Automation in Construction 176 (2025) 106226

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

AEC Co-design workflow for cross-domain querying and reasoning using Semantic Web Technologies

Diellza Elshani [a,*], Alessio Lombardi [b], Daniel Hernandez [c], Steffen Staab [c,d], Al Fisher [b], Thomas Wortmann [a]

[a] Institute for Computational Design and Construction, Department for Computing in Architecture (ICD/CA), Faculty of Architecture and Urban Planning, University of Stuttgart, Stuttgart, Germany
[b] Buro Happold, London, United Kingdom
[c] Department for Analytic Computing (AC), Institute for Artificial Intelligence, University of Stuttgart, Stuttgart, Germany
[d] Electronics and Computer Science, University of Southampton, University Road, Southampton, United Kingdom

ARTICLE INFO

Dataset link: https://doi.org/10.18419/darus-4360

Keywords:
Data integration
Co-design workflow
Ontology
Knowledge graphs
Tool integration
Federated querying

ABSTRACT

The Architecture, Engineering, and Construction (AEC) industry faces data integration challenges due to fragmented silos and diverse data representations, hindering cross-domain queries and early detection of design constraints. Semantic Web Technologies (SWTs) address data integration challenges.

This paper evaluates the impact of SWTs on co-design workflows by comparing them with alternative approaches to assess their effectiveness in supporting interdisciplinary collaboration and design constraint detection. Using Design Science Research, a co-design methodology is developed that integrates SWTs with AEC tools for reasoning and federated querying. A component of this methodology is a bidirectional mapping strategy for translating object-oriented data models, demonstrated with the Building Habitat Object Model (BHoM), an AEC interoperability framework.

Findings reveal that integrating SWTs enables reasoning and complex queries across federated datasets, improving co-design efficiency. These findings support AEC professionals in advancing co-design and data-driven decision-making, while also informing future research on integrating SWTs into AEC design workflows.

1. Introduction

The integration of digital technologies in the Architecture, Engineering, and Construction (AEC) industry has primarily focused on specific phases of the building process or individual disciplines [1]. While this approach has driven advancements in isolated areas, it often results in fragmented workflows that hinder interdisciplinary collaboration. Co-design is a collaborative methodology that facilitates the simultaneous and feedback-driven development of design methods, analysis techniques, manufacturing processes, and building systems across disciplines [2]. In a co-design process, it is crucial to integrate design data, analysis methods, and relevant information from multiple disciplines, as constraints in one domain influence decisions in another [3]. However, the focus on isolated phases or disciplines frequently leads to data being compartmentalized within distinct silos, which hampers effective co-design [4]. The challenge of achieving seamless data integration across disciplines remains substantial due to the varied tools and applications employed, each with its own modeling and information representation methods [5–8].

The absence of an integrated overview of project data hampers AEC project stakeholders' ability to make logical inferences, conduct cross-domain queries, and derive information from all involved disciplines, thereby impeding their effectiveness in identifying design restrictions. Cross-domain querying refers to the ability to retrieve data from different disciplines or domains, allowing queries across them. Current AEC practice often considers limitations arising from disciplinary design or construction only in the later stages of a linear design process. However, research has shown that the most critical decisions in building design are made during the conceptual design phase [1,9,10]. These early decisions profoundly impact not only construction costs [9] but also the subsequent energy use of the building [10].

Several solutions have been proposed to address data integration challenges in the AEC industry, including common data schemas such as the Industry Foundation Classes (IFC), collaborative interoperability frameworks, and Semantic Web Technologies (SWTs). IFC, an international standard, facilitates data exchange across AEC software applications throughout various building lifecycle stages [5,11]. However, it

* Corresponding author.
E-mail address: diellza.elshani@icd.uni-stuttgart.de (D. Elshani).
URL: https://www.icd.uni-stuttgart.de/team/Elshani (D. Elshani).

https://doi.org/10.1016/j.autcon.2025.106226
Received 5 January 2025; Received in revised form 9 April 2025; Accepted 19 April 2025
Available online 14 May 2025

primarily serves as a standardized reference rather than a bidirectional exchange mechanism, often resulting in data loss due to software inconsistencies [12]. Collaborative interoperability frameworks, such as the Building and Habitats Object Model (BHoM) [13] and Speckle [14], support bidirectional data flows and visual data flow modeling, facilitating interdisciplinary exchange. Visual data flow modeling represents operations as nodes linked by data flows [15]. However, these frameworks rely on mapping rather than direct data linking [16], limiting integrated data analysis and cross-disciplinary queries. Additionally, their imperative coding approach complicates logical inference and declarative reasoning [17]. SWTs, including the Resource Description Framework (RDF) and the Web Ontology Language (OWL), enable decentralized data representation, machine-readable semantics, and complex reasoning [18,19]. Several studies have explored their potential in the AEC industry [1,19,20]. However, SWTs remain under-integrated with AEC design tools, particularly visual data flow modeling systems [1,21], and their impact on workflow efficiency and holistic data utilization remains unexplored.

In addressing these challenges, we have previously undertaken several foundational research efforts, including a review of interoperability paradigms in the AEC industry [1], the preliminary translation of BHoM's terminological layer to OWL [16], and explorations of semantic reasoning within BHoM building data [3,22]. Building on this work, this paper develops and evaluates a co-design workflow that integrates SWTs with collaborative interoperability frameworks to enhance cross-domain querying and reasoning. While several studies have explored the integration of SWTs into AEC workflows [23–25], they mainly address feasibility rather than providing a systematic evaluation. This research compares the developed co-design workflow against alternative approaches to assess its impact on cross-domain querying, design constraint identification, collaboration, and holistic data use. To support this evaluation, we developed several key components:

1. A co-design workflow that integrates SWTs with collaborative interoperability frameworks, enabling cross-domain querying and reasoning to enhance data interoperability in AEC projects;
2. A mapping strategy for the bidirectional translation of object-oriented data models used by collaborative interoperability platforms — aligned with visual data flow modeling principles — into OWL/RDF formats and vice versa;
3. A comparative evaluation of the workflow's effectiveness in representing semantic and disciplinary data, managing class definitions and extensibility, supporting cross-domain querying, and synchronizing data, in contrast to alternative approaches. This assessment quantifies the impact of SWTs in AEC contexts, reinforcing their practical value beyond previous implementations.

This paper follows the Design Science Research (DSR) methodology to develop and evaluate a co-design workflow [26], integrating SWTs with AEC design tools. The artifact was developed with continuous feedback from our industry partner, Buro Happold, and evaluated in workshops to ensure it met practical requirements. Core co-design requirements were defined, guiding the development of use cases and a translation mechanism. A comparative evaluation against alternative approaches assessed its ability to manage semantic and disciplinary data, support cross-domain queries, and detect design constraints.

The paper is structured as follows: Section 2 reviews AEC data schemas, interoperability standards, and data exchange approaches in the AEC industry. Section 3 identifies the research gap and objectives. Section 4 outlines the methodology for developing the co-design workflow by defining requirements and falsifiable use cases, and by introducing bidirectional translation between object-oriented data models and OWL/RDF. Section 5 presents a comparative case study of three co-design approaches: (1) without an interoperability framework, (2) with an interoperability framework (BHoM), and (3) the proposed co-design workflow using both SWTs and BHoM. Section 6 evaluates the workflows based on the criteria from the methodology. Section 7 discusses limitations and future directions, and Section 8 concludes the paper.

2. Background

AEC projects are inherently decentralized [27], requiring multidisciplinary solutions [28]. Different disciplinary designers employ various design tools, each with its own schema [27]. This diversity extends to modeling approaches among architects, engineers, and subcontractors [6,8], posing challenges like fragmented collaboration and data interoperability issues [29,30]. Integrating knowledge representations in AEC projects remains a significant challenge [31], but SWTs offer a promising solution [32,33]. While various data schemas and interoperability standards have been established to support information exchange, limitations in consistency, bidirectional data flow, and cross-domain reasoning persist [34]. This section explores the aspects of data exchange and interoperability in AEC. We begin by discussing AEC data schemas and interoperability standards, like IFC and other structured frameworks that facilitate digital data exchange. Next, we examine approaches to co-design and data exchange, presenting workflows that rely on multiple data models versus shared data models, highlighting their scalability and integration challenges. We then introduce SWTs, which enable structured, machine-readable representations of data, allowing for improved linking, querying, and reasoning across disciplines [33]. By understanding these approaches, we establish the basis for evaluating how SWTs can enhance cross-domain querying, reasoning, and workflow automation in AEC co-design processes.

2.1. AEC data schema and interoperability approaches

IFC, an international standard (ISO 16739-1:2018 [35]), is a data schema that supports the exchange of digital data between heterogeneous applications [36]. Based on the EXPRESS data model, IFC schema allows encoding in various formats such as XML, JSON, STEP, or ifcOWL [1]. The IFC data model exhibits a well-defined graph structure comprising multiple entities, types, and associated data properties, covering concepts from all disciplines involved in an AEC project [25,37]. While IFC serves as the industry standard for data schema, complementary standards like ISO 19650 [35], EIR Guidance [38], and DIN SPEC 91400 [39] play significant roles in enhancing data interoperability and data exchange within the AEC industry, synergizing with the research's objective of refining co-design workflows.

2.2. Approaches to co-design and data exchange in AEC

In this subsection, we discuss data exchange approaches in AEC that facilitate collaborative design. These include: (1) exchange based on multiple data models, where the data interchange between each pair of tools differs from how data is exchanged between other pairs of tools; and (2) exchange based on a shared data model, such as IFC-based data exchange and visual data flow modeling workflows like BHoM and Speckle. In this approach, all tools exchange data with the same shared data model. The data of each tool has a representation on the shared data model, and the representation of the data of different tools can overlap. Thus, tools can exchange data using this representation overlap.

2.2.1. Exchange based on multiple data models

This approach facilitates data exchange by establishing custom mappings between pairs of tools. By supporting both proprietary data models and APIs, it enables interoperability across diverse software applications. In scenarios where multiple tools are involved, to exchange data between two tools using different data models we must use an exchange data model (which usually differs from the data models used by both tools). The exchanged data can be both proprietary data models via file transfers and APIs [1,40]. For instance, a workflow involving Rhino, Robot Structural Analysis, and ArchiCAD, where each tool exchanges data with another tool using different formats or data models, demonstrates this method. For example, ArchiCAD and

Rhino/Grasshopper communicate through the ArchiCAD-Grasshopper connection, while Rhino and Robot Structural Analysis share data via exporting and importing files in DXF and CSV formats. While effective for specific workflows, this approach does not scale well as the number of tools increases. Managing multiple direct connections becomes increasingly complex, leading to interoperability challenges. Additionally, this approach can lead to fragmented workflows, potential data loss, and synchronization challenges [40].

