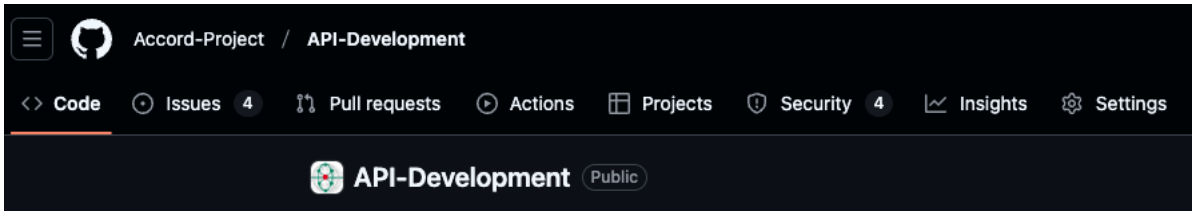
D4.2 ACCORD Building Compliance Checking Components, Microservices and APIs

March 24, 2025



This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement no. 101056973.

Funded by the European Union

UK Participants in Horizon Europe Project ACCORD are supported by UKRI grant numbers [10040207] (Cardiff University), [10038999] (Birmingham City University) and [10049977] (Building Smart International).

Innovate UK

Funded by the European Union. The views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HaDEA). Neither the European Union nor the granting authority can be held responsible for them.

Project Title:	ACCORD - Automated Compliance Checks for Construction, Renovation or Demolition Works
Deliverable:	D4.2 ACCORD Building Compliance Checking Components, Microservices and APIs
Type:	REPORT
Dissemination level:	PUBLIC
Work package:	WP4
Deliverable leader:	FUI
Contributing partners:	CU, FUNITEC, SOL, ONTO, CP, FHG, VTT
Due date:	31 March 2025
Date:	24 March 2025
Status - version, date:	Version 0.6, 24.3.2025
Authors:	Thomas Beach (CU)
	Gonçal Costa (FUNITEC)
	Pasi Paasiala (SOL)
	Rick Makkinga (FUI)
	Nataliya Keberle (ONTO)
	Arkadiusz Chadzynski (ONTO)
	Ilkka Mattila (CP)
	Katja Breitenfelder (FHG)
	Rafael Horn (FHG)
	Petr Hradil, Rita Lavikka (VTT)
Reviewer 1:	Mahmood Al-Doori (UNIKO)
Reviewer 2:	Kiviniemi Markku (VTT)
Final review	Rita Lavikka (VTT)

DOCUMENT HISTORY

Ver-sion	Date	%	Comments	Main Authors (affiliation)
0.1	10.1.2025	5	First Version of ToC	Thomas Beach (CU)
0.2	30.1.2025	50	Contribution by CU/FUI	Thomas Beach (CU) / Rick Makkinga (CU)
0.3	28.2.2025	85	Contributions from other partners	Petr Hradil, (VTT), Rafael Horn (FhG), Katja Breitenfelder (FhG), Ilkka Mattila (CP), Arkadiusz Chadzynski (ONTO), Pasi Paasiala (SOL), Gonçal Costa (FUNITEC)
0.4	15.03.2025	90	Reviewing and Comments	Mahmood Al-Doori (UNIKO), Kiviniemi Markku (VTT)
0.5	19.03.2025	90	Dealing with comments from the reviews	Tom Beach (CU)
0.6	24.3.2025	100	Added missing words to the terminology, added suggestions to the executive summary, and final review	Rita Lavikka (VTT)

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation, or both.

Executive Summary

This deliverable documents the results of the ACCORD project's Task *4.3 Implementation of ACCORD Building Compliance Checking Components, Microservices and APIs and Task 4.4 Implementation of Consortium Microservice APIs and integrations in ACCORD Service-Setups*. These tasks belong to Work Package 4 Solutions development.

Task 4.3 has implemented the ACCORD core building compliance checking components, and their Application Programming Interfaces as specified within the Cloud Architecture Model designed in T4.2. This development has reused open-source and existing solutions of consortium partners and has built upon the results of recent research and innovation projects at the EU level. Finally, the deliverable describes the extensibility of the ACCORD framework and how extra novel components can continually be integrated beyond the scope of the project.

Task 4.4 has developed and integrated microservice applications related to the demonstration deployments. It has provided integration between the consortium partners' software and ACCORD cloud components. Furthermore, it has managed the mapping of these components to the demonstrations through the service setups.

This deliverable starts by re-capping the ACCORD cloud architecture model, covering any needed changes that have been made in the process of software development. The document then covers the development of the three main elements of the ACCORD cloud architecture; (1) **Core Components**, (2) **Microservices and information services** and (3) **The APIs (Application Programming Interface) that integrate these elements.**

In total, the ACCORD project has developed 9 core components, produced 3 API specifications, and integrated 8 demo-specific microservices. These are:

- Core Components:
  - Rule Formalization Tool
  - Data Dictionary Repository Component
  - Rule Repository Component
  - Information Requirements Repository
  - IDS Generation Component
  - Model- and Data Requirement Validation Component
  - Process Execution Component
  - Data Storage Component
  - Orchestrating Microservices Component
- API Specifications:
  - Building Codes and Rules API
  - Data APIs
  - Results API
- Microservices:
  - Urban Regulations

- Solibri
- Finnish LCA Calculator
- Clearly.BIM
- Eurocode Calculations
- Land Use Checking
- German LCA
- Timber Construction Compliance Checker

Each of the developments will be described, as well as their relationships to the ACCORD demonstration activities via the ACCORD Service Setups.

The ACCORD framework is designed to be extensible, allowing for the customization of permitting processes and the addition of new microservices as needed. Individual nationals and municipalities are able to select the components and their integration based on the needs of their permitting processes. The framework is applied in practice to country-specific real-life building projects, advancing the automation of building compliance checks and improving efficiency.

Thus, this deliverable will present a set re-used components, that can be integrated together to deliver the requirements of, not only the ACCORD demonstrations, but also permitting authorities across Europe. These developed components consist of core components that can be reused across multiple permitting authorities and microservices that deliver specific compliance-checking functionality. This key ethos, of smaller re-usedable software components is what will enable the ACCORD approach to be reused across multiple jurisdictions and deliver its wider applicability.

Contents

List of Figures

List of Tables

Terminology

Term	Definition
AEC3PO	An ontology used in the rule formalization process
API	Application Programming Interface
BCRL	Building Compliance Rule Language developed in ACCORD
BIM	Building Information Modeling
BNB	Assessment system for sustainable buildings
BREEAM	Building Research Establishment Environmental Assessment Methodology
bsDD	buildingSMART Data Dictionary
DGNB	German Sustainable Building Council
FME	Feature Manipulation Engine (Orchestration SoftwarE)
GraphQL	Graph Query Language
IDS	Information Delivery Specification
IFC	Industry Foundation Classes
IR	Information Requirements
JSON-LD	JavaScript Object Notation – Linked Data
LCA	Life Cycle Assessment
LOIN	Level of Information Needs
NLP	Natural Language Processing
ÖKOBAUDAT	A platform for the ecological evaluation of buildings
QNG	German Sustainability Building Certification Scheme
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Indicator
YAML	Yet Another Markup Langauge

1 Introduction

This deliverable presents the results of the ACCORD project's Task 4.3 Implementation of ACCORD Building Compliance Checking Components, Microservices and APIs and Task 4.4 Implementation of Consortium Microservice APIs and integrations in ACCORD Service-Setups. These tasks belong to Work Package 4 Solutions development. This introduction section first outlines the ACCORD project, states the aims of this document and then summarizes its structure.

1.1 The ACCORD Project

The ACCORD project's objective is to provide a framework for digitalising building permitting and compliance processes using BIM and other data sources, with the end goal of improving the productivity and quality of design and construction processes, supporting the design of climate-neutral buildings and advancing a sustainable built environment in line with the EU Green Deal and New European Bauhaus initiative.

ACCORD is based on the principles that these digitised processes must be human-centred, transparent, and cost-effective for the permit applicants and authorities and, above all, relevant to the industry within which they are to be employed.

To achieve this, ACCORD is developing a Semantic Framework for European digital building permitting processes, regulations, data, and tools. This framework will drive rule formalisation and integration of existing compliance tools as microservices. Solutions and tools are to be developed, providing consistency, interoperability and reliability with national regulatory frameworks, processes, and standards. It will enable the integration of technical solutions for automating compliance checking of buildings in their design, construction, and renovation/demolition lifecycle phases.

To ensure the industry relevance of the project work, the first work package of the ACCORD project analysed the complex landscape of built environment compliance checking and building permitting across Europe to ascertain the requirements for the future digitalisation of this complex interdisciplinary field. The project partners conducted a landscape review and analysis of the current adoption of the concept of digitalisation of building permit- and compliance checking, including a survey into the attitudes of stakeholders to the prospective digitalisation of this domain in a range of European countries. This work was reported in the D1.1 Landscape Review Report that focused on a) academic projects and methods, b) relevant software tools and technologies, and c) national adoption efforts in the field.

This solid basis paved the way for a framework that has the potential to achieve real change and drive forward the digitalisation of this area. Evidence of this will be collected through the implementation and demonstration of construction projects in various EU regulatory contexts: UK, Finland, Estonia, Germany, and Spain. This work will be reported in Deliverable 5.1.

This work has led to the specification of the ACCORD Semantic Framework that will integrate and provide access to building compliance and permitting services, allowing for the storing, processing, analysis and retrieval of administrative and regulatory information related to construction, renovation and demolition works (documented in Deliverable 1.2). Building on this, a technical specification was developed. This was created by defining the different requirements and usage scenarios and aligning the requirements with the

components defined in the ACCORD Framework. This has been formalized into the ACCORD cloud architecture model (documented in Deliverable 4.1).

1.2 Aims and Objectives

This deliverable has three aims:

- Document the results of development and/or the use of open-source codes for the implementation of the core ACCORD Building Compliance Checking Components.
- Document the re-used and new APIs developed within the ACCORD project.
- Report the results of microservice implementation and integrations outlined in the ACCORD Service-Setups.

1.3 Deliverable Structure

This deliverable is structured into 7 sections. Section 2 outlines the ACCORD cloud architecture and presents the modifications that have been made during development since the release of deliverable 4.1. Section 3 presents the ACCORD cloud components, documenting their developed functionality and the implementation decisions made during their development. Section 4 documents the ACCORD APIs, describing the existing APIs that have been re-used as well as those that have been developed specifically within the ACCORD project.

Section 5 then documents the ACCORD integration strategies covering the differing methods of integration of microservices with the ACCORD core components that have been utilised within the project. Secondly, It documents the integrations required by the demonstration activities by describing the ACCORD service set-ups used in each demonstration. Finally, this section describes the extensibility of the ACCORD framework and how new novel components can be continually integrated.

Section 6 then outlines each of the demo-specific microservices that have been developed within the project, it describes their functionality, how they have been developed and the problem’s they solve within their respective demonstrations. Finally, Section 7 concludes the deliverable.

2 ACCORD Cloud Architecture

This section will recap the ACCORD cloud architecture originally specified in Deliverable 4.1. Subsequently, the evolution of this architecture as the development process has proceeded will be documented, together with the presentation of the architecture in its finalized state.

The ACCORD cloud architecture itself is designed to be flexible, enabling individual nationals and municipalities to select the components and their integration based on the needs of their permitting processes. Thus, each ACCORD demonstrator will use a subset of the components defined below. This is described more in Section 5.

2.1 The Original ACCORD Cloud Architecture

The ACCORD Cloud Architecture was originally defined in Deliverable 4.1. It stems from two main sources: (1) the ACCORD Semantic Framework (defined in D1.2) and (2) the elicited ACCORD technical requirements (defined in D4.1).

The final structure of the ACCORD Cloud Architecture, as defined in D4.1, contains twelve main components:

1. **Rule Formalization:** The Rule Formalization Tool is responsible for all the various activities on the building authority side. Its subcomponents oversee various tasks that are related to formalising building codes and regulations, utilises a domain-specific rule language and managing building codes and rules.

2. **Data Dictionaries:** This component provides access to data dictionaries with the aim of mapping explicit definitions and terms being present in regulatory documents. It also provides a reconciliation capability to conduct fuzzy mapping and other lookups when terminologies do not precisely align.

3. **The Rule Repository and Provision:** This component provides the ability to store and translate or interpret the codes and rules provided by the Rule Formalization Tool.

4. **Information Requirements Repository:** This component will be responsible for providing formalized definitions of information requirements for building permit processes to other components of the ACCORD Cloud Architecture.

5. **The Cloud-based Building Permit Services:** This component manages the overall process of building permitting and acts as an intermediary between the rule provision component and the compliance checking microservices.

6. **Model and Data Requirement Validation:** This component validates the building's model (IFC format) against bSDD and IDS specifications and validates additional external data sources (e.g. a zoning/land use plan) against input requirements.

7. **Process Execution:** This component executes the process flow for a building permit, from the first initial application to the final granting of the permit. It needs to interact

with the microservice orchestration component to execute checks as part of the process flow.

8. **Data Storage:** This component is a ‘single source of truth’ location to store the building models in IFC and cityGML format and land use plans in XPlanung format. It is accessed by other components via API for retrieving (parts of) the models with the aim of executing compliance checks.

9. **Orchestrating Microservices:** This component orchestrates (coordinates) the API communication between components of the ACCORD Cloud Architecture.

10. **Compliance Checking Microservices:** This element of the architecture encompasses the various compliance checking services of ACCORD. It runs compliance checks based on the input data (building models in IFC and others) and rules and returns the results back to the process execution component.

11. **Information Services:** The information services provide additional information and reasoning capabilities to compliance checking microservices such as geospatial or environmental data.

12. **APIs:** The APIs act as an intermediary between all the various components of the ACCORD Cloud Architecture. 7 APIs are distinguished as follows:
    - API (1) Definitions API
    - API (2) Building Codes and Rules API
    - API (3) Information Services APIs
    - API (4) Data APIs
    - API (5) Management APIs
    - API (6) Results
    - API (7) Reconcilian API

The initial ACCORD Cloud Architecture with its various components is illustrated in Figure 1.

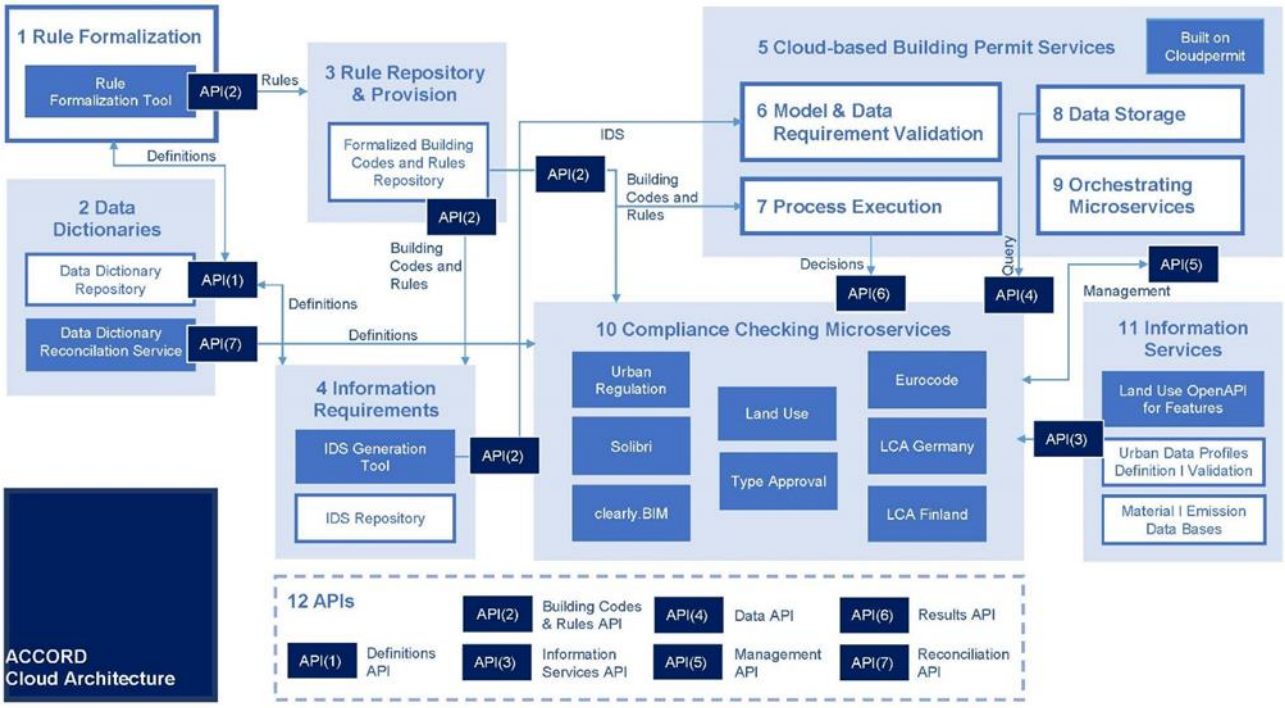


Figure 1. Initial ACCORD Cloud Architecture

2.2 Revised ACCORD Cloud Architecture

During the process of development of the components, as well as evolution in the definition of the demonstration cases, several modifications have been made to the ACCORD cloud architecture. These modifications are summarised in Table 1.

