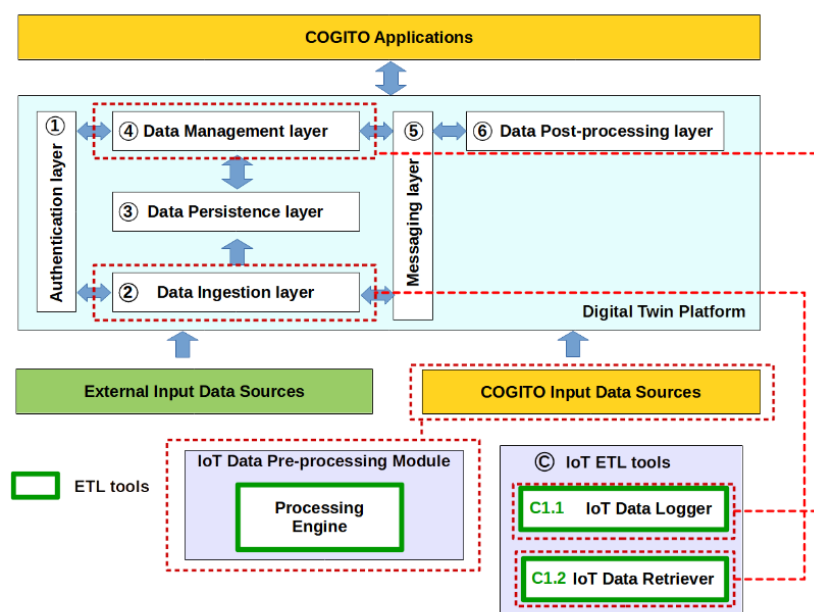
If the MC tools find errors, these are highlighted to the user through the DTP interface using the B-rep Generator to create a visual representation (a BCF export is still a work in progress). The modeller needs to fix these errors and provide an updated model². Model checks and re-modelling to correct errors repeat until the IFC file becomes error-free. Then the IFC model passes through the IFC optimizer for lossless compression. If construction element-construction spatial zone containment relationships have not been modelled, the ETL GCC tool can enrich the model with this information. The enriched IFC is passed to the Construction RDF ETL to generate the RDF TTL file. This file is the first step to filling parts of the COGITO semantic graph model with BIM-sourced information.

Since BIM-only information is insufficient to meet COGITO's data requirements, this initial RDF file is further enriched with non-BIM data by invoking three ETL tools to complete Project, Process and Resources data.

² It is also important to develop modelling guidelines, so that elements are modelled properly. This is considered out of scope for this deliverable, which assumes they are available.

It should be mentioned here that this example describes one possible exchange. The situation, in reality, is much more complex, with multiple such interactions required. The actor-based approach described in D7.2 and D7.10 provides a compositional framework that allows the creation of various data integration tasks. These actors (many of which are the tools we describe here) then serve all the use cases and prepare outputs (or receive) inputs as described in D2.5. The result can be an RDF graph stored in a triple store or a response to an application. The DTP supports the deployment of multiple such execution pipelines, which can then serve the real-world requirements of all three COGITO applications. We do not dwell on these aspects here but refer to D7.10 and WP8 deliverables for further discussion.

2.2 ETL tools for dynamic data



The IoT data from the tracking devices contain noise and possible outliers, removed by the Processing engine in the IoT Data Pre-processing module. We show the processing engine in Figure 4 in green; more details on the Processing engine can be found in D3.6. The IoT data pre-processing module performs:

- **Filtering:** IoT data contain outliers referring to sudden changes in the positions of the sensed construction entities. These Processing Engine filters out these outliers.
- **Aggregation:** 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.

After the IoT Data Pre-processing module preprocesses the IoT data, they are sent to the DTP for further processing. The ETL tools that operate on dynamic data are included in a third category, **Category C** (cf. Categories A and B in Figure 1):

- The **IoT Data Logger** manages storing received data in the time series database (C1.1 in Figure 4).
- The **IoT Data Retriever** queries the data structures formed by the IoT data logger and applies linear interpolations to respond to queries from external-to-DTP COGITO tools (C1.2 in Figure 4).

2.2.1 The operational sequence of ETL tools on dynamic data

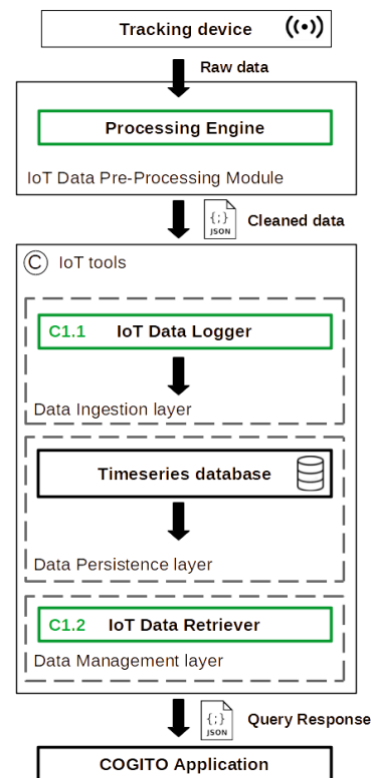


Figure 5: Operational sequence of ETL tools acting on dynamic data

The ETL tools working with dynamic data support: (1) event-based logging in the time series database of the DTP, and; (2) create responses to applications by querying the internal database structures and formulating the responses.

The raw data from IoT location tracking devices pass through the three ETL tools pictured in Figure 5. These tools are parts of COGITO’s architecture described in D2.5 and D7.2. As Figure 5 illustrates, the processing steps of IoT data are as follows:

1. Initially, raw data originating from location tracking devices contain noise and outliers, filtered out by the Processing Engine in COGITO’s IoT Data Pre-processing module (see D3.6).
2. The DTP receives the “cleaned data” and passes them to the **IoT data logger** (C1.1 in Figure 5), which stores these in the Timeseries database according to defined data structures.
3. Finally, when Applications request dynamic data, the stored data structures are interpolated linearly, forming responses sent in JSON format by the IoT data retriever ETL tool (C1.2 in Figure 5).

The responses in step 3 contain the location data (points in cartesian coordinates) of the requested resources for the requested time interval. The approach described here is the first step to generating a more general abstraction for time series data, even though it is unclear if such functionality will be helpful in the COGITO demonstration activities.

3 Tools that work with BIM data as input

The ETL and MC tools that work with BIM data as input include:

(A1.1) **MVD Checker** validates IFC BIM models by checking the existence of certain IFC classes and related attributes;

(A2.1) **B-rep Generator component**, which produces mesh-based representations of BIM data that conform to the IFC4x3 standard;

(A2.2) **IFC Optimiser**, which performs lossless compression and is capable of significantly reducing the original file size of an IFC to speed up the data transformation processes;

(A2.3) **GCC tool**, which enriches input IFC files with (construction element)-(construction zone) containment information;

(A2.3) **Construction RDF ETL** produces the initial knowledge graph model containing BIM data from an input IFC BIM file.

3.1 Model-View Definition (MVD) Checker

COGITO's input IFC BIM files contain geometric and non-geometric data that refer to construction elements and their related properties and associations. Although supported by the IFC schema, these data may not always be present in input models. The reasons vary from modelling errors and omissions by the BIM modeller to errors in BIM/IFC exporting routines. Consequently, a checking mechanism is required to ensure that COGITO input BIMs have the necessary information for every COGITO application.

This data quality checking mechanism is required in several COGITO use cases since all COGITO applications use IFC BIM data. These use cases include:

UC1.1: PMS tool requires the 4D BIM file that contains 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 with H&S information.

UC3.2: Proactive Safety uses 4D BIM data during construction 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.

The MVD Checker tool provides the completeness checking functionality. This tool was first introduced in D7.1 and is described in more detail here.

3.1.1 Prototype Overview

The MVD Checker ensures that a predefined subset of the populated IFC classes in the input IFC BIM files of COGITO contains correct and complete data. The MVD Checker has been developed according to the MVD specification defined by buildingSMART, which enables the view of only a subset of the overall IFC schema to facilitate data exchanges. This “view” of the general IFC schema is constructed using a set of predefined Concept Templates and Rules. The exchange requirements are captured in a machine-readable mvdXML file produced by buildingSMART's IfcDoc tool. The checking rules defined inside the mvdXML file, as Concept Templates, are essentially subsets (“views”) of the IFC schema, which contain the necessary Concepts, Rules, and Constraints.

To perform its model-checking operations, the MVD Checker requires two files as input:

- The model to be checked for completeness.
- An mvdXML file with the exchange requirements (defining the required IFC instances, properties, and inter-relationships).

The output of the MVD Checker is a JSON file containing the results of the checking process. We illustrate these input/output file operations in Figure 6.

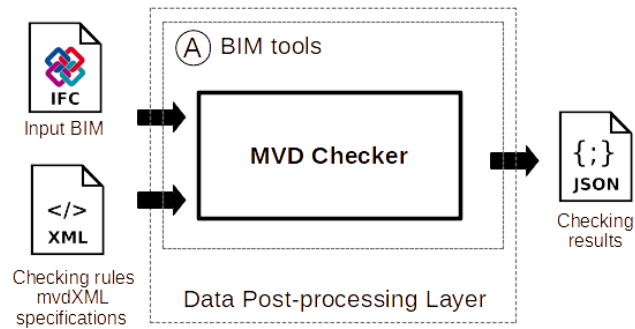


Figure 6: Block diagram of MVD Checker's MC operation

The MVD checking tool comprises two subcomponents: (a) the **MVD parser** and (b) the **MVD-checking** algorithm. The MVD parser reads the mvdXML and IFC files and generates respective memory objects. Then:

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

After the above two steps are complete for all search paths and respective checking rules, the results of the checking operations are used to create the output JSON file.

3.1.2 Technology Stack and Implementation Tools

The MVD Checker uses the technologies presented in Table 2.

Table 2: Libraries and Technologies used in MVD Checker

Library/Technology Name	Version	License
IFC SDK	1.0.0	UCL Proprietary

3.1.3 Input, Output, and API Documentation

The MVD Checker tool takes as input two files: (a) a BIM file 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 3.5.3. As illustrated in Figure 7, input (b) is an XML file containing two essential data items:

- **Concept Templates:** This item contains entity and attributes rules which are nested on one another and define a searchable path within the IFC interconnected classes and their attributes. These concept templates are illustrated with numbered (1 and 2) dashed blocks Figure 7.
- **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 at the bottom screenshot of Figure 7.

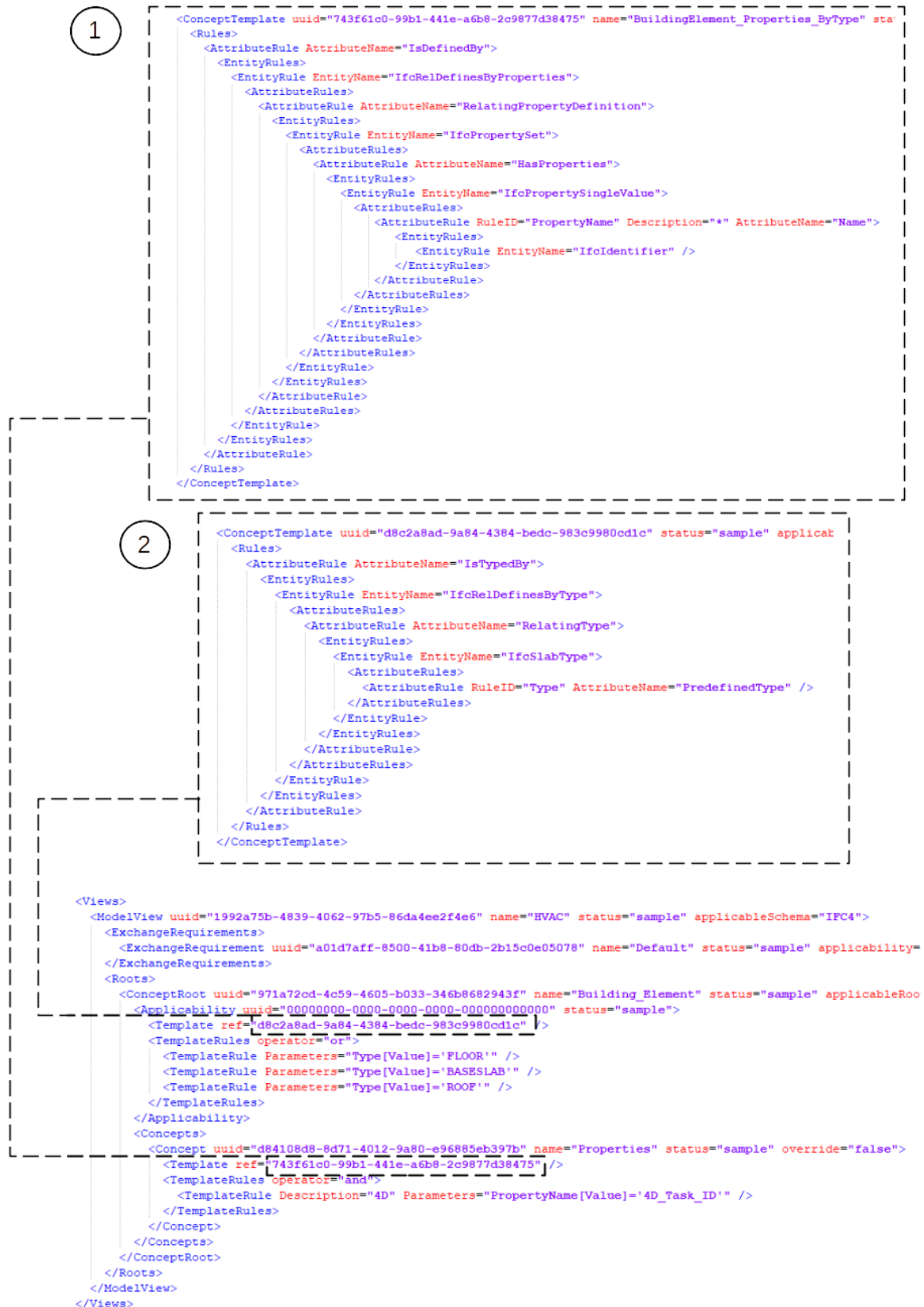


Figure 7: Input mvdXML file format of MVD Checker

The output of the MVD checker contains the results of the checking operation 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 8. All entities defined in the input mvdXML document are checked in the JSON output. For every entity checked, the JSON file reports the following values:

- **valid:** is a boolean value (true/false) representing the checking results of the entity checked.
- **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:** are a complex entity containing the rules that are checked for the current entity and the checking results for every individual rule. A rule further contains the following entities:
 - **valid:** is a boolean value (true/false) representing the checking results of the current rule.
 - **description:** is a string describing the current rule.
 - **parameters:** is a complex entity containing the parameters of the current rule. Every parameter further contains:
 - **valid:** is a boolean value (true/false) representing the checking results of the checked parameter.
 - **name:** is a string describing the checked parameter.
 - **value:** is a string describing the value of the currently checked parameter.

The MVD checking results are communicated back to the COGITO application, which initiates the checking process for appropriate corrective actions.

```

1  [{
2    "entities":
3      [
4        {
5          "valid":false,
6          "name":"Floor:Generic Concrete 300mm:133345",
7          "global":"2E3cuzY0LCc9ztHo7GNgl4",
8          "express":151151,
9          "rules":[
10             {
11               "valid":false,
12               "description":"4D",
13               "parameters":[
14                 {
15                   "valid":false,
16                   "name":"PropertyName",
17                   "value":"4D_Task_ID"
18                 }
19               ]
20             }
21           ]
22        }
23      ]
24    ]
25  }
```

Figure 8: JSON output file format of MVD Checker

3.1.4 Application examples

The MVD checker tool is integrated into the GUI 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 9, related to a detected issue in a column element of an IFC file of a demonstration building highlighted green. Of course, this is useful when the error is linked to a geometric construct (which is most of the time); the resulting JSON file will contain all completeness errors.