2.2.2. Exchange based on a shared data model

In this approach, data exchange between tools is structured around a shared data model. This model can have a unified representation for all disciplines or a modular approach with discipline-specific data representations. A key example of a unified representation is IFC-based exchange, where tools map their internal schemas to the IFC standard. Widely used in the AEC industry, IFC facilitates interoperability across software and collaboration throughout project lifecycles. It supports direct software exchanges (e.g., Revit to Tekla), database management (e.g., IFC-to-SQL with PostgreSQL), and linked data integration (e.g., GraphDB) [11]. However, maintaining a single data representation requires consensus among participants. While the IFC schema restricts modifications, custom properties can be added through IfcProxy entities and property sets (PSet) [41]. However, IFC prevents internal class references, though the buildingSMART Data Dictionary (bSDD) API provides an alternative for defining custom data structures. At the same time, to address the rigidity of a unified model, a modular approach within IFC allows tools to adopt discipline-specific models [40,42]. Discipline-specific IFC files can be exchanged independently or linked to accommodate diverse requirements [43,44]. However, IFC primarily serves as a standardized reference rather than a bidirectional exchange mechanism, often leading to data loss. Studies have shown that data loss can occur during IFC-based transfers due to inconsistencies in software implementations [11]. For instance, Sibenik et al. [12] observed variations in exported entity counts and missing elements across different software, highlighting challenges in maintaining data integrity [11,45]. In current AEC practice, an increasing portion of design work relies on parametric design and visual programming tools [15,46]. Literature highlights the growing use of visual data flow modeling in the AEC industry to integrate technologies beyond traditional design tools, including SWTs [47]. As demonstrated by Reiss and Renieris [48], visual data flow modeling provides an effective means of working with dynamic data—information that undergoes frequent changes and updates throughout the design process [49]. However, integrating IFC into these workflows remains challenging, as IFC serves more as a structured reference than a dynamic exchange platform. Frequent mappings between parametric environments and IFC can lead to data loss, as noted by Sibenik [12]. Several frameworks facilitate data exchange in visual programming environments via plugins, including Geometry Gym, VisualARQ, Rosetta, BHoM, and Speckle. This study examines BHoM and Speckle, as both enable bidirectional data exchange across multiple AEC software. As open-source platforms, they use internal data schemas to translate information between BIM, engineering tools, and visual programming interfaces [50]. Both platforms allow tool-specific data representation rather than enforcing a unified model across disciplines. Speckle provides 50 predefined classes and prioritizes software connectivity [1], while BHoM offers over 1251 classes with a federated data structure, supporting multiple representations of building elements [13]. This federated structure aligns with the multidisciplinary nature of AEC workflows, where different domains describe building components differently. BHoM also includes predefined functions for data manipulation [16]. However, neither maintains persistent cross-platform data links, limiting seamless multidisciplinary queries. This means information is sent from one platform to another, but it is not linked. The absence of a linked data representation complicates cross-disciplinary queries, making it difficult for designers to access information seamlessly from multiple disciplines. Therefore, both platforms can be complemented with external databases for more robust data storage and management, particularly in projects requiring extensive data retention or cross-platform data integration.

2.3. SWTs in the AEC industry

SWTs in AEC can provide assistance in three crucial aspects: (1) interoperability, as it can enhance the collaboration of various systems and software within the AEC sector; (2) linking data, as it excels in connecting information from different domains; (3) and logic and proof, as it can aid in logical reasoning and proof processes in AEC applications [33]. Key components of SWTs include ontologies and knowledge graphs. Ontologies provide a structured framework for domain-specific knowledge [18], while knowledge graphs link entities and relationships, enhancing data integration [51]. These are implemented using RDF [52] and OWL [18], which support semantic reasoning and structured data queries via SPARQL [53]. Existing AEC applications leverage linked data and ontologies like OWL [33,54–56]. Notably, ifcOWL represents IFC using OWL, addressing IFC schema limitations such as the lack of rigorous logic [54]. AEC ontologies also include LBD modular ontologies like BOT [57], Building Product Ontology [58], Brick Ontology [59], and OPM [56], which offer discipline- or domain-specific data representation, aligning with AEC's federated nature. Ontologies and knowledge graphs enable extensibility by expanding data dictionaries and representing evolving information [60]. Shen et al. emphasize the need for tighter integration between SWTs and AEC tools to maximize their potential [21]. Various research initiatives have explored the application of SWTs in AEC design workflows. For example, Cao et al. [61] demonstrate how ontologies can automate manufacturability analysis in industrialized construction. Pauwels et al. [62] provide an overview of SWTs in AEC, highlighting their role in structuring and linking data but also noting the challenges of integrating them holistically into industry workflows. Perisic et al. [63] propose the Extensible Orchestration Framework, which improves interdisciplinary co-design through SWTs. Additional efforts aim to link BIM data with semantic models. The BIM Semantic Bridge [64] facilitates semantic enrichment by connecting BIM models to ontologies, though its focus remains on ontology development rather than dynamic data integration. Other approaches explore graph-based and cloud-based BIM models to support real-time information exchange. For example, Wang et al. [65] propose a cloud-based BIM framework that structures domain-specific models as interlinked graphs, enabling automated inconsistency detection. Similarly, Cloud BIM (CBIM), introduced by Sacks et al. [23], incorporates knowledge graphs into federated BIM environments, enhancing semantic relationships between project components. However, these approaches lack comparative evaluations against other interoperability methods. Visualization tools such as the LD-BIM Web App and LBD-viz [66] facilitate interaction with linked data in BIM systems but only support reading BIM data and exporting RDF, without enabling the reverse process, which limits their ability to provide customization for co-design workflows.

Despite promising developments, existing research, to the best of our knowledge, does not quantitatively assess the impact of SWTs on design collaboration, workflow automation, or interdisciplinary reasoning. To fully leverage SWTs in co-design workflows, systematic evaluation is necessary to measure their effectiveness and practical benefits.

3. Research gaps and objectives

Interdisciplinary co-design in the AEC industry faces persistent challenges related to data integration, interoperability, and reasoning. Exchange based on multiple data models is inefficient when multiple tools are involved, as it requires direct communication between each pair of software applications, leading to increased complexity and potential data inconsistencies [40]. While IFC provides a standardized schema, it is not optimized for repeated import/export cycles, often leading to data loss [12]. Collaborative interoperability frameworks such as BHoM and Speckle facilitate data mapping across AEC tools, but they do not establish semantic links between information, limiting their

ability to support cross-domain queries and logical inferences across disciplines [1]. As a result, workflows rely on manual intervention, impeding knowledge-driven decision-making and reasoning in co-design. While SWTs offer a promising solution by enabling structured data integration and reasoning, their direct integration into AEC design software remains limited. Despite promising developments, existing research, to the best of our knowledge, does not quantitatively assess the impact of SWTs on design collaboration, workflow automation, or interdisciplinary reasoning. To leverage SWTs in co-design workflows, systematic evaluation is necessary to measure their effectiveness and practical benefits. This study identifies three key research gaps that need to be addressed:

- Gap 1: Semantic Representation and Cross-Domain Querying: AEC workflows require discipline-specific data representations rather than a single, uniform representation across all domains [28]. While modular approaches support discipline-specific representations, they often result in fragmented silos that hinder interoperability. Linking discipline-specific datasets to enable cross-domain querying and reasoning remains a challenge [16].
- Gap 2: Consistent and Scalable Data Exchange: Direct data exchanges between tools that rely on custom mappings between individual software applications do not scale well because each pair of tools requires a different mapping, which can lead to data loss and a lack of synchronization [12,40]. While SWT-supported workflows exist, their integration into AEC design software remains limited.
- Gap 3: Evaluation of SWTs in Co-Design Workflows: While studies explore SWT integration in AEC workflows [23], most focus on feasibility rather than systematic assessment of their impact on co-design efficiency, cross-domain querying, and collaboration.

To address these gaps, this study establishes three key objectives. First, it aims to enhance cross-disciplinary querying and extensibility by leveraging SWTs through linked data principles and knowledge graphs. This approach strengthens semantic representation, facilitates disciplinary integration, and improves extensibility, directly addressing the challenge of semantic representation and cross-domain querying. Second, it aims to develop a data exchange mechanism by developing a bidirectional translator between object-oriented building data models from AEC design software and OWL/RDF, ensuring synchronization and scalability, thereby addressing research gap 2. Third, it evaluates the effectiveness of SWTs in co-design workflows through a comparative assessment of an SWTs-integrated workflow against alternative approaches. This evaluation measures the impact on semantic representation, disciplinary data handling, class definition and extensibility, cross-domain querying, information inference, and data synchronization, directly addressing research gap 3. To achieve these objectives, this research develops a co-design workflow that integrates SWTs to enhance interoperability in AEC. The Methodology section defines the requirements necessary for its implementation, while the Evaluation section assesses its effectiveness through a comparative analysis of three co-design workflows.

4. Methodology: Integrating SWTs with design tools via data translation

This section outlines the methodology for developing a co-design workflow that integrates SWTs with collaborative interoperability frameworks to enable cross-domain querying and reasoning. A key component of this approach is a mapping strategy for bidirectional translation between object-oriented data models—aligned with visual data flow modeling principles—used by interoperability frameworks and OWL/RDF formats. The development approach is based on the principles of DSR and follows a systematic strategy to design, develop, and evaluate the artifact known as the co-design workflow. We applied the DSR methodology by developing the converter with constant feedback from our industry partner and evaluated it in workshops. The initial design phase of the artifact defines the requirements for co-design and develops use cases derived from the background section and industry collaboration feedback. The development phase focused on implementing the translation mechanism illustrated in Fig. 1. The following subsections present the requirements for co-design and use cases, followed by the bidirectional translation of data models.

4.1. Requirements to co-design and use cases

In line with the DSR methodology, based on the background section, research gap, and objectives, as well as in collaboration with the industry partner key requirements and falsifiable use cases for an integrated co-design workflow are outlined. These requirements address the gaps in current AEC data exchange practices, focusing on semantic representation, flexible class definitions, linked data for reasoning, and efficient data synchronization. The use cases presented herein will serve as evaluation criteria in Section 6.

- Requirement 1: Semantic Representation and Disciplinary Representation
  Enable semantic representation and allow different disciplines to model building elements based on their unique perspectives and requirements, granting freedom in schema design. Recognizing the diversity in the way disciplines define and represent building elements enhances flexibility and accommodates the varied analysis needs.
  - Use Cases for Requirement 1: (a) Architect represents a column by its location and other relevant properties; Structural engineer defines it as an analytical bar with start/end points and cross-section properties.
- Requirement 2: Class Definition and Extensibility
  Designers should be able to define new classes and extend existing ones, including the addition of new attributes/properties within the data dictionary, all within the design software. This allows designers to accommodate new project-specific concepts and enhance existing classes with project-specific attributes.
  - Use Cases for Requirement 2: (a) Extend existing classes, e.g., add simulation results as attributes to the "Bar" class; (b) Define a new class to represent elements not covered within the existing schema.
- Requirement 3: Linked Data for Reasoning, and Cross-Disciplinary Queries
  Enable linked data representation to facilitate cross-disciplinary queries, allowing designers to access information from multiple disciplines. Building elements should be depicted from disciplinary perspectives, and linked data enables holistic decision-making.
  - Use Cases for Requirement 3: (a) Determine optimal acoustic ceiling placement considering structural properties; (b) Decide machinery lab location based on column node reactions; (c) Optimize floor slab openings in the machinery lab, considering mesh bending moments.
- Requirement 4: Data Synchronization - Read back data from the graph database to the design platform.
  In a co-design process, AEC designers need to share information with each other and be able to read data from different disciplines involved in the project. This process should be smooth and ideally not involve many manual steps of importing and exporting information, where information loss can occur. With the approach

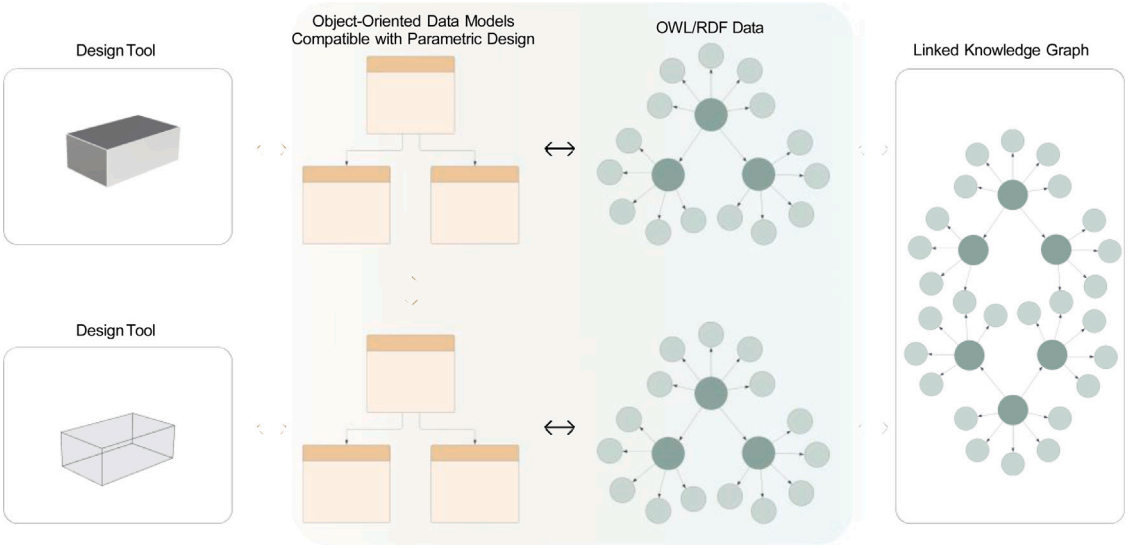

Fig. 1. Key component of the developed artefact.

using a graph, designers need to be able to read data from the graph to their software to ensure seamless integration and collaboration across different disciplines. This capability allows designers to use disciplinary tools for designing and analyzing options while having access to a linked data graph of the entire project. If changes occur in the graph, designers should be able to propagate these changes into their design platform.