Table 1. ACCORD Cloud Architecture Change Log

No	Change	Justification
1.	Clarify that the cloud-based permitting service is not a component.	In D4.1, the cloud-based permitting service was described, in error, as a component. Instead, it is a collection of components.
2	Remove the Data Dictionary Reconciliation Service and API7	This was originally defined as it may be needed to deliver the Finnish/German LCA checking demonstrations, but these demonstrations no longer need this service.
3	Remove the management API (API5)	Upon detailed design, this was found to duplicate the functionality of API6, so it has been removed.
4	Remove Land Use OpenAPI for Features Information Service	This was originally specified as being required for the Land Use Microservice but is no longer required.
5	Add Timber construction database	This new information service was required by the German Demonstration.

6	Remove the Urban Data Profile Definition Information Service	This was originally required by the Urban Regulation microservice, but the data was unable to be provided by an external party via an API, so this was not implemented.
7	Rename Type Approval to Timber Construction Compliance	In D4.1, the name of the Type Approval service was still generic. The demonstration has not determined the type of construction that it will be checking, so it has been renamed the Timber Construction Compliance Service.

Following these modifications, a new architecture figure has been drafted; this is shown in Figure 2. Additionally, a simpler version has also been created to aid communication with external parties. This is shown in Figure 3.

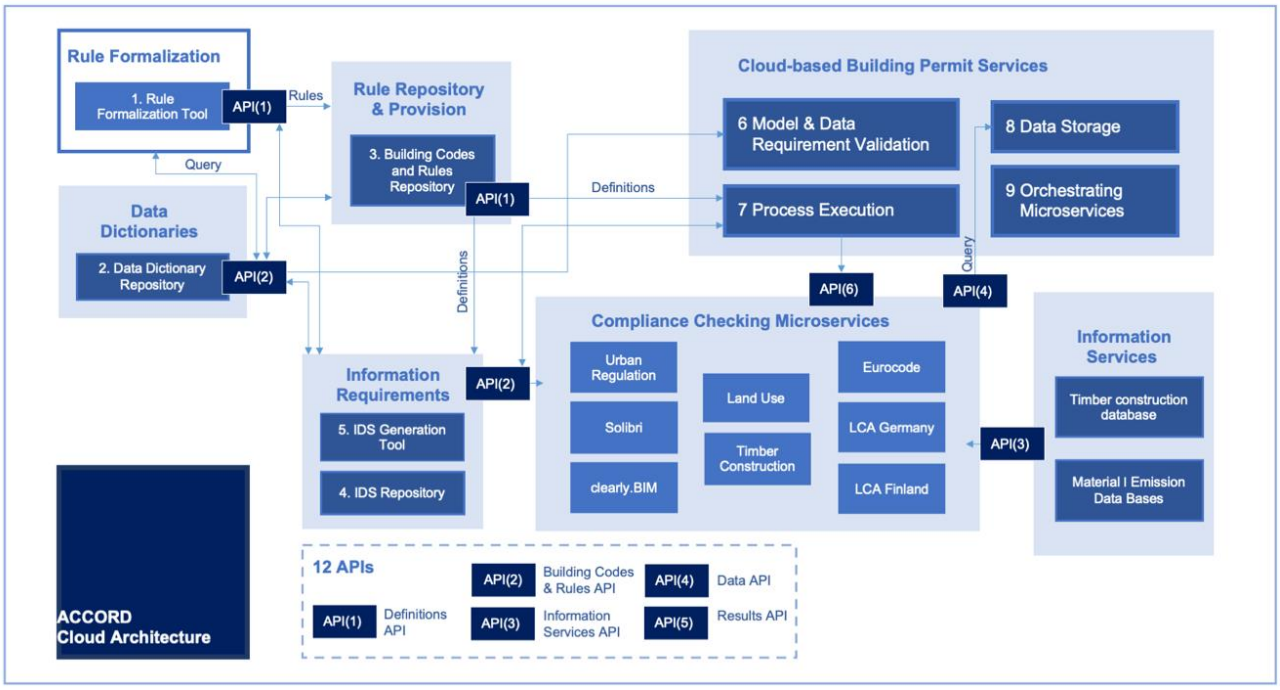


Figure 2. Revised ACCORD Cloud Architecture.

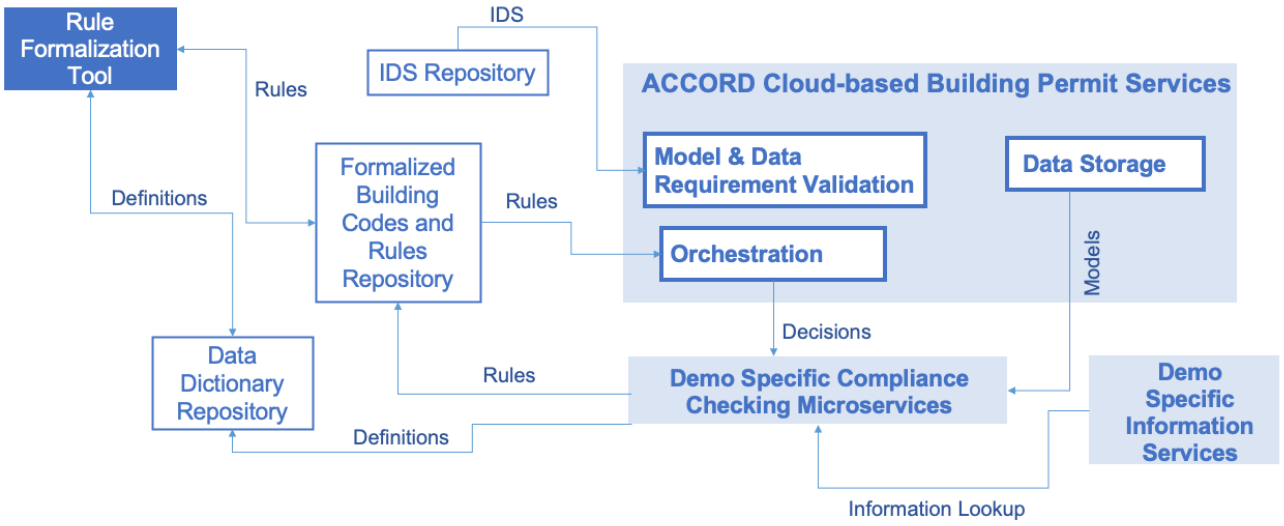


Figure 3. Revised ACCORD Cloud Architecture - Simplified Version.

The list of assigned partners for each component in the revised architecture is shown in Table 2.

Table 2. ACCORD Cloud Architecture Partner Allocation.

Component Number/Name	**Responsible Partners**
1 Rule Formalization	FUNITEC
2 Data Dictionaries Repository	BSI
3 Rule Repository	ONTO, CU
4 IDS Repository	CU
5. IDS Generation Tool	CU
6 Model- and Data Requirement Validation	BSI, FUI, SOL
7 Process Execution	CP
8 Data Storage	FUI, ONTO
9 Orchestrating Microservices	CU
10 Compliance Checking Microservices	
(1) Solibri	SOL
(2) Future Insight Clearly.BIM	FUI
(3) LCA Finland	VTT
(4) LCA Germany	FhG
(5) Eurocode Compliance Checking	AE

(6) Urban Regulations Checking	FUNITEC
(7) Land Use Building Compliance Checking	ONTO/FhG
(8) Timber Construction Compliance Checking	FhG
11 Information Services	
(1) Timber Construction Database	FUNITEC
(2) Material Emissions Database	VTT
12 APIs	
API (1) Definitions API	BSI
API (2) Building Codes and Rules API	CU
API (3) Information Services APIs	FUNITEC, VTT
API (4) Data APIs	FUI, ONTO
API (5) Results API	CU

Finally, the open-source status and links to open-source repositories are provided in Table 3.

Table 3. Component Open-Source Status.

Component Number/Name	Open Source Status
1. Rule Formalization	Available on GitHub Here[1].
2. Data Dictionaries Repository	Components provided by BSI here[2].
3. Rule Repository	Available on GitHub here.[3]
4. IDS Repository	
5. IDS Generation Tool	Available on GitHub here.[4]
6. Model- and Data Requirement Validation	Component Provided by BSI here.[5]
7. Process Execution	Not open source, developments are built on top of an existing commercial tool.
8. Data Storage	Not open source, developments are built on top of an existing commercial tool.
9. Orchestrating Microservices	Available on GitHub here.[6]

[1] https://github.com/Accord-Project/RuleFormalizationTool
[2] https://www.buildingsmart.org/users/services/buildingsmart-data-dictionary/
[3] https://github.com/Accord-Project/API-Development/tree/main/BuildingCodesAndRules/Implementation
[4] https://github.com/Accord-Project/RegulationProcessing
[5] https://technical.buildingsmart.org/services/validation-service/
[6] https://github.com/Accord-Project/RegulationProcessing

10. Compliance Checking Microservices	
(1) Solibri	Not open source, developments are built on top of an existing commercial tool.
(2) Future Insight Clearly.BIM	Not open source, developments are built on top of an existing commercial tool.
(3) LCA Finland	Available on VTT Gitlab – here.[7]
(4) LCA Germany	Not open source, developments are built on top of existing commercial tool
(5) Eurocode Compliance Checking	Available on GitHub Here[8]
(6) Urban Regulations Checking	Available on Github Here.[9]
(7) Land Use Building Compliance Checking	Available on Github Here.[10]
(8) Timber Construction Compliance Checking	Available on Github Here.[11]
11. Information Services	As information services are provided by external third parties, no source code can be provided.
12. APIs	Due to their nature, APIs are documented via Open API, but there is no associated code.

[7] https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fextgit.vtt.fi%2Fpetr.hradil%2Fac-co2-rd&data=05%7C02%7Cbeachth%40cardiff.ac.uk%7C1c4a93afe10a4f4f40ce08dd5fefc8a0%7Cbdb74b3095684856bdbf06759778fcbc%7C1%7C0%7C638772205655307150%7CUnknown%7CTWFpbGZsb3d8eyJFbXB0eU1hcGkiOnRydWUsIlYiOiIwLjAuMDAwMCIsIlAiOiJXaW4zMiIsIkFOIjoiTWFpbCIsIldUIjoyfQ%3D%3D%7C0%7C%7C%7C&sdata=0inIMwPMFUb3QT7KLPTZnzFBOsneD3NxcJCDa2%2FV2r8%3D&reserved=0

[8] https://github.com/Accord-Project/UK-Demo-Microservice

[9] https://github.com/Accord-Project/Urban-Regulation-Microservice

[10] https://github.com/Accord-Project/Tegel

[11] https://github.com/Accord-Project/Timber-Construction-Compliance-Checking-Microservice

3 ACCORD Core Components

This Section presents the ACCORD core cloud components in detail. It documents their developed functionality and the implementation decisions made during their development. Finally, it compares the developed components to the technical requirements specific to those components.

3.1 Rule Formalization

The purpose of the rule formalisation tool is to enable public administrations, such as city councils and other governmental bodies responsible for generating building codes and regulations in European countries, to be able to formalise the regulations provided in plain text documents into a machine-processable format according to a semantic data schema, an ontology, with the purpose of it becoming a standard. It provides a software implementation of the ACCORD rule formalisation processes, integrating additional NLP tools.

This rule formalisation process has already been defined in Task 2.3, “Machine-executable Regulations”, and reported in Deliverable D2.2 “, BC Ontology and Rule Format”, under the name Regulation Digitalisation Methodology. The tool integrates components already defined in Task 2.3 and developed in Task 2.2, “Development of the Building Compliance Ontology”. These components are the AEC3PO ontology and the BCRL language. The tool also integrates an automated annotation process using NLP techniques, which is described in Deliverable D2.3, “Rule Toolset”.

Given that the technical implementation of this component is described in another deliverable, it will not be repeated here. However, a comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 4.

Table 4. Rule Formalization - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
20	The system must provide a rule configurator allowing the user to evaluate rules and record provenance of the contribution (who/when said so).	Basic configuration is also provided as part of the rule formalization. Given the prototype nature of this tool, full commercial features are not yet developed.
21	The system must provide a rule tool allowing rule authors to bind rules to data (IFC, bSDD, various data representations).	This is done via the creation of bindings and transmitting them to the data dictionary.
22	The system should provide a rule configurator allowing the selection of rule calculators depending on the data representation.	
23	The system should provide a rule tool assisting rule authors with auto-completions, in-place documentation, and lookups of IFC and bSDD definitions.	This is done via a connection to the bSDD. However, due to the fact that no dictionary was defined at the time of implementation, it has not yet been implemented.
25	The system must provide NLP with a function to break regulations into subclauses, down to the atomic level, and explicate clause structure (using RASE and/or logical connectives).	This has been implemented through the integration of the NLP tools, which were developed and documented in D2.3 with the rule formalization tool.
26	The system must provide a rule configurator that allows users to select from a list of available rule	Basic configuration is provided as part of the rule formalization tool. Given the prototype

	calculators to execute the same rule, with at least one rule calculator provided for each calculation.	nature of this tool, full commercial features are not yet developed.
27	The system must be able to detect conflicting rules and provide a strategy to select the rule to be applied in such cases (filter the conflicting rules based on the country or some specific conditions or follow a pessimistic approach that prioritizes the worst case).	Due to the methodology employed by the rule formalization process, this issue, originally foreseen, did not actually manifest. Thus, no implementation was necessary.
28	The system should be able to leverage emerging Artificial Intelligence techniques, such as semantic deep learning or Natural Language Processing (NLP).	See TR 25.
29	The system may optionally have NLP to find entities, properties and measures (e.g. 2000 mm).	
30	The system must integrate the results of NLP processing of building regulations and codes together with other inputs to be used in the rule formalization tool.	
32	The system must provide an intuitive method allowing regulation experts to digitise building codes/regulations and embed rules within them, without the need to write code.	This is done by the implementation of the rule formalization process defined in D2.2.
132	The system must allow the matching of services to logical statements within the digitised regulation.	This is done via the creation of bindings and transmitting them to the data dictionary.
94	The system's rule formalization tool must be able to communicate with the database where the rules will be stored.	This is implemented through the RFT integration with the building codes and rules database.
93	The system must allow for the lookup and visualisation of the text of the original regulation clause with any given result.	These elements are integrated into the RFT interface.
90	The system must be able to elaborate regulation clauses into checks.	
89	The rule formalization tool should link information on the regulation's document version to the rules being created. The system should enable updating the rule database in response to changes in regulatory documents. It should provide building permit authorities with data storage for archiving checking results that are linked to outdated rules and regulation document versions.	This is implemented through the RFT integration with the building codes and rules database.
88	The system shall provide a plugin for an integrated development environment (IDE) allowing to edit and run the rules.	Due to the complex nature of the rule formalization tool, it was developed as a standalone tool as opposed to a plugin for an existing IDE.
86	The system must add and display RASE tags to the rules.	This is a key element of the implementation of the rule formalisation tool.
82	The system must allow the visualisation of regulation documents in a human-readable form.	

3.2 Data Dictionary Repository

3.2.1 General Component Description

The data dictionary repository functionality is provided by bSDD (buildingSMART Data Dictionary). The bSDD - buildingSMART Solution for Data Dictionaries is an online service hosting classes (terms) and properties, allowed values, units, translations, relations between

those and more. It supports a standardised workflow to guarantee data quality, information consistency and interoperability. BIM modellers use the bSDD for easy and efficient access to all kinds of standards to enrich their models. BIM Managers use the bSDD to reference Information Delivery Specifications (IDS) and check BIM data for validity. Content creators benefit from having one entry point to various BIM tools and platforms.

ACCORD will utilise bSDD to store regulation-specific dictionary mappings between regulatory terms and (a) model storage, (b) calculation methods (i.e. which compliance checking microservice should perform the calculation) and (c) classification codes.

Creating data dictionaries for ACCORD regulatory terms in bSDD with immediate access to the data from the software solutions integrated with the bSDD. Allowing bSDD to form a central source of true for the dictionary mappings used as part of the ACCORD cloud architecture.

The motivation for selecting an existing solution such as bSDD is: (a) it is already widely used within the construction sector, (b) it meets the requirements of the ACCORD project, and (c) it will allow ACCORD development resources to focus on more innovative areas.

3.2.2 Implementation and Functionality

Since the bSDD service already exists, only a minimal implementation was required. All that needed to be done was a mapping between the data items required by ACCORD and the bSDD schema. This schema is shown in Figure 4.