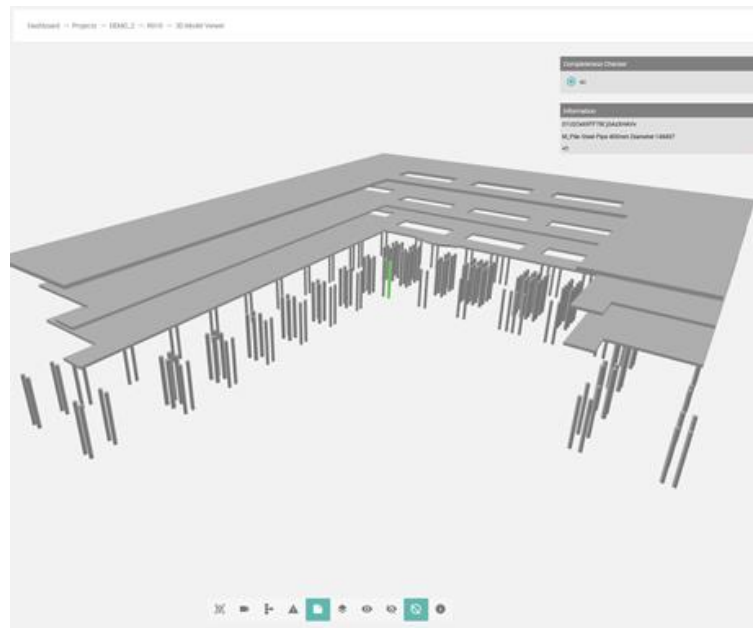


Figure 9: Demonstration of MVD Checker's issue reporting using DTP's GUI

To better understand how the MVD Checker works, the results of the demo construction IFC BIM file referring to a school building construction project using the checking rules defined in the mvdXML file are presented in Figure 9. As we see in the concept template displayed in part 2 of Figure 7, 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 7, a specific property with the description “4D” of these slab types is checked to determine whether it obtains the value “4D_Task_ID”. In effect, the rule checks all IfcSlab elements of type FLOOR, BASESLAB and ROOF to decide whether or not they are connected to construction task elements. In simple terms, the particular check ensures that all slabs have a construction time for these slab elements. When an error is identified (e.g. a construction schedule has not been assigned for a specific element), the results of this checking process are presented in a JSON format, as in Figure 8.

3.1.5 Installation Instructions

The MVD Checker is an internal component of the data post-processing layer of DTP. Consequently, no API is involved, and therefore no installation is required.

3.1.6 Development and integration status

Currently, we tested the MVD checker with several models, including the example of the school building construction project. 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 DTP's GUI can visualise the checking results of the MVD checker. This integration allows the user of DTP to locate in 3D space problematic elements of the construction project that violate one or more checking rules.

Currently, we tested the MVD Checker with available BIM models for exchanges linked to the Construction RDF model. As the validation progresses, new model requirements will emerge, which will introduce new checking rules. These new rules will define new particular checking procedures for the MVD Checker, supporting the data needs of COGITO tools. This process might create additional checking scenarios, which will be served with separate mvdXML files, defining the specific exchange requirements.

3.1.7 Requirements Coverage

The requirements coverage of the MVD Checker tool, according to D2.5, is summarized in Table 3. The component successfully receives as-planned construction data from IFC files (BIM models, Req 1-2). The tools can be scaled up (Req-2.1), and new checking rules can be added as the project progresses. The MVD

Checker is highly responsive (Req-2.2) and available (Req-2.4) and can be executed in a parallelizable and containerized fashion communicating via the message broker (Figure 2).

Table 3: MVD Checker's requirements coverage from D2.5

ID	Description	Type	Status
Req-1.2	Receives as-planned data (BIM models)	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

3.1.8 Assumptions and Restrictions

Currently, the MVD Checker receives as input an IFC model [1] and an mvdXML file according to the MVD v1.1 specification defined by buildingSMART [2]. Ensure that both input files have a structure according to these specifications for the correct tool operation. Any structure violation will result in the termination of the tool's execution without any output file generation or any checking results reported.

For the correct operation of the MVD Checker, and especially when a tool other than the IfcDoc tool is used to generate the mvdXML file, it is assumed that the following conditions are satisfied:

- 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. [1]
- The checking rule definitions inside the Concepts and 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 [1].

3.2 B-rep Generator

The geometric representation within COGITO follows the structures and methods supported in IFC4. However, this format is unsuitable for visualisation due to the compact geometric descriptions of construction elements used to reduce excess data verbosity. Adding to this complexity, the geometric definitions of some of the construction elements might appear in a parametric form which is not suitable for viewing purposes. To convert the geometric descriptions of the construction elements from the IFC4 geometric representation to a visualisation-friendly form (correctly oriented triangulated boundary representations), the B-rep Generator tool (BRG tool) is developed.

The larger the IFC file, the more time and processing resources are required to produce files for use by 3D visualisation software. The computational and memory resources required to generate this information, precludes the direct use of whole IFC files as direct input to hand-held devices (e.g. oculus) due to these devices' size, hardware resources, and processing power limitations.

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

UC 4.1: DCC, requests from the DTP certain construction elements from the 4D BIM model to be displayed in computer displays.

UC 4.2: DigiTAR, requests from the DTP certain construction elements from the 4D BIM model to be displayed in hand-held devices.

To support this need to visualise 3D BIM data, we developed the B-rep Generator (BRG)³, an ETL tool that is part of the Data Post-processing layer of the DTP. The BRG tool is displayed in block 5 in Figure 1. BRG is described in more detail in the following subsections.

³ A first version of the B-rep Generator has been developed prior to COGITO. This has been extended to cover the geometric data structures and cover more recent versions of the IFC.

It is essential to point out that although DCC and DigiTAR tools in UC 4.1 and UC4.2, respectively, have implemented for development purposes their own IFC to OBJ converters using the ifcOpenShell (IFCtoMesh Generator reported in D7.5), their final implementation will use the BRG tool, which can cover infrastructure visualisation scenarios.

3.2.1 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 created OBJ files contain triangulated boundary representations of the 3D BIM elements augmented with the normal vectors of the generated triangle vertices. This representation is in the simplest polygon form (triangle) and can be used directly by the GPU hardware to create visualisations in parallel by the multiple GPU cores. Additionally, visualisation tools can handle the representation triangles to manipulate and correct to make a water-tight rendering of the boundary representation. A water-tight boundary representation is an accurate solid representation with no misalignments among its neighbour boundary surfaces. Such correcting operations are more complex in polyhedral representations.

To produce these triangulated OBJ files, the B-rep Generator parses an input XML file produced by the IFC Geometry Exporter 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 10.

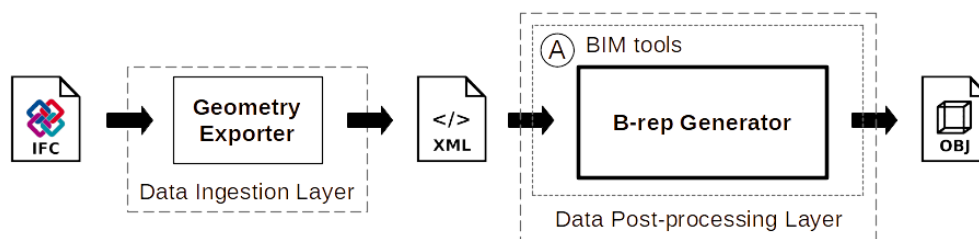


Figure 10: Block diagram of B-rep Generator's ETL operation.

The operation of the 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. 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, implemented using the clipper library [3], (a library based on Vatti's algorithm [4]). Ear-clipping-based polygon triangulation works for polygons without holes. In case any polygon from the sets obtained from step 4 contains holes, it is converted to a degenerate polygon without holes according to [5], 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 every polygon set generated in step 4).
6. The sets of triangles obtained from the polygonal sets of Step 5 are exported as anOBJ output of the BRG tool (every triangle in a triangle set is written as the face of the same object in the OBJ file).

3.2.2 Technology Stack and Implementation Tools

As reported in D7.3 BRG tool is based on two technologies presented in Table 4.

Table 4: 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

3.2.3 Input, Output, and API Documentation

B-rep Generator's input XML file contains deserialized geometric descriptions (IfcDefinitionShapes) of COGITO's input IFC4x3 file elements. These descriptions are organized in a tree-like structure, presented in blue colour, in the left part A of Figure 11. At the root level, this structure contains the description of the projects' district (if it exists), followed by the building subtree description (if any), which is also considered a construction facility, and non-building facility subtrees (these subtree descriptions include roads, rails, bridges). Every facility subtree consists of parts with specific 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.

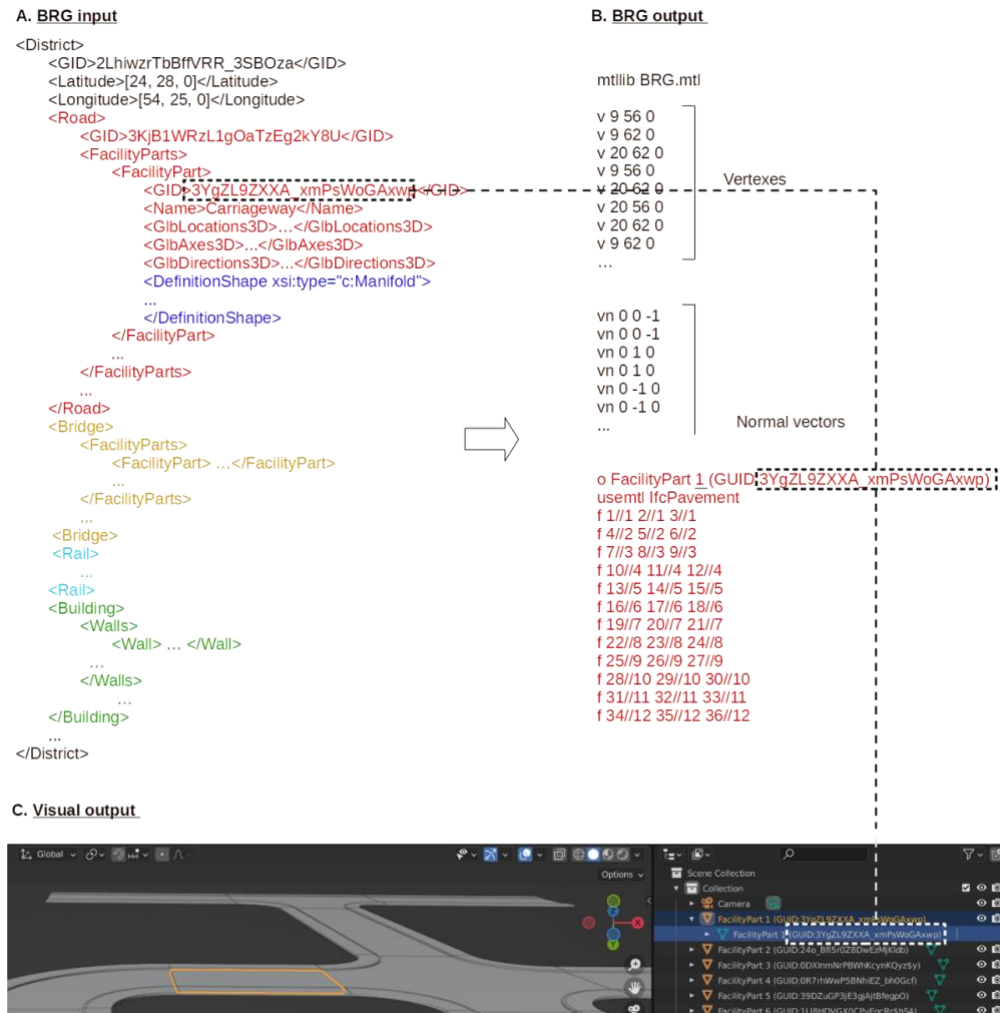


Figure 11: Input and output of B-rep Generator tool.

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-hand side of Figure 11 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 blocks across the parts of Figure 11. We can also use geometric mesh viewers such as Blender to visualise the resulting OBJ file as illustrated for a specific road segment in part C of Figure 11.

The initial version of the BRG tool supported the architectural domain of buildings. Consequently, that version mainly supported definition shapes as solid representations whose parametric descriptions contained non-curved elements. The only non-curved geometric parametric description supported by the v1 of the tool was the trimmed curve based on a curved profile (circle, ellipse, ...) description.

Since COGITO is expected to cover other domains apart from the architectural building domain, new geometric definition shapes must be supported by the BRG tool. Therefore, the newly added definition shapes in the second version of the BRG tool also changed the XSD definitions of BRG's input XML file.

These new solid definition shapes are:

- **IfcAdvancedBrep:** This solid definition shape consists of multiple surfaces that are not necessarily planar. We added support for Cylindrical, BSpline and SurfaceOfRevolution surfaces in the new definition shapes.
- **IfcRevolvedAreaSolid:** This solid definition shape requires a planar profile definition, an axis definition around which a revolution takes place, and the starting and ending angles of this revolution.
- **IfcSurfaceCurveSweptAreaSolid:** This solid definition shape requires a profile surface definition and a directrix definition (curve definition) along which a swept operation takes place. We added new 3D curve definitions for the directrix for this definition shape, including the BSplineCurve.
- Additionally, we added new planar profile definitions for the IfcRevolvedAreaSolid and IfcSurfaceCurveSweptAreaSolid shapes. The added new profile definitions are parametrized profiles frequently encountered in construction projects (supporting structural beams, railway tracks). These newly added profiles are IShapeProfile, CShapeProfile, ZShapeProfile, LShapeProfile, TShapeProfile, UShapeProfile, AssymmetricIShapeProfile and TrapeziumShapeProfile. All these profile definitions can also be found in [6].

3.2.4 Application examples

We tested the BRG tool with IFC data from the RSRG demonstration site. The modelled objects include a few hundred meters of train rail structure and its supporting structure (slippers and foundation layers). Since the construction will be completed in different phases, several different constant-height rectangular construction zones were created. The boundary representations of the BRG tool-generated construction zones along the rail track are displayed in Figure 12.

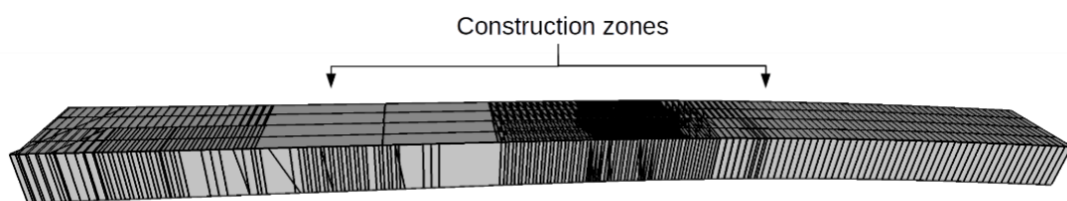


Figure 12: Boundary representations of construction zones of COGITO's RSRG rail project, generated by the B-rep Generator tool.

Inside the site's construction zones, the rail tracks are supported by the slippers and several foundation layers, as displayed with wireframe representation in Part 1 and solid representation in Part 2 of Figure 13.