- Use Cases for Requirement 4: (a) Upon determining the function of a room (e.g., as a machinery lab) by inferring or calculating data from a linked database, propagate the machinery lab label information from the graph into the design software.

4.2. Bidirectional translation of data-models

The translation approach converts object-oriented building models into OWL/RDF to enhance interoperability across diverse software tools. This involves mapping key object-oriented elements — classes, interfaces, properties, and datatypes — into corresponding vocabularies within SWTs. Fig. 2 illustrates this mapping process, showing the flow from an object-oriented data model to OWL/RDF. We use BHoM as an example of an object-oriented data model within an interoperability framework (See Appendix A for the BHoM Abstract Model) . While both BHoM and Speckle utilize such models, BHoM was chosen due to its (1) federated data approach, allowing multiple discrete representations of building elements, and (2) extensive set of predefined classes, reducing the need for new ontologies. This translation pattern applies to any interoperability framework employing object-oriented models, provided it can define object properties and functions separately, ensuring compatibility with SWTs and linked data principles. By generalizing this approach, interoperability across AEC design environments is enhanced. Existing research, such as Tong et al. [67,68], has mapped object-oriented database models to OWL/RDF, but existing converters posed challenges. First, BHoM, as an AEC design plugin, has specific development requirements. Second, deserializing OWL/RDF into BHoM and integrating it into design platforms proved complex. While research on integrating existing mappers could potentially offer a solution, the unique constraints of our use case made direct adoption difficult. Instead, we developed a tailored mapping strategy for direct translation within AEC design tools. Classes in the object-oriented model are mapped to OWL classes, while interfaces, which often represent taxonomic concepts, are also translated into OWL classes. The inheritance relationships between classes and the implementation relationships between classes and interfaces are represented using the rdfs:subClassOf relationship. Object properties and data properties are extracted from each class and mapped to OWL object properties and datatype properties, respectively. Unique identifiers (GUIDs) for each object are converted into URIs, ensuring each instance is uniquely identifiable within the RDF graph. Primitive data types such as strings, integers, and booleans are mapped to their corresponding XML schema datatypes. Complex data structures such as lists are represented using RDF sequences to maintain order. The mapping process ensures that both the structure and semantics of the original data model are preserved in the OWL/RDF representation.

A key aspect of this framework is integrating geometric data with semantic information, enabling cross-domain queries and reasoning. The BHoM to OWL/RDF converter/serializer supports two geometric representation approaches, balancing storage efficiency and semantic depth. The first serializes geometry (points, lines, meshes) into strings, optimizing storage and query performance but limiting geometric reasoning. In contrast, the second approach represents geometric entities as OWL ontology classes, where each entity is constructed hierarchically. For instance, a polyline is represented as an ordered collection of points, with each point stored as an instance containing X, Y, and Z coordinates. While this method enhances the framework's reasoning capabilities, allowing inferences about spatial relationships, it also increases computational complexity and storage requirements. In this study, the ontology-based representation was selected to demonstrate the framework's semantic reasoning capabilities, particularly in handling architectural and structural models. The implications of this choice on graph complexity and performance are explored in the following sections.

In this research, the generated data includes both RDF-based representations and ontology-enriched data, referred to as "bhRDF" and "bhOWL", respectively. The bhRDF component provides structured data optimized for storage and querying in graph databases, while bhOWL encapsulates ontology definitions based on the BHoM schema. During BHoM's data model translation, we developed tools for seamless integration with SWTs. While BHoM's classification structure was pre-existing, our contributions include: (1) A BHoM to OWL/RDF converter/serializer encoding object models into RDF/OWL for SWT interoperability; (2) An OWL/RDF to BHoM deserializer reconstructing BHoM objects from RDF/OWL for AEC visual modeling tools; and (3) A GraphDB connector for storing, querying, and retrieving BHoM data within a graph database. These tools are collectively referred to as

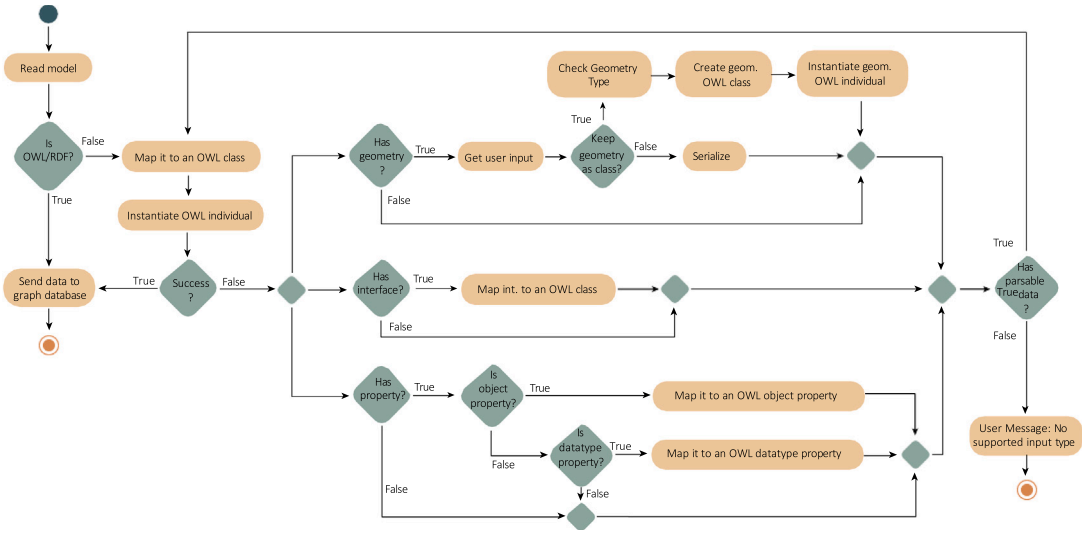

Fig. 2. Activity diagram of the mapping process.

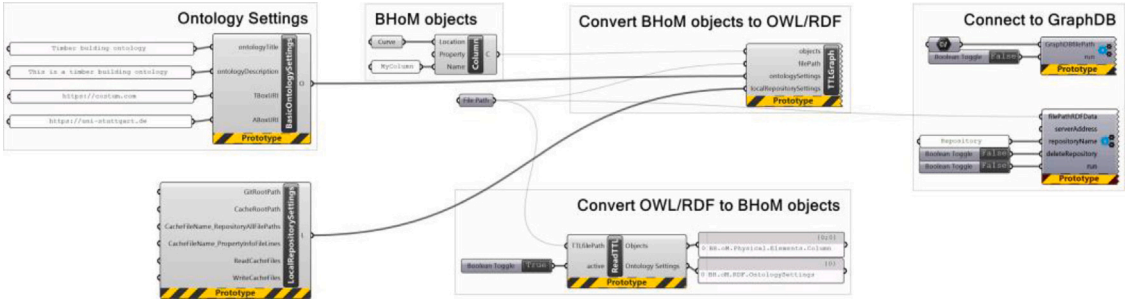

Fig. 3. Grasshopper implementation overview of the translation of BHoM objects to OWL/RDF data (and vice-versa) while designing.

the bhOWL toolkit for co-design workflows. Installation and update instructions are provided in Appendix D. The prototype is available at GitHub [69]. For a detailed exploration of BHoM to OWL/RDF Translation and Vice Versa, refer to "Appendix B: BHoM to OWL/RDF Translation and Vice Versa. Fig. 3 shows the mapping components developed in Grasshopper and illustrates how to interact with the RDF data in Grasshopper. A detailed figure of the SWT components within Grasshopper is provided in Appendix C.

5. Case study: Comparative study of design workflows

In this section, we present a comparative study of three distinct design workflows for designing a timber building: (1) Approach 1 – a design workflow that uses dedicated AEC software for each discipline but lacks an interoperability framework for data exchange, (2) Approach 2 – a design workflow that incorporates BHoM as an interoperability framework, and (3) Approach 3 – the proposed co-design workflow that utilizes the interoperability framework BHoM and SWTs, employing the developed mappers for object-oriented data models (within BHoM) to OWL/RDF and vice versa. The "co-design workflow" in this study refers to the structured iterative process that facilitates collaborative design efforts between architectural modeling and structural analysis disciplines. A detailed overview of the tools involved in each approach is shown in Fig. 4. In addition, besides the tools, white boxes indicate scalability with additional software or disciplines. In Approach 1, tools exchange data based on multiple data models, as illustrated by the top box showing that the addition of a new tool or discipline requires direct export/import between each tool. Approach 2 connects new tools or disciplines through the collaborative interoperability framework BHoM. Approach 3 integrates new disciplines via a graph database, where all project data is synchronized using SWTs (OWL/RDF graphs) and is mapped through BHoM. We evaluate these approaches, following DSR, against the criteria based on the requirements defined in our Methodology section, which include semantic representation, disciplinary data handling, class definition and extensibility, cross-domain querying capabilities, information inference capabilities, and data synchronization workflows.

The case study was developed in collaboration with an industry partner to ensure that the scenarios reflect practical multidisciplinary design challenges. The timber building features a grid structure composed of timber columns and slabs and is designed as a machinery lab with an entrance hall, covering 70 square meters (10 m × 7 m). The facade consists of a curtain wall. A 3D architectural representation is shown in Fig. 5.

We present four co-design iterations between architecture and structural engineering to demonstrate the workflow for each approach. Initially, an architect initiates the design process using a visual programming AEC design tool. Second, a structural engineer receives the design, generates a structural representation, and performs the analysis. Third, an architect designs a suspended ceiling to function as an acoustic insulator on a timber structure. To determine the optimal height for the acoustic ceiling, the architect considers potential deformation and displacement of the structural ceiling, determined through structural analysis and simulation. In the fourth iteration, the task is to determine the best location for the machinery lab within the timber building. The structural engineer provides column node reaction data, advising against placing heavy equipment near columns with high node reaction forces to avoid structural problems. The information exchange requirements for the four steps are defined and illustrated in Fig. 6.

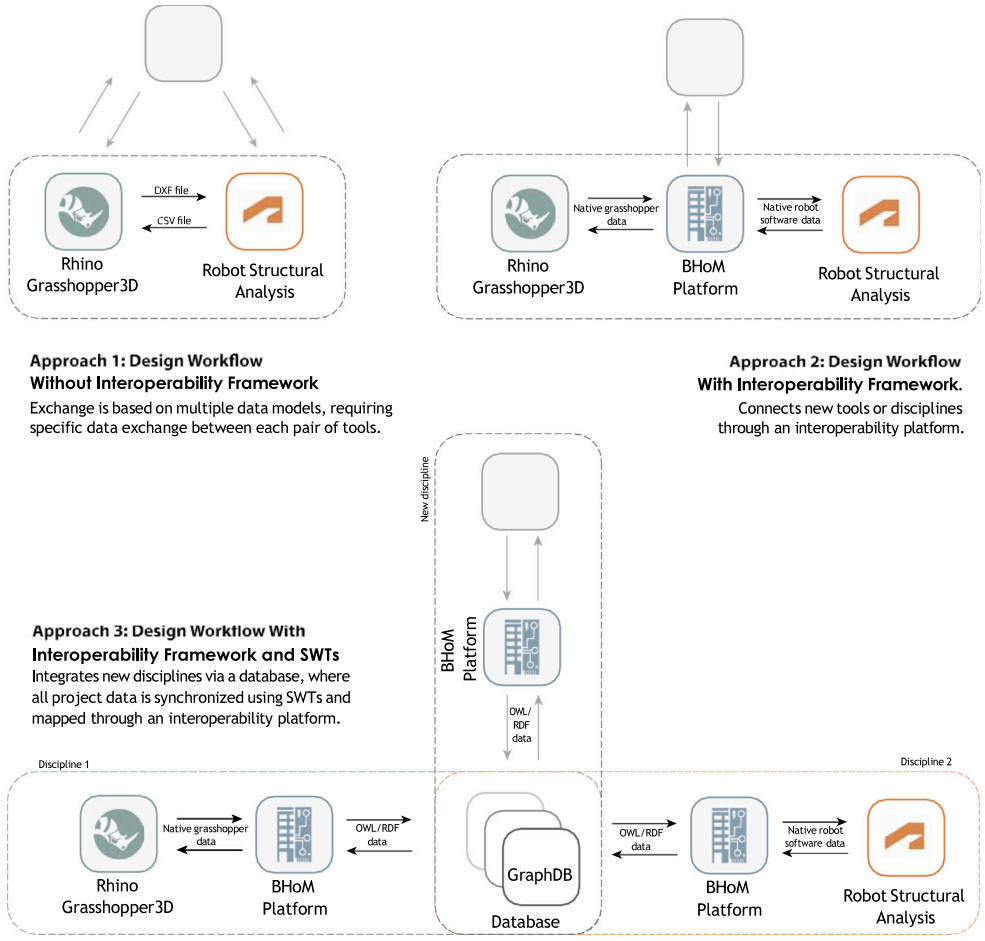

Fig. 4. Tools involved in the design workflow approaches.