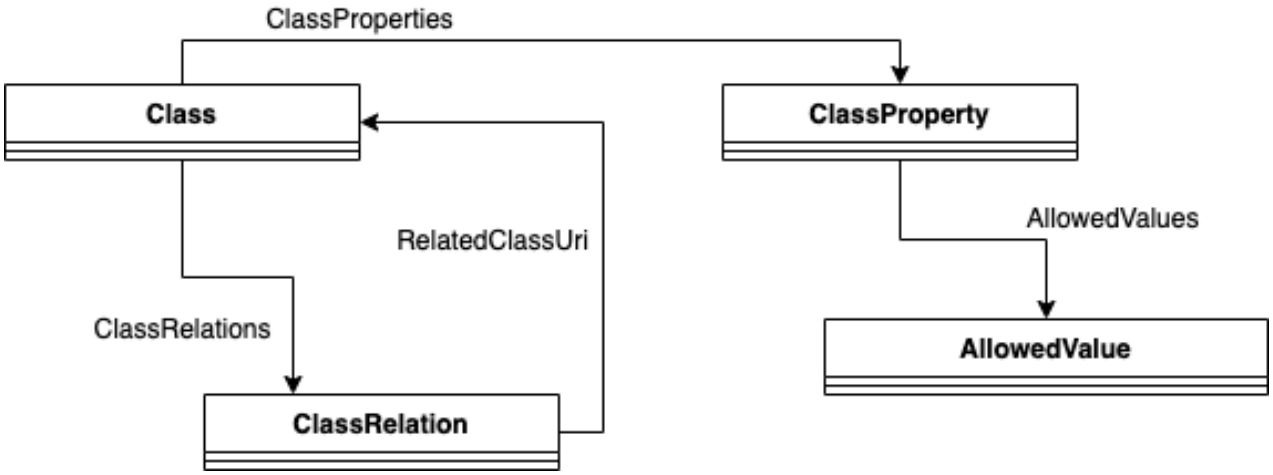


Figure 4. bSDD Schema (Deliverable 2.2).

At the heart of bSDD is a database with all dictionaries. Each dictionary may contain a list of classes and properties. Classes mainly define physical objects (e.g. a door) and properties are the attributes that describe classes (e.g. thickness). The content of dictionaries can be related to each other, creating a connected graph.

To map a given term to bSDD, the following process is followed. Each term generated by the ACCORD rule formalization process is categorised as either an object or a property. If a property, it should be added to bSDD as a ClassProperty; if it is an object, it should be a *Class*.

If the term is an object (e.g., Door): A *Class* should be created in bSDD. If the term has an equivalent in any of the existing bSDD dictionaries (e.g., IFC classes or Uniclass), then

relationships between the new Class and these related Class are created using a ClassRelation object.

If the term is a property (e.g., width), the Class of the property currently being considered should be inferred from the RASE metadata present within the document (e.g., if the term is width, then the type of object to which the width is a property should be inferred). This can be done automatically. Then, depending on the selected execution type, one of the following should be performed:

1. **Simple data lookup**: A relationship should be established between the inferred *Class* and the appropriate *ClassProperty* in the target bSDD dictionary (depending on what model format is the target). This serves to relate the term to a data item in the IFC model schema.
2. **Calculation Method**: A new *ClassProperty* should be created. The bSDD IsDynamic property is set to true and a URI for the process application is also saved.
3. **Human Decision:** A new *ClassProperty* should be created, but no other action taken.

3.2.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 5.

Table 5. Data Dictionary - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
21	The system must provide a rule tool allowing rule authors to bind rules to data (IFC, bsDD, various data representations).	This component supported this TR by providing a means to host bindings and vocabularies in the data dictionary. Allow mapping between terms, how they can be calculated, and how they should be stored in model files.
22	The system should provide a rule configurator allowing the selection of rule calculators depending on the data representation.	
23	The system should provide a rule tool assisting rule authors with auto-completion and in-place documentation, as well as lookups into IFC and bSdd-definitions.	
31	The system must enable mapping between abstract concepts in regulatory documents and concrete concepts in BIM data file format. E.g. data may be stored in a different place in different versions of the IFC model.	
59	The system must allow for human input where no automated result is available and provide the ability for human override of automated results (assuming a correctly qualified user).	This component supports achieving this by allowing defined terms within the data dictionary to be flagged as human assessed.
60	The system should be able to describe how checks are implemented: "simple": implemented in a declarative language like BIM SPARQL; "complex": implemented by invoking specialized calculations (but the purpose and input/output of these routines are semantically described).	See TR 21.
80	The system should support the use of classification systems.	This component supports achieving this TR by enabling the mapping between regulatory terms and classification systems.

81	The system should support the integration of data dictionaries to enable mappings between regulatory terms and data schemas.	See TR 21.
85	The system must provide vocabularies that are both human and machine readable.	This component supports achieving this TR by providing the hosting of a machine-readable data dictionary of terms.

3.3 Rule Repository Component

3.3.1 General Component Description

The purpose of this component is to provide a single repository of machine-operable building codes and rules within the ACCORD framework. This component, together with the API (2) Building Codes and Rules:

- Provides access to all other components of the digitised regulations in a suitable machine-readable format.
- Provides the ability for digitised regulations to be created/updated/deleted by Rule Formalisation Tool in a frame of the Component 1 Rule Formalization.

3.3.2 Implementation and Functionality

The component is implemented as a dedicated repository **aec3po** running in the ACCORD GraphDB instance. The repository aec3po contains named graphs, each of which stores the digitised rules of a particular building code/ruleset. The name of such a graph should conform to the following pattern: https://graphdb.accordproject.eu/resource/aec3po/{jurisdiction}/{classification}/{language}/{date}
Technical details of the implementation are given in the D2.4 Section "GraphDB as the storage for regulation graphs" [D2.4].

3.3.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 6.

Table 6 Rule Repository and Rule Provision - Comparison against Technical Requirements

TR No	Requirement Description	Implementation Detail
33	The system must be able to classify rules by country.	Each digitized building code/ruleset is stored as a named graph. The name of the graph conforms to the pattern, part of which is {jurisdiction} - which country defines the building code/ruleset. Hence, the URI of the named graph carries the necessary information to classify rules by country.
34	The system must provide country dependency between rules in the database and regulation documents.	
36	The system must provide the ability to store a database of rules.	
79	The system must provide versioning to all assets (models, rules, dictionaries).	When a new version of some building code or building ruleset is created and uploaded into the Rule Repository after the digitization process with the Rule Formalization Tool, a new named graph is created. Its name reflects the date of creation of such a digitized version.

87	The system should allow for the maintenance of links between results and maintenance of regulatory clauses even across document version changes.	Links between results are enabled due to the linked data nature of RDF chosen as the internal language for storage of building code/rulesets.
127	The system's data must be modelled as linked data in the sense that URI/ URLs shall be used when referencing definitions and instances.	Each AEC3PO ontology distinguishable element has its URI for referencing purposes. URI dereferencibility is provided by a proxy server operating with Ontotext GraphDB instance for the ACCORD project.

3.4 Information Requirements Provision Component

3.4.1 General Component Description

This component is responsible for providing IDS data for a respective building code. This allows other ACCORD components to retrieve an IDS file for a given building code as required. This component implements the ACCORD building code and rules API for the serving of IDS files.

3.4.2 Implementation and Functionality

This component is implemented relatively simply. When a request for an IDS file is received, the service either returns the IDS file (if it already exists) or calls the IDS generator to generate an IDS file. This is shown in the sequence diagram in Figure 5.

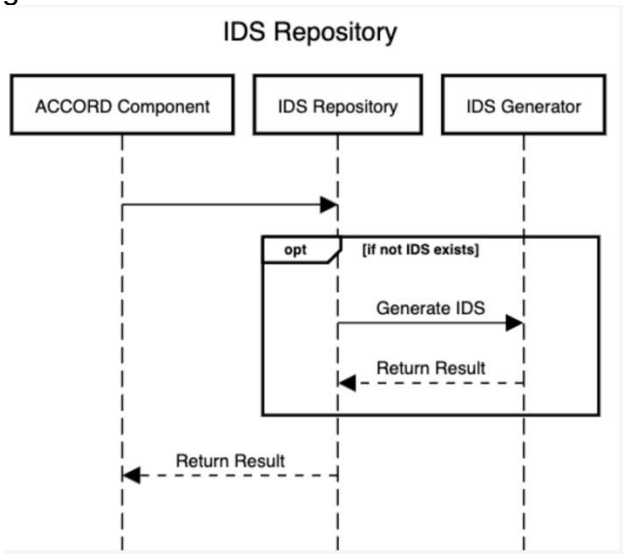


Figure 5. IDs Repository Functionality.

3.4.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 7.

Table 6. Information Requirements - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail

39	The system should allow for the configuration of varying requirements for BIM models (modelling guidelines and Level of Information Needs) required for differing submission stages and building permit types.	This component contributes to this TR by hosting IDS files that are either generated manually or by the IDS generation component.
48	The system should provide IDS checking services that are able to process any IDS that meets the buildingSMART IDS specification.	

3.5 IDS Generation Component

3.5.1 General Component Description

The IDS generation tool is an experimental component developed to automatically generate IDS from the formalized building codes and rules developed within the ACCORD project. This will be useful as it provides an automated method for generating IDS files, eliminating the need for manual development by a person.

3.5.2 Implementation and Functionality

The IDS generation tool is invoked by the IDS provision component when a manually created IDS file is not available for a given building code. Upon receiving this request, it retrieves data for the building code from the Building Codes and Rules Repository and the definitions used by the building code from the data dictionary repository. It will then produce an IDS and store it in the IDS repository.

The IDS generation itself is done in three steps:

1. Parsing the RASE embedded within BCRL documents.
2. Generating the minimum set of data requirements needed by the given building code. This is important as certain information is only needed in certain circumstances. Thus, it would be undesirable to fail an IFC model as part of model validation processes if the data is not required for that given building. Thus, the output of this step is a list of data requirements that are always required to check a given building.
3. Generation of IDS.

3.5.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 7.

Table 7. IDS Generation Component - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
38	The system must provide the ability to extract information requirements for digital building permit- and compliance processes and to represent these as a standardised data schema using BuildingSMART standards (i.g. IDS).	This component implements this TR by converting BCRL documents into the minimum IDS required to ensure the model is ready for checking.
40	The system should provide the ability to generate documentation of BIM (model)	This component supports the implementation of this TR by providing the ability to generate

	requirements for differing submission stages and building permit types and to adapt to locally differing sets of predefined requirements.	IDS for a building regulation if no manually created IDS exists.

3.6 Model and Data Requirement Validation Component

3.6.1 General Component Description

The Model and Data Requirement Validation Component ensures that the building model (IFC) and additional information (i.e., GIS data sets or supporting product data) that is to be checked on regulation compliance is of a good enough quality to be subject to such checks.

It is common that many BIM models lack the required detail, structure or information that is needed to execute detailed regulation compliance checks. For example, it is still common practice in the construction sector for supporting information to remain in PDF documents and not be transferred into building models. As this might result in false positives or negatives and thus misleading results, the Model and Data Requirement Validation Component executes several gatekeeper checks, which can be roughly split into four parts:

- **Validation of IFC Models on IFC validity:** Will ensure any IFC models provided to the ACCORD Cloud Architecture are valid IFC models with regards to one of buildingSMART's published schemas. This is required because any invalid models will not be able to be parsed and understood by ACCORD components.
- **Validation of IFC Models against IDS specifications**: Will ensure that IFC models provided to the ACCORD Cloud Architecture meet the requirements of the required IDS files. This is required to ensure that sufficient information is present in the IFC file to allow compliance checking against it to take place.
- **Validation of additional GIS Data:** Will ensure that any submitted GIS data conforms to agreed GIS specifications and profiles.

3.6.2 Implementation and Functionality

The implementation and functionality of the Model and Data Validation Component will be further described based on its four subcomponents.

Validation of IFC Models on IFC validity: BuildingSMART has been offering their IFC Validation tool[12], which offers:

a. STEP Syntax checks: The STEP Physical File syntax
b. IFC Schemac checks: An up-to-date (not withdrawn and latest revision) IFC schema referenced in the file, including formal propositions and functions encoded in the EXPRESS schema language
c. Normative IFC Rules checks: Other normative rules of the IFC specification (e.g. implementer agreements and informal propositions)
d. Additionally: Industry Practices: Checking the IFC file against common practice and sensible defaults. None of these checks render the IFC file invalid. Therefore, any issues identified result in warnings rather than errors

[12] https://validate.buildingsmart.org

e. Additionally: bSDD Compliance Checking whether references to classifications and properties from bSDD, found in an IFC file, comply with the source definitions in bSDD

In order to integrate the buildingSMART IFC Validation tool, a preview version of the IFC Validation Service API (REST) has been developed[13]. The API performs all of the checks mentioned above. After some initial authentication issues, it has been successfully tested using the orchestrator component (in this case, FME).

Results are returned as a list of all validation outcomes, optionally filtered by request_public_id or validation_task_public_id in JSON.

Validation of IFC Models against IDS specifications: For implementation of IDS validation we made the assumption that a valid IDS was provided. Creating valid IDS was thus out of the scope of this implementation, which solely focused on executing IDS checks as part of the ACCORD Architecture Framework.

For IDS checking, two tools have been used:

1. Solibri Cloud Checking: IDS checks for Solibri are stored in cset files and can be invoked through the Solibri Cloud Checking API (Solibri Checking as a Service).[14] Any cset file can be provided as payload to the API using an array of URIs. This has been successfully tested using the orchestrator component (in this case FME) and returns the check results both directly in JSON or in BCF (1.0, 2.0, 2.1 or 3.0).
2. Clearly.BIM: IDS checks for Clearly.BIM have to be configured in the Clearly.BIM backend and can be invoked through API based on their checkUuid[15]. It is not yet possible to dynamically provide an IDS definition as payload to the API. This has been successfully tested using the orchestrator component (in this case FME) and returns the check results both directly in JSON or in BCF.

3.6.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 9.

Table 8. Model and Data Requirement Validation - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
37	The system must support the ability to perform model verification - checking that a submission complies with relevant modelling schema.	The buildingSMART Validation service (preview API version) provides such a check.
48	The system should provide IDS checking services that are able to process any IDS that meets the buildingSMART IDS specification.	Both Solibri and Clearly.BIM are able to provide IDS checks through API.
49	The system must support the ability to perform model validation – checking that it	This is covered by the IDS checks as described in requirement 48. Of course, this

[13] https://dev.validate.buildingsmart.org/api/swagger-ui/

[14] https://checking-api-docs.solibri.com/

[15] https://hub.clearly.app/connect/bim-open-ap(REST) or https://hub.clearly.app/connect/bim-graphql(GraphQL)

	contains the required information to perform compliance checking.	does require the input of an IDS that checks on the required information.
50	The system must provide the ability to pre-check application data for compliance prior to formal submission.	All checks described in this chapter can be implemented through API and thus at any 'moment in time' of a permit submission process. This is just a requirement that is more relevant to the process execution component.
51	The system should provide a gatekeeping functionality to allow detection of invalid data format, while accepting that often partially incomplete submissions should be allowed.	By checking on IFC Validity and bSDD compliance (buildingSMART Validation Service) and IDS compliance (Solibri and Clearly.BIM), the gatekeeping functionality is provided.
96	The system must provide and facilitate BIM-model validation.	By checking on IFC Validity and bSDD compliance (buildingSMART Validation Service) and IDS compliance (Solibri and Clearly.BIM), the BIM-model validation is covered.
123	The system must support the provision of reporting results of model verification and validation.	Results of model verification and validation are provided in JSON and/or BCF.

3.7 Process Execution Component

3.7.1 General Component Description

The Process Execution component acts as the process backbone of the permitting process. Every permit request follows a predefined process. That process is often prescribed by local law, but in general, consists of steps like:

- An applicant applies for a permit.
- The applicant uploads relevant documentation, including models.
- The local permitting authority reviews the permit request and evaluates it against the relevant laws and regulation.
- The local permitting authority comes to a decision, the permit is issued, additional information or changes in the plans is required, or the permit is denied.

This component will project the coordinating implementation of the ACCORD digitised processes for building permitting. It will thus integrate and coordinate the other components of the ACCORD Cloud Architecture, enabling the execution of building permitting process flows specific to the local process required by permitting authorities.

3.7.2 Implementation and Functionality

For the implementation of this component, two parallel tracks were explored during the ACCORD project:

Use an existing permitting and licensing platform and extend it with functionality to invoke external functionalities through API: In ACCORD, this platform was provided by Lupapiste, a Cloudpermit permitting service.