Part A images of Figure 13 display the boundary representations of the contents of the construction zones, and part B images, display the obtained triangulations by the BRG tool of the construction zones. The solid representations are used for rendering purposes here (Part A of Figure 13) as they display the actual surfaces of each solid object. The triangulated representations (parts B of Figure 13) are more detailed since boundary polygons of more than three sides are split further into triangles. This operation is performed by the BRG tool to produce geometries that can be easily manipulated and rendered in GPUs. The wireframe representation displays the objects' interior (Part 1 of Figure 13), which cannot be accomplished by the solid representation (Part 2 of Figure 13).

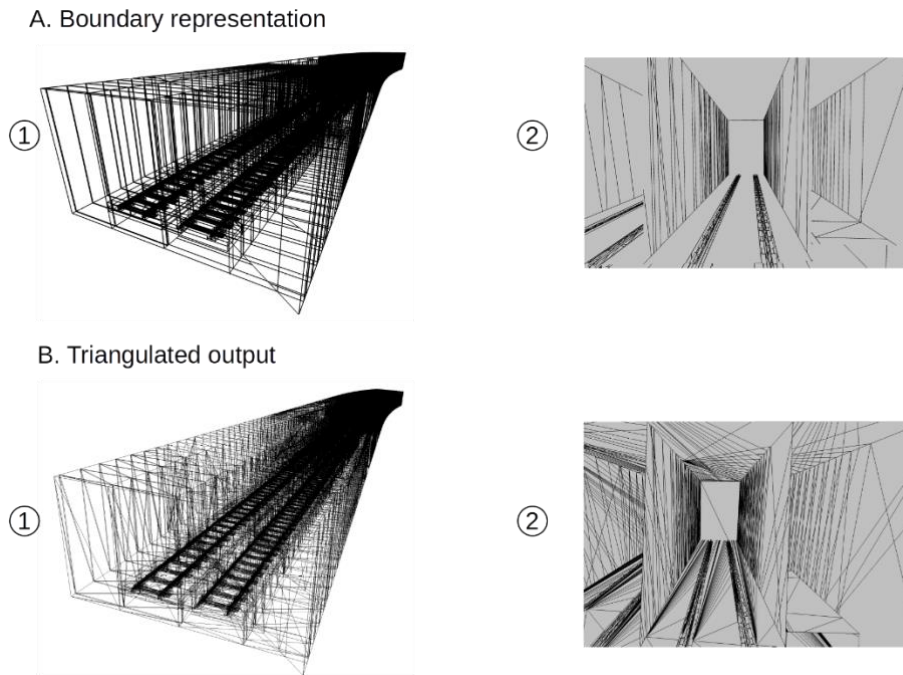


Figure 13: Boundary surface extraction (A) and triangulated output (B) of BRG tool applied on COGITO's RSRG demonstration site (1: Wireframe display, 2: Details inside a construction zone).

3.2.5 Installation Instructions

The B-rep Generator is an internal component of the data post-processing layer of DTP. Consequently, no API is involved, and therefore no installation is required.

3.2.6 Development and integration status

The initial version of the BRG tool was only for 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). In line with the additional infrastructure elements (road, bridge, rail) in the IFC4x3 specification, we updated the tool to support these new facilities. To simplify building domain complexity, we adopted the nested facility part abstraction of IFC4x3. In the second version, the tool's operations remain unchanged at the leaf level under the facility part.

We also updated the tool to support the new solid geometries related to the new IFC4x3 domain extensions, to support new parametric curved solid geometries appearing in non-building constructions domains (railways, tunnels, etc.).

3.2.7 Requirements Coverage

The requirements coverage of the BRG tool, according to D2.5, is summarized in Table 5. The tool successfully receives as-planned construction data from IFC files (BIM models, Req 1-2). The tool can be scaled up (Req-2.1) as the project progresses by including more construction elements extracted from IFC and displayed as triangulated B-reps. The BRG tool is highly responsive (Req-2.2) and available (Req-2.4),

and can be executed in a parallelizable and containerized fashion communicating via the message broker (Figure 2). BRG uses internally low-level C++ libraries for fast geometric operations and multithreading routines which are very efficient on multi-core CPU hardware resources.

Table 5: BRG tool's requirements coverage from D2.5.

ID	Description	Type	Status
Req-1.2	Receives as-planned data (BIM models)	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

3.2.8 Assumptions and Restrictions

Currently, the B-rep Generator covers the COGITO's user's 3D visualization requirements as it supports fast, accurate and on-demand BIM geometry visualisation for non-building domains such as roads, rails, and bridges. The IFC4x3 schema supports non-building domains as facility parts, and their geometric information is exported in OBJ format by BRG.

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.
- Curved geometric representations are approximated linearly in the OBJ file output by polyhedral boundary representations. The resolution level of these approximate polyhedral boundary representations can be adjusted.

3.3 IFC Optimizer

IFC exporters in BIM authoring tools often generate multiple instances of the same object, increasing IFC text verbosity. This "same-object" repetition becomes even more frequent when including the geometric content of IFC models. In this context, 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 in 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 IFC entities.

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

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

These UCs benefit from a lossless reduction of the IFC file sizes and helps expedite the operations of these applications in case large IFC files are required to be processed. The IFC Optimizer provides this compression functionality. (see Figure 1). We describe the operational structure of the IFC Optimizer in the following subsections.

3.3.1 Prototype Overview

The IFC Optimizer reduces 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 IFC objects and to merge them when they are identical. Furthermore, it updates the express identifiers of the deleted objects to maintain the original connections. This optimization operation is applied to an input IFC₀ to produce an output IFC₁ file, as illustrated in Figure 1.

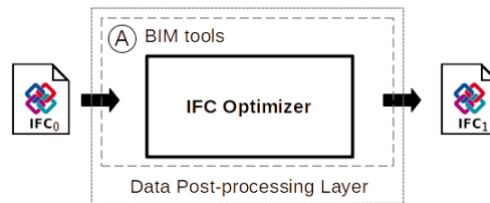


Figure 14: Block diagram of IFC Optimizer's ETL operation.

The operation of the 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 to hash keys and stored in 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 matching IFC text lines and corresponding hash key values. The algorithm keeps one hash key value and corresponding IFC line entry for every one of these sets.
3. For every set isolated in Step 2, all references to its hash key values in the IFC text are replaced by the set's single representative hash key-value entry. This helps maintain the IFC structure intact.
4. Finally, after all the replacements of Step 3 are completed, for every set extracted in Step 2, all elements (hash key entries and respective IFC lines) except the representative ones are removed.

3.3.2 Technology Stack and Implementation Tools

IFC Optimizer is based on the technologies presented in Table 6Table 7.

Table 6: Libraries and Technologies used in IFC Optimizer.

Library/Technology Name	Version	License
IFC SDK	1.0.0	UCL Proprietary

3.3.3 Input, Output, and API Documentation

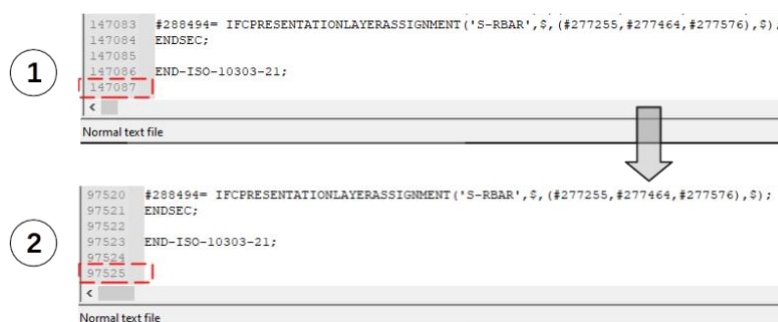


Figure 15: Input and Output files of IFC optimizer

The IFC optimizer uses as input any BIM file generated by the exporters of openBIM authoring tools, such as Autodesk's Revit or Graphisoft's Archicad, as described in subsection 3.5.3. The IFC optimizer deduplicates repeated definitions and keep only one main segment as a reference point. Additionally, the tool updates cross-references. The resulting IFC file is usually significantly reduced in size. f Figure 15 shows the input and output files, with the original file having a total of 147087 lines and the resulting file having a total of 97525 lines – a very significant reduction.

3.3.4 Application example

More detail on the compression operations in the input IFC files and the production of optimized output IFC Files by the IFC optimizer is shown in Figure 16.

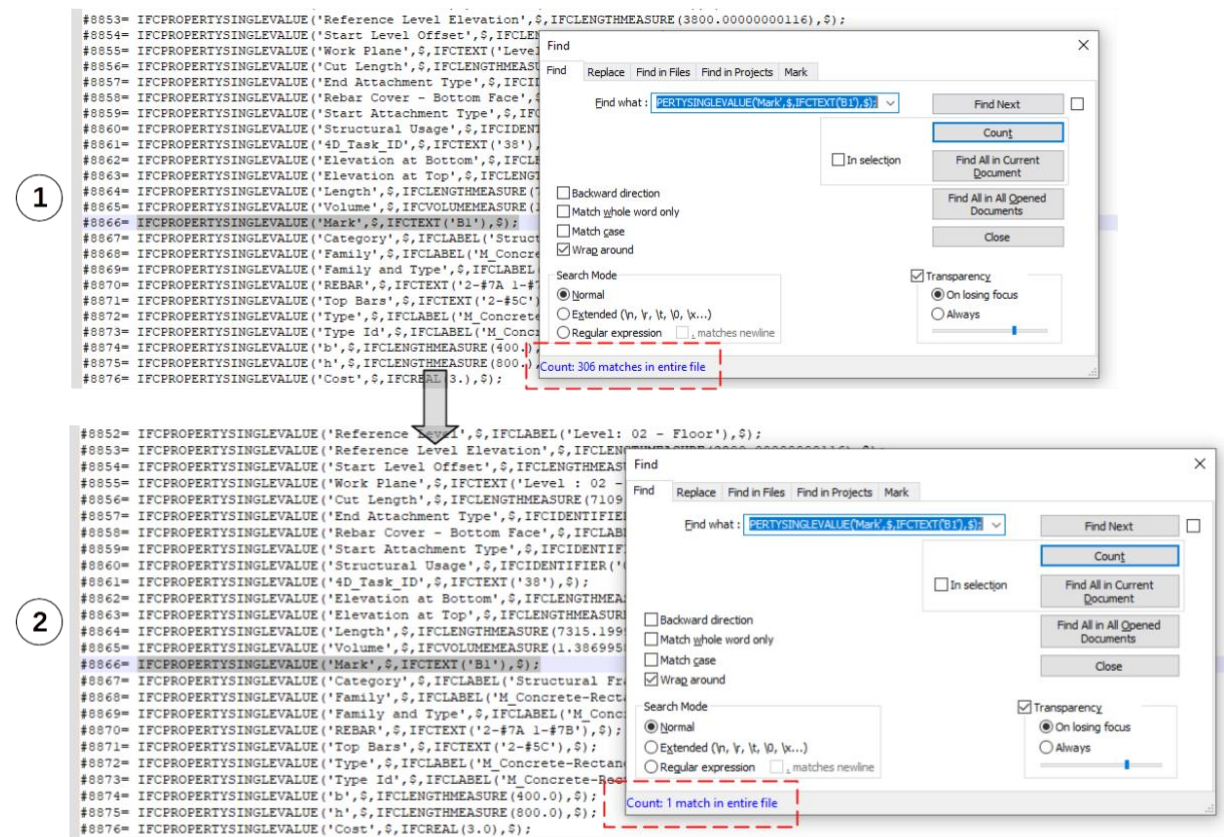


Figure 16: Example of IFC file optimization operations of IFC Optimizer tool.

As we can see, the snippet:

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

in the original IFC file, appears a total of 306 times. After the text operations finish, the same line is counted only once in the tool's optimized output IFC file. 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 highlights that the IFC Optimizer has added the necessary references to maintain the IFC schema structure.

3.3.5 Installation Instructions

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

3.3.6 Development and integration status

Currently the IFC optimizer is tested on a demo IFC BIM file of a school building. 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 0.

3.3.7 Requirements Coverage

Currently, the IFC optimizer receives as input open BIM files conforming to the IFC4 standard [1]. For the correct operation of the tool, the input file needs to have a structure according to the standard specification. Any violation of this structure in the input file will result in the tool's execution termination without output file generation or compression.

3.3.8 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 similar lines in the IFC textbased on string matching. These strict criteria apply to all text line checks performed by the tool. Geometric criteria for identifying similar lines in a 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 two lines will refer to a single `IFCCARTESIANPOINT((0.,0.))` element.

3.4 Geometric Clash Checker

Since COGITO's as-planned input BIM IFC data originate from different BIM authoring tools, they often contain incomplete information. They do not meet the data input requirements of other COGITO tools. For example, in COGITO, the (spatial construction zone) – (construction element) containment relationship is of paramount importance for creating a complete and semantically linked COGITO data model according to the COGITO ontology described in D3.4. This containment information may not exist in IFC files due to several reasons, which include:

- Human design errors where the BIM designer has not specified all of these containment relationships between construction elements and their spatial zones.
- The inability of the BIM authoring tool to populate its BIM model with these relationships.
- Exporter errors where a BIM authoring tool might not export the relations correctly.

Therefore, it is crucial to enrich the IFC files with all containment relationships using an appropriate geometric clash-containment checking mechanism. This checking mechanism benefits COGITO's correct knowledge graph generation since every element should be linked to the zone it belongs. This checking mechanism is offered within the DTP by an MC tool called Geometric Clash Checker (GCC). The GCC tool is described in more detail next.

3.4.1 Prototype Overview

To GCC tool identifies and enriches the (construction element) - (construction zone) containment relationships in the BIM input files. The GCC tool parses an input XML file produced by the IFC Geometry Exporter of DTP's Data Ingestion layer (see Figure 17). This XML file contains the geometric descriptions (parametric and non-parametric) of the 3D BIM elements (construction elements and construction zones) that conform to the `IfcProductDefinitionShape` class definition. The GCC tool outputs are one OBJ file, for viewing the detected containments and an IFC file enriched with the containment information in the form

of populated **IfcRelReferencedInSpatialStructure** classes. This ETL operation process of the GCC tool is illustrated in the block diagram of Figure 17.

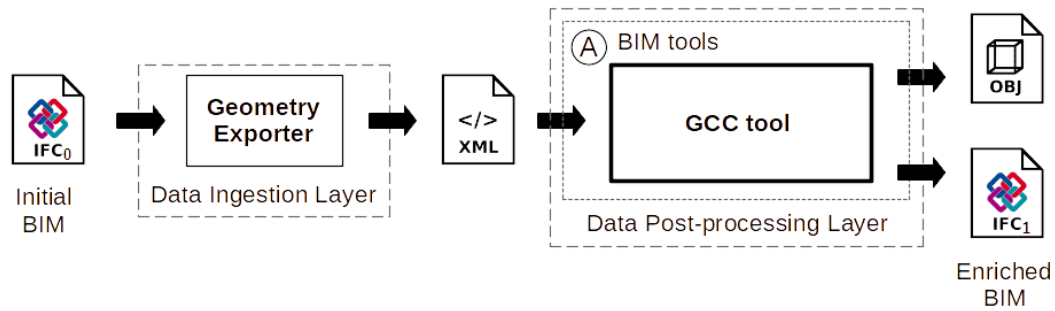


Figure 17: Block diagram of GCC tool's ETL operation.

3.4.2 Technology Stack and Implementation Tools

The GCC tool uses the technologies presented in Table 7.