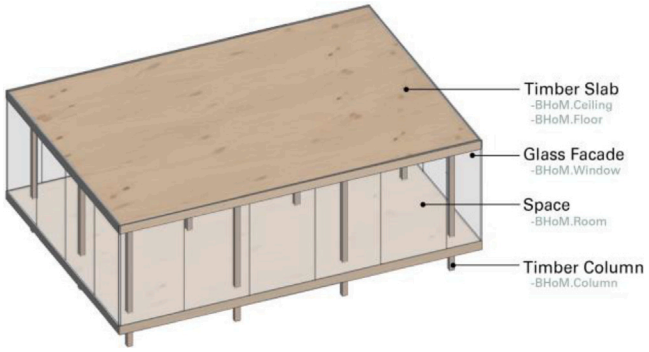

Fig. 5. 3D Architectural representation of the timber structure.

5.1. Approach 1: Design workflow without interoperability framework

In Approach 1, the design process involves the use of Rhino 3D and Grasshopper for architectural modeling, and Robot structural analysis software for engineering calculations.

1. Co-Design Iteration 1 - The Architectural Representation: We used Rhino to build the 3D geometry of the building, the grid structure, and the curtain wall. A parametric definition is created in Grasshopper, allowing adjustments to column spacing, slab thickness, and opening locations. After having an architectural design, we exported the geometry from Rhino to a DXF file.
2. Co-Design Iteration 2 - The Structural Representation and Structural Analysis Setup: We imported the DXF file into Robot Structural Analysis and redrew the structural elements according to the architectural geometries. Columns and slabs were identified and labeled using Define function. Material properties were assigned, and sections for columns and slabs were defined based on the design intent. We applied loads, including dead weight (self-weight of building elements) and live loads (occupancy and machinery), and performed structural analysis. Results — including member forces, deflections, and reactions — are exported as spreadsheets and graphical charts for further review.
3. Co-Design Iteration 3 - Placement of an acoustic ceiling under the structural ceiling: The slab deflection results from the structural analysis are reviewed. The architect then identifies critical areas and determines the highest Z coordinate for ceiling placement (see Fig. 7). The ceiling is manually modeled in Rhino3D or Grasshopper using imperative coding, and is exported as a DXF file for further structural analysis. This approach relies on DXF file exchanges.
4. Co-Design Iteration 4 - Decide the location of the machinery lab: Positioning the machinery lab is challenging in this approach because we intentionally do not use a classification (interoperability) tool. Therefore, we approach the task by identifying the room's location manually. We import a CSV file from the structural analysis software containing Z-direction node reactions for columns into Grasshopper. Using this data, we color-code the columns based on their Z-direction reactions. By visualizing these color-coded columns, we identify those with the highest reactions and adjust the machinery lab placeholder position accordingly (see Fig. 8). This process involved exchanging DXF and

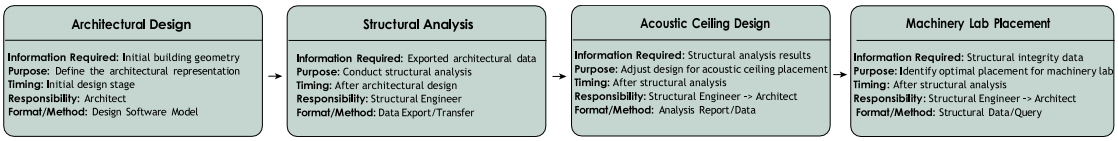

Fig. 6. Process diagram: Required information exchange.

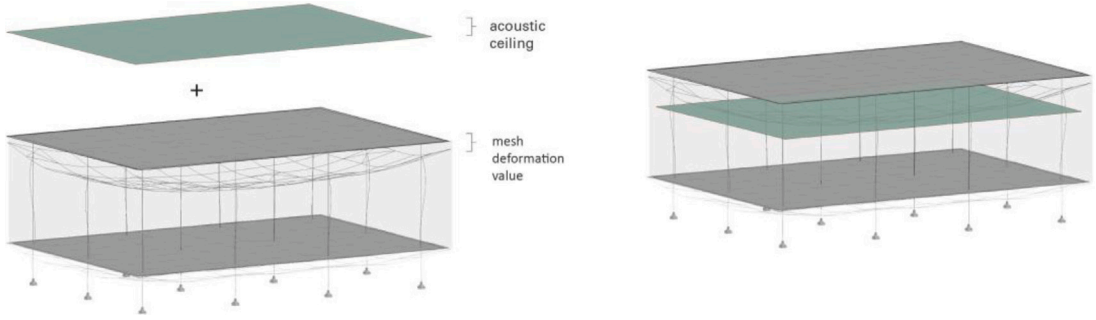

Fig. 7. Co-design Iteration 3: Placement of an acoustic ceiling under the structural ceiling, taking into consideration structural properties of the building.

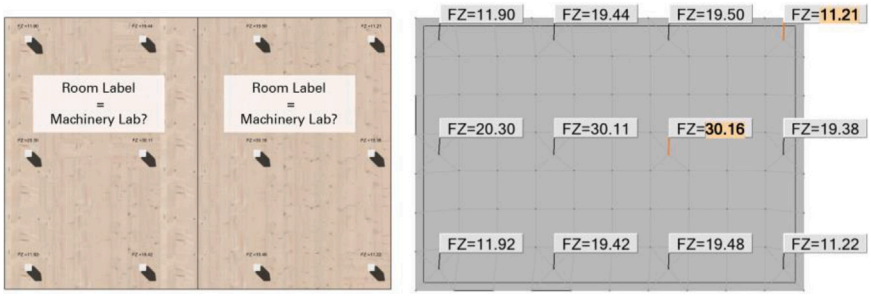

Fig. 8. Co-Design Iteration 4: Decide the location of the machinery lab depending on the node reaction of columns.

CSV files and required manual data import into Grasshopper, utilizing imperative coding for sequential task execution rather than rule-based placement.

5.2. Approach 2: Design workflow with interoperability framework

In Approach 2, we use Rhino3D, Grasshopper and BHoM for both architectural and structural engineering workflows, but with different modeling methods. Additionally, Robot Structural Analysis is used for structural representation and analysis.

1. Co-Design Iteration 1 - The Architectural Representation: To represent the building architecture, we used Rhino 3D and Grasshopper, integrating the BHoM plug-in for schema specifications. We utilized BHoM objects such as level, room, wall, column, floor, and window, along with material-related classes (e.g., wood, framing properties) and geometry-related classes (e.g., point, curve, surface). Additionally, we defined a custom class, ArchitecturalBuilding, which encapsulates these elements as properties (e.g., hasLevel for levels, hasColumn for columns), allowing us to relate objects to one another.
2. Co-Design Iteration 2 - The Structural Representation and Structural Analysis Setup: The process starts by extracting data from the architectural model using BHoM's adapters. Non-load-bearing elements (e.g., walls, windows) are excluded, focusing on load-bearing components such as columns and slabs.BHoM functions like AnalyticalBars and Surface map these elements to their structural counterparts, automatically computing the necessary parameters. The model is then prepared for structural analysis in Robot Structural Analysis, where loads, material properties, and boundary conditions are assigned manually. Various analyses, including deformation, stresses, and node reactions, are performed and integrated into Grasshopper via BHoM. Additionally, we introduce a custom class, "Structural Building", which relates bars, panels, and self-weight data to the building through properties like "hasBar", "hasPanel", and self-weight.
3. Co-Design Iteration 3 - Placement of an acoustic ceiling under the structural ceiling: We used BHoM functions to extract displacement values (e.g., maximum displacement at specific nodes) from the structural analysis results. We then manually calculated the maximum height for the acoustic ceiling, ensuring it is positioned below points of maximum deflection to avoid interference. In Grasshopper, we created a definition to query and calculate the Z coordinate for the acoustic ceiling based on these displacement values. Once the Z coordinate was determined, we modeled the ceiling manually in Rhino3D.
4. Co-Design Iteration 4: Decide the location of the machinery lab: We utilized BHoM functions to extract structural data such as load capacities and stress distribution, which were compared with building design requirements. In Grasshopper, we developed a definition to query and identify suitable machinery lab locations based on this data. A BHoM function (or Grasshopper script) was used to compute the optimal lab location. After obtaining the query results, we manually updated the architectural model in Grasshopper to reflect the chosen lab location. This process relies on imperative coding, requiring manual coding and scripting.

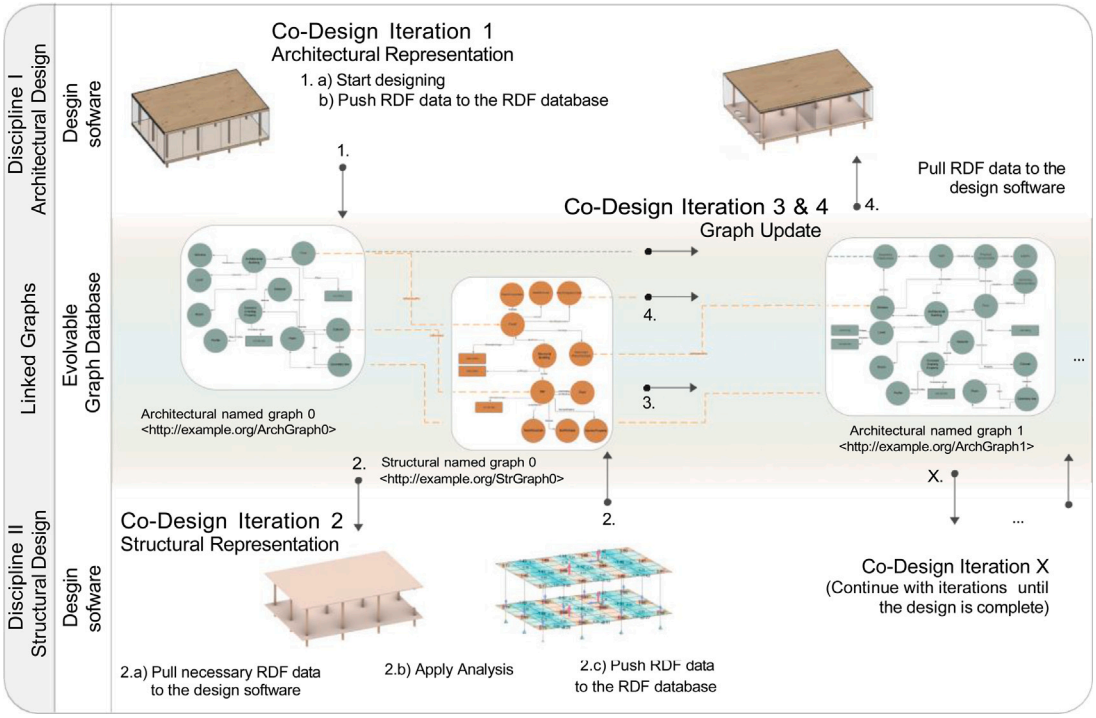

Fig. 9. Approach 3: Design workflow with an interoperability framework and SWTs.