The permitting service contains the state machine, a configurable component to manage each application's stage and status. The main stages are usually pre-consultation, application and construction. Under the stages the process is divided into phases. Examples of typical statuses in the application stage are draft, submitted, in review, changes requested, and permit issued. The process is often iterative.

Automatic compliance checking adds new statuses under the application stage. When the models are submitted to the service for automatic checking, the application itself is not yet submitted. Thus, there is a pre-checking status, where the designers can make sure their submission will be compliant with the data requirements and the building code.

The status of each model is managed separately

- The first status of a model is submitted.
- The first check is about data validation (IDS or proprietary format). Data-validated status means that the IFC model has been done according to the local BIM specifications, and it contains the required information in the correct places. Having the data correct is a prerequisite for compliance checks.
- The second check is about the building code compliance. If the checks return errors, a corrected version of the model needs to be submitted for checking, or the designers must provide the reason for exceptions before the application can be submitted.

Execution of checks is configurable. The checks usually apply to only certain types of projects or buildings, not all. Although the orchestration components and microservices could manage which rules should be checked, Cloudpermit can make it more efficient by only providing the relevant checks in the context. The configuration is managed by the system administrators and can be changed when there are new or changed checks. The Cloudpermit process execution is designed to work well with integration strategy B (chapter 5) as it is agnostic to the content and result of each atomic check.

Use an external orchestration component (with a wider scope than just the compliance checking microservice orchestration as described in 3.9): This track was not part of the initial scope of the ACCORD project but during the project the need for better understanding 'orchestration' arose. Future Insight experimented with using FME as an orchestration component.

The aim is to technically and practically explore the workings of a permit process orchestrator. A permit process orchestrator combines multiple tools/services (check) suppliers and can do this following a logical order. For example, this could be: Determine the type of checks to be executed; Validate the IFC-model; Perform checks; show results.

For the exploration of the orchestration concept within the scope of BIM-based permit checking, we defined a 'happy flow' for a permit process. We refer to this as a happy flow because it follows a path that is not blocked by, for example, invalid data or the unavailability of actual checks. The main goal was to a) keep momentum and progress and b) to check to what extent the components in the ACCORD Architecture are already 'orchestratable'.

On an abstract level, the happy flow consists of the following steps:

- The process is initiated by e.g. a permit and licensing platform.
- The IFC model is stored in some centralized storage (see 3.8 Data Storage Component) and picked up by the orchestrator from there.
- The model is then sent to an IFC/bSDD Validation service.

- After successful IFC/bSDD validation, it is sent to an IDS checking service.
- After successful IDS checking, we consider two possible paths:
  - Normal 'Compliance checking' which only requires a check definition and the contents of the IFC.
  - Zoning plan/geospatial checking, which, in addition, also requires data from an external source that is possibly location specific, e.g. a zoning plan providing a location-specific value for the maximum building height.
- End of happy flow

A visualization of this “happy flow” can be found here.[16]

FME has been used as a tool for testing this happy flow, as it has the capabilities of designing an automated workflow, including the consumption of API’s.

In addition, FME allows us to do quick data manipulations on models, input GIS data, rules and others, allowing us to quickly solve (and document that we took a shortcut) problems that would otherwise fully block progress.

Separate FME 'Workspaces' are used to call separate API microservices and process results. These are the yellow blocks in the flow. FME Flow is used to tie the workspaces together. That allows us to easily remove or replace entire API microservices from the flow, e.g., replacing compliance checking service A with service B.

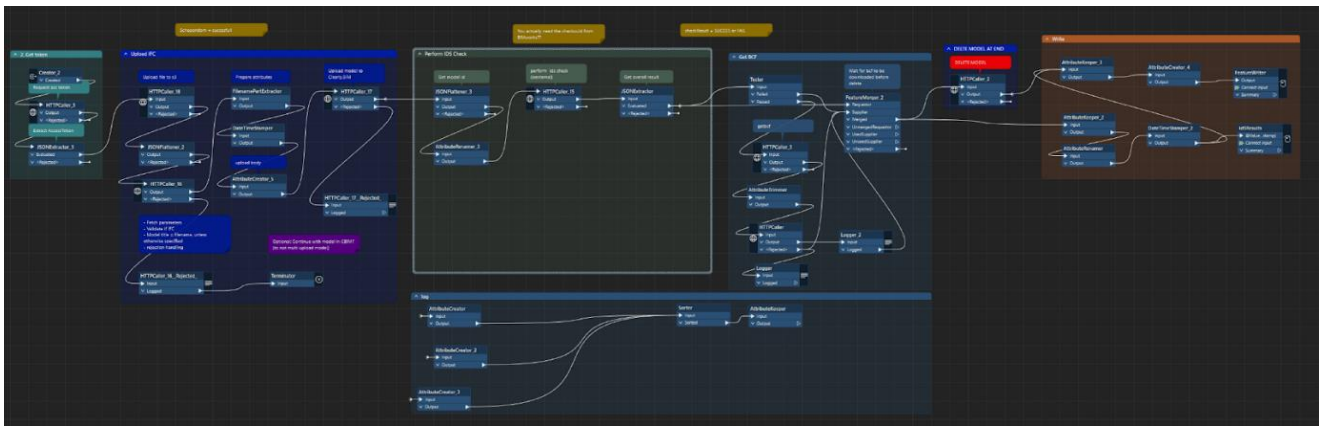


Figure 6. Example of a workspace. Accessing Clearly.BIM, upload an IFC model, perform an IdS check and process the results.

The results of the happy flow experiment are listed for each of the steps described above.

- Process initiation
  - The process initiation was successfully tested in two ways
    - Manually initiate the process in FME itself.
    - Use an FME Flow API endpoint that automatically spins up the FME Flow instance and starts the process.
- Retrieving model from storage
  - The model was successfully retrieved from:
    - An open URI

[16] https://link.excalidraw.com/l/hdqXUlvpPN/48s6dvgX8Vt

        - The Clearly.BIM BIM storage using a protected URI with authentication (token).
- IFC/bSDD validation
    - IFC/bSDD validation was successfully initiated from FME in the buildingSMART IFC Validation (preview) environment through their REST API. Results were returned in JSON.
- IDS checking
    - IDS checking was successfully initiated through Clearly.BIMs REST API interface. Results were returned in JSON and BCF.
    - IDS checking in Solibri still needs to be tested.
- Compliance checking
    - Compliance checking was successfully initiated through Clearly.BIMs REST API interface. Results were returned in JSON and BCF.
    - Compliance checking in Solibri still needs to be tested.
- Zoning plan/geospatial checking
    - As there were no ready-to-use zoning plan/geospatial checks that use an external variable from, e.g. a zoning plan WMS/WFS, this has not been tested.

It should be noted that multiple solutions for these processes have been developed within the ACCORD project, as the execution of the building permitting process varies between demonstration cases. However, for brevity, only the general case is described here.

3.7.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 10.

Table 9. Process Execution Component - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Details
52	The processing services should run asynchronously for long operations.	In FME, asynchronous operations were successfully used in two variants: • Polling execution progress every X amount of time. • With an active push from the service when results were ready.
54	The processing services must provide persistent activity and results logs with output links as relevant.	The microservices themselves generally purge the results after they've been executed. E.g. because duplication of models needs to be prevented. Results can however be persistently stored by the process execution component, but this hasn't been implemented in FME. In Cloudpermit's solution, the results from the checks are retrieved in JSON and BCF format and stored in the data storage. Each checking round is executed independently, and the previous rounds' results do not affect the following rounds.
57	The system must provide configurable components and services allowing to adjust core functionalities according to local regulations.	The FME orchestration experiment proved that one can easily switch between different similar components, e.g. switch between Solibri or Clearly.BIM for IDS and compliance checking.

92	The system should allow users to select the set of rules to be used for the checking.	For both Solibri and Clearly.BIM one can execute any of the available checks in each system. The process for invoking the checks is standardized in both cases.

3.8 Data Storage Component

3.8.1 General Component Description

The data storage component forms the central location where data that is used in the cloud-based permitting service is stored. The 'single store multiple use' concept is important here. Information should be stored only once to create a single source of truth and be useable and used by multiple applications. The primary use of the data storage component will be to store BIM and GIS models, along with any metadata relevant to these models.

Two methods will be utilised to provide this storage: (1) standard static file storage, and (2) semantic storage. These will both be described in the following section.

3.8.2 Implementation and Functionality

- Standard static file storage. This component will store models as a single file. Access to these files will be provided via the OpenCDE Documents API[17].
    - Clearly.BIM static storage has been implemented, and models can be retrieved through a protected URI.
    - CloudPermit static storage contains the models and pre-processed Solibri BCRL rulesets.
- Semantic Data Storage. This storage component will allow retrieval of only parts of a model. i.e. 'give me all doors' or 'give me all objects related to fire safety'. There is no open standard for retrieving only specific parts of a model yet. So, SPARQLwill be utilised as a medium to provide access to this data. To store BIM data in this format, it will be converted into RDF and stored in GraphDB.

3.8.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 11.

Table 10. Data Storage - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
55	The system should provide archival of submitted BIM models.	Clearly.BIM supports storing different versions of a model. In the Semantic Data Storage implementation, each BIM model in IFC is submitted to GraphDB after its conversion into RDF and is stored in a newly named graph created as a storage of the submitted cityGML data, converted into RDF/Turtle format.
56	The system should support the upload of BIM model files in an appropriate format.	Clearly.BIM supports uploading of BIM models in IFC. In the Semantic Data Storage implementation, a BIM model after conversion into RDF/Turtle was uploaded manually.

[17] https://github.com/buildingSMART/documents-API

79	The system must provide versioning to all assets (models, rules, dictionaries).	For BIM models, Clearly.BIM offers versioning. In the Semantic Data Storage implementation, a new named graph created for a new version of a BIM model.
125	The system should facilitate that multiple microservices communicate the simultaneous process of compliance checking using parts of the IFC model.	Clearly.BIM can provide the model to multiple microservices simultaneously.
126	The system should provide the ability to export submitted (BIM) models.	Clearly.BIM supports exporting BIM models in various formats, e.g. IFC and CityGML. In the Semantic Data Storage implementation, BIM model in cityGML can be exported in various formats (JSON, JSON-LD, RDF/XML, N3, N-triples, other) manually using Ontotext GraphDB UI.

3.9 Orchestrating Microservices Component

3.9.1 General Component Description

The function of the orchestrating microservices component is to identify and manage the execution of compliance checking across the ACCORD compliance checking microservices. This service is required since the ACCORD Cloud Architecture will consist of multiple microservices, each of which may well have several different instances running.

Thus, this component performs two tasks. (1) It maintains a list of current ACCORD microservices, and (2) it selects and triggers compliance-checking microservices in order to execute compliance-checking against a given Building Code.

3.9.2 Implementation and Functionality

This component has implemented X key elements of functionality:

1. Ability to maintain a list of available microservices
2. Ability to call microservices to retrieve their functionality.
3. Ability to parse BCRL and execute compliance checking on a given BCRL document by calling one or more compliance-checking microservices.

This will be delivered through two sub-components:

1. **Microservice Functionality register:** A list of microservices with a set of their functionality. This functionality is retrieved by calling each microservice (using the results API) to retrieve their capabilities.
2. **BCRL Execution:** This component performs the steps listed below for each building code that requires execution. This is shown in the sequence diagram, Figure 7.
    - **a.** Retrieve and parse BCRL from the repository of building codes.
    - **b.** Determine and order of execution as defined within the BCRL.
    - **c.** For each individual compliance check to be executed
        - i. Lookup the definition stored in the bSDD.
        - ii. Select a microservice for execution based on the microservice functionality register and the data in bSDD.
        - iii. Call the microservice using the Results API.

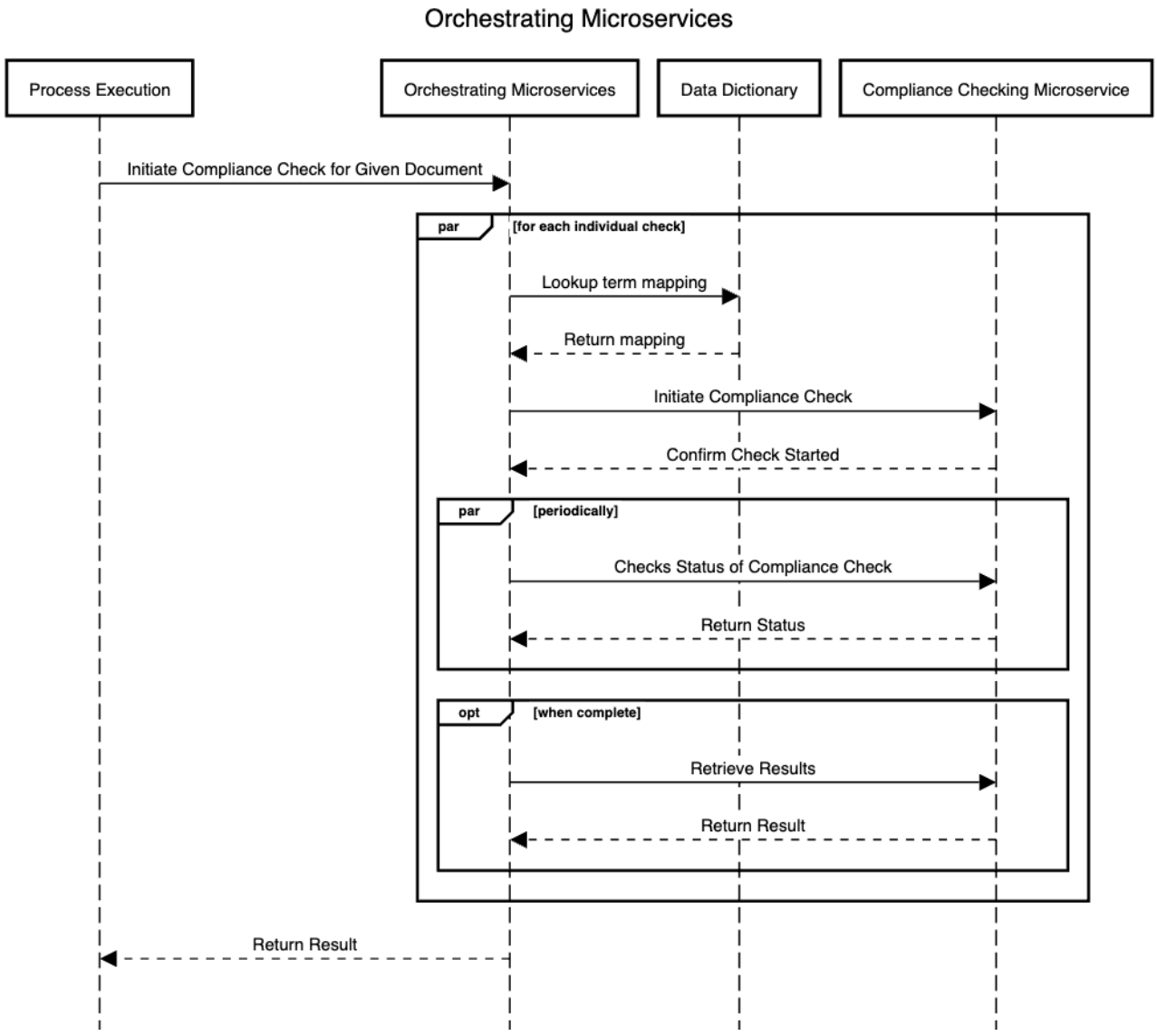


Figure 7. Orchestrating Microservices Sequence Diagram.

3.9.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this component and the relevant technical requirements defined in D4.1 is given in Table 12.

Table 11. Orchestrating Microservices - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
52	The processing services should run asynchronously for long operations.	This TR is implemented through the adoption of asynchronous operation through the Results API.
53	The system shall allow the optimisation of the execution of automated compliance checking	This component delivers this TR by determining, based on the contents of BCRL, what regulations should be checked, in what

	based on the computation expense (time/cost) of a given logical operation.	order and by which compliance checking microservice.
54	The processing services must provide persistent activity and results logs with output links as relevant.	This component supports this TR by recording results provided by the compliance-checking microservices and the building code clauses that they are relevant to.
57	The system must provide configurable components and services, allowing for the adjustment of core functionalities according to local regulations.	This component supports this TR by delivering the integration between the re-useable components (the compliance checking microservices)
58	The system shall provide several reusable calculation components.	
70	The system must be able to automatically determine the regulations to be checked against based on building criteria.	See TR 53.
84	The system must allow for the execution of high-level logic behind building permit processes and regulation documents.	
87	The system should allow for maintenance of links between results, and maintenance of regulatory clauses even across document version changes.	See TR 54
91	The system must link or show the corresponding rule(s) that led to a given result.	
134	The system must allow for the "registration" of services and component capabilities (e.g. what can each component do).	The component supports this TR by providing a list of all compliance checking microservices.