Table 7: Libraries and Technologies used in GCC

Library/Technology Name	Version	License
Clipper	6.1.3	Boost Software Licence
RapidXML	1.13	Boost Software Licence
IFC SDK	1.0.0	UCL Proprietary

3.4.3 Input, Output, and API Documentation

The input of the GCC tool is an XML file containing one or more “Check” fields. Each of these check fields has three additional fields:

1. A string field “CheckType” indicates the check type, which can take only two possible values: “SelfCheck” and “CrossCheck”.
2. A string field “CheckRelation” indicates the geometric relationships that will be checked. This field can take values such as: “Clash”, “Containment”, and others.
3. An element set (in case the CheckType is “SelfCheck”) or two element sets A and B (in case the CheckType is “CrossCheck”). Each set component has one or multiple product definition shapes (IfcProductDefinitionShape classes) and one unique global identifier (GUID). Each product definition shape contains parametric or non-parametric geometric definitions of a solid geometrical object.

The input of the GCC tool is illustrated in part A of Figure 18.

The output of the GCC tool is an OBJ file used as input to the DTP’s GUI for viewing purposes and contains the results of the detection process. This OBJ file is a text file with three parts:

1. A set of vertices that are described by text lines starting with the “v” letter followed by the x,y, and z coordinates of a 3D point representing the respective vertex
2. A set of normal vectors is described by text lines starting with the “vn” letters followed by the x,y, and z coordinates of a 3D point representing the respective normal vector.
3. A set of faces form the boundary representations of the checked element pairs (Element 1, Element 2) and faces related to the detected geometric relationship (Clash X Surfaces, for example). Each face is defined by an individual line in the OBJ file starting with the letter “f”, followed by pairs of vertex and normal vector indexes, separated by space “ ”. Each pair of vertex and normal vector indexes are denoted using an “(X/Y)” notation, where X is the vertex index, and Y is the normal vector index.

The output of the GCC tool is shown on the right of Figure 18.

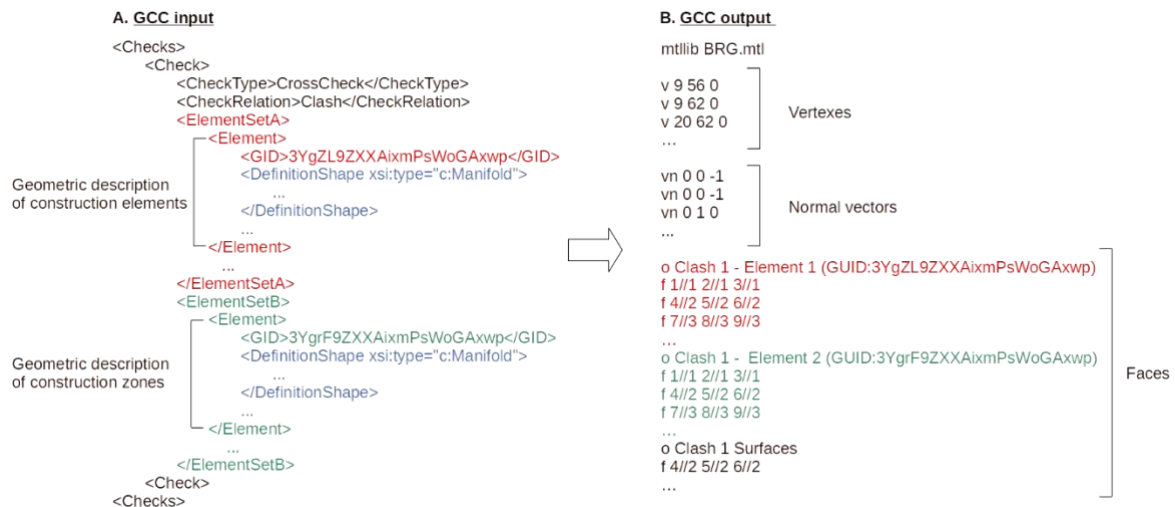


Figure 18: I/O file illustration of GCC tool

3.4.4 Application examples

The checking operation of the GCC tool is demonstrated in Figure 19, where a clash between a column and a space volume is detected. As it is illustrated in Figure 19, the OBJ file output of the GCC tool is received by DTP's embedded viewer, and it is displayed together with the inheritance tree of IFC elements. Through using the inheritance tree (right part of Figure 19), the user can hide or show groups of elements to facilitate the location of the detected clashes in the 3D space. The detected clashes by the GCC tool are displayed in red and highlighted in yellow.

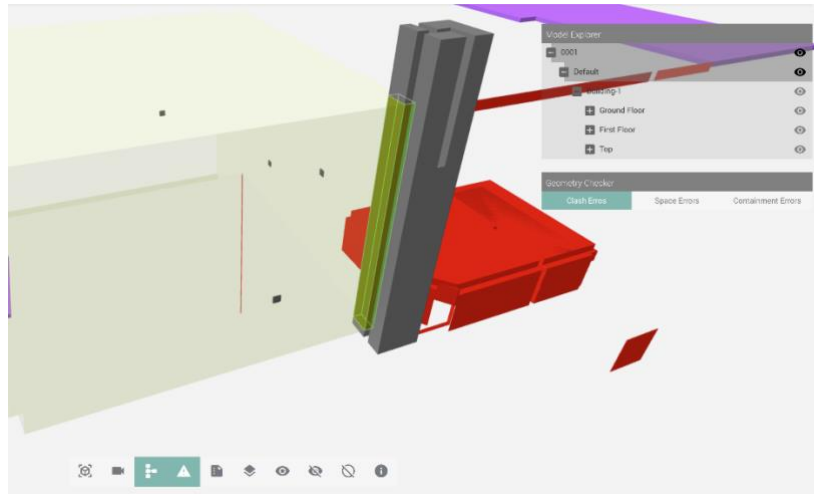


Figure 19: Demonstration example of GCC tool checker (clash checking) using DTP's GUI

3.4.5 Installation Instructions

The GCC tool is an internal tool of DTP's Data Post-Processing layer, and no installation is required.

3.4.6 Development and integration status

At the moment, the GCC tool supports two geometric relations:

- **Clash**, where two or more solid geometric representations intersect, and the intersection is not equal to any of the involved solid representations.
- **Containment** is where a solid geometric representation is inside another solid representation.

Generally, containment can be considered a special case of a clash where all the surfaces of one element are inside the volume of the other.

It is essential to point out that GCC is not a compliance checking tool as COGITO's Geometric Quality Control is. Its goal is to ensure data quality by ensuring a spatial zone containment relationship for every construction element. If other more complex geometric checking relationships are required, they will be implemented in future versions of the tool.

The tool is integrated with the DTP's GUI environment. It works with other BIM-related tools, such as the MVD checker and the B-rep generator, to provide visual feedback to the user of COGITO related to the containment or clash relationships that exist among the construction elements and their related construction spatial zones.

3.4.7 Requirements Coverage

GCC uses internally low-level C++ libraries for fast geometric operations and multithreading routines which are very efficient on multi-core CPU hardware resources. The requirements coverage of the GCC tool, according to D2.5, is summarized in Table 8. The tool successfully receives as-planned construction data from IFC files (Req 1-2). The tool can be scaled up (Req-2.1), and as the project progresses, new checking relationships, apart from clash and containment, can be added. The tool can be scaled up (Req-2.1) as the project progresses by including more construction elements extracted from IFC and displayed as triangulated B-reps. The GCC tool is highly responsive (Req-2.2) and available (Req-2.4), and can be executed in a parallel and containerized way, communicating via the message broker (Figure 2).

Table 8: MVD Checker's requirements coverage from D2.5

ID	Description	Type	Status
Req-1.2	Receives as-planned data (BIM models)	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

3.4.8 Assumptions and Restrictions

Like the requirements of the MVD Checker analysed in the previous subsection, the GCC tool requires that its input IFC data files have a data structure according to the IFC4 schema. Any data structure violation of the tool's input data will result in an incomplete operation without any OBJ file and any visible results displayed in the DTP's GUI.

Internally the GCC tool uses boundary representations of solid objects to perform its geometric clash detection operations. This restriction requires all objects with a curved solid geometric description to be approximated as polyhedral solid representations. These approximations limit the tool's ability to detect geometric relations in the volume differences between the solid curved geometries and their polyhedral approximations. The approximation step can be added as a user-adjustable parameter to overcome this difficulty. By increasing the approximation step, other geometric clashes can be detected at the location of these volume differences.

3.5 Construction RDF ETL

According to the use cases defined in D2.5, several COGITO applications require access to the information related to the elements of COGITO's input 3D BIM files. These tools require the geometric descriptions of these construction elements and their associated properties such as tasks, material properties, point clouds, etc. Within the BIM files, these elements, with their geometry and properties, are organized in a hierarchical tree-like structure formed by inheritance relations. In this inheritance 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, at the end leaves of the tree, the elementary facility parts (or construction components which cannot be further subdivided). Inside this tree-like structure of the input 3D BIM files,

the semantic links of the construction elements are intertwined (serialized and compressed) with their geometrical representations and properties of these elements to avoid data verbosity. This fusion results in inefficient data query times due to required data deserialization operations.

Specific applications, including the ones at COGITO, do not always require detailed geometric information – this is reflected in the COGITO ontology of D3.5. The RDF models represent a flat structure that focuses on specific information requirements. This way, by constructing application-specific models, instantiating and managing the data becomes much simpler. For visualization purposes, the geometric content of the input BIM files can be extracted and converted into a graphics-friendly form (such as OBJ format by the BRG tool) when geometric information is needed.

Many COGITO's use cases and respective tools need access to the COGITO data models, not the complete geometric information. An example of such a query could be retrieving the set of as-planned construction elements in a specific time window. The UCs which do not require a construction data model include the following:

UC2.1: Geometric QC tool requests point cloud data and as-planned 4D BIM data.

UC2.2: Visual Data Pre-processing tool construction components and tasks.

UC4.1: DCC tool requests as planned 4D BIM data.

UC4.2: DigiTAR tool requests QC/3D BIM with geometry data and H&S/3D BIM with geometry data.

The Construction RDF ETL tool performs the data transformation to generate part of the complete RDF graph and is described in more detail in the following subsections.

3.5.1 Prototype Overview

The operation of the Construction RDF ETL tool is outlined in the block diagram of Figure 20. The Construction RDF ETL produces a knowledge graph containing links between different construction zones (that have implicit geometry) with corresponding construction elements (without geometry). These links are extracted from the 3D BIM data contained in the tool's input IFC file.

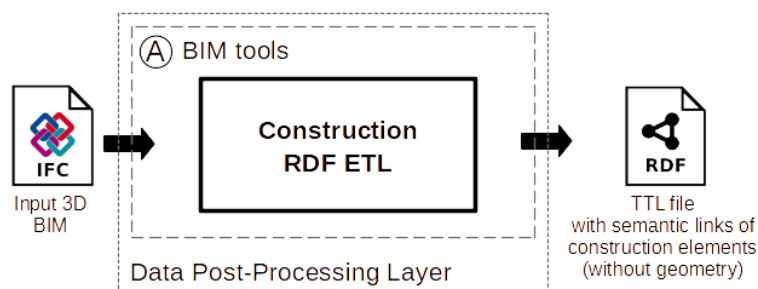


Figure 20: Block diagram of Construction RDF ETL's ETL operation.

The operation of the Construction RDF ETL 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 objects are stored in the memory.
3. The set of the construction elements is extracted from the loaded IFC classes on memory, including their properties (area, volume, related construction task and possible containment relations to other construction elements).
4. Based on the COGITO ontology, the relevant parts of the construction ontology are instantiated.
5. The in-memory instances get serialised to produce the final TTL RDF file, which is the tool output that gets returned.

3.5.2 Technology Stack and Implementation Tools

The Construction RDF ETL uses the technologies presented in Table 9.

Table 9: Libraries and Technologies used in Construction RDF ETL

Library/Technology Name	Version	License
IFC SDK	1.0.0	UCL Proprietary
stellar-base-sseclient	0.0.21	MIT License
Wheezy template	3.1.0	MIT License

3.5.3 Input, Output, and API Documentation

The construction RDF ETL tool receives as input an IFC BIM file in IFC4 format [1] (including the latest IFC4x3 schema), as illustrated in the screenshot in part 1 of Figure 21 that refers to an IFC4 file exported from Autodesk's Revit 2021. The IFC model is defined in EXPRESS with instances exported by modelling tools in STEP as shown in Figure 21. IFC4x3 has extended the previous versions to cover infrastructure constructions. These files can act as a shared 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. The IFC4 files used as input to the Construction RDF ETL contain geometric and non-geometric information on all the elements 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, is processed by the BIM management component of the 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 3.2. The non-geometric content of the IFC file includes properties of the construction objects, some of which are used by the Construction RDF ETL tool, such as area and volume properties. Additionally, the non-geometric content of the IFC file contains relations among the construction elements used by the Construction RDF ETL tool, such as the containment relations produced by the GCC tool analyzed in section 3.4.

A screenshot of the TTL RDF file output of the Construction RDF ETL tool is presented in Figure 21. This TTL RDF file contains the construction elements of the input IFC file described as subjects 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 Figure 21, 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. Appropriate prefixes are added in the header of the output TTL RDF file, referring to the used ontologies.

The elements contained in the TTL RDF output of the Construction RDF ETL 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 the current element in the IFC input file.
- **props:hasVolume**: Points to a double number equal to the current element's volume.
- **props:hasArea**: Points to a double number equal to the current element's area, if the element is aligned to a horizontal plane (for example floor slab).



Figure 21: Input/Output operation of Construction RDF Generation tool

3.5.4 Application example

An example of a TTL RDF file produced by the Construction RDF ETL is displayed in Figure 22. In this example, the BIM IFC4 file of a demo building from a school construction project is used as input to the Construction RDF ETL. 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 at 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 is displayed in Figure 22.



Figure 22: 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 ETL tool for these examined elements is displayed in part 2 of Figure 22. As shown in Figure 22, the Construction RDF ETL 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).

3.5.5 Licensing

The Construction RDF ETL 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 requires a docker container and 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

The Construction RDF ETL has been tested on BIM IFC files related to the school building construction project. Later in the project, and as BIM data from COGITO demonstration sites become available, the tool will be tested on these sites' respective IFC data files. If additional information requirements appear, the tool will be updated to capture additional information. Requirements Coverage

The requirements coverage of the Construction RDF ETL tool, according to D2.5, is summarized in Table 10. The tool successfully receives BIM-related data (Req-1.2) and generates an RDF graph populating the COGITO ontology (Req-1.5). The tool can be scaled up (Req-2.1) as new projects and related construction elements are tested.

Table 10: Construction RDF ETL requirements coverage from D2.5.

ID	Description	Type	Status
Req-1.2	Receives as-planned data (BIM models)	Functional	Achieved
Req-1.5	Populates COGITO's ontology	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

3.5.8 Assumptions and Restrictions

The correct operation of the Construction RDF ETL requires a 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 for modelling the construction elements.

All element containment relations should be modelled in the input BIM file to obtain the hierarchy tree for construction elements. To ensure the relationships exist, the following actions are suggested:

- Issuing 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 zones containing construction elements, using a BIM authoring tool (e.g., Revit).
- Checking whether the IFC exporter of the BIM authoring tool exports these containment relations in its output IFC file.
- Use of the GCC tool if the above two conditions do not apply.

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

- The input IFC BIM file should conform to the IFC4 standard [1].
- 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 and steel.
- All required property values of the construction elements in the input IFC should be included in the appropriate format, e.g. integer or string.