5.3. Approach 3: Design workflow with interoperability framework and SWTs

Similar to Approach 2, Approach 3 utilizes BHoM as a collaborative interoperability framework while continuing to utilize Rhino 3D, Grasshopper, and Robot Structural Analysis software. However, what distinguishes this approach is the incorporation of a GraphDB as a central repository where project data is stored, queried, and updated using OWL/RDF representations. This enables seamless data integration and reasoning across disciplines, facilitating inference and cross-domain queries. A high-level overview of this workflow is depicted in Fig. 9.

1. Co-Design Iteration 1 - The Architectural Representation: This step is similar to Approach 2, we used Rhino 3D and Grasshopper along with the BHoM plug-in for the schema specifications. After modeling objects with existing BHoM classes and adding new ones (highlighted on the right in Fig. 10), we employed the developed converter to translate the building schema and its instances into an OWL/RDF graph. The converter allows toggling between serializing geometric information as strings or representing BHoM geometry as ontology classes. We chose to represent each geometry class as an ontology class to enable querying both geometric and semantic information. The resulting data graph was then pushed to the project repository in GraphDB using the named graph <http://example.org/ArchGraph>.
2. Co-Design Iteration 2: The Structural Representation and Structural Analysis Setup: This iteration follows Approach 2 up to the creation of the structural representation in BHoM objects using Rhino3D, Grasshopper and Robot structural analysis software. To connect structural and architectural objects, a new property named “isRelatedTo” was created. This property stores the GUID of the corresponding architectural objects when creating structural objects. We used BHoM’s setProperty function in Grasshopper to assign this “isRelatedTo” property. This relationship is essential for tracing the origin of structural objects and linking them to their architectural counterparts.Relations were established between the architectural floor and structural panel, as well as between the architectural column and structural bar (see Fig. 10). The structural graph, including the ‘isRelatedTo’ link, was pushed to the project repository in GraphDB using the named graph <http://example.org/StrGraph>. The linked ontologies of the architectural and structural graphs are illustrated in Fig. 10, where newly added classes and properties — expanding beyond the existing BHoM schema — are highlighted in green and orange.
3. Co-Design Iteration 3 - Cross-Domain query: Placement of an acoustic ceiling under the structural ceiling, taking into consideration the structural properties of the structural ceiling: Unlike the previous two approaches, this method uses a declarative language to determine the optimal placement of the acoustic ceiling while considering the structural properties of the structural ceiling. Specifically, a federated query is executed to retrieve the Z coordinate of the panel by accessing data from both the architectural and structural graphs. The query to find the Z coordinate using linked named graphs is illustrated in listing 1. This procedure is carried out in GraphDB, and the data is synchronized simultaneously from both architectural and structural sources. The retrieved information is then imported into Grasshopper via the BHoM platform and used to parametrically model the acoustic ceiling.
Query: What is the maximum height we can place the acoustic ceiling while taking into account structural factors?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
    syntax-ns#>
PREFIX : <https://schema.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT (?maxPanelHeightBar AS ?maxPanelHeight) (?
    minMeshValue AS ?
    minMeshDisplacementsUZZValue)
(?maxPanelHeightBar + ?minMeshValue AS ?
    difference)
{
  {
    SELECT (max(?barZendNode) AS ?
    maxPanelHeightBar)
    WHERE {
```

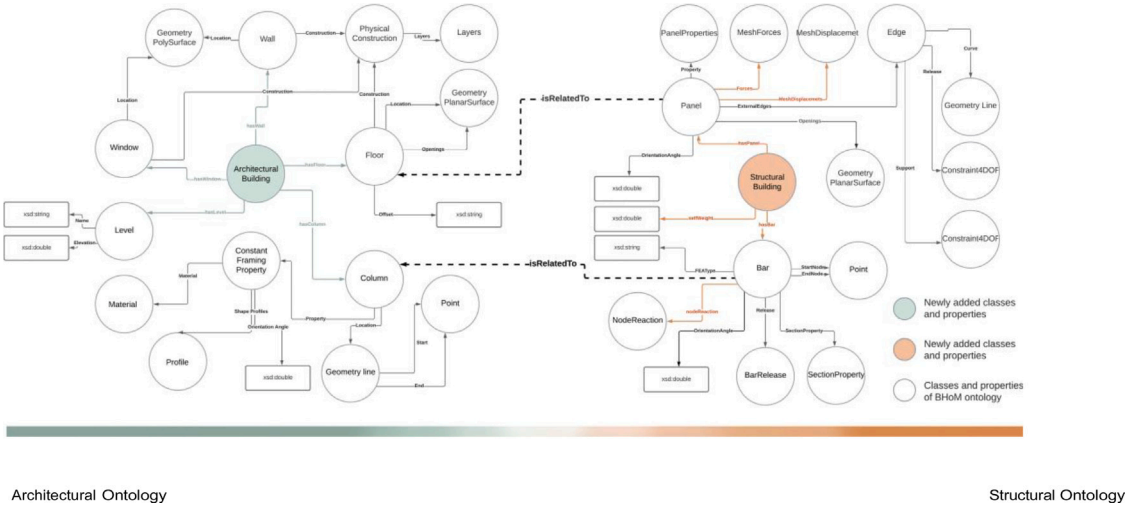

Fig. 10. Simplified visualization of the linked architectural and structural ontologies.

```
      GRAPH <http://example.org/StrGraph>
              {
                ?bar rdf:type :BH.oM.Structure.
    Elements.Bar.
                ?bar :BH.oM.Structure.Elements.
    Bar.EndNode ?EndNode.
                ?EndNode :BH.oM.Structure.
    Elements.Node.Position ?position.
                ?position :BH.oM.Geometry.Point.Z
     ?barZendNode.
              }
        }
  }
  {
    SELECT (MIN(?maxValue) AS ?minMeshValue)
    {
      GRAPH <http://example.org/StrGraph>
      {
        ?MeshDisplacements rdf:type <https://
    schema.org/MeshDisplacements>.
        ?MeshDisplacements <https://schema.org/
    MeshDisplacements.UZZ> ?
    MeshDisplacementsUZZValue.

        BIND(REPLACE(STR(?
    MeshDisplacementsUZZValue), "NaN", "0") AS ?
    stringValue)
        BIND(xsd:double(?stringValue) AS ?
    maxValue)

        FILTER (isNumeric(?maxValue))  # Filter
    out non-numeric values
      }
    }
  }
}
```

Listing 1: SPARQL Query for Determining Maximum Acoustic Ceiling Height Based on Structural Data

4. Co-Design Iteration 4: Reasoning: Infer the location of the machinery lab depending on the node reaction of columns
   In contrast to the first two approaches for design iteration 4, this approach uses a declarative language to determine the machinery lab's location based on column node reactions. It utilizes architectural and structural graph data alongside rule-based inference in RDF Format (RIF Core Syntax). The reasoning step relies on 23 facts, defining the rule type, description, conditions, and consequences. The rule consists of three main components. The conditions define constraints on node reactions, spatial boundaries, and permissible locations for the machinery lab. The constraints filter out locations where structural stress exceeds acceptable levels. The consequence specifies the final room label based on the inferred safe zone.

Before reaching the final placement rule, intermediate rules were applied to refine the decision-making process. The spatial constraints evaluate the minimum and maximum permissible (X, Y, Z) coordinates, with the rule incorporating functions such as (*e.g.*, *swrlb* : $greaterThanOrEqual(?x, ?minX)$). The input values—such as $minX$, $maxX$, $minY$, $maxY$, and $maxNodeReaction$—are manually input into GraphDB as hard-coded constraints. These values represent the results of earlier structural analysis, and they define the permissible placement range for the machinery lab, establishing spatial and structural boundaries within which the lab must be placed.

To enforce the rule, we use GraphDB's OWL Horst reasoning capability, which handles OWL 2 DL ontologies and integrates class descriptions and property axioms. This reasoning infers the optimal machinery lab placement based on structural and spatial constraints. To retrieve the inferred location data, we execute a SPARQL query against the GraphDB repository, which returns the optimal coordinates or room for placing the machinery lab. We then populate the architectural graph with a label indicating the machinery lab's location directly within the graph structure. When accessed via BHoM, the updated architectural graph will already reflect this designation, reflecting the results of the semantic reasoning into the overall model. The rule is provided in Listing 2.

Semantic Rule: Placing Machinery Lab in Column-Safe Zones

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-
    syntax-ns#> .
@prefix : <https://schema.org/rules> .
@prefix swrlb: <http://www.w3.org/2003/11/swrlb#>
    .

:PlacingMachineryLabRule
    rdf:type :Rule ;
    :description "Rule to place the machinery lab
     in a room without columns with high node
    reactions." ;
    :hasCondition [
        rdf:type :Condition ;
        :hasBar ?bar ;
        :hasNodeReaction ?nodeReaction ;
        :nodeReactionUZ ?uz ;
        :hasPosition ?pos ;
        :hasEndNode ?bar ;
        :hasPosition ?pos ;
        :hasRoom ?room ;
        :roomName "MachineryLab" ;
        :hasBoundary ?boundary ;
        :hasPoints ?pointBoundary ;
        :xCoordinate ?x ;
        :yCoordinate ?y ;
```

```
        swrlb:greaterThanOrEqual(?x, ?minX) ;
        swrlb:lessThanOrEqual(?x, ?maxX) ;
        swrlb:greaterThanOrEqual(?y, ?minY) ;
        swrlb:lessThanOrEqual(?y, ?maxY) ;
        swrlb:lessThanOrEqual(?uz, ?
    maxNodeReaction)
    ] ;
    :hasConsequence [
        rdf:type :Consequence ;
        :isLocatedIn ?bar :room
    ] .
```

Listing 2: Semantic Rule for Placing Machinery Lab in Column-Safe Zones (SWRL Syntax)

6. Results: Evaluation and comparison of approaches

The evaluation criteria for the case study are based on the requirements outlined in Section 4.1. These criteria include semantic representation, handling of disciplinary data, class definition and extensibility, cross-domain querying capabilities, information inference capabilities, and data synchronization workflows. To assess the performance of the approaches, we use two evaluation scales: low to high and none to extensive. The scale "None" indicates a lack of capability or reliance on fully manual processes, with minimal scalability. "Moderate" reflects partial or semi-automated integration, often requiring manual intervention or ad-hoc methods. "High" or "Extensive" signifies robust and scalable capabilities.

6.1. Criterion 1: Semantic representation and disciplinary representation

Approach 1 lacks a formal semantic representation, resulting in a rigid data structure primarily composed of geometric information. Although some semantic details are captured in the building's structural representation using Robot software's internal schema, it remains limited. Approach 2 improves semantic representation by utilizing the BHoM interoperability framework, which incorporates semantic information. This approach supports diverse disciplinary models by allowing different disciplines to represent building elements according to their specific needs, regardless of the software used. Approach 3 advances both semantic and disciplinary representation by modeling architectural and structural aspects using BHoM schema classes and custom definitions, and translating them into OWL/RDF. The architectural graph includes 1250 classes, 32 object properties, 51 datatype properties, 85 individuals, 221 nodes, and 1099 edges (see Table 1). It primarily features geometric information, with a large number of nodes representing primitive and constructive geometries. This results in large graph sizes and lengthy query response times, especially for complex geometries. The structural graph, which includes 2000 classes, 29 object properties, 69 datatype properties, 704 individuals, 847 nodes, and 5314 edges, is even larger due to the detailed geometry required for structural analysis (see Table 1). For visual reference, a simplified ontology of both representations is shown in Fig. 10, while the complete ontologies with instances are available in [70]. This dataset also includes a neutral building model without BHoM schema specification, ensuring a more generalized and software-independent representation.

In summary, Approach 1 lacks semantic and disciplinary representation. Approach 2 utilizes the BHoM schema to address these needs, while Approach 3 integrates BHoM with SWTs to enhance both semantic and disciplinary representation.

Table 1
Statistics on the resulting ontologies using the developed toolkit, including the number of OWL classes, properties, individuals, total nodes, and number of edges.