4 ACCORD API Definitions

This section documents the ACCORD APIs. It firstly, describes the existing APIs that have been re-used as well as those that have been developed specifically within the ACCORD project. For the APIs that have been developed, the functionality is described as well as links to API documentation being provided. Finally, the functionality of each of these APIs is compared against the technical requirements defined in D4.1.

In total, 5 APIs are defined by the ACCORD project, 3 are newly developed, and two are re-used:

- API 1– Definitions API (re-used)
- API 2 – Building Codes and Rules API (developed)
- API 3 – Information Services APIs (re-used)
- API 4 – Data API (developed)
- API 5 – Results API (developed).

Given that the re-used APIs already exist, they will be described briefly here, along with a comparison against the technical requirements in Table 13.

Definitions API: The definitions API, used to retrieve definitions from a data dictionary, that will be used by the project is the standard Building Smart Data Dictionary API. Details on this API can be found here[18].

Information Services APIs: Information Services APIs are a collective term for any API that provides a connection between a microservice and a source of data (operated by an external third party) that it requires. Within the project, only one of these is needed (as shown in the ACCORD cloud architecture diagram), which is for the Finnish LCA Calculation Microservice. Due to how the external third party exposes this data – a simple HTTP request was all that was required to be implemented.

Table 12. Re-used API - Comparison against Technical Requirements.

TR No	API	Requirement Description	Implementation Detail
21	1	The system must provide a rule tool allowing rule authors to bind rules to data (IFC, bSDD, various data representations).	These are provided using the definitions API that allows components to download and update the bindings between rules, model data, the method of calculation, or if results should be provided by a human.
22	1	The system should provide a rule configurator allowing to select rule calculators depending of the data representation	
23	1	The system should provide a rule tool assisting rule authors with auto- completions, in-place documentation and lookup into IFC- and bSDD-definitions.	
31	1	The system must enable mapping between abstract concepts in regulatory documents and concrete concepts in BIM data file format. E.g. data may be stored in a different place in different versions of the IFC model.	
59	1	The system must allow for human input where no automated result is available and provide the ability for human override of automated results (assuming a correctly qualified user).	
60	1	The system should be able to describe how checks are implemented: "simple": implemented in a declarative language like BIM SPARQL; "complex": implemented by invoking specialized calculations (but the purpose and input/output of these routines are semantically described).	

[18] https://app.swaggerhub.com/apis/buildingSMART/Dictionaries/v1

61	1	The system should provide the ability for BIM checks using a proprietary check definition (as there is no standard available).	
78	All	The system must facilitate the communication between system components through standardized APIs.	This is achieved through the re-use of existing commonly used APIs where possible.
80	1	The system should support the use of classification systems.	See TR 21 above
81	1	The system should support the integration of data dictionaries to enable mappings between regulatory terms and data schemas.	
83	1	The system must allow marking rules with subjective judgement as “isSubjective" (requires human judgement).	
85	1	The system must provide vocabularies that are both human and machine readable.	This is provided using bSDD and its API, through which ACCORD components can download vocabularies.
130	3	The system must provide support to various Coordinate Reference Systems (CRSes) supporting spatial data integration from external sources.	See TR 72 above

4.1 Building Codes and Rules API

The building codes and rules API is designed to allow other components within the ACCORD cloud architecture to retrieve building codes and rules in a standardised format. A full API description for the developed API is available[19].

This API provides the following endpoints:

1. Retrieval of regulations server metadata.
2. Retrieval of the latest version of a building code in both YAML and JSON-LD.
3. Returns a specified version of the building code.
4. Execute a graphQL query over a building code.
5. Return IDS for a given building code (and or version).
6. Creates a new building code
7. Returns a specific section or paragraph of the building code.

It should be noted that the building codes and rules API returns data in either JSON-LD or YAML to give integrators the freedom to choose the format that best suits their use case.

A comparison of the functionality of this API and the relevant technical requirements defined in D4.1 is given in Table 14.

Table 13. Results API - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
24	The system must provide a rule tool converting rules from building compliance rule language (YAML) to RDF and allowing rule authors to edit and explicate the rules.	This is achieved by allowing data to be returned in either JSON-LD or YAML.
33	The system must be able to classify rules by country.	This is provided through the use of country classifications in the API endpoints, allowing users to retrieve building codes based on country code.
34	The system must provide country dependency between rules in the database and regulation documents.	

[19] https://accord-project.github.io/API-Development/buildingcodesandrules.html

36	The system must provide the ability to store a database of rules.	This API is designed to provide the interface between such a database and other ACCORD components.
76	The system must provide a generic API framework to allow third parties to create new services that can interact with core components.	This API is one of the ACCORD developed APIs that will contribute to the generic API framework.
78	The system must facilitate the communication between system components through standardized APIs.	
79	The system must provide versioning to all assets (models, rules, dictionaries).	The API supports versions of building codes and rules.
87	The system should allow for maintenance of links between results and maintenance of regulatory clauses even across document version changes.	This API contributes towards achieving this by providing access to building codes and rules (and their paragraphs and clauses) via a URI reference.
89	The rule formalization tool should link the regulation's document version to the rules being created. The system should allow updating the rule database, accordingly, that is providing a data storage for archiving checking results including links to relevant rules and regulation document versions.	This API contributes towards this TR by allowing the upload of building codes and rules by the rule formalization tool.
127	The system's data must be modelled as linked data in the sense that URI/ URLs shall be used when referencing definitions and instances.	See TR 87.

4.2 Data APIs

The data API is designed to allow other components within the ACCORD cloud architecture to retrieve building model data. A full API description for the developed API is available[20].

This API provides the following endpoints:

1. Return a given model based on a model ID
2. Execute a graphQL query (if supported by the model format) of a given model ID.

A comparison of the functionality of this API and the relevant technical requirements defined in D4.1 is given in Table 15.

Table 14. Results API - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
55	The system should provide archival of submitted BIM models.	This is provided through the ability to retrieve a model through its model ID.
56	The system should support the upload of BIM model files in an appropriate format.	This is supported by the use of the model ID to retrieve models.
76	The system must provide a generic API framework to allow third parties to create new services that can interact with core components.	This API is one of the ACCORD-developed APIs that will contribute to the generic API framework.
77	The system must provide a Data Accessor Configurator to enable access to specific data from BIM models via external APIs.	This API provides the needed access to BIM models.

[20] https://accord-project.github.io/API-Development/data.html

78	The system must facilitate the communication between system components through standardized APIs.	See TR 76.
79	The system must provide versioning for all assets, including models, rules, and dictionaries.	BIM model versioning is implemented by the fact that each model has an immutable model ID.
126	The system should provide the ability to export submitted (BIM) models.	See TR 77.

4.3 Results API

The results API is designed to allow other components within the ACCORD cloud architecture, especially the orchestrating microservices components, to interact with compliance-checking microservices. All compliance-checking services should implement this API to integrate with the rest of the ACCORD cloud services. A full API description for the developed API is available[21].

This API provides the following endpoints:

1. Retrieve the metadata about compliance checking microservice and its capabilities. This includes:
   a. A list of model formats that the microservice supports
   b. A list of building code terms (as described in the data dictionary) that this microservice can support.
2. Initiates compliance checks. This operates in two modes to support differing integration scenarios:
   a. Initiating a compliance check across an entire building code.
   b. Initiating a compliance check for a given building code term (or set of terms).
3. Retrieving the status of a compliance check.
4. Retrieving the results of a compliance check in either JSON or BCF format.

It is important to note that the API utilises an asynchronous design, this is to enable compliance checks (which may take some time) to operate in parallel. This is shown in the sequence diagram shown in Figure 8.

[21] https://accord-project.github.io/API-Development/results.html

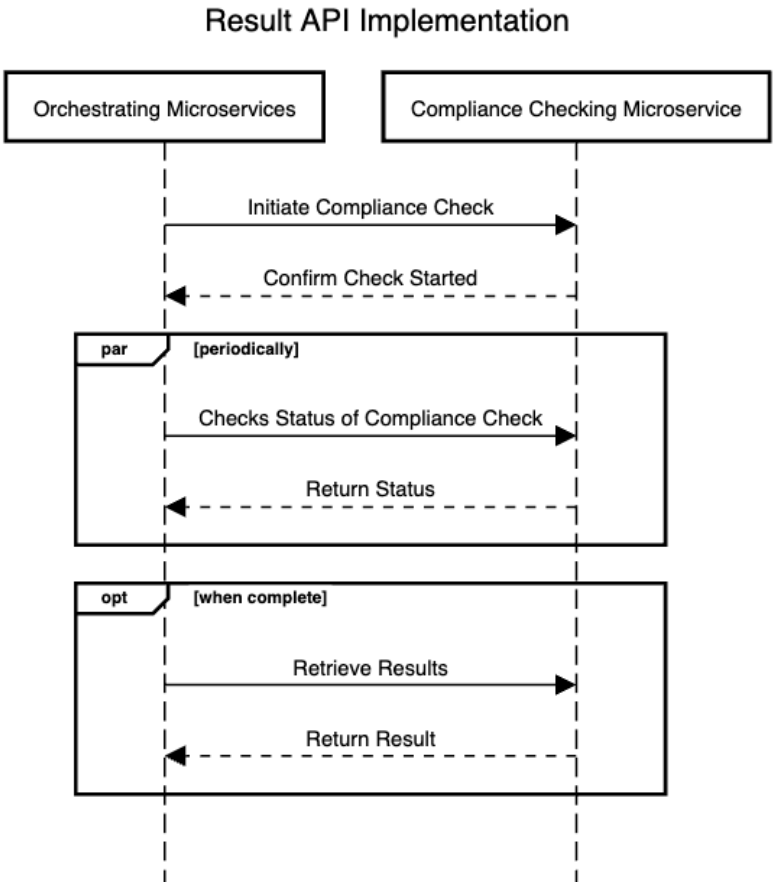


Figure 8. Result API Implementation.

A comparison of the functionality of this API and the relevant technical requirements defined in D4.1 is given in Table 16.

Table 15. Results API - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
52	The processing services should run asynchronously for long operations.	This is provided as per the asynchronous implementation shown in Figure 7.
54	The processing services must provide persistent activity and results logs with output links as relevant.	This is provided via the step-by-step checking and results supported by this API.
54	The processing services must provide persistent activity and results logs with output links as relevant.	This is supported by the data returns as part of the BCF/JSON results from this API.
57	The system must provide configurable components and services allowing to adjust core functionalities according to local regulations.	This API provides the method for the core services and this configurable component to communicate.
58	The system shall provide several reusable calculation components.	
76	The system must provide a generic API framework to allow third parties to create new services that can interact with core components.	This API is one of the ACCORD developed APIs that will contribute to the generic API framework.
78	The system must facilitate the communication between system components through standardized APIs.	

87	The system should allow for maintenance of links between results and maintenance of regulatory clauses even across document version changes.	This API allows this by using the building code URI reference as part of the response data.
91	The system must link or show the corresponding rule(s) that led to a given result.	
134	The system must allow for the "registration" of services and component capabilities (e.g. what each component can do).	This API provides this via its functionality to retrieve the capabilities of compliance-checking microservices.

5 ACCORD Integration Strategies

This section documents the ACCORD integration strategies, covering the differing methods of integration of microservices with the ACCORD core components that have been utilised within the project. It will also document the integrations required by the demonstration activities by describing the ACCORD service set-ups used in each demonstration. Finally, this Section describes the extensibility of the ACCORD framework and how new novel components can be continually integrated.

5.1 Summary of Microservice Integration Approaches

The ACCORD cloud architecture can be integrated with compliance-checking microservices in two ways, depending on the requirements of the permitting process being executed.

Integration Strategy A: This integration strategy makes use of the Orchestrating Microservices component. In this case, the Orchestrating microservices component parses the rules provided by the building codes and rules repository and invokes one or more microservices to complete compliance checking. It is also responsible for integrating the results of the compliance checking from these microservices into a result. The sequence diagram for this execution strategy is shown in Figure 9.

Integration Strategy B: This integration strategy bypasses the Orchestrating Microservices component and instead invokes a single microservice to perform the entire compliance checking process for a given building code. This microservice will be responsible for parsing the BCRL, fetching data and performing all required compliance checks before returning results to the process execution component. The sequence diagram for this execution strategy is shown in Figure 10.

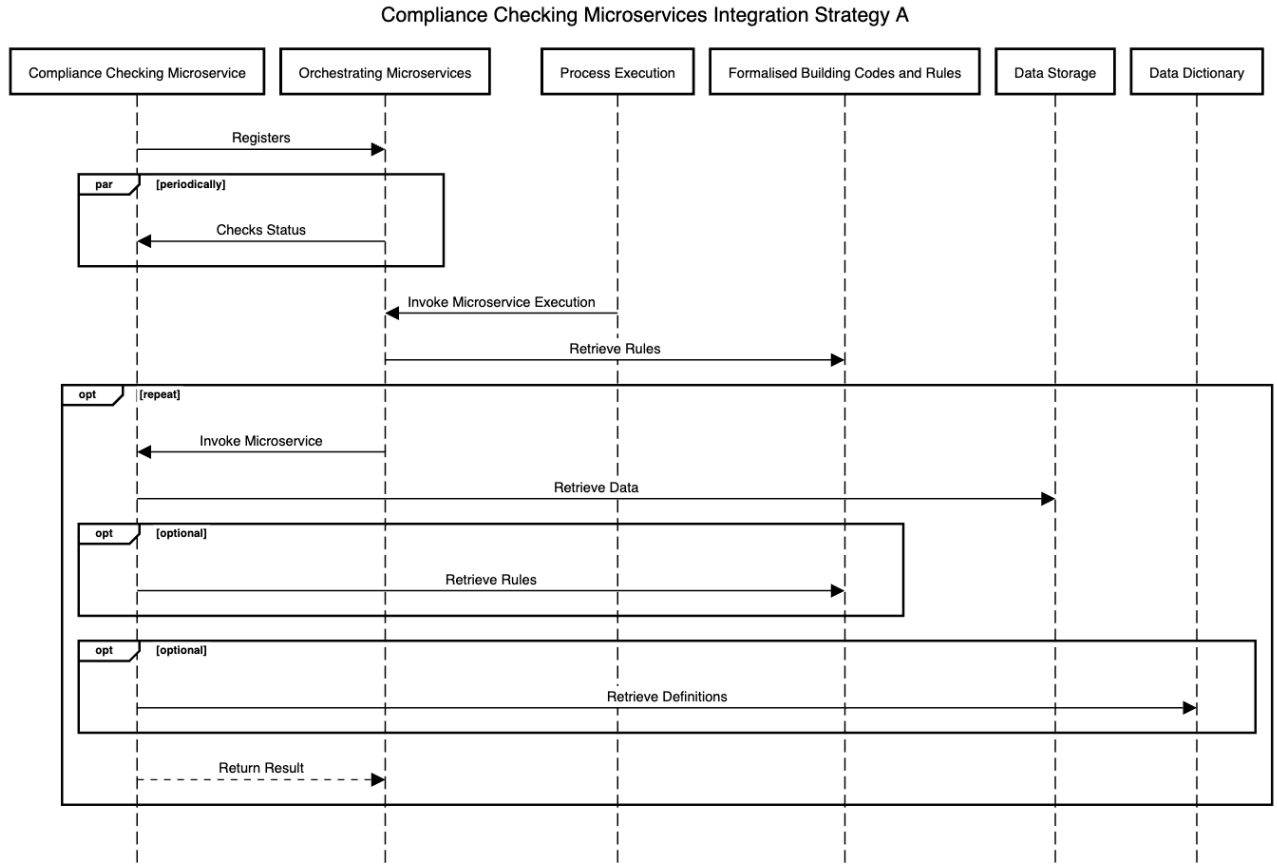


Figure 9. Integration Strategy A.