4 Tools that work with other (non-BIM) data as input

The ETL and MC tools that work with BIM data as input include (Category B in Figure 1) include:

(B1.1) **Project RDF ETL** enriches the knowledge graph data model with construction project-related data.

(B1.2) **Process RDF ETL** enriches the knowledge graph data model with construction schedule-related data.

(B1.3) **Resources RDF ETL** enriches the knowledge graph data model with construction resources-related data.

(B2.1) **RDF Validator** validates the final enriched COGITO's knowledge graph data model using predefined checking rules.

4.1 Project RDF ETL

Data traffic in COGITO among data sources, providers and consumers is diverse. Without data structures acting as a common point of reference, the related data exchanges will be left unorganized, leading to confusion. In this case, project identification data (name, description, and project id) are a common reference point to bind all other data structures of a single construction project.

The project data are generated when the project is created after the DTP's Authentication layer applications authenticate the COGITO user. 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, the list of projects is returned from the DTP in:

- UC1.1 to the PMS UI.
- UC1.2 to the WOEI.
- UC2.1 to the Visual Data Pre-processing.
- UC2.2 to the Visual Data Pre-processing UI.
- UC3.1 to SafeConAI UI.
- UC3.2 to SafeConAI UI.
- UC4.1 to DCC.
- UC4.2 to DigiTAR.

The 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. The data are provided in text JSON form by the authentication layer of DTP. This project-related information should be converted to TTL format to link the JSON data to the TTL RDF graph models produced by other RDF ETL tools. This conversion is performed by the Project RDF ETL tool, the first ETL tool in the series of three semantic graph enrichment tools (ETL tools of block B), presented in Figure 1. We describe the operation of Project RDF ETL next.

4.1.1 Prototype Overview

The Project RDF ETL tool is a file translation component. It transforms an input JSON text file (containing project-related data) to a TTL RDF file in the output, as summarized in Figure 23.

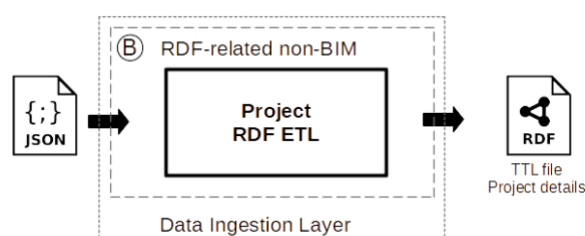


Figure 23: Block diagram of Project RDF ETL's ELT operation.

The TTL RDF file of Project RDF ETL contains the triplets formatted project-related data classes for the input JSON file.

The operation of Project RDF ETL follows a series of steps:

1. The input JSON file is de-serialised to create in-memory objects.
2. Based on facility domain ontology (published at <http://cogito.iot.linkeddata.es/def/facility>) appropriate objects are initiated.
3. The entities formed in step 2 are populated using data from the in-memory objects from step 1.
4. The populated memory objects from step 3 export the final TTL RDF serialisation.

4.1.2 Technology Stack and Implementation Tools

The Project RDF ETL uses the technologies presented in Table 11.

Table 11: Libraries and Technologies used in Project RDF ETL

Library/Technology Name	Version	License
stellar-base-sseclient	0.0.21	MIT License
wheelzy.template	3.1.0	MIT License
rdflib	6.1.1	BSD 3-Clause License
requests	2.27.1	Apache License 2.0

4.1.3 Input, Output, and API Documentation

The input of Project RDF ETL 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 Figure 24:

- **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 ETL is a TTL RDF file displayed in Part 2 of Figure 24. This output file contains three triplets referring to the JSON input file classes. This RDF output file and the TTL RDF output files of the other three RDF ETL 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 SPARQL queries can query it originated from other COGITO applications.

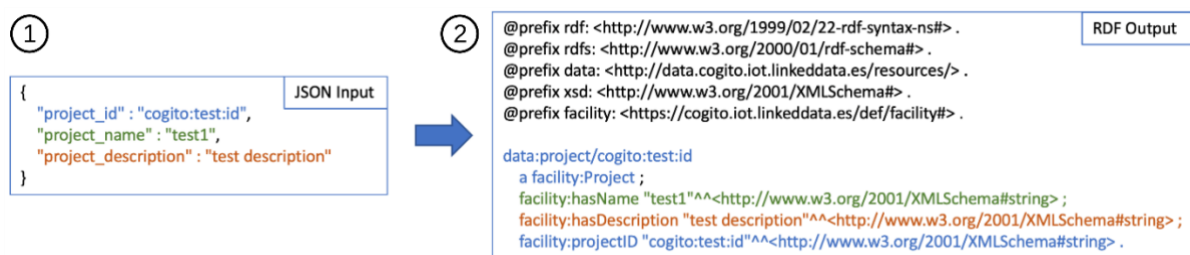


Figure 24: Input/Output of Project RDF ETL

4.1.4 Application example

This tool parses a queue of server-sent events, where events are published to perform specific actions. The project RDF ETL, in particular, is in charge of collecting the events related to generating knowledge graphs for the projects to be registered in the platform. After collecting the project event, the project RDF ETL extracts the information encapsulated in JSON format. The Knowledge Graph is generated using this information combined with the previously defined mappings and the Helio translation tool. When the translation process finishes, the data generated in the knowledge graph is validated; if the data pass

validation, the RDF is sent to the system. Then after the event is published, it is saved as the knowledge graph in the Triple-Store.

As an example of an application for this ETL tool, the data, once translated, validated and stored in the Triplestore, will be published in the domain of <https://data.cogito.iot.linkeddata.es/resources>. This enables the query using a SPARQL endpoint deployed in <https://data.cogito.iot.linkeddata.es/sparql>. An example of a published project is shown below.

Resources > http://data.cogito.iot.linkeddata.es/resources/Project_079637bb-87d5-4fef-9704-d556364d90bb

[Properties \(3\)](#)
[Relations \(1\)](#)
[Types \(1\)](#)
[Equivalences \(1\)](#)
[Download](#)

Filter ...

Property	Value
https://cogito.iot.linkeddata.es/def/facility#isRelatedToProcess	http://data.cogito.iot.linkeddata.es/resources/FE090C7C-7AEC-42FE-9452-AD0AA8643D28
https://cogito.iot.linkeddata.es/def/facility#projectID	079637bb-87d5-4fef-9704-d556364d90bb^^http://www.w3.org/2001/XMLSchema#string
https://cogito.iot.linkeddata.es/def/facility#hasDescription	test description^^http://www.w3.org/2001/XMLSchema#string
https://cogito.iot.linkeddata.es/def/facility#hasName	test1^^http://www.w3.org/2001/XMLSchema#string
http://www.w3.org/2002/07/owl#sameAs	http://data.cogito.iot.linkeddata.es/resources/project/079637bb-87d5-4fef-9704-d556364d90bb
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://cogito.iot.linkeddata.es/def/facility#Project

Figure 25: Published Project triples example

4.1.5 Licensing

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

4.1.6 Installation Instructions

The installation requires a docker container and a configuration file indicating the type of event to pick up from the server-sent events queue, the Helio tool's endpoints and the server providing the events, and the RDF Validator tool.

4.1.7 Development and integration status

After the development and integration of the tool into the system are completed, the developed system then be deployed to be used by the other COGITO tools responsible for the knowledge graph generation and validation process.

4.1.8 Requirements Coverage

The requirements coverage of the Project RDF ETL tool, according to D2.5, is summarized in Table 12. The tool successfully receives project-related metadata (identifier, name and description) and generates an RDF graph populating the COGITO ontology (Req-1.5). The tool can be scaled up (Req-2.1), and as the project progresses, more information can be added to the Project RDF graph generated. The tool is highly responsive (Req-2.2) and available (Req-2.4) and can be executed in a parallelizable and containerized fashion using the Flask framework.

Table 12: Project RDF ETL requirements coverage from D2.5.

ID	Description	Type	Status
Req-1.5	Populates COGITO's ontology	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

4.1.9 Assumptions and Restrictions

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

- 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 retrieve the data received from the Server-Sent Events queue.
- The identifier of the project must be a UUID.

4.2 Process RDF ETL

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 pre-defined 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). The workflow process stays at a higher-than-task level, 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 who 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 XML file format.

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

- UC1.1: PMS tool requests 4D BIM data. These data contain this construction schedule information.
- UC2.2: Visual Data Pre-processing tool requests building components and tasks.
- UC4.1: DCC in requests as planned 4D BIM data.
- UC4.1: DCC requests 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 XML files of COGITO should be translated into a semantic graph format and linked to respective construction elements contained in output of the Construction RDF ETL, presented in the previous subsection. The tool that performs this translation is defined as the Process RDF ETL and is described in the following subsections.

4.2.1 Prototype Overview

The operation of the Process RDF ETL is summarized in the block diagram of Figure 26. Essentially, the Process RDF ETL takes the construction schedule input file (in XML format) as input and populates the data classes according to the COGITO's process domain ontology, forming an output RDF file in TTL file format.

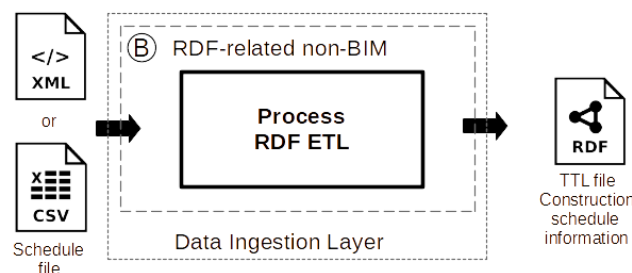


Figure 26: Block diagram of Process RDF ETL's operation.

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

1. Depending on the type of 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.

4.2.2 Technology Stack and Implementation Tools

Process RDF ETL is based on the technologies presented in Table 13.

Table 13: Libraries and Technologies used in Process RDF ETL

Library/Technology Name	Version	License
stellar-base-sseclient	0.0.21	MIT License
wheelzy.template	3.1.0	MIT License
rdflib	6.1.1	BSD 3-Clause License
requests	2.27.1	Apache License 2.0

4.2.3 Input, Output and API Documentation

Process RDF ETL input XML file contains the information related to the construction process (construction schedule and related tasks), which will be translated into RDF format in the tool's output.

The tool's input 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 using the Work Breakdown Structure (WBS), Predecessor and SubTask task properties. Part 1 of Figure 27 indicates the properties of each task which are contained in the input to the Process RDF ETL tool, XML file:

- **ID:** Is an integer used for the identification of each task.
- **Status:** A string indicating the task status, such as "Active".
- **Task_Name:** A string describing the name of a task.
- **Duration:** A value indicating the duration of the task in time units.
- **SubTasks:** A list of the subtasks of the task.
- **Predecessor:** A task preceding 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 ETL are shown in Figure 27. This output file is a TTL RDF file, which contains the triples referring to the process that includes several construction tasks and triplets referring to individual construction tasks and their related properties. This RDF file instance, generated by the Process RDF ETL tool, will be stored later in a triple store where SPARQL queries can query it.

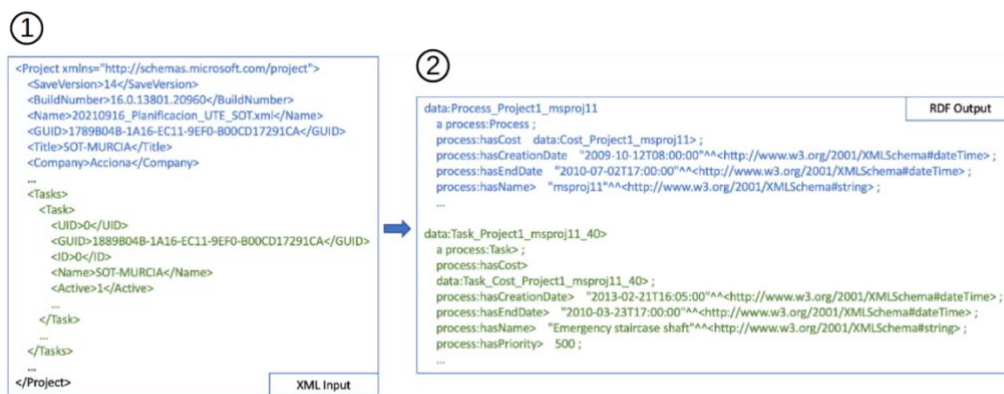


Figure 27: O/I file ETL operations of the Process RDF ETL tool

4.2.4 Application example

An example of a triple that is generated by the Process RDF ETL component in the leaf and parent task elements, is presented in parts 1 and 2 of Figure 28, respectively. What differentiates the triplets of a leaf task from the triplets of a parent task, as the dashed blocks in Figure 28 highlight, is the existence of an `isSubTaskOf` triplet in the leaf task and the existence of the `hasSubTask` triplet in the parent task. If both `isSubTaskOf` and `hasSubTask` exist in a task element, the task element is an intermediate task node.

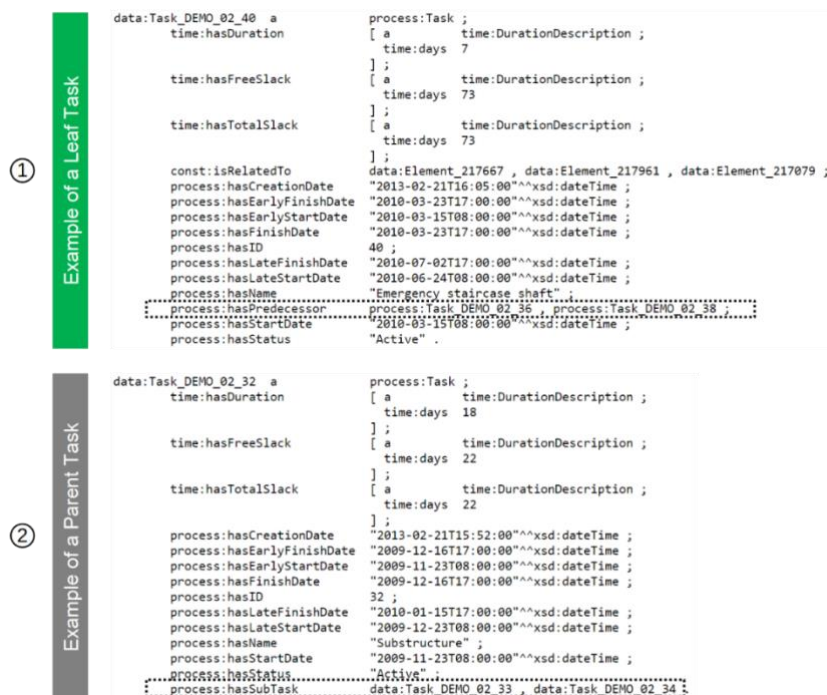


Figure 28: Example of triple generation of a Leaf and Parent Tasks by the Process RDF generation tool

As an example of an application for this ETL tool, the data, once translated, validated and stored in the Triplestore, will be published in the domain of <https://data.cogito.iot.linkeddata.es/resources>, being able to be queried using a sparql endpoint deployed in <https://data.cogito.iot.linkeddata.es/sparql>. An example of a published task is shown below.

Resources > http://data.cogito.iot.linkeddata.es/resources/Task_079637bb-87d5-4fef-9704-d556364d90bb_FE090C7C-7AEC-42FE-9452-AD0AA8643D28_117

[Properties \(7\)](#)
[Relations \(2\)](#)
[Types \(1\)](#)
[Equivalences \(1\)](#)
[Download](#)

Filter ...