Category	Architecture ontology	Structural ontology
Classes	49	41
Object Properties	32	29
Datatype Properties	51	69
Individuals	85	704
Nodes	221	847
Edges	1099	5314

6.2. Criterion 2: Class definition and extensibility

Approach 1 does not provide direct access to defining new classes within the structural analysis software. While class definitions exist, they are embedded within the software's API, requiring additional complexity to extend via scripting or API calls. This makes customization possible but more challenging. Approach 2 utilizes BHoM's custom objects, allowing users to define new classes directly within the Grasshopper interface, without coding. This feature, which we used extensively in our case study, enables computational designers to create custom classes for both architectural and structural representations. For example, we introduced the custom class "ArchitecturalBuilding" to encapsulate various building elements and provide a holistic view of the architectural structure. Similarly, the custom class "StructuralBuilding" represents the entire structural system, including bars, panels, and their associated properties (see Fig. 11). Approach 3 incorporates all the benefits of the BHoM framework as well as the use of RDF to define classes and properties across disciplines, providing seamless extension and integration capabilities. Thus, class definition and extensibility are enhanced by both SWTs and BHoM. In both ontologies depicted in Fig. 10, the highlighted nodes and edges represent newly defined classes, properties, or relations.

In conclusion, Approach 1 is limited in terms of class definition and extensibility, relying on manual tool definitions. Approach 2 leverages BHoM for easier class creation and extension, and Approach 3 enhances these capabilities further by integrating BHoM with SWTs, providing comprehensive and flexible data representation.

6.3. Criterion 3: Linked-data for reasoning and cross-disciplinary queries

In terms of cross-domain querying, the three approaches demonstrate distinct levels of capability. Approach 1 lacks the ability to perform cross-domain querying, as the data remains disconnected across disciplines. Any data import or export is carried out manually, which limits the possibility of integration or reasoning across sources. Approach 2 introduces the BHoM framework, enhancing interoperability between different disciplinary models through a common schema. However, the lack of a linked data framework means that each query must be manually scripted for specific tasks, resulting in limited scalability. So, like Approach 1, it lacks native support for linked data, restricting cross-disciplinary queries and still relying on imperative coding. Approach 3 allows for reasoning and cross-disciplinary queries by using SWTs, representing data in OWL/RDF to create an interconnected knowledge graph. This approach leverages RDF's declarative querying and reasoning capabilities to discover implicit relationships across disciplines. For instance, Fig. 12 visualizes a SPARQL query that determines the position of an acoustic ceiling relative to a structural ceiling, illustrating how architectural and structural data interact. The visualization aids in understanding the query's logic and data connections, as outlined in [71].

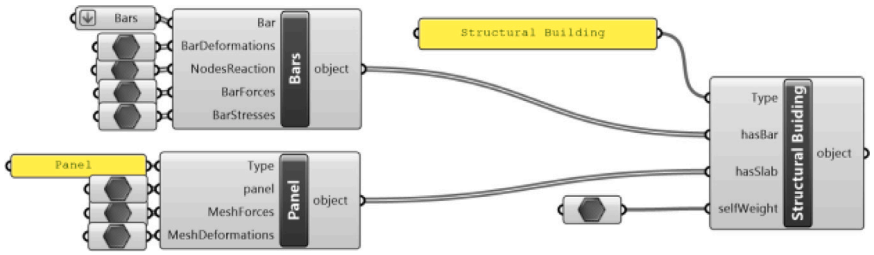

Fig. 11. Type definition of newly defined custom classes using the BHoM framework in Grasshopper.

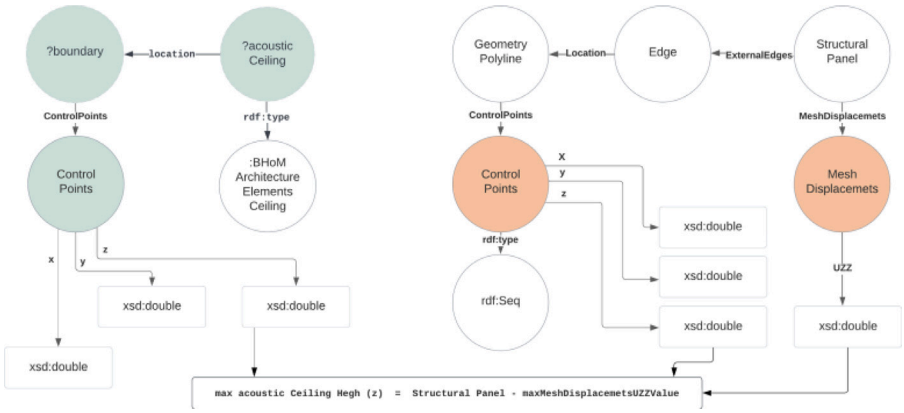

Fig. 12. Visualized SPARQL query of interacting parameters of architectural and structural data to determine the acoustic ceiling position in Approach 3.

Similarly, the approaches differ in their support for information inference. Approach 1 requires users to interpret results manually, offering no support for reasoning or rule-based decision-making. Approach 2 provides semi-automated inference capabilities, where some calculations are supported through scripting using imperative coding. However, this approach still requires manual intervention, particularly for more complex tasks. Approach 3 enhances inference capabilities by employing SWTs and declarative logic, integrating ontology-based inferences and rule-based reasoning. Ontology-based inferences derive facts from the BHoM schema, leveraging subclass relationships and object properties to generate additional links between architectural and structural data. Rule-based reasoning determined the optimal placement for a machinery lab based on column node reactions. Before finalizing the placement, multiple intermediate rules were applied using RDF rules such as: identify high-stress columns, assess adjacent rooms, evaluate spatial feasibility, and compute permissible (X, Y, Z) coordinates. Ultimately, GraphDB's OWL Horst Reasoning ensured placement in zones with minimal structural impact. Overall, these inferencing processes resulted in 4115 additional inferred facts, demonstrating enhanced reasoning capabilities, as illustrated in Fig. 13.

6.4. Criterion 4: Data synchronization

To evaluate the three design approaches in terms of data synchronization, we will consider the number of steps required for each approach to complete the presented design iteration, as well as whether these steps are achieved manually or if they are automated.

All approaches require a minimum of 2 steps to complete iteration 1. Approach 1 involves the following steps: (1) Model Building Geometry and (2) Exporting the Rhino model as .dxf. The other two approaches differ in the way we export the data. Approach 2 saves BHoM objects, while Approach 3 sends data to GraphDB. Thus, Approach 2 and 3 focus on data transfer, while Approach 1 relies on file formats.

The approaches diverge in the second design iteration. Approach 1 requires eight steps: importing the .dxf file into Robot, defining structural components manually, cleaning up or remodeling geometry, assigning structural elements, defining loads, setting up supports, performing structural analysis, and interpreting results. Approaches 2 and 3 each require seven steps, omitting the three manual steps of Approach 1 by using BHoM mappers to map architectural objects (columns to bars, slabs to panels) to structural ones, ready for structural analysis. For Approach 2, the steps are pulling load-bearing architectural objects from the architectural model to the structural environment in Grasshopper, mapping BHoM architectural concepts to structural ones, pushing structural objects from Grasshopper to Robot, defining loads, setting up supports, performing structural analysis, and sending results to Grasshopper. For Approach 3, the steps are similar, with data sent to and read back from the graph database.

Iterations 3 and 4 show similar step counts for each approach but differ in execution. Approaches 1 and 2 use imperative coding, while Approach 3 uses declarative languages. For iteration 3, Approach 1 requires two steps: analyzing results manually and modeling the acoustic ceiling, with optional iterations of pushing objects to Robot for additional analysis. In iteration 4, Approach 1 requires three steps: exporting CSV files from Robot, importing Z-direction node reaction data into Grasshopper, adjusting the location of the machinery lab, and optionally exchanging data using DXF and CSV files. Approach 2, with the same number of steps, differs in data import/export methods. For Iteration 3, it includes extracting displacement values for modeling the acoustic ceiling using the BHoM framework, with optional iterations of pushing objects to Robot for additional analysis. In iteration 4, data is exchanged via BHoM, with steps including pulling relevant structural data, creating a query to identify suitable locations, and updating the architectural model in Grasshopper. The steps of Approach 3 for iteration 3 include extracting displacement values to model the acoustic ceiling and running a SPARQL query to model. For iteration 4, the

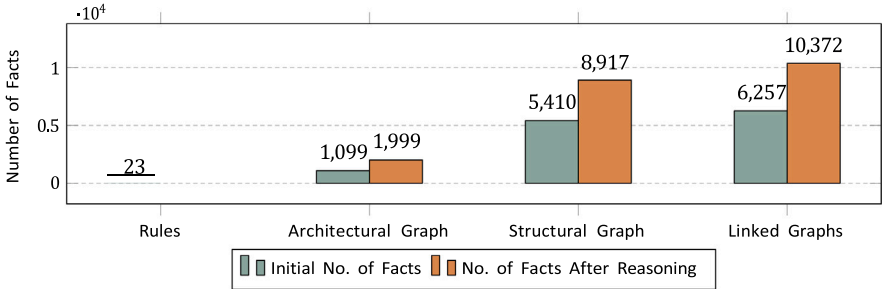

Fig. 13. Number of RDF triples (facts) in the graph before and after running a reasoner in Approach 3.

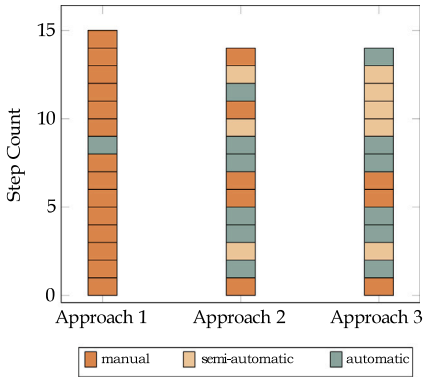

Fig. 14. Criterion 4: Data synchronization: Automation level classification for approach steps.

Table 2
Number of required steps for each iteration in the three different approaches.

Co-Design iteration	Approach 1	Approach 2	Approach 3
Iteration 1	2	2	2
Iteration 2	8	7	7
Iteration 3	2	2	2
Iteration 4	3	3	3
Total	**15**	**14**	**14**

steps involve pulling relevant structural data from the graph, creating a rule to place the machinery lab in zones without high-node reactions, and updating the architectural model in the database graph. The total number of steps can be found in Table 2.

However, the count of steps alone does not fully convey the impact on data synchronization. Therefore, in Fig. 14, we have categorized each step in each iteration as manual, semi-automatic, or automatic based on human intervention. Manual steps entail significant human involvement. Semi-automatic steps combine automated processes with human supervision. Automatic steps operate without human intervention once initiated. Overall, Approach 1 involves the most manual steps, particularly related to data import/export from design platforms. Approaches 2 and 3, with the same step count, differ in data mapping and storage. Approach 2 maps data from one discipline to another, losing the original link and object provenance. Thus, Approach 3 particularly stands out for its declarative approach, underscoring its advancement in ensuring data synchronization and enhanced design decision-making capabilities.

6.5. Summary

The comparison of the three approaches, summarized in Table 3, highlights a progression in capabilities from Approach 1 to Approach 3. Approach 1 represents the most manual and least integrated workflow, with no formal semantic representation, limited class definition, and a reliance on manual data transfer. Consequently, this Approach lacks scalability, is prone to errors, and offers no support for cross-disciplinary reasoning or inference. Approach 2 introduces improvements through the BHoM interoperability framework. It incorporates semantic capabilities and facilitates disciplinary data representation using a unified schema. The framework supports the creation and extension of classes, which enhances modularity and adaptability for different disciplines. However, despite these advancements, Approach 2 still relies on imperative coding for querying and reasoning, which limits automation and consistency. Information inference required manual intervention, particularly for complex tasks. Approach 3 demonstrates the most advanced capabilities by combining BHoM with SWTs. It provides comprehensive semantic representation and enhanced class extensibility, enabling linked-data reasoning and cross-disciplinary querying. This Approach supports declarative logic and rule-based reasoning, allowing the deduction of implicit relationships and insights directly from linked graphs. For example, as illustrated in Figs. 12 and 13, Approach 3 achieves information inference by leveraging SWTs and reasoning engines, which enhance the accuracy and scalability of the workflow. Furthermore, Approach 3 ensures data synchronization by maintaining object provenance and leveraging declarative languages, reducing the risk of inconsistencies across iterations. In addition to its technical advancements, Approach 3 demonstrates a reduction in manual steps across design iterations (as shown in Table 2), achieving higher levels of automation compared to Approaches 1 and 2. This improvement is particularly evident in Iterations 3 and 4, where the use of declarative languages and RDF graphs eliminates the need for manual scripting. Overall, the evaluation demonstrates a trend toward increased scalability and integration as approaches evolve from manual processes (Approach 1) to interoperable frameworks with semantic reasoning (Approach 3). While Approach 1 may still be applicable for small scale projects with minimal data integration

Table 3
Comparison of the three co-design Approaches: Approach 1: Design workflow without interoperability framework; Approach 2: Design workflow with interoperability framework; and Approach 3: Design Workflow with interoperability framework and SWTs.