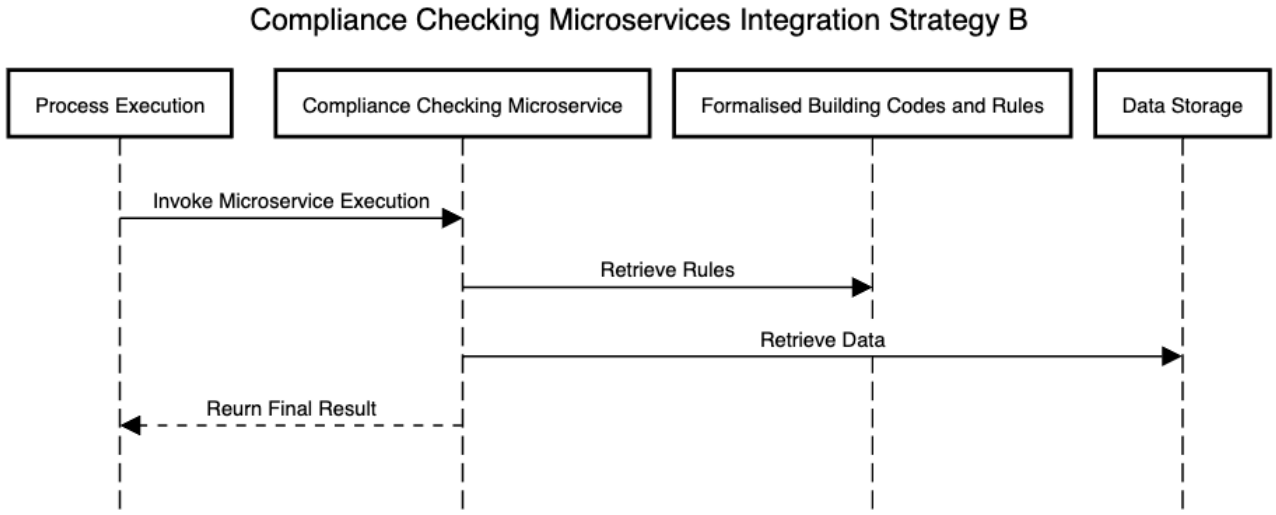


Figure 10. Integration Strategy B.

The advantages and disadvantages of this integration approach are compared in Table 17.

Table 16. Integration Strategy Comparisons.

Integration Strategy A	**Integration Strategy B**

Enables use of multiple microservices to perform compliance checking on a given building code.	Microservices must parse and understand BCRL.
Microservices are not required to implement BCRL parsing.	Self-contained checking.
Microservices are loosely coupled.	Gives microservice developers full control over the checking process.
The orchestrator can use multiple microservices to calculate the most efficient route to determining compliance.	Cannot make use of multiple microservices.

5.2 ACCORD Service Setups

ACCORD Service Setups are defined as subsets of the refined ACCORD Cloud Architecture by presenting the integration of use case-specific solution developments for all demonstration countries.

ACCORD deliverable D4.1 “Technical Requirements Elicitation, Analysis and Cloud Architecture Model” documented the demo-specific mapping of ACCORD Cloud Architecture components to the ACCORD demo use cases.

Besides the ACCORD Cloud Architecture components (see Chapter 2), ACCORD Service Setups were developed and refined iteratively. The Service Setups showcase what elements of the ACCORD Cloud Architecture will be deployed at each demo and in what context. The following listing and numbering of the ACCORD Cloud Architectures components and demo use cases were used for the mapping:

Demo Use Cases:

- DE - Germany
    1. Planned Land Use Permitting
    2. Environmental Compliance
    3. Architectural Design Compliance of Timber Construction Systems
- ES - Spain - Urban Regulations
- UK – United Kingdom - Steel Modular Housing
- FI - Finland
    1. Building registry data
    2. Accessibility
    3. CO2 calculation
    4. Operational Safety
- EE - Estonia
    1. Accessibility / Education (School/ Kindergarden).
    2. CO2 calculation (it should be noted this uses the same technology as FI use case 3 so will not be described separately)

ACCORD Cloud Architecture Components:

1. Rule Formalization
2. Data Dictionaries Repository
3. Rule Repository

4. IDS Repository
5. IDS Generation Tool
6. Model- and Data Requirement Validation
7. Process Execution
8. Data Storage
    a. File-Based Data Storage
    b. Semantic Data Storage
9. Orchestrating Microservices
10. Compliance Checking Microservices
    (1) Solibri
    (2) Future Insight Clearly.BIM
    (3) LCA Finland
    (4) LCA Germany
    (5) Eurocode Compliance Checking
    (6) Urban Regulations Checking
    (7) Land Use Building Compliance Checking
    (8) Timber Construction Compliance Checking
11. Information Services
    (1) Timber Construction Database
    (2) Material / Emissions Databases.

Two modes of deployment exist for a given component: (1) Normal, where a component will be used to deliver the demo and (2) experimental, where a more experimental component developed by the ACCORD project will be tested as well, to validate its functionality and usability.

Table 18 presents the mapping of the ACCORD Cloud Architectures components and demo use cases, with demo cases illustrated as columns and the components as rows. In the table, blue-indicated components are used as normal components within the demo and orange-indicated (O) components are used in an experimental context. The table also identifies which integration approach is used by each demonstration. In row 8 the A/B indicators state which type of data retrieval this given use case will use; (a) file based or (b) semantic.

Table 18. Refined alignment of ACCORD Cloud Architecture components to demo use cases.

	DE 1	**DE 2**	**DE 3**	**ES**	**UK**	**FI 1**	**FI 2**	**FI 3**	**FI 4**	**EE1**	**EE2**
Int	A	A	A	A	A	B	B	A	B	A	A
	ACCORD Core Cloud Components										
1											
2											
3											
4											
5				O			O				
6											
7											

8	B	A	A	A	A	A	A	A	A	A	A
9											
	ACCORD Microservices										
(1)											
(2)											
(3)											
(4)											
(5)											
(6)											
(7)											
(8)											
	Information Services										
(1)											
(2)											

5.3 Extensibility of ACCORD Framework

Extensibility of the ACCORD framework is achieved via two methods: (1) Customisation of Permitting Process and (2) Addition of the new Microservices.

Customisation of Permitting Process: The ACCORD process execution component provides the ability to execute a custom permitting process. The ACCORD project proposes a default process (described in Section 3.7), but the intention is for each user of the ACCORD framework to specify their own process. This allows each permitting authority to customize their process and the ways in which ACCORD components are composed to perform their own permitting process.

Addition of new Microservices: As new digitised regulations are added to the ACCORD system, new microservices can also be added to support their execution. The process of adding new microservices consists of the following generic steps:

- **Determine if a new microservice is needed:** Firstly, it should be verified that a new microservice is indeed necessary and that no existing microservice already supports the functionality being implemented.
- **Analyse required BCRL Terms:** The BCRL terms identified via the ACCORD rule formalization process should be analysed, and the implementation of the microservice should be aligned with this so that each individual functional check to be implemented within the microservice aligns with each BCRL term. An example of this is shown in Figure 11. This is method is described fully in D2.2.

- **Select checking method:** Once the functional checks have been defined their implementation method should be defined. This could include (a) manual implementation, (b) use of external software tool, (c) use of external component via API.
- **Implement checking method:** The final checking method should be implemented/integrated.
- **Implement Data API:** To access model data (either IFC or GIS) the microservice should utilise the Data API to retrieve it from the core ACCORD data storage service.
- **Implement Results API:** Once the implementation is complete it should be exposed to the other ACCORD components via the implementation of the results API.
- **Register Microservice in Data Dictionary:** Finally, the microservice should be registered in the data dictionary and associated with appropriate BCRL terms to enable it to be utilised by the core ACCORD components.

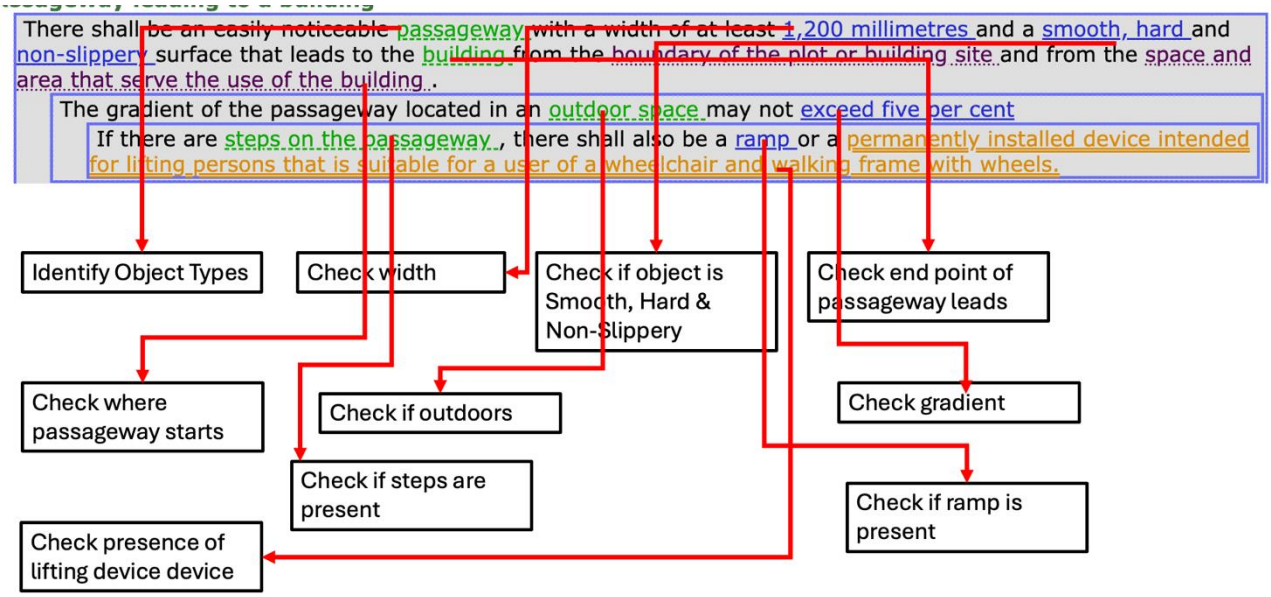


Figure 11. Example Mapping of BCRL Terms to Microservice Functions.

6 Integration of ACCORD Compliance Checking Microservices and Information Services

This section outlines each of the demo-specific microservices that have been developed within the project. For each of them, it first describes the problems they solve within their respective demonstrations. Then, their functionality and how they have been developed are described, along with a comparison to the technical requirements that govern their development.

6.1 Urban Regulations Microservice (Spain)

6.1.1 General Microservice Description

The purpose of this microservice is to automate the process of checking the compliance of building project proposals – for new construction buildings and the renovation of existing buildings – with the urban parameters defined in municipal urban planning plans. This type of check is typically conducted by municipal technicians in city or town councils in countries such as Spain.

6.1.2 Microservice Implementation and Functionality

This microservice is composed of four parts, each of which is executed in a specific execution order, as shown in Figure 12.

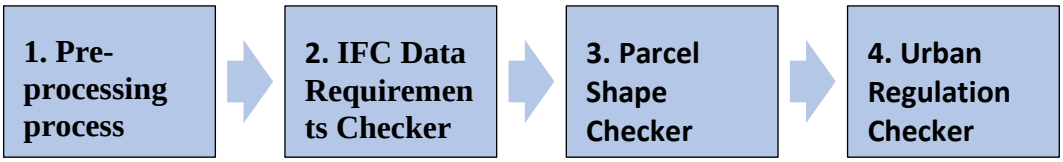


Figure 12. Urban Regulations Structure.

All the parts have been implemented using Python language and Django, a high-level Python web framework, for their deployment on a server. The source code is available on GitHub[22]. The inputs for this microservice are a BIM model (in IFC format) and a cadastre file (in GML format).

Pre-processing process: This process enriches the data in the cadastre input files with the necessary information so that it can be properly processed by the Urban Regulations checker microservice, as well as removing data that is not necessary. This service is executed only once to generate this information for each cadastre input file.

IFC Data Requirements Checker: This part is responsible for checking that the BIM of the building project complies with the data requirements necessary to ensure that the part for checking compliance with urban planning regulations is executed. These data requirements are provided in a file specified in the IDS standard format.

[22] https://github.com/Accord-Project/Urban-Regulation-Microservice

Parcel Shape Checker: This part checks if the perimeter of the parcel provided in the cadastre file coincides with the parcel's perimeter specified in the building project's BIM.

Urban Regulation Checker: This part checks that the building project complies with urban planning parameters defined in the municipal urban planning plans. To perform the checking, data from four different sources are required: (1) Cadastre data, (2) Topographic data, (3) urban data, and (4) project data provided in a BIM model represented in the IFC standard.

The functions of this microservice are exposed to the ACCORD core components via an implementation of the Results API.

6.1.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this compliance-checking microservice and the relevant technical requirements defined in D4.1 is given in Table 18.

Table 17. Urban Regulations - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
68	The system's checking of compliance with urban regulations must be automated. The permitting process will be applicable to both public and private buildings during the design and construction phases, using BIM and GIS as data input.	This microservice implements the urban regulation checking as described above.
69	The system must allow to integrate GIS- and BIM data to carry out checking against urban regulations using one unified model.	As part of the pre-processing and parcel shape checking elements of this microservice, the GIS and BIM data are integrated for the purposes of compliance checking.

6.2 Solibri and Cloudpermit Microservice Integration (Finland)

6.2.1 General Microservice Description

Solibri Checking as a Service (CaaS) checks IFC files against building codes. The service is based on the existing industry-proven Solibri technology that is adapted to be used through an API. An integrating party can send their IFCs and building code specifications through the API. The results are provided in BuildingSmart's BCF format and JSON. The building code specifications can either be in Solibri's proprietary Ruleset format or in the BCRL language format that is developed in the ACCORD project. Thus, to implement this microservice, Solibri has added the ability to fetch and process BCRL from the rule repository to perform checks.

In the context of the ACCORD project and this work, the integrating party is Lupapiste, a permitting service by Cloudpermit that leverages the Solibri CaaS checking capability through the API. In the ACCORD architecture schema, it also implements some functionalities of the process execution and data storage components. Lupapiste is used as a user interface and process management system in any permit application process. It collects and hosts all the data for each individual permit application and contains all checks and configurations required to run the automatic checks. The Lupapiste-Solibri implements Integration Strategy B, presented in Chapter 5.

6.2.2 Microservice Implementation and Functionality

This implementation is divided into two stages:

Pre-processing of BCRL: Solibri has a utility that takes BCRL as input and converts it to a Solibri Ruleset, containing one rule for each BCRL section. Each rule contains the BCRL of that section as a YAML string. The pre-processed BCRL Rulesets are stored in Cloudpermit's local rule repository for later use (Figure 13).

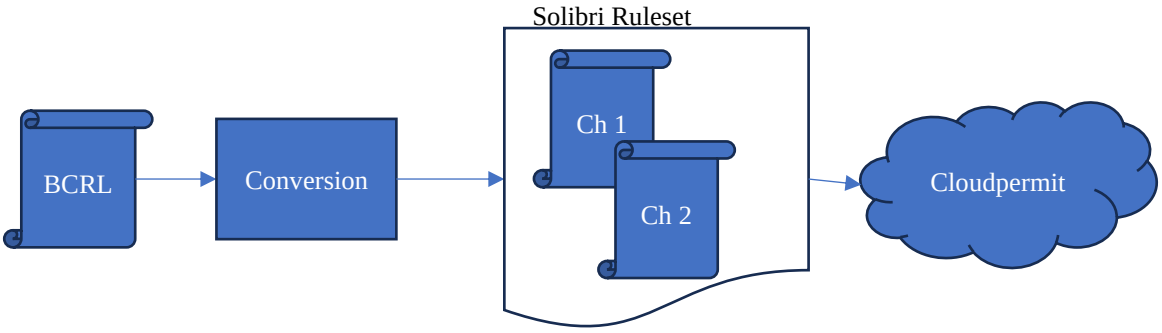


Figure 13. Conversion of BCRL to Ruleset.

Running BCRL against a model: When an IFC model is submitted in a permit application and needs to be checked, Cloudpermit calls the Solibri API and sends the model and the Rulesets to Solibri CaaS for checking. Cloudpermit selects the Rulesets based on the type of the project and the type of the building to speed up the automatic checking.

Solibri evaluates each BCRL rule against each component in the model. When a component matches all applicabilities and does not match any exception and finds itself in a requirement, it is checked against the requirement. If the requirement is not met, an issue is created.