Property	Value
https://cogito.iot.linkeddata.es/def/process#hasStatus	PT0H0M0S ^{^^} http://www.w3.org/2001/XMLSchema#string
https://cogito.iot.linkeddata.es/def/facility#isRelatedToProject	http://data.cogito.iot.linkeddata.es/resources/Project_079637bb-87d5-4fef-9704-d556364d90bb
https://cogito.iot.linkeddata.es/def/process#taskId	480ACC2C-BD92-4C0C-B230-901AA90DFCF ^{^^} http://www.w3.org/2001/XMLSchema#string
https://cogito.iot.linkeddata.es/def/process#isSubTaskOf	http://data.cogito.iot.linkeddata.es/resources/Task_079637bb-87d5-4fef-9704-d556364d90bb_FE090C7C-7AEC-42FE-9452-AD0AA8643D28_116
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://cogito.iot.linkeddata.es/def/process#Task
https://cogito.iot.linkeddata.es/def/process#hasPriority	500 ^{^^} http://www.w3.org/2001/XMLSchema#integer
https://cogito.iot.linkeddata.es/def/process#hasCreationDate	^{^^} http://www.w3.org/2001/XMLSchema#dateTime
https://cogito.iot.linkeddata.es/def/process#taskId	117 ^{^^} http://www.w3.org/2001/XMLSchema#string
https://cogito.iot.linkeddata.es/def/process#hasProgress	PT0H0M0S ^{^^} http://www.w3.org/2001/XMLSchema#string
https://cogito.iot.linkeddata.es/def/process#hasName	Emergency staircase shaft ^{^^} http://www.w3.org/2001/XMLSchema#string
http://www.w3.org/2002/07/owl#sameAs	http://data.cogito.iot.linkeddata.es/resources/task/079637bb-87d5-4fef-9704-d556364d90bb_FE090C7C-7AEC-42FE-9452-AD0AA8643D28_117

Figure 29: Published Task triples example

4.2.5 Licensing

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

4.2.6 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, the server providing the events and the RDF Validator tool.

4.2.7 Development and integration status

The status of the development and integration of the tool into the system is completed and deployed to be used by the other COGITO tools responsible for the knowledge graph generation and validation process.

4.2.8 Requirements Coverage

The requirements coverage of the Process RDF ETL tool, according to D2.5, is summarized in Table 14. The tool successfully receives as-planned construction schedule data in the form of XML files (Req-1.2) and generates an RDF graph populating the COGITO ontology (Req-1.5). The tool can be scaled up (Req-2.1) and as the project progresses, more information can be added to the Process RDF graph generated. The tool is also highly responsive and available (Req-2.2 and Req-2.4); it can be executed as a service in a parallelizable and containerized fashion through the Flask framework using a queue system.

Table 14: Process RDF ETL requirements coverage from D2.5

ID	Description	Type	Status
Req-1.2	Receives as-planned data (construction schedule)	Functional	Achieved
Req-1.5	Populates COGITO's ontology	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved

Req-2.4	High availability	Non-Functional	Achieved
----------------	-------------------	----------------	----------

4.2.9 Assumptions and Restrictions

The operation of the Resources RDF ETL tool relies on the following assumptions and restrictions applied to the input 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 related to 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...).

4.3 Resources RDF ETL

The completion of every construction project requires a certain number of human and non-human resources. Within COGITO this resources-related information is described in an input CSV file together with the description of the construction task schedules. This information contains resource types rather than the instances of the actual resources. For example, it determines the specific task that is going to be executed by a crane rather than 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 into a TTL RDF form and linked to other TTL RDF files, produced by the previous RDF ETL tools (Project, Construction and Process RDF ETLs), to form the COGITO semantic graph model. This TTL RDF translation of the project resource-related information, from the native CSV format, is performed by the Resources RDF ETL tool, and analysed in the following subsections.

4.3.1 Prototype Overview

The Resources RDF ETL, as the RDF ETL tools analysed in the previous subsections, is a file translation tool which transforms construction resources-related data from CSV format to TTL RDF file format, as summarized in Figure 30.

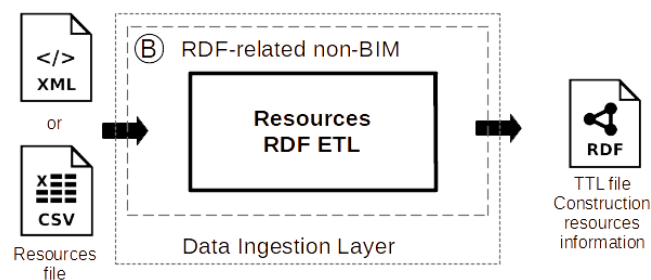


Figure 30: Block diagram of Resources RDF ETL's operation.

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

1. Depending on the type of 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.

4.3.2 Technology Stack and Implementation Tools

Resources RDF ETL is based on the technologies presented in Table 15.

Table 15: Libraries and Technologies used in Resources RDF ETL

Library/Technology Name	Version	License
stellar-base-sseclient	0.0.21	MIT License
wheelzy.template	3.1.0	MIT License
rdflib	6.1.1	BSD 3-Clause License
requests	2.27.1	Apache License 2.0

4.3.3 Input, Output, and API Documentation

Resources RDF ETL input 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 31.

As presented in part 1 of Figure 31, the input CSV 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 produced by the Resources RDF ETL is shown in part 2 of Figure 31, where we can see the triples reflecting the information in RDF format associated with the input process CSV file and the related resource type instances and their properties described previously. This generated RDF file will be stored later in a triple store where it can be queried by SPARQL queries.

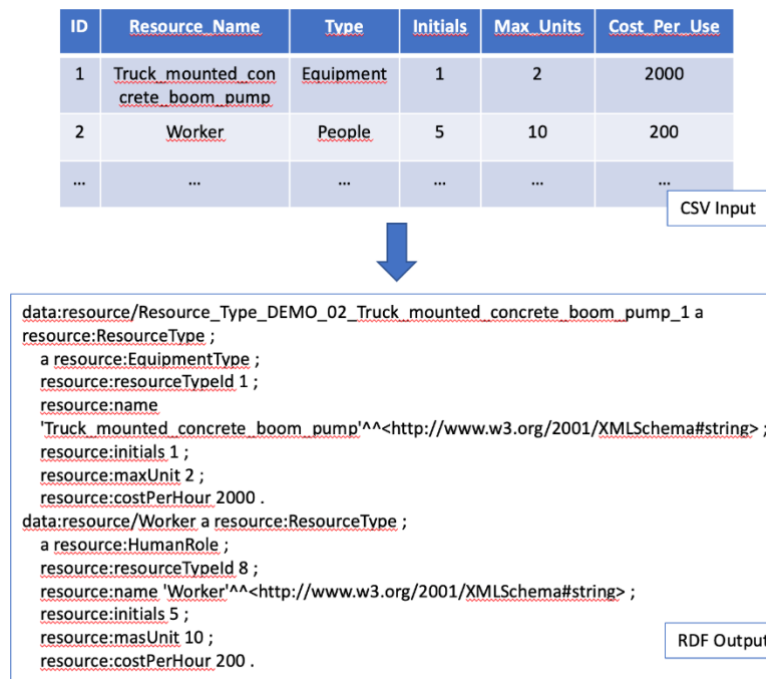


Figure 31: O/I file ETL operations of the Resources RDF ETL tool

4.3.4 Application example

An application example of the Resources RDF ETL's ETL file operations is illustrated in Figure 32. In part 1 of Figure 32, an example of a CSV file instance, which is used as input to the Resources RDF ETL 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 32. 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".

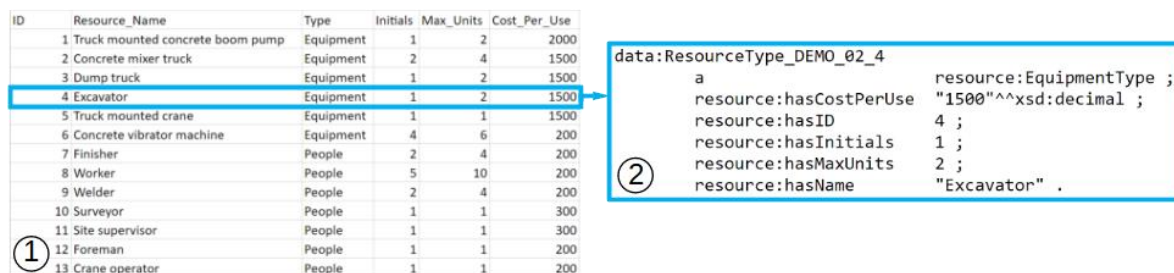


Figure 32: Illustration of the I/O file operations of the Resources RDF ETL tool.

As an example of application for this ETL tool, the data once translated, validated and stored in the Triplestore, will be published in the domain of <https://data.cogito.iot.linkeddata.es/resources>, being able to be queried using a sparql endpoint deployed in <https://data.cogito.iot.linkeddata.es/sparql>. An example of a published resource_type is shown below.

Resources > http://data.cogito.iot.linkeddata.es/resources/ResourceType_079637bb-87d5-4fef-9704-d556364d90bb_Truck_mounted_concrete_boom_pump_1

[Properties \(5\)](#)
[Relations \(0\)](#)
[Types \(2\)](#)
[Equivalences \(0\)](#)
[Download](#)

Filter ...

Property	Value
https://cogito.iot.linkeddata.es/def/resource#costPerHour	2000 ^{^^} http://www.w3.org/2001/XMLSchema#integer
https://cogito.iot.linkeddata.es/def/resource#masUnit	2 ^{^^} http://www.w3.org/2001/XMLSchema#integer
https://cogito.iot.linkeddata.es/def/resource#initials	1 ^{^^} http://www.w3.org/2001/XMLSchema#integer
https://cogito.iot.linkeddata.es/def/resource#name	Truck_mounted_concrete_boom_pump ^{^^} http://www.w3.org/2001/XMLSchema#string
https://cogito.iot.linkeddata.es/def/resource#resourceTypeId	1 ^{^^} http://www.w3.org/2001/XMLSchema#integer
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://cogito.iot.linkeddata.es/def/resource#EquipmentType
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://cogito.iot.linkeddata.es/def/resource#ResourceType

Figure 33: Published Resource Type triples example

4.3.5 Licensing

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

4.3.6 Installation Instructions

The installation requires a docker container and a configuration file indicating the type of event to pick up from the server-sent events queue, the Helio tool's endpoints and the server providing the events, and the RDF Validator tool.

4.3.7 Development and integration status

The status of the development and integration of the tool into the system is completed and deployed to be used by the other COGITO tools responsible for the knowledge graph generation and validation process.

4.3.8 Requirement Coverage

The requirement coverage of the Resources RDF ETL tool, according to D2.5, is summarized in Table 16. The tool successfully receives as-planned available resources data in CSV files (Req-1.2) and generates an RDF graph populating the COGITO ontology (Req-1.5). The tool can be scaled up (Req-2.1) and as the project progresses, more information can be added to the Process RDF graph generated. The tool is also highly responsive and available (Req-2.2 and Req-2.4); it can be executed as a service in a parallelizable and containerized fashion through the Flask framework using a queue system.

Table 16: Resources RDF ETL requirements coverage from D2.5.

ID	Description	Type	Status
Req-1.2	Receives as-planned data (available resources)	Functional	Achieved
Req-1.5	Populates COGITO's ontology	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

4.3.9 Assumptions and Restrictions

The operation of the Resources RDF ETL tool relies on the following assumptions and restrictions applied to the input CSV 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.

4.4 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 database of the Data Persistence layer of DTP. These RDF files follow the TTL file format and are generated using four separate ETL processes described in subsections 4.1, 4.2, 4.3 and 3.5, by the four RDF ETL modules displayed in Figure 1. These files are parts of COGITO's semantic graph. After these parts are generated by the ETL processes, they are linked together to enrich the overall COGITO semantic graph.

Multiple COGITO use cases (UCs) involve queries to the COGITO semantic graph, which is formed by multiple inter-linked data. These UCs include:

- UC1.1: PMS tool in transaction 5, requests from the DTP 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 the COGITO data model should contain the necessary populated data structures. To ensure this, a checking mechanism should be established to report possible: data errors, missing content or semantic links. These are potential 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. As in the case of BIM data and the checking operations applied on these data by the MVD Checker described in a previous section, RDF data have to be checked for possible issues, since these data come from different sources and they have questionable quality. To detect possible 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.

4.4.1 Prototype Overview

The RDF Validator checks the conformity of the generated semantic graphs against a set of predefined checking rules (as in the case of the MVD Checker) in the form of a SHACL shapes file. This checking operation performed by the RDF validator is illustrated in the block diagram of Figure 34.

As illustrated in Figure 34, 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 the data graph.
- **Checking rules:** this is a TTL RDF file containing the checking rules that will 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 a shapes graph.

Finally, as displayed in Figure 34, the RDF Validator outputs the checking or validation results into a JSON text file.

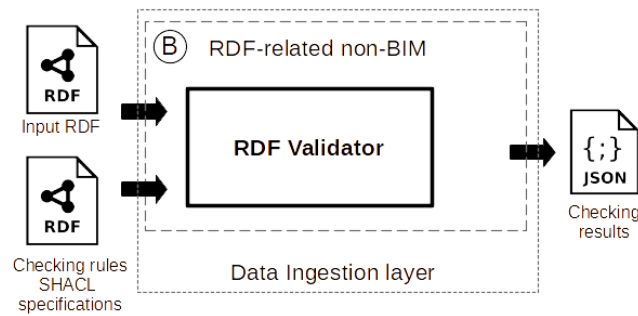


Figure 34: Block diagram of RDF Validator's MC operation.

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

1. Initially, through using Apache/Jena RDF processing tools, the triplets contained in the input RDF files (input RDF graph and SHACL shapes graph) are converted into memory objects.
2. A checking rule is formed for every entity in the objects generated from the shapes graph a checking rule is formed.
3. Using the rules formed in step 2, all objects originating from the RDF data file are checked against these rules.
4. The results of the checking procedure in step 3 are gathered and exported in the output JSON file. If the test results are satisfactory, "Conforms" will be reported in the JSON file. In any other case, the detected issue will be reported.

4.4.2 Technology Stack and Implementation Tools

RDF Validator is based on the technologies presented in Table 17.

Table 17: Libraries and Technologies used in RDF Validator

Library/Technology Name	Version	License
spark-core	2.9.3	Apache License 2.0
sparql-template-velocity	2.7.1	Apache License 2.0
jackson-databind	2.6.3	Apache License 2.0
h2	2.1.210	EPL 1.0 - MPL 2.0
hibernate-core	5.6.5.Final	LGPL 2.1
jena-shacl	4.5.0	Apache License 2.0

4.4.3 Input, Output, and API Documentation

The input RDF files of the RDF validator have been developed based on the defined COGITO ontologies. Based on these ontology definitions certain SHACL shapes have been defined to validate these RDF data. The SHACL Shapes defined for validation consist of numerous constraints used as validation parameters. These parameters include:

- **sh:NodeShape**: responsible for indicating which RDF term from the Knowledge Graph has to be validated. All these defined nodes have to indicate which explicit value is referred to within the RDF.
- **sh:PropertyShape**: indicate the validation on a term that is the existing property between two **sh:NodeShape**.
- **sh:targetNode** or **sh:targetClass**: predicate used to specify the value to which it refers inside the RDF.**sh:property**: used to identify the associated **sh:PropertyShape** to a specific **sh:NodeShape**.
- **rdfs:label** and **sh:name**: used to indicate the specific name and description of the associated node.
- **sh:nodeKind**: used to indicate whether the node is a URI, Literal or Blank Node.