Indicator	Approach 1	Approach 2	Approach 3
Allow semantic representation	None: No formal semantic representation	Moderate: Yes. Uses BHoM schema for data representation	Extensive: Semantic representation through BHoM and SWTs
Allow disciplinary data representation	None: No formal disciplinary data representation handling	Moderate: Allow disciplinary data representation through BHoM	Extensive: Enhanced disciplinary data representation handling via SWTs and BHoM
New class definition ability and class extensibility	Low: Limited; manual definition in tools	Moderate: Extensive; utilizes BHoM class definitions	High: Class definition flexibility extended by SWTs or BHoM
Cross-Domain Querying	None: No cross-domain querying; manual data import/export	Moderate: Relies on manual scripting	Extensive: Federated queries across linked graphs
Information Inference	Low: Results interpreted manually	Moderate: Semi-automated inference; requires manual calculations via imperative coding	High: Automated inference using declarative logic and SWTs; reasoning engine significantly enhances insights
Data synchronization	Low: Prone to inconsistencies due to manual data transfer	Moderate: Synchronization ensured through the interoperability framework	High: Enhanced data synchronization using SWTs

needs, Approach 3 stands out as a more effective solution for complex, multidisciplinary co-design scenarios, offering both efficiency and decision-making capabilities.

7. Discussion, challenges and future work

This paper advances the application of SWTs in the AEC domain by developing and evaluating a co-design workflow that facilitates cross-domain querying and reasoning. By integrating SWTs, we have improved semantic representation, data interoperability, and design decision-making processes through knowledge inference. The proposed co-design workflow offers a robust approach to interdisciplinary data integration. By leveraging linked named graphs, it enables complex queries across disciplines, enhancing the ability to make informed design decisions. Incorporating SWTs for knowledge representation and cross-domain querying and reasoning aligns with the emphasis on ontology and logic as foundational methods for formalizing knowledge, as discussed by Hartmann and Trappey [24]. Despite these advancements, challenges persist within the developed workflow, along with opportunities for extensions, including broader challenges and research directions. In the following subsections, we discuss these aspects.

7.1. Challenges with the developed workflow

Key issues include the complexity of formulating queries, the deserialization, and handling geometric information.

Query Formulation Complexity: While linked named graphs support cross-disciplinary queries, formulating queries for AEC problems remains complex due to their intricate nature and the limited expressiveness of SWTs. Pauwels et al. [33] note that while linked data applications in the AEC industry have been demonstrated, technologies like reasoners and proof engines have not been widely adopted. They attribute this to the higher-layer positioning of these tools in the SWT stack, which is perceived as more challenging to implement. Expanding on this, we argue that the inherent complexity of AEC problems exacerbates these challenges, as SWTs often lack the expressiveness needed to fully capture intricate domain-specific relationships. To mitigate this, we propose integrating a data coordinator within the co-design workflow to support and streamline query formulation.

Deserialization Challenges: Ensuring that queries return graphs parsable into BHoM objects proved challenging. Deserialization functions well with unmodified graphs but encounters errors when manipulated graphs deviate from the BHoM structure. Issues arise when undeleted properties from removed nodes or new relations introduce inconsistencies. Implementing automatic delete cascade functions within the knowledge graph could improve consistency by ensuring the proper update or removal of associated data. Similar issues occur with reasoners, where multiple type assignments can confuse mappers, leading to incorrect classifications in custom BHoM objects. Future research could explore using SHACL to validate RDF graph shapes before deserialization, improving consistency across disciplines and reducing errors.

Handling Geometric Information: Representing geometric data within the graph introduced computational challenges and increased storage requirements. To address this, we implemented a flexible converter capable of toggling between serializing geometric data as strings and representing BHoM geometry as ontology classes. For instance, a polyline boundary of a space was represented as an ordered collection of point instances with X, Y, and Z coordinates. This approach facilitated the handling of complex geometries, such as meshes for structural analysis. Meshes consist of points and edges, which made the disciplinary graphs large in storage. For example, the structural graph using meshes was four times larger than the architectural graph (847 nodes vs. 221 nodes). Future research could explore SWT-compatible methods to optimize geometry storage, querying, and manipulation, referencing studies like [72]. Additionally, investigating hybrid RDF-based databases that integrate semantic and geometric data may provide greater flexibility and efficiency. A hybrid approach — storing semantic data in RDF while maintaining complex geometric data in optimized formats — warrants further evaluation to enhance performance while preserving interoperability [23].

7.2. Future integrations and possible extensions

Advancing the integration of SWTs in AEC requires both technical enhancements and considerations of broader interoperability challenges. While this study primarily focuses on semantic and technical aspects, further refinements and extensions are necessary to fully leverage SWTs for seamless cross-domain collaboration. This section discusses potential improvements to the developed workflow, including expanding ontology support, enhancing cross-platform compatibility, and addressing challenges related to automation, scalability, and governance structures.

- Enhancing Technical Capabilities: The current toolkit focuses on OWL and RDF vocabularies, incorporating OWL classes, properties, and datatypes relevant to our needs. However, it does not yet support constructs such as disjointWith, equivalentTo, functionalProperty, or inverseFunctionalProperty, which are essential for defining new object relationships. Expanding the toolkit to incorporate these constructs would require additional components

capable of leveraging the generated graph as input, further improving graph manipulation and reasoning capabilities. While our approach supports linked data and cross-domain queries, deserializing graphs aligned with ontologies beyond the BHoM schema was not within the scope of this study. However, the methodology is not inherently restricted to BHoM and could be adapted to alternative models such as Brick, ifcOWL, or Siemens' industrial ontologies. Since the translation mechanism maps object-oriented data models to OWL/RDF, extending it to other structured data schemas following similar principles is feasible. For broader generalizability, the methodology could be applied across different environments by modifying the translation layer. For example, replacing BHoM with Speckle would require mapping its data to RDF triples, while adapting it to Brick would ensure alignment with building automation systems while maintaining interoperability. Future research could explore incorporating these ontologies and improving RDF data reading and exporting for seamless integration with AEC ontologies like BOT and ifcOWL [56]. Beyond computational and infrastructure constraints, the broader application of SWTs in AEC still presents challenges. The lack of standardized vocabularies across AEC subdomains complicates interoperability, making seamless data exchange more difficult. Additionally, knowledge graphs introduce issues related to graph completion, accuracy, and consistency. Integration efforts remain largely semi-automated, requiring manual intervention for schema alignment and data transformation. Future research could explore AI-driven approaches, such as machine learning-assisted ontology alignment and automated graph completion, to enhance the efficiency and scalability of SWTs in AEC workflows.

- Addressing Governance and Organizational Challenges: While technical advancements play a crucial role, the successful implementation of SWTs also depends on organizational and legal considerations. Aligning governance structures and ensuring compliance with cross-border legal frameworks is critical for achieving seamless integration. Future research could explore these areas to support comprehensive interoperability in AEC workflows, following guidelines such as the European Interoperability Framework [73].

This study's interoperability framework ensures mappings and connectivity between tools, highlighting the importance of linking software through shared mapping methods before integrating disciplinary data. A key question that emerges is: What is the optimal balance between connecting software via interoperability frameworks and linking data across disciplines? Additionally, while integrating SWTs into design software enhances accessibility for designers, it is important to consider how workflows can incorporate these technologies without requiring substantial modifications to existing design processes.

8. Conclusion

This paper has developed and evaluated an AEC co-design workflow that integrates SWTs with collaborative interoperability frameworks to enhance cross-domain querying and reasoning. By addressing the challenges of fragmented data silos and inconsistent data representations in the AEC industry, the proposed approach demonstrates advancements in semantic representation, cross-disciplinary data integration, and reasoning capabilities.

A comparative evaluation of three design workflows — (1) without an interoperability framework, (2) with a framework but without SWTs, and (3) integrating both SWTs and an interoperability framework — revealed that the third approach performs best across multiple aspects. The integration of SWTs enables reasoning, federated queries, and improved data synchronization, facilitating more efficient decision-making and constraint identification. The use of declarative logic and rule-based reasoning enhances information inference, reducing reliance on manual intervention and imperative coding. Furthermore, the study highlights the scalability of the SWTs-integrated workflow, ensuring adaptability to evolving project requirements while preserving data integrity across disciplines. This integration is particularly valuable in early-stage design decisions, where cross-domain dependencies must be identified promptly to optimize building performance and minimize costly modifications in later project phases.

Future research should explore further optimizations in query performance and computational efficiency, as well as better alignment with existing AEC ontologies such as BOT and ifcOWL. Additionally, expanding the workflow's capabilities to integrate emerging technologies — such as AI-driven approaches, machine learning-assisted ontology alignment, and automated graph completion — could further enhance co-design efficiency.

By advancing interdisciplinary collaboration through linked-data methodologies, this research provides a foundation for a more interconnected and automated AEC industry. Ultimately, this study offers valuable insights for both scholars and industry professionals seeking to optimize the application of semantic technologies in AEC workflows. By acknowledging both the challenges and opportunities associated with SWTs, this work contributes to the advancement of more effective, collaborative, and data-driven design processes in the AEC sector.

CRediT authorship contribution statement

Diellza Elshani: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Alessio Lombardi:** Writing – review & editing, Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Daniel Hernandez:** Writing – review & editing, Validation, Supervision, Methodology, Formal analysis. **Steffen Staab:** Writing – review & editing, Supervision, Project administration, Investigation, Funding acquisition, Conceptualization. **Al Fisher:** Supervision, Project administration, Investigation. **Thomas Wortmann:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT-4 in order to improve the grammatical correctness of sentences. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2120/1 – 390831618 and COFFEE - STA 572_15-2. We thank our industry partners, Buro Happold, for their efforts in the paper, as well as for their help in defining use cases for evaluating the workflow. Additionally, we thank our student assistant, Aaron Wagner, for his support with the BHoM to GraphDB adapter development. We also acknowledge Shermin Sherkat, Lior Skoury, and Ali Nahkaee for their valuable discussions that informed parts of this paper. Finally, we appreciate the reviewers for their valuable comments that helped improve this work.

Appendix A. BHoM abstract model

BHoM's object model consists of C# programming language types (like classes, interfaces or enums.) and methods that form BHoM's object models and related functionality. Object models are classes like Column, Wall or Room, and they are designed to be a collection of types that embed only data and not functionality. The functionality is defined separately from the types and bound to them when required. BHoM type development follows the composition over inheritance principle, using interfaces with semantic meaning rather than subclassing; for instance, all 2D representation classes such as Room or Panel implement the IElement2D interface, thereby avoiding issues like tight coupling or increased complexity [16].

Hence, to present BHoM abstract model, we assume a finite set **C**, called the *set of classes*, which represents BHoM classes and interfaces. The BHoM data model includes standard C# types (e.g., numbers and strings) and classes that do not belong to BHoM. We therefore assume a finite set **T**, called the *set of data types*, which represents standard C# types and non-BHoM classes and interfaces. Finally, we assume a finite set **P**, which represents the properties of the BHoM classes and interfaces. We assume that these three sets **C**, **T**, and **P** are pairwise disjoint.

An essential aspect of the BHoM model is that its elements are described differently depending on the designer's discipline. The following examples illustrate this aspect showing a BHoM class having different attributes in structural and acoustic design.