The API hides the complexity of the microservice checks from the permitting service and returns the complete result sets in a standard way independent of the format of the check. Cloudpermit is agnostic to the format of the Rulesets; they can be proprietary Solibri format or BCRL. (Figure 14)

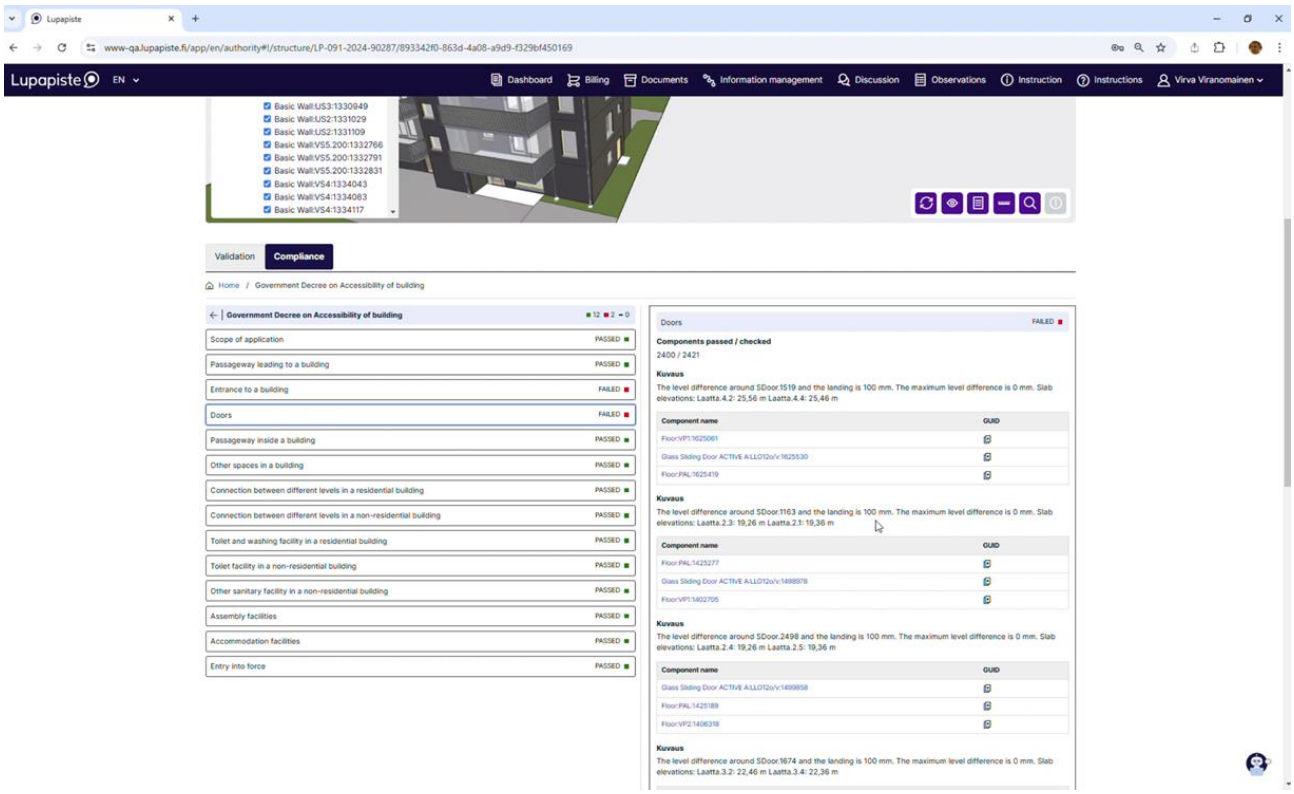


Figure 14. Results from BCRL checks are shown in Lupapiste, a Cloudpermit service.

It is also possible to run the BCRL check in Solibri UI without the integration to Cloudpermit, as shown in Figure 15.

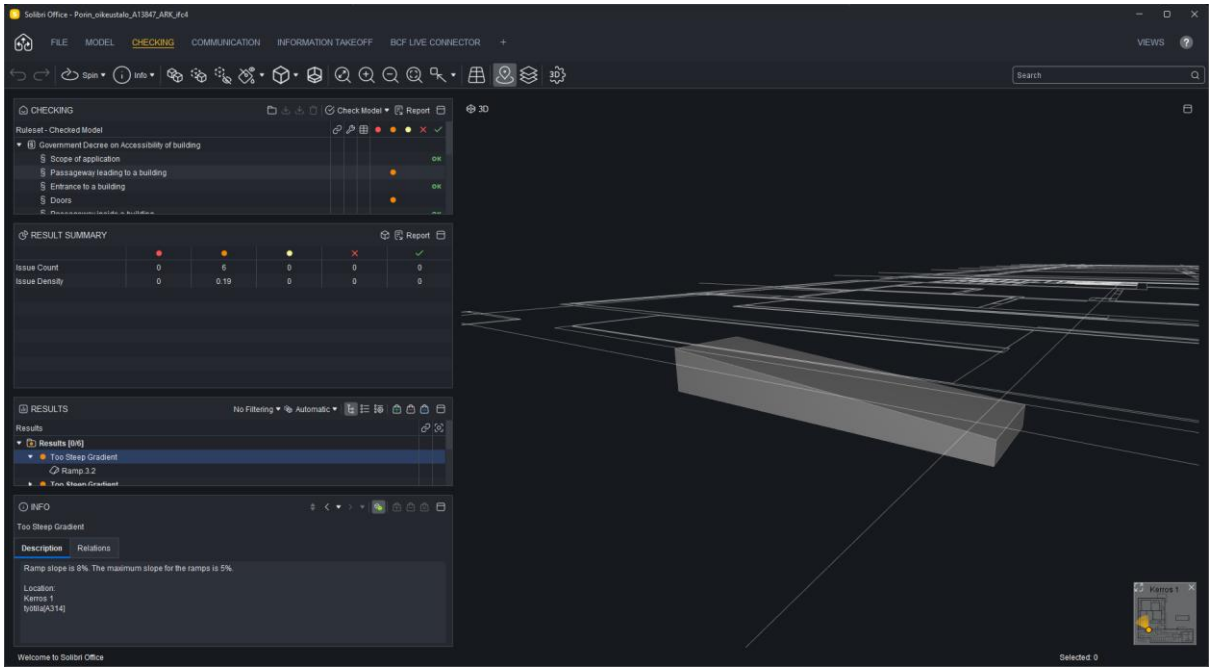


Figure 15. Running BCRL check in Solibri desktop application.

BCRL Implementation in Solibri: Solibri has an algorithm evaluating BCRL. This algorithm is independent of the given BCRL. The only thing that is BCRL-specific is the terms. When the algorithm needs to evaluate a term, it asks for it from the Solibri terms library. If one is found, it is evaluated. Now, mainly terms related to accessibility are implemented.

6.2.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this compliance-checking microservice and the relevant technical requirements defined in D4.1 is given in Table 19.

Table 18. Solibri - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
61	The system should provide the ability to perform BIM checks using a proprietary check definition, as there is no standard available.	The Solibri Integration allows for the performing of proprietary geometric checks (as described above) without the ACCORD cloud architecture.

6.3 AC(CO2)RD whole life carbon assessment microservice (Finland, Estonia)

6.3.1 General Microservice Description

AC(CO2)RD is a Proof-of-Concept microservice that automatically extracts information from the IFC model and relevant environmental databases, combines them, and generates a Life Cycle Assessment (LCA) report without requiring user interaction. For the project purpose, only the Global Warming Potential (GWP) category is used, and the results are expressed in $kgCO_2e$. The process is designed to overcome barriers caused by insufficient or mismatched data in the model and/or databases by automatically generating conservative assumptions and providing feedback to the designer on areas of the model that can be improved.

6.3.2 Microservice Implementation and Functionality

The AC(CO2)RD microservice is an open-source Python tool developed on the ifcopenshell platform, including ifctester and ifcopenshell.validate modules for the input validation.

The following inputs are used by the tool:

- STEP Physical file (IFC 2X3 to IFC 4.3) together with optional additional information in JSON format,
- JSON file with material quantities and additional building information

The AC(CO2)RD microservice can retrieve LCA information about the building materials and products from different databases using HTTP calls:

- Finnish Emissions database for construction via CO2data service at co2data.fi,
- ILCD+EPD standard databases such as environdec.com.

In addition to responding to compliance checks via the results API, the tool has the following file based output options in JSON or Microsoft Excel:

- LCA report (minimum report with the overall whole life carbon footprint of the model, standard LCA report according to EN 15978 (Sustainability of Construction Works), or full report with all the calculation results of each LCA stage by material, by type and by entity) in tonnes of CO_2e,
- Material report (bill of materials, or material flows report including the recycling rates and recycled content) in tonnes, square meters or cubic meters of each material,
- Environmental declaration according to the Finnish Decree on Climate Declaration in $kgCO_2e$/sqm of heated area.

The microservice is based on two basic principles: (1) The calculation shall be fully automated and (2) in the case of any uncertainties, the most conservative results shall be produced. The specific mechanisms how this is achieved are described in the following sections:

Selection of the appropriate entities: All the model data (all entities represented by 3D geometry) may not apply to the LCA calculation. For instance, the building regulations are restricted only to certain product groups. The conservative approach is to calculate the impact of each entity unless it can be explicitly excluded from the assessment. The exclusion of certain IFC entities can be specified in the accompanying JSON file.

Missing information about the material*: I*n certain cases, the entities may not have associated materials, or the material names may be ambiguous (“Undefined”, “Default”, etc.). The tool assumes conservative values of CO_2 emissions for such entities as the highest possible value in the database for a certain product class.

Material not matching the records in the environmental database*:* In a common design practice, the only information about the material in the permitting phase is its name assigned by the designer, the BIM authoring tool, or its template. This name is typically not pointing directly to any specific record in the environmental database. The tool, therefore, uses an extensive mapping dictionary of common material names linked to the Finnish national database. If the material name is not in the dictionary, the tool automatically uses conservative emissions for this material.

Missing information about the quantities*:* Environmental data is typically related to a certain unit flow, for instance, mass, volume, surface area or just the number of products. The relevant quantities must be recognisable from the model. The tool mainly relies on quantities stored in the IFC file, but if the information is not found in the model, a thorough search of all element properties is performed. The last option is to calculate the quantities directly from the model geometry to ensure a conservative result is delivered in any situation. The tool uses bounding boxes of the elements to optimise the performance of the calculation, but other methods (such as tetrahedral meshing) may be implemented in the future if they ensure conservative results and reasonable computational time.

Missing entities: During the permitting stage, verifying whether the designer has provided a digital model of all components required for the compliance check is impossible. Therefore, in this case, the tool cannot guarantee the conservativeness of the calculated results.

6.3.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this compliance-checking microservice and the relevant technical requirements defined in D4.1 is given in
Table 19.

Table 19. Finnish LCA Calculator - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
64	The system must provide a microservice supporting Lifecycle Assessments for Green Building Certification.	AC(CO2)RD microservice supports LCA for Finnish climate declaration compliance and for general certification purposes.
129	The system's compliance-checking microservices must allow the extraction of data from national databases and/or check against it.	The Finnish national database is implemented, including the mapping dictionary. International ILCD+EPD data can be accessed only using a specific GUID in the model.
131	The system must integrate with a microservice to check building CO2 compliance.	AC(CO2)RD rule checker is a Python script generating a specific AC(CO2)RD report (for the Finnish Climate Declaration) and comparing the overall carbon footprint to the given limit.

6.4 Clearly.BIM integration (Estonia)

6.4.1 General Microservice Description

Clearly.BIM is a user-friendly online solution for storing, viewing, sharing, querying, and checking BIM models in IFC. It offers a scalable solution for whoever works with BIM models: municipalities, contractors, the fire department, building owners, housing corporations, etc. Clearly.BIM's main capabilities are:

- Storing IFC models in an online database (fulfils the Data Storage Component)
- Viewing and interacting with the model in detail.
- Sharing the BIM models within an organization or between organizations.
- Streaming BIM models in 3D Tiles to be used directly in 3D City Models or other 3D visualizations.
- Exporting BIM models in various formats.
- Executing IDS checks. (Model and Data Requirement Validation Component and subject to the Orchestrating Microservices Component)
- Executing compliance checks for Building Code or Zoning Plan rules (subject to the Orchestrating Microservices Component)
- Publishing check results in BCF.
- Clearly.BIM is fully API based. All functionalities are also available through OpenAPI and GraphQL endpoints and can be easily integrated in other software systems. Through these APIs Clearly.BIM is subject to the Process Execution Component and Orchestrating Microservices Component

In the ACCORD project and in the integration with the ACCORD Architecture, Clearly.BIM is used through API and less through its UI.

6.4.2 Microservice Implementation and Functionality

For the Estonian use case in ACCORD, Clearly.BIM has been mainly used for storing BIM models and executing various checks from the Estonian Building code. These functionalities have been implemented through API in:

- The Estonian Building Registry (EHR / e-ehitus)
- The draft Process Execution Component
- The draft Orchestrating Microservices Component

The general flow that Clearly.BIM executes in this use case is as follows:

- A BIM model is uploaded by the end user (e.g. a permit applicant) directly in the Clearly.BIM web application and/or in the external interface (in this case of the Estonian Building Registry) as an IFC file.
- Through API this IFC file is sent to and imported in the Clearly.BIM database, which is based on BIM.works.
- From the database, the model is directly viewable and queryable in the Clearly.BIM web application and/or external interface.
- Through API, both IDS and compliance checks (building code or zoning plan) can be initiated.
    - Many different checks were configured, but the initiation process is the same.
- The results of the checks are provided in both JSON and BCF.

6.4.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this compliance-checking microservice and the relevant technical requirements defined in D4.1 is given in Table 20.

Table 20. Clearly.BIM - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
61	The system should provide the ability to perform BIM checks using a proprietary check definition, as there is no standard available.	Checks are implemented using the BIM Check Language (BCL) in the BIM.works backend. This is a proprietary language, as there is no (open) standard for this yet.
62	The system should be able to extract building- and spatial information from IFC files and visualize the results.	Clearly.BIM can extract both building and spatial information from IFC files. This information is also used in the actual checks and check results are visually represented in the model viewer using the related elements.
63	The system should be able to extract information from IFC files and check it against building environmental data from national Digital Twin if IFC files are georeferenced.	Clearly.BIM can extract information from the IFC, both on the model level and on an object (within a model) level. Currently the compliance checks do not support using external information from e.g. a national Digital Twin as a variable in the check. It is, however, possible to use information extracted from the IFC in externally run checks.
67	The system should use open data formats for delivering results of	Clearly.BIM outputs result in JSON and BCF (buildingSMART open standard).

	calculation software used for finite element analysis.	
68	The system's checking of compliance with urban regulations must be automated. The permitting process will be applicable to both public and private buildings during the design and construction phases, using BIM and GIS as data input.	If those urban regulations are configured into a compliance check in Clearly.BIM, the whole process can be automated. It can be applied to both public and private buildings. As described with requirement 63, using dynamic GIS input in those checks is not yet possible.
69	The system must allow to integrate GIS- and BIM data to carry out checking against urban regulations using one unified model.	See 63.
70	The system must be able to automatically determine the regulations to be checked against based on building criteria.	Clearly.BIM compliance checks and IDS checks can be configured as such that applicability of requirements/regulations can be automatically determined. E.g. executing a check only if the building is a hospital.
72	The system's spatial checking rules definitions should use estabilished vocabularies and query languages (like GEOSPARQL).	Clearly.BIM uses standarized IDS definitions for IDS checks. It uses JSON for compliance checks definitions.
75	The system must provide a microservice supporting compliance checks of timber construction systems.	Checks for timber construction systems can be configured using the BIM Check Language. Individual checks can then be started through API as a microservice.

6.5 Eurocode verification microservice (UK)

6.5.1 General Microservice Description

This microservice has been developed to perform structural compliance checks on building construction elements in accordance with the Eurocodes, a comprehensive set of design standards that govern the structural integrity of the built environment throughout most European countries. Its implementation leverages structural-analysis-related IFC data and requires that structural analysis model results be available for each applicable verification check.

In the absence of such analysis results, the specialized open-source tool IFC2CA[23] can be utilized to conduct the structural analysis of IFC building models using the Finite Element Method (FEM). The FEM engine employed is Code_Aster[24], a robust finite element solver developed by EDF (Électricité De France) and validated in the nuclear sector. The tool operates on the IFC data and provides three primary high-level functionalities:

- Mesh Generation: Creation of a discretized numerical model (mesh) for a given IFC

[23] https://github.com/Jesusbill/ifc2ca/

[24] https://code-aster.org/

structural analysis model.

- Analysis Execution: Generation of a Code_Aster-specific input file incorporating the model properties, load specifications, and configuration parameters necessary for the analysis.
- Data Enrichment: Enhancement of the original IFC model with corresponding analysis responses, including internal forces, displacements, and reactions for each associated IFC entity.

These enriched IFC result data are then employed to carry out structural design verifications in full compliance with the relevant design standards.

6.5.2 Microservice Implementation and Functionality

The microservice is built around an API developed in Python and integrated with the ACCORD Results API[25]. This integration enables its use with the core components of the ACCORD system via Integration Strategy A, using the Eurocode-based Ruleset.

The required input for this service comprises an IFC file, accessed through the Data Storage Component, and the GlobalId of a structural analysis model that contains the necessary analysis results. Upon processing these inputs, the system generates a comprehensive data dictionary for each structural member within the model, indexed by its GlobalId. This dictionary encapsulates all verification parameters needed for every applicable check.

Designed to streamline the validation process, the API facilitates a standardized workflow that includes:

- Target Type Applicability Validation: Confirming that a specific target type, as a feature of interest, is applicable to a given structural member.
- Verification Parameter Check: Ensuring that all required verification terms have been computed.
- Compliance Check Generation: Producing the identified structural compliance verifications based on the enriched analysis data.