4.4.4 Application examples

The checking operations applied to the TTL RDF files produced by the RDF ETL, refer to specific classes of COGITO ontology and their respective relations (semantic links) which are to going to be checked. Examples of COGITO classes and respective relations to be checked are presented in Table 18.

Table 18: Examples of involved entities in the RDF validator checking operations.

Involved classes (Subject/Object)	Relationships to be checked (Predicates)
facility:Project / process:Process	facility:isRelatedToProcess
process:Process / process: Task	process:hasTask
facility:Element / process: Task	(process:addsElement AND process:isAddedByTask) XOR (process:removesElement AND process:isRemovedByTask) XOR (process:repairsElement AND process:isRepairedByTask) XOR (process:controlsElement AND process:isControlledByTask)
process:Task / resource:Resource	process:hasAssignedResource

Table 18 is not final and will be updated as the project evolves, with new classes and respective semantic links to be checked.

A specific application example of the RDF Validator is displayed in the following Figure 35. A Schedule XML file translated to an RDF graph file by the Process RDF ETL is validated in this example. The top-left image of Figure 35 contains information about the shapes graphs, which are the rules used to perform the validation (input checking rules following the SHACL specifications of Figure 34). For this example, six shapes graph rules have been defined, the first three using the **sh:NodeShape** predicate to validate that the elements are generated from classes defined by the ontology. In this case, these elements include projects, processes and tasks. For the other three defined rules, the predicate **sh:PropertyShape** has been used to validate the existing relations between the different classes of elements defined in the RDF graph to be validated, i.e., **process:hasTask** that is a relation between a **process:Process** and a **process:Task**. The top-right image contains the information in RDF graph format of the input data that are going to be validated (input RDF of Figure 35). Finally, the bottom image of Figure 35, contains the result of the validation, where two errors have been detected. The first error is related to wrong use of the **process:hasTask** predicate, where in the graph it can be seen that a **facility:Project** is associated with the predicate **process:hasTask** to a process: Task. The second error refers to incorrect spelling in the RDF graph, i.e., the process:Process in the example below, is not related to a process:Task entity, but rather a number ("1.1").

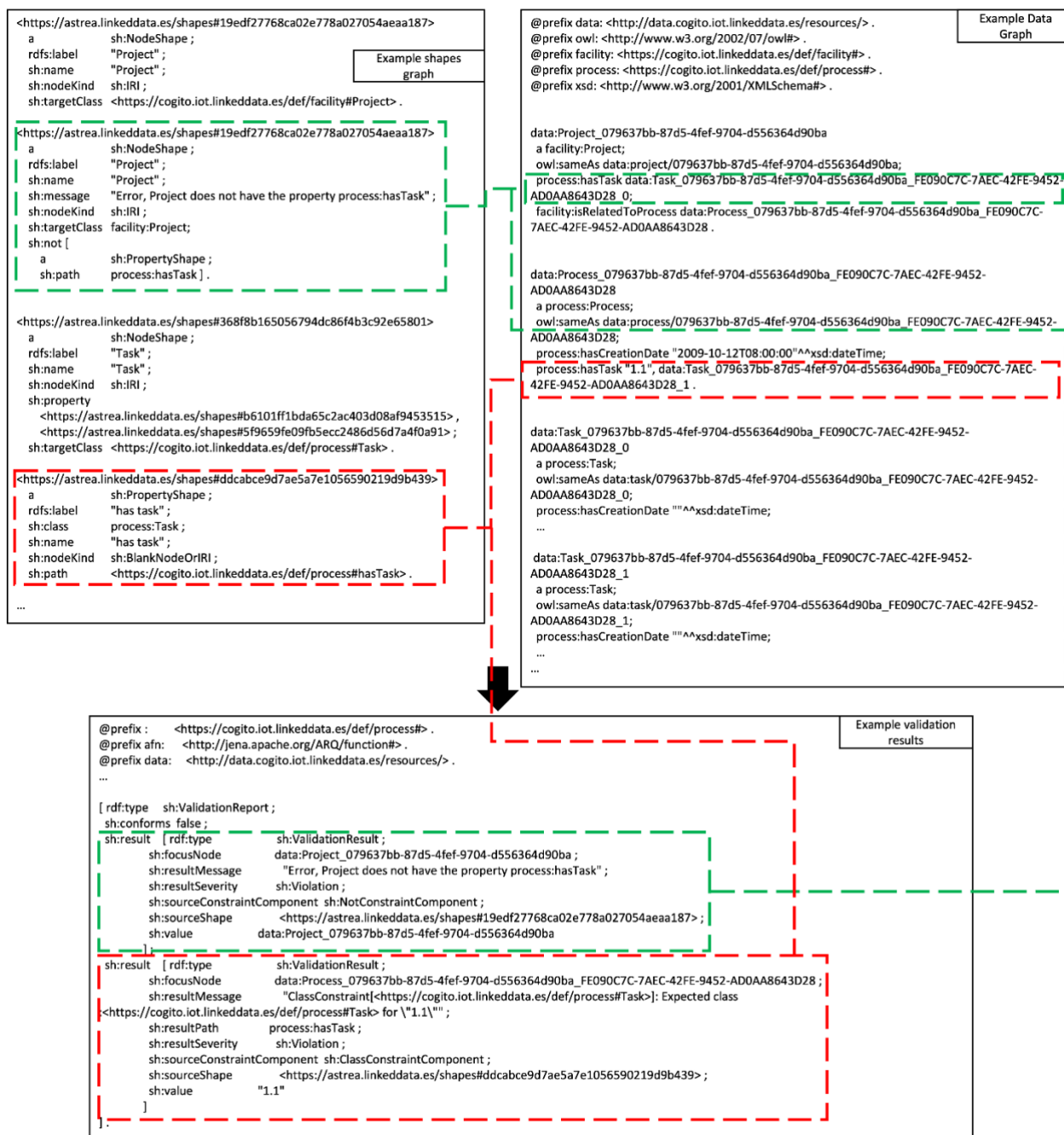


Figure 35: Application example of RDF validator

4.4.5 Licensing

The license of the code used for creating this tool is available in the GitHub repository (<https://github.com/oeg-upm/coppola>).

4.4.6 Installation Instructions

RDF Validator will be used internally in DTP and will not be connected with an external UI, allowing external user interventions. Therefore, the tool is available without any installation.

4.4.7 Development and integration status

The RDF validator's development and integration into the system are completed. The local operation of the tool works correctly and makes the validation properly using SHACL shapes files developed using the COGITO ontologies.

4.4.8 Requirements Coverage

The requirements coverage of the RDF Validator tool, according to D2.5, is summarized in Table 19. The tool successfully receives RDF data from translated as-planned data in TTL files (Req-1.2) and validates the RDF graph using SHACL shape files, providing assurance that the data to be used on the platform will not be erroneous (Req-2.3). The tool can be scaled up (Req-2.1), and as the project progresses, more SHACL Shapes can be added to provide the most accurate validation for the RDF data generated. The tool is also highly responsive and available (Req-2.2 and Req-2.4), for the following reasons: it can be executed as a service in a parallelizable and containerized fashion through the Spark framework. Additionally, this framework offers multithreading capabilities, allowing tool execution and validating multiple different RDF graphs simultaneously.

Table 19: RDF Validator requirements coverage from D2.5.

ID	Description	Type	Status
Req-1.2	Receives as-planned data	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.3	Security	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

4.4.9 Assumptions and Restrictions

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 input data graph file structure should follow the COGITO ontology defined in D3.3.
2. The shapes graph should be formatted according to the SHACL shapes language described in [7].

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

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

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

5 ETL Tools for dynamic data

IoT data originating from tracking devices attached to (human and non-human) construction resources can be useful within COGITO. The time-series database in the DTP can store this information. The database is updated in real-time with streaming data. It can respond to queries related to the location of the construction resources for predefined periods.

The ETL tools that work with dynamic data include:

(C1.1) **IoT Data Logger** is a persistence tool in charge of writing data to the time-series database.

(C1.2) **IoT Data Retriever** queries the time-series database, applies linear interpolation, and creates the responses to requesting applications.

5.1 IoT Data Logger

Once the streaming data have been cleaned from noise and outliers by the IoT data pre-processing module, such data are sent for storage in a database. The IoT data from location tracking devices are points in 2D space (x and y coordinates) at specific time instances. Based on the tracking device's sampling frequency, the time between successive samples is between 200 and 500 ms. Storing all received data is not required. Therefore, a procedure to select values that should be stored in the database should be put in place. This event-driven approach (based on Change of Value and other approaches) is part of the IoT Data Logger ETL.

5.1.1 Prototype Overview

According to Figure 36, the IoT Data Logger receives data from the IoT data pre-processing module in JSON format and performs the following two operations:

1. An event-based data compression.

An event that triggers the sample generation is defined by the following two conditions:

- a. The Euclidean distance between successive data points (locations of the tracked resource) exceeds a certain threshold value. In case this distance exceeds the threshold, the tracked resource is moving, and this value can be written in the DB. In any other case, the tracked resource is unmoving, so no Change of Value needs to be recorded.
- b. Even for still resources, a value might be written after a specific (long) interval.

The threshold value and the fixed time interval are adjustable parameters.

2. Storage of the compressed data in the time series database in the Data Persistence layer of DTP.

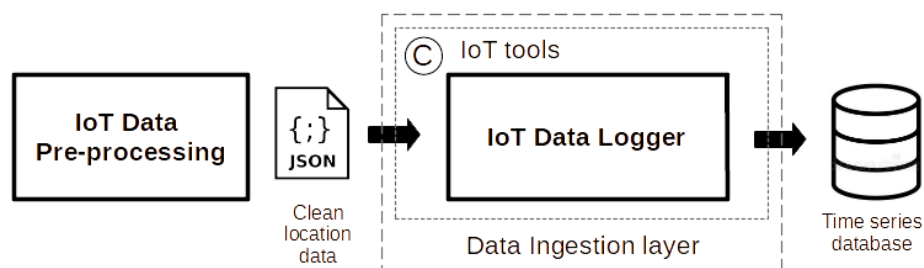


Figure 36: Block diagram of IoT Data Logger's operation.

5.1.2 Technology Stack and Implementation Tools

IoT Data Logger is based on the technologies presented in Table 20.

Table 20: Libraries and Technologies used in B-rep Generator

Library/Technology Name	Version	License
KAFKA	3.3.1	Open source
Java	18	Apache licence

5.1.3 Input, Output, and API Documentation

The input of IoT Data Logger is a JSON file containing the tracking data of construction resources as timestamped instances of points in 3D cartesian space (x,y,z coordinates) as also presented in D3.6. The input specification illustrated in part A of Figure 37. More specifically, for every “**tagid**” class which contains a string of characters as an attribute, multiple “**coords**” classes are assigned. Each “**coords**” class contains four additional classes:

- “**coordTimestamp**”: contains the time instance of the sample in time format as an attribute.
- “**coordX**”: contains the X coordinate of the sample in numerical format as an attribute.
- “**coordY**”: contains the Y coordinate of the sample in numerical format as an attribute.
- “**coordZ**”: contains the Z coordinate of the sample in numerical format as an attribute.

The output of the tools is a database entry with the following five fields: **tag_id**, **Timestamp**, **coordX**, **coordY** and **coordZ**, as highlighted in part B of Figure 37.

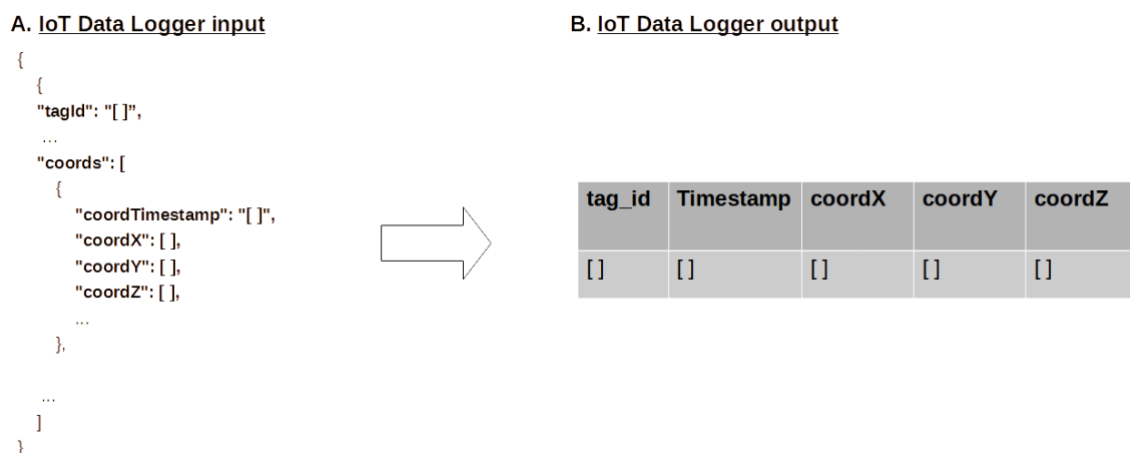


Figure 37: I/O illustration of IoT Data Logger.

5.1.4 Application examples

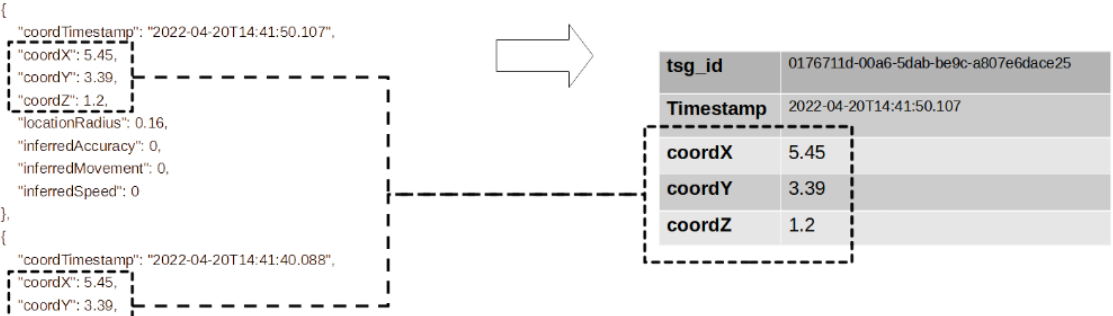
An application example of the IoT Data Logger is presented in Figure 38. In this example, the two coordinate instances retrieved from the IoT data pre-processing module are compressed into one instance in the timeseries database as the coordinates (x,y,z) of the two instances do not change. Since there is no change in the coordinates, neither of the two conditions (a,b) introduced in subsection 5.1.1 is violated, therefore no additional storing event is triggered in the database. Consequently, the tool writes only one instance in the database and not two.

A. IoT Data Logger input

```

{
  {
    "tagId": "0176711d-00a6-5dab-be9c-a807e6dace25",
    "tagActive": 1,
    "lastSeen": "2022-04-20T14:41:50.107",
    "assignedToGroup": {
      "groupId": "7ec6bf7e-14bd-42a4-8cad-9d5d60e789b2",
      "groupName": "Workers",
      "groupType": 0,
      "assignedToProject": {
        "projectId": "b305f83a-6674-4ba5-99c1-091e5938b2dc",
        "projectTitle": "Example COGITO IoT project"
      }
    }
  },
  "coords": [
    {
      "coordTimestamp": "2022-04-20T14:41:50.107",
      "coordX": 5.45,
      "coordY": 3.39,
      "coordZ": 1.2,
      "locationRadius": 0.16,
      "inferredAccuracy": 0,
      "inferredMovement": 0,
      "inferredSpeed": 0
    },
    {
      "coordTimestamp": "2022-04-20T14:41:40.088",
      "coordX": 5.45,
      "coordY": 3.39,
      "coordZ": 1.2,
      "locationRadius": 0.16,
      "inferredAccuracy": 0,
      "inferredMovement": 0,
      "inferredSpeed": 0
    },
    ...
  ]
}

```

B. IoT Data Logger output


tsg_id	0176711d-00a6-5dab-be9c-a807e6dace25
Timestamp	2022-04-20T14:41:50.107
coordX	5.45
coordY	3.39
coordZ	1.2

Figure 38: Application example of the IoT Data Logger**5.1.5 Development and integration status**

The tool is part of the DTP platform.