It is important to note that the implementation relies on predefined property sets that include properties for materials, cross-sections of uniform members and required design coefficients for structural verifications. The existence of these property sets is checked in the IDS validation phase, but the values themselves are not part of the current Eurocode-based Ruleset. Nonetheless, all required data are captured within the IFC file with applicable templates, and this setup can be readily extended for such checks via the Integration Strategy A.

6.5.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this compliance-checking microservice and the relevant technical requirements defined in D4.1 is given in Table 22.

Table 21. Eurocode Calculator - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail

[25] https://github.com/Accord-Project/UK-Demo-Microservice/

65	The system must support the integration of a Finite Element Analysis compliance checking microservice.	The service is based on the use of FEM results and compliance data in IFC schema
66	The system must link BIM data inputs with finite an element analysis tool to facilitate automated checks.	The results from a FEM analysis, based on an IFC structural model, are integrated into the IFC schema, ready to be used for verifications
67	The system should use open data formats for delivering results of calculation software used for finite element analysis.	Indeed, the FEM engine is open-source, and all non-IFC-related data are based on open data formats. Computation-needed resources (scripts, meshes) that allow regenerating calculation results are also provided. However, the verification checks are based solely on IFC data.

6.6 Land Use Checking microservice (Germany)

6.6.1 General Microservice Description

The purpose of this service is to automate the compliance checks of land use regulations applicable to buildings in Germany. It can be used in scenarios where land use regulations are specified in the XPlanung format, and building designs are stored in formats convertible to CityGML 2.0 or 3.0 regarding their spatial design and specification of uses of spaces. An example of such a format is IFC, also commonly used in Germany for Building Information Modelling.

XPlanung is an open, XML-based data exchange format based on Geography Markup Language Version 3 (GML 3.2.2), the extensible standard for spatial data developed by the Open Geospatial Consortium (OGC) and ISO TC211. The CityGML standard defines a conceptual model and exchange format for the representation, storage and exchange of virtual 3D city models. The standard provides a framework for integrating, storing, and exchanging 3D geospatial data encoded in GML/XML.

Transformation of BIM data in IFC format into CityGML 2.0 is implemented as a custom FME flow.

6.6.2 Microservice Implementation and Functionality

This service transforms XPlanung and CityGML models into RDF graphs and topology relations between their geometries are materialised using predicates of the GeoSPARQL ontology. Materialised topological relations are equivalent to outputs of GeoSPARQL topological functions with sets of 2D and 3D geometries transformed from XPlanung and CityGML provided as arguments. Materialisation of topology relations between geometries in both models serves as a link between two models in one RDF graph, making both models semantically interoperable, as shown in Figure 16.

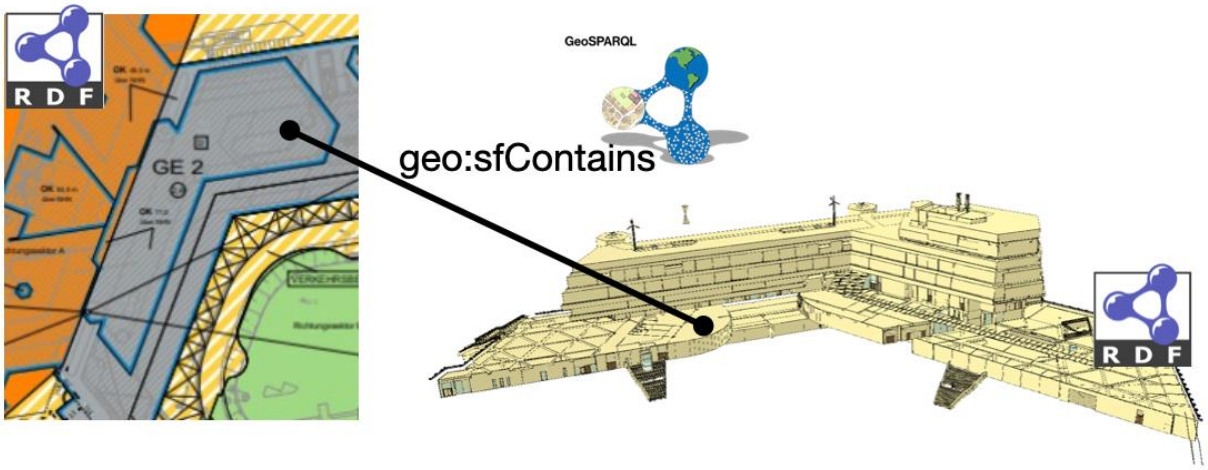


Figure 16. Relationship between IFC Data and XPlanung data.

SPARQL enables querying information from databases or any data source that can be mapped to RDF. The SPARQL standard is designed and endorsed by the W3C and helps users and developers focus on what they would like to know instead of how a database is organised. This service makes use of those capabilities of the query language to retrieve geometries of parts of a building that are topologically related to land parcels' geometries, and check using the use of more semantically rich descriptions of such building geometries whether the building complies with the requirements of the XPlanug.

The service is implemented in Java using Spring Boot framework. The information flow, illustrated in the activity diagram (Figure 17), is as follows:

- Upon initialisation, the service listens to HTTP Post requests.
- Upon receiving a service description request, it returns a description of its functionality and the numbered list of implemented rules it is capable of checking in HTTP response.
- Upon receiving the request to check all rules, it sends SPARQL queries, corresponding to every implemented check, to GraphDB and returns a list of passed and failed checks when all queries are finished, in HTTP response.
- Upon receiving the request to check one or more specific rules, it sends SPARQL queries, corresponding to each check from the list, to GraphDB and returns a list of passed and failed checks when all queries are finished, in HTTP response.

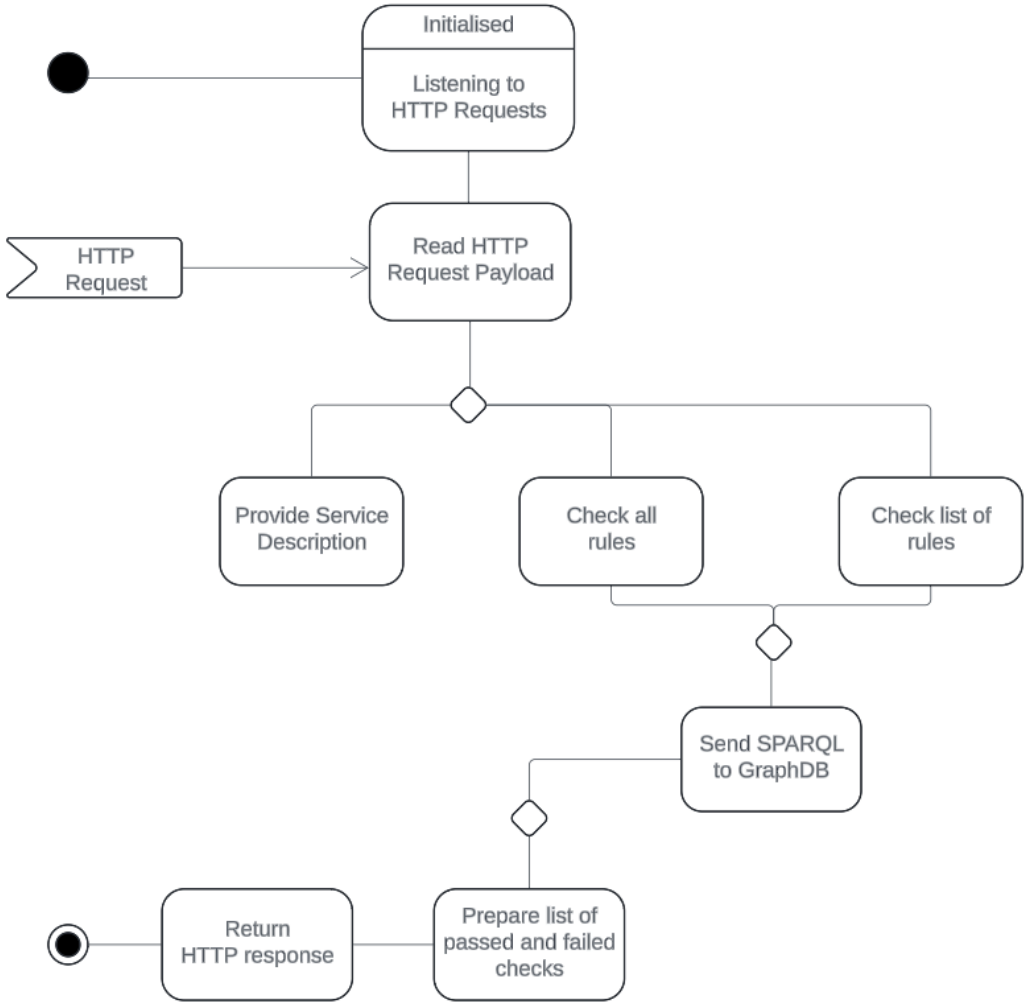


Figure 17. Land Use Checking Service Implementation.

A comparison of the implemented functionality of this compliance-checking microservice and the relevant technical requirements defined in D4.1 is given in Table 22.

Table 22. Land Use Checking - Comparison against Technical Requirements

TR No	Requirement Description	Implementation Detail
69	The system must allow to integrate GIS- and BIM data to carry out checking against urban regulations using one unified model.	GIS and BIM data are transformed into RDF/Turtle format and uploaded to a dedicated knowledge graph operated by Ontotext GraphDB. The details of the organization of such knowledge graph are given in ACCORD deliverable D2.4 (here). GIS and BIM data are thus

		available for unified querying with SPARQL.
71	The system must allow local authorities to check submitted BIM models against land use requirements provided in XPlanGML format.	This microservice implements checking of IFC data against XPlanung requirements, meeting all these TRs.
73	The system must be able to extract building- and spatial information from IFC-files (and CityGML, if applicable) and check against requirements provided in the standardised data format XPlanGML in the German context (and INSPIRE PLU in European context).	
74	The system should be able to extract information from IFC-files being required for formal building permit applications and convert it into the standardised data format XBauXML in the German context.	This requirement was not, in the end, required. As the process of converting IFC to CityGML and then to RDF allowed the problem to be solved without this more complex step.
128	The system must read specific attribute values from OGC services (originated from INSPIRE PLU data).	This was not required in the final implementation as all required data can be retrieved from the XPlanung data files.

6.7 Environmental compliance microservice (Germany)

6.7.1 General Microservice Description

This microservice facilitates the checking of environmental compliance with the DGNB green building certification requirements. It builds on a service setup using web-based building LCA tools that create and submit an XML building LCA model to DGNB direct submission API for compliance checking and performance point calculation based on DGNB requirements on building LCA models as defined in the DDNB requirement specification DNG 1.1.

The model can build on IFC but needs to be translated to XML and to be enriched to be checked by DGNB. The XML schema definition and a test API are provided by DNGB upon request.

6.7.2 Microservice Implementation and Functionality

The microservice builds on the DGNB API for environmental compliance checking, point calculation and result submission. The requesting agent is a building LCA web software such as Generis®, where users can create building LCA models aiming at environmental compliance with DGNB certification requirements. The models are directly submitted via API (as specified by DGNB; XSD, API link and documentation are available upon request) and checked against LCA calculation rules, model consistency, and environmental performance achievement.

Within the Generis® microservice, the following services are provided:
- Model/Data requirement validation through quality and compliance pre-checking routines
- Building LCA model creation and manipulation

- Data storage on building and construction level
- Data export and direct model submission via API

6.7.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this compliance checking microservice and the relevant technical requirements defined in D4.1 is given in Table 23.

Table 23. German LCA Calculator - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
64	The system must provide a microservice supporting Lifecycle Assessments for Green Building Certification	This is implemented through Generis®' base functionality. LCA models can be built in compliance with DGNB, BNB, BREEAM and QNG, and model creation is supported by various building model imports and internal pre-checking routines on compliance.
129	The system's compliance checking microservices must allow to extract data from national databases and/or check against it.	This is implemented in Generis® building on the ÖKOBAUDAT API that allows to involve environmental data on product level for building level LCA[26]
131	The system must provide integration with a microservice to check building CO2 compliance.	This is implemented through the following workflow. LCA models are created (optionally using IFC) in Generis®, and a XML building on the schema definition by DGNB (version 2.0) is created. The XML can then directly be checked via API call or manual checking routine over the DGNB API, receiving compliance and performance results (certification points).

6.8 Timber Construction Compliance Checking Microservice (Germany)

6.8.1 General Microservice Description

The *Timber Construction Compliance Checking Microservice* is designed to assist professionals in ensuring that timber construction projects comply with relevant building regulations and standards. The microservice leverages BIM data and integrates with external tools to perform comprehensive compliance checks.

The key feature of the microservice is a *Timber Construction Compliance Checker*, which verifies compliance with specific timber construction regulations. The prototype is developed on the example of timber panel construction in German building classes 4 and 5, for which the German Model Timber Construction Building Code ("Musterholzbaurichtlinie") applies. As an outlook, the legal repository will integrate Eurocode 5, ensuring that timber construction projects meet European standards. The prototype includes an *Information Requirements* component, which ensures that the BIM model meets the necessary information requirements (IRs) provided in LOIN (required for German federal buildings) and

[26] https://www.oekobaudat.de/en/guidance/software-developers.html

IDS formats. The microservice allows for integration with other external tools for specialized compliance checks, and its API integration exposes functionalities for seamless integration with other systems.

6.8.2 Microservice Implementation and Functionality

The microservice is implemented using Python and Django, a Python web framework, for deployment on a server. The inputs for this microservice include IRs, BIM models and construction-related data. The microservice is integrated in the ACCORD Cloud Architecture via the Integration Strategy A.

The microservice's *Timber Construction Compliance Checker* component verifies that the building project complies with timber construction regulations, using data from IFC models, "Dataholz.eu" constructions, and project-specific data. API endpoints provide functionalities for initiating compliance checks and retrieving results, which were developed using a Python-compliant framework for efficient request handling. Additionally, the microservice integrates with external tools, sending API requests to these tools and consolidating the results.

The workflow of the microservice begins with the service sending an API request to the BIM Portal Germany to assess the IRs in LOIN format, applying for the relevant German federal state or federal building owner. The *Information Requirements* component allows the mapping of those IRs with those applying to selected timber construction regulations and the export of the dataset. The microservice allows for both, external model validation, such as checking solutions provided via the BIM Portal Germany, and internal checking. For both, initiating validation checks and final compliance checks, the service sends an API request to the *Timber Construction Compliance Checker* tool. The tool processes the data and performs the compliance checks, sending the result back to the service. Finally, the service consolidates the results and sends them back to the tool's UI, which displays them to the user.

6.8.3 Comparison against Relevant Technical Requirements.

A comparison of the implemented functionality of this compliance-checking microservice and the relevant technical requirements defined in D4.1 is given in Table 24.

Table 24. Type Approval - Comparison against Technical Requirements.

TR No	Requirement Description	Implementation Detail
75	The system must provide a microservice supporting compliance checks of timber construction systems.	This requirement has been fulfilled.
129	The system's compliance checking microservices must allow to extract data from national databases and/or check against it.	The microservice ensures that the technical requirement is met by leveraging data for IRs in LOIN format for federal buildings via the BIM Portal Germany. Additionally, it references construction data from the database "Dataholz.eu" and other sources.

7 Conclusions

This deliverable has documented the results of ACCORD's Tasks 4.3 and 4.4.

Specifically, the outputs of the work are:

- Documentation of the results of developing the core ACCORD compliance checking components, describing what has been newly developed and what has been reused from existing components or open-source tools.
- Documentation of the results from developing compliance checking microservices.
- Documentation of the ACCORD APIs including those that are re-used and newly developed.
- Report the results of microservice implementation and integration outlined in the ACCORD Service Setups.

In the upcoming tasks of ACCORD WP4, these outputs will be key in (a) performing quality assurance across the ACCORD cloud architecture and (b) producing technical documentation for external audiences. In the context of the wider ACCORD project, the core compliance checking components, as well as the microservices, will be used to deliver the demonstration projects, as defined in the ACCORD service set-ups described in this deliverable.

References

[D2.2] ACCORD D2.2 BCO Ontology and Rules Format
Available at http://accordproject.eu/wp-content/uploads/2024/02/ACCORD_D2.2_BCO_Ontology_and_Rules_Format.pdf
(Accessed 24 January, 2025).

[D2.3] ACCORD D2.3 Rules Toolset

[D2.4] *ACCORD D2.4 Semantics Documentation*.
Available at: https://docs.accordproject.eu/#graphdb/#graphdb-as-the-storage-for-regulation-graphs
(Accessed 14 January, 2025).

[D4.1] *ACCORD D4.1 Technical Requirements Elicitation, Analysis and Cloud Architecture Model*.
Available at: https://accordproject.eu/wp-content/uploads/2024/03/ACCORD_D4.1_Technical-Requirements-Elicitation-Analysis-and-Cloud-Architecture-Model.pdf
(Accessed 24 January, 2025).