5.1.6 Requirements Coverage

The requirements coverage, according to D2.5, of the IoT Data Logger is summarized in Table 21. IoT Data Logger successfully handles real-time data from location tracking devices attached to construction resources (Req-1.4). Additionally, the tool is responsive (Req-2.2) and highly available (Req-2.4) as it can receive multiple input JSON files streamed using the KAFKA framework.

Table 21: IoT Data Logger requirements coverage from D2.5.

ID	Description	Type	Status
Req-1.4	Handles real-time data of location tracking sensors	Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

5.1.7 Assumptions and Restrictions

One of the main requirements for the correct operation of the IoT Data Logger is that the structure and content of the input JSON data file are according to the input specification presented in subsection 5.1.3. For example, every "coord" instance should contain x,y, and z coordinates values. Any violation of these conditions will cause incorrect results to be written in the time-series database.

5.2 IoT Data Retriever

As explained in the previous subsection, the IoT data stored in the database follow an event-based rationale. To this extend, the locations of the tracked resources during periods of inactivity are not stored in the database to reduce the total size of data. On the other hand, in COGITO several tools and respective use cases request the location of construction resources during a predefined time interval $[t_1, t_2]$.

These use cases and respective tools include:

UC1.2 where in data transaction 10 PMS requests from DTP resource location tracking data.

UC3.2 where in data transaction 3 Proactive Safety requests from DTP resource location tracking data.

UC4.1 where in data transaction 13 DCC requests from DTP resource location tracking data.

To support the above requests related to resource location tracking data, and software component must be implemented that search in the timeseries database of DTP for available location data samples related to the tracked resource and for the specified, by the requesting Application, time interval $[t_1, t_2]$. This software mechanism is an ETL tool called IoT Data Retriever described next.

5.2.1 Prototype Overview

From a broad perspective, the IoT Data Retriever, as a component of DTP's Data Management layer uses an actor-based rationale and connects the time-series database component of DTP's Data Persistence layer to any external COGITO application that wants to query location tracking data of construction resources. The response to the querying COGITO application is via the use of a JSON file. This connection between the timeseries data base and the external-to-DTP querying application is illustrated by the block diagram of Figure 39. According to this diagram, upon request from an external-to-DTP Application, the IoT Data Receiver: (i) queries the timeseries database using as input a predefined time interval $[t_a, t_b]$ (specified from the Application request), (ii) applies linear interpolation in 2D Euclidean space, if necessary and (iii) forms the final output as a JSON file to be sent back to the Application that made the request.

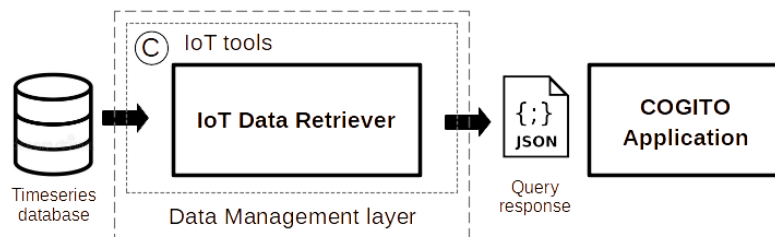


Figure 39: Block diagram of IoT Data Retriever's operation.

The ETL process of IoT Data retriever in terms of the tracked location data of a construction resource is illustrated in Figure 40. IoT Data Retriever queries the time-series database and extracts time samples which are within the time interval $[t_a, t_b]$ (t_2, t_3) together with the location sample before t_a (t_1) and after t_b (t_4). These location samples are not equidistant in the time scale since the data storage in the database is event-based. Therefore, the samples at the beginning of the interval (t_1 and t_2) are interpolated linearly and produce a new sample $t_a(x_a, y_a, z_a)$ and the samples at the end of the interval (t_3 and t_4) are also interpolated linearly and produce a new sample $t_b(x_b, y_b, z_b)$. The in-between samples (t_2 and t_3), together with the new ones (t_a and t_b), are collected in the final JSON response.

During the inter-sample time, the speed of the tracked resource is assumed to be constant; therefore, the weight for the linear interpolation for the t_a sample is $w_a = (t_a - t_1) / (t_2 - t_1)$ and for the t_b sample: $w_b = (t_b - t_3) / (t_4 - t_3)$. Using this weight, the interpolated coordinates (C_a and C_b), with C being either X, Y or Z , become: $C_a = C_1 + w_a(C_2 - C_1)$ and $C_b = C_3 + w_b(C_4 - C_3)$.

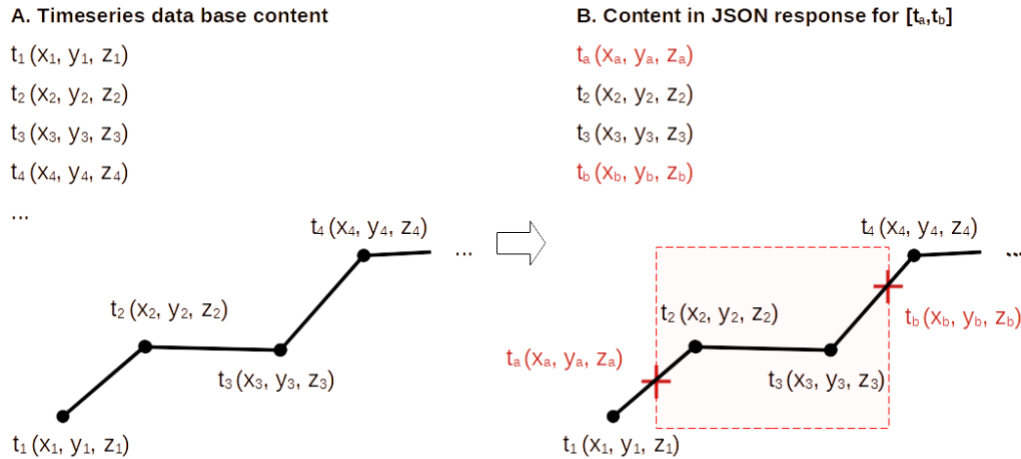


Figure 40: Illustration of ETL process on location tracking data performed by the IoT Data Retriever.

5.2.2 Technology Stack and Implementation Tools

IoT Data Retriever is developed using the technologies in Table 22.

Table 22: Libraries and Technologies used in IoT Data Retriever

Library/Technology Name	Version	License
JAVA	18	Apache licence

5.2.3 Input, Output, and API Documentation

The I/O specification of the IoT Data Retriever is the reverse I/O specification of the IoT Data Logger. More specifically, the input of IoT Data Retriever is time-series database entries pictures in part A of Figure 41. Figure 41: I/O illustration of IoT Data Retriever.

The output of the IoT Data Retriever is a JSON file containing the tracking data of construction resources as timestamped instances of points in 3D cartesian space (x,y,z coordinates) as presented in part B Figure 41. Figure 41: I/O illustration of IoT Data Retriever. The output of the IoT Data Retriever is a more compact version of the input of the IoT Data Logger presented in the previous subsection. It contains multiple “**tagid**” classes (one per tracked resource), which contain a string of characters as attributes and multiple “**coords**” classes. Each “**coords**” class contains four additional classes:

- “**coordTimestamp**” contains the time instance of the sample in time format as an attribute.
- “**coordX**” contains the X coordinate of the sample in numerical format as an attribute.
- “**coordY**” contains the Y coordinate of the sample in numerical format as an attribute.
- “**coordZ**” contains the Z coordinate of the sample in numerical format as an attribute.

A. IoT Data Retriever input

tag_id	Timestamp	coordX	coordY	coordZ
[]	[]	[]	[]	[]



B. IoT Data Retriever output

```
{
  {
    "tagId": "[ ]",
    "coords": [
      {
        "coordTimestamp": "[ ]",
        "coordX": [ ],
        "coordY": [ ],
        "coordZ": [ ],
      },
      ...
    ]
  }
}
```

Figure 41: I/O illustration of IoT Data Retriever.

5.2.4 Application examples

An example of the IoT Data Retriever is given in Figure 42. In this example, four location samples are queried from the time series database related to the tracked resource with **tag_ID**: 0176711d-00a6-5dab-be9c-a807e6dace25.

The four samples were obtained at the following times:

```
t1= 2022-04-20T14:35:50.17
t2 = 2022-04-20T14:35:51.22
t3 = 2022-04-20T14:35:53.45
t4 = 2022-04-20T14:35:54.32
```

The t_a , t_b limits of the query interval were:

```
ta = 2022-04-20T14:35:50.56, and
tb = 2022-04-20T14:35:54.11.
```

Based on these interval limits the final JSON response of the tool contains four cords instances:

- An instance at t_a produced by linear interpolation of the t_1 and t_2 samples.
- The t_2 instance.
- The t_3 instance.
- An instance at t_b produced by linear interpolation of t_3 and t_4 samples.

A. IoT Data Retriever input

tag_id	Timestamp	coord X	coordY	coord Z
0176711d-00a6-5dab-be9c-a807e6dace25	2022-04-20T14:35:50.17	1.1	1.2	1.7
0176711d-00a6-5dab-be9c-a807e6dace25	2022-04-20T14:35:51.22	1.5	1.8	1.7
0176711d-00a6-5dab-be9c-a807e6dace25	2022-04-20T14:35:53.45	2	1.9	1.7
0176711d-00a6-5dab-be9c-a807e6dace25	2022-04-20T14:35:54.32	4.5	4.8	1.7

B. IoT Data Retriever output

```
{
  {
    "tagId": "0176711d-00a6-5dab-be9c-a807e6dace25",
    "coords": [
      {
        "coordTimestamp": "2022-04-20T14:35:50.56",
        "coordX": 1.24,
        "coordY": 1.42,
        "coordZ": 1.7,
      },
      {
        "coordTimestamp": "2022-04-20T14:35:51.22",
        "coordX": 1.5,
        "coordY": 1.8,
        "coordZ": 1.7,
      },
      {
        "coordTimestamp": "2022-04-20T14:35:53.45",
        "coordX": 2,
        "coordY": 1.9,
        "coordZ": 1.7,
      },
      {
        "coordTimestamp": "2022-04-20T14:35:54.11",
        "coordX": 3.89,
        "coordY": 4.1,
        "coordZ": 1.7,
      },
    ],
  }
}
```

Figure 42: Application example of IoT Data Retriever.

5.2.5 Development and integration status

The tool is part of the DTP platform.

5.2.6 Requirements Coverage

The requirements coverage, according to D2.5, of IoT Data Retriever is summarized in Table 23. IoT Data Retriever successfully handles real-time data from the time-series database of DTP, which are location data

from the tracking devices attached to construction resources (Req-1.4). Additionally, the tool is responsive (Req-2.2) and highly available (Req-2.4), as it can respond to multiple requests.

Table 23: IoT Data Logger requirements coverage from D2.5.

ID	Description	Type	Status
Req-1.4	Handles real-time data of location tracking sensors	Functional	Achieved
Req-2.2	Responsiveness	Non-Functional	Achieved
Req-2.4	High availability	Non-Functional	Achieved

5.2.7 Assumptions and Restrictions

The flawless operation of IoT Data Retriever can be guaranteed if the times series database is populated for the requested time interval by the querying COGITO application. The JSON response will contain no entries if the database has no location samples for the requested time interval.

6 Conclusions

In this demonstrator deliverable, we presented the final version of eleven components or tools of the Digital Twin Platform, demonstrating their operation. Eight of these tools apply Extraction, Transformation, and Loading (ETL) operations, and two perform model checking (MC) operations. Depending on their purpose, these tools work with static or dynamic data from location-tracking devices. These tools are elements towards the population of the COGITO data models, according to the COGITO ontology presented in D3.2. Using the shapes constraint language, we can *a posteriori* verify that the required information is there. These elements are ingredients in the actor-based architecture presented in D7.10. The DTP stores Dynamic data in a time series database, and the data are made available to COGITO applications.

Following the architecture presented in D7.2, we grouped tools into three *ad hoc* categories: (1) Category A, which includes ETL and MC tools that work with BIM data as input. These require solid management of the BIM models and rely on the IFC SDK and the geometric and enrichment operations. (2) Category B includes tools that work with non-BIM data and generate parts of the RDF graph. This category includes the shapes-based RDF validation component. (3) Category C cover ETL tasks on dynamic IoT data originating from location-tracking devices attached to the construction resources. At this point, these tools provide all the functionalities required to start with the demonstration activities or COGITO. Should additional needs arise, we may extend existing components or develop new ETL tools.

BIM-based components in Category A may require significant computational resources and cannot run synchronously. The DTP reflects this consideration by using containerisation technologies to allow easy scaling and parallelisation. We designed the DTP following the micro-services design pattern and adopting an event-driven architecture. The message broker allows asynchronous communication to invoke dependent tasks sequentially. Consequently, simultaneous, parallel and asynchronous executions of these tools are possible, significantly reducing the overall query-response time of COGITO tools, requesting ETL and MC tool executions as separate services. The message broker accomplishes the coordinated invocation of these services to facilitate internal data transformation operations between the data providers (external data sources or stored data) and communication with external data consumers (COGITO tools).

This deliverable focused on presenting and demonstrating a second (final) version of the ETL and MC tools. This second version includes more detailed information related to the technology stack used for the implementation and updates based on additional testing of the tools on construction site data arriving from the demonstration sites. As the project evolves and further data from the demonstration sites becomes available, we will test the developed tools further. When quality- and H&S-related data structures and data originating from construction workflow applications become available, additional RDF ETL tools will be required to input these new data structures and produce semantic graph outputs in TTL RDF format. We will incorporate these tools into DTP's Knowledge Graph Generator component and further report on this development in the context of the demonstration activities.

References

- [1] W3C, “Shapes Constraint Language (SHACL),” [Online]. Available: <https://www.w3.org/TR/shacl/>
- [2] ISO 16739, “Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries (ISO 16739:2013),” CEN, 2016.
- [3] buildingSMART, “Model View Definition,” 2022. [Online]. Available: https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-1/mvdXML_V1-1-Final.pdf
- [4] A. Johnson, “Clipper - an open source freeware library for clipping and offsetting lines and polygons,” 2014. [Online]. Available: <http://www.angusj.com/delphi/clipper.php>
- [5] B. R. Vatti, “A generic solution to polygon clipping,” *Communications of the ACM*, vol. 35, no. 7, pp. 55-36, 1992.
- [6] G. Mei, J. C. Tipper and N. Xu, “Ear-Clipping Based Algorithms of Generating High-Quality Polygon Triangulation,” in *International Conference on Information Technology and Software Engineering* , 2015.
- [7] buildingSMART, “Ifc Parametrized Profiles,” IFC, [Online]. Available: https://standards.buildingsmart.org/IFC/DEV/IFC4_3/RC2/HTML/schema/ifcprofileresource/lexical/ifcparameterizedprofiledef.htm



COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu



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