Example 1. Structural Panel: In the structural design, a panel is a planar surface defined by a list of planar edges (over the external contour), a list of openings that are coplanar with the contour of the panel, a property definition that describes the thickness and the material of the panel, and the angle. Panels are modeled using the BHoM class Panel with the attributes:

1. ExternalEdges of type list(Edge),
2. Openings of type list(Opening),
3. Property of type ISurfaceProperty,
4. OrientationAngle of type double.

Elements Panel, Edge, Opening, and ISurfaceProperty are classes (i.e., belong to the set **C**); element double is a data type (i.e., belong to set **T**); and elements ExternalEdges, Openings, Property, and OrientationAngle are properties (i.e., belong to **P**). The word list modifies the type of the property to indicate that the type is not an atomic value, but a list of elements of the specified type. For instance, the value for the property ExternalEdges has to be a list of elements of the type Edge.

Example 2. Acoustic Panel: In acoustic design, a panel is defined by a mesh, an identifier, and a dictionary that maps frequencies to numbers. Panels are modeled using the BHoM class Panel with the attributes:

1. Surface of type Mesh,
2. PanelID of type int,
3. R of type dict(Frequency, double).

Elements Panel and Mesh belong to set **C**; elements Frequency, int, and double belong to set **T**, and elements Surface, PanelID, and R belong to **P**. The word dict is used to indicate that the type is not atomic, but a dictionary.

Examples 1 and 2 show that concept Panel has different properties in each discipline. Since designers of the building structure do not need information regarding the building acoustic (and vice versa), irrelevant information is hidden for them. However, designers from both disciplines need to interchange information to design and construct a building. In the current workflow, designers translate their designs from one discipline (e.g., structure) to another (e.g., acoustic). In the BHoM abstract model, a design consists of a set of objects of one discipline. In the translation from the structure to the acoustic discipline, an object representing a panel losses its structural attributes (e.g., ExternalEdges) and acquires new properties (e.g., R). Furthermore, the result is not the same object since it changes its identity (i.e., is copied with a different object ID), and a heuristic is needed to fill out the missing values. The whole design of the building is a set of disciplinary design models where there is no explicit relationship between objects representing the same entity in the building (e.g., the same panel) and no information about how the missing properties were filled. These limitations motivate the translation of BHoM models to OWL. The main advantage of this translation is to provide disciplinary and integrated views of the design. In our translation to OWL, we represent a disciplinary view as a subset of the RDF dataset whose scope is limited to the objects in the discipline, whereas the integrated view is the merge of the disciplinary views and the following metadata:

1. *Provenance relations among design objects.* There are multiple reasons to derive design objects. Examples are the translation above between disciplines and updates in the data in the same discipline. The Wide Web Consortium (W3C) recommends the PROV-O ontology to annotate artefacts with their provenance.
2. *Flexible data scope*. In the current BHoM system, objects are grouped in disjoint sets, where each set defines a building design. Objects in the structural design are not beyond the scope of structural design, and objects of acoustic design are not in the scope of structural design. In an RDF dataset integrating all building designs, elements can be organized more flexibly. For instance, we can see an update of a building design as the act of removing and adding elements. Removed elements are not in the scope of the new design, but preserved elements are in both. Objects can be beyond the scope of design disciplines. For instance, a constraint C in the acoustic design may not be satisfied in the structural design, leading to an update in the structural design. Hence, the constraint is beyond the scope of acoustic design. Furthermore, constraint C is valid in the new structural design but not in the old one. This flexibility on the validity of a constraint illustrates the need to annotate scopes more flexibly. RDF allows annotating data by using multiple reification schemes and named graphs.
3. *Current and outdated designs.* It is desirable that designers focus on the current design and that the outdated designs be hidden from their working view. As we already mentioned, designs are updated to satisfy constraints. Hence, an outdated design can be reconsidered if a constraint is no longer be valid. To this end, the metadata can allow identifying the data that have to be in the current view and old data that may be reconsidered.
4. *The identity of objects.* Currently, a new panel object is created when a design is translated from the structural to the acoustic discipline. Similarly, a panel can be updated, changing its properties. We have to answer whether the old and the new panels are the same panel (i.e., with a single identity). If we do not consider both panels the same, we must add the corresponding metadata to indicate that the new panel is derived from the old panel.

A.1. BHoM's disciplinary schema

In this section, we will discuss the BHoM abstract model as it pertains to a specific discipline within the AEC industry. The discipline schema within the BHoM framework serves to organize and manage discipline-specific data in a structured manner. Its primary function is to ensure that the data is effectively organized and tailored to meet the unique requirements of each discipline within the AEC industry. The disciplinary schema forms a crucial part of the BHoM framework, facilitating efficient data management, interoperability, and collaboration within the AEC industry by offering a clear and structured representation of discipline-specific data. In essence, a disciplinary schema comprises class definitions, a hierarchical relationship between these classes, and properties associated with class definitions. Each property is constrained by a type, indicating the possible values objects can possess for that property.

Definition 1. Given a set of classes $C \subset \mathbf{C}$, the set of *BHoM types* over C, denoted $\text{types}(C)$, is the minimal set defined recursively as follows. If $t \in C \cup \mathbf{T}$ then $t \in \text{types}(C)$. If $t_1, t_2 \in \text{types}(C)$ then $\text{set}(t_1)$, $\text{bag}(t_1)$, $\text{list}(t_1)$, and $\text{dict}(t_1, t_2)$ belong to $\text{types}(C)$.

Example 3. Consider both definitions of class Panel in Examples 1 and 2, and the set $C = \{\text{ISurfaceProperty}, \text{Edge}\}$. ISurfaceProperty is a BHoM type over C because $\text{ISurfaceProperty} \in C$, double is a BHoM type over C because $\text{double} \in \mathbf{T}$, $\text{list}(\text{Edge})$ is a BHoM type over C because $\text{Edge} \in C$, and $\text{dict}(\text{Frequency}, \text{double})$ is a BHoM type over C because $\text{Frequency} \in \mathbf{T}$ and $\text{double} \in \mathbf{T}$

Alternatively, we could have defined types over $\mathbf{C}$. Definition 1 restrict types over a subset C because we want to define types for a specific discipline. We next state (Definition 2) that the schema of discipline is restricted to a finite set of classes.

Definition 2. A *BHoM disciplinary schema* S is a quin $(C, P, \sqsubseteq, \text{dom}, \text{range})$ where:

1. C and P are finite subsets of $\mathbf{C}$ and $\mathbf{P}$, respectively,
2. $\sqsubseteq$ is a partial order in $\mathbf{C}$,
3. $\text{dom} : \mathbf{P} \to C$ is a function called the *property domain*,
4. $\text{range} : \mathbf{P} \to \text{Types}(C)$ is a function called the *property range*.

Given a disciplinary schema $S = (c, P, \sqsubseteq, \text{dom}, \text{range})$, a class $c \in C$ and a property $p \in P$, we said that property p is a *property* of class c in the disciplinary schema S, denoted $c.p \in S$, if there exists a class $c' \in C$ such that $\text{dom}(p) = c'$ and $c \sqsubseteq c'$. We write $\text{properties}(c, S)$ to denote the set $\{p : c.p \in S\}$.

Example 4. Another example:

In C# it is possible to define two classes with properties with the same name and different data types. This is precluded by Definition 2. Indeed, it is not difficult to see that if two different classes c_1 and c_2 have the same property p in a disciplinary schema S (i.e., $c_1.p \in S$ and $c_2.p \in S$), then there exists a class c such that $c_1 \sqsubseteq c$ and $c_2 \sqsubseteq c$ and $c.p \in S$.

We next define the databases that satisfy a BHoM disciplinary schema.

Definition 3. A *database* over a disciplinary schema $S = (C, P, \sqsubseteq, \text{dom}, \text{range})$ is a triple $(O, \text{type}, \text{value})$ where:

1. O is a finite set disjoint with $\mathbf{C}$, $\mathbf{T}$, $\mathbf{P}$, called the *set of objects*,
2. $\text{type} : O \to C \cup \mathbf{T}$ is a function,
3. value is a partial function such that $\text{dom}(\text{value}) \subseteq \{(o, p) : \text{type}(o).p \in S\}$, and if $\text{value}(o, p)$ is defined then:
   (a) $\text{type}(\text{value}(o, p)) \sqsubseteq \text{range}(p)$ if $\text{range}(p) \in \mathbf{C}$,
   (b) $\text{type}(\text{value}(o, p)) = \text{range}(p)$ if $\text{range}(p) \in \mathbf{T}$.

By defining disciplinary schemas, the BHoM framework enables the encapsulation of discipline-specific attributes and constraints, facilitating efficient data management and interoperability across diverse design domains. This illustrates how the BHoM framework supports domain-specific modeling and data integration, laying the groundwork for further advancements in collaborative parametric design.

A.2. The parallels between BHoM's and SWTs

The object-oriented data model structures data into objects—instances of classes that encapsulate attributes and methods. Object-Oriented Programming (OOP) integrates data and behavior within objects, a concept widely used in CAD and BIM software. Object-Relational Mappers (ORMs) are tools that map object-oriented programming classes to relational database tables, enabling developers to interact with databases using their programming language's syntax [74]. BHoM follows an object-oriented model but separates functions from data. This separation makes BHoM similar to a virtual object database like those used to convert objects to relational databases — ORMs. Separating data from behavior (functions) enables more flexible and loosely coupled code, as seen in the BHoM approach for compatibility with visual data flow modeling [16]. This separation also aligns BHoM's data representation more closely with that of SWTs. Table 4 compares BHoM and SWTs in terms of identifiers, database models, data schema, and querying methods. SWTs rely on RDF Schema (RDFS), OWL, and Shapes Constraint Language (SHACL) to define schema constraints and terminological knowledge for RDF graphs. However, the actual data schema for transferred content is determined by domain-specific ontologies. These ontologies define structured representations of AEC concepts [75]. BHoM uses JSON key–value pairs for data exchange, while SWTs support formats such as TTL, N-Triples, JSON-LD, and RDF/XML. Querying in SWTs is performed using SPARQL, whereas BHoM relies on framework-specific functions in the BHoM_Engine Query classes [76]. For example, a representation of a room in the Architectural BHoM namespace is:

Table 4
BHoM and SWTs comparison [16].

	Identifiers	Database model	Data schema	Data exchange format	Querying
BHoM	GUID	Object-Oriented	BHoM Namespaces and BHoM classes	JSON	BHoM_Engine Query and MongoDB
SWTs	URI	RDF	Domain Ontologies (e.g., ifcOWL, BOT, or project-specific ontologies)	TTL, N-Triples, JSON-LD, RDF/XML	SPARQL

```
namespace BH.oM.Architecture.Elements
{
    public class Room : BHoMObject, IRegion, IElement2D
    {
        public virtual ICurve Perimeter { get; set; } = null;
        public virtual Point Location { get; set; } = null;
    }
 }
```

Listing 3: Representation of a room in the Architectural BHoM namespace

Appendix B. BHoM to OWL/RDF translation and vice-versa

To ensure that building designers use SWTs, this paper suggests that tools in the AEC world should be able to read and write RDF data. This is one of the main reasons that in this paper we develop a bidirectional converter from building data to OWL/RDF using BHoM, because the BHoM is already accessible through many AEC software. To make this integration possible, we develop BHoM methods that translate any object to RDF, including its ontology specifications within a design software. Furthermore, the RDF graph output is directly linked to a graph database, where data exploration and semantic queries can be done. Similarly, if changes are made, or new graphs are constructed, the RDF graph can be fed into AEC design tools. The translation is made with the idea of backwards compatibility in mind (OWL/RDF to BHoM objects). The BHoM to OWL/RDF converter is an open-source project written in C# and is available on GitHub.

In terms of data translators, in this paper three key components are developed: (1) the converter from BHoM objects to OWL/RDF, (2) the converter back from OWL/RDF to BHoM objects, and (3) a connector to a graph database GraphDB [77]. Through this paper, we exemplify this process on Grasshopper, as a popular software among AEC professionals [46] and has a user-friendly visual interface that makes working with SWTs easier. However, the translation to graph can be done in any design software supported by BHoM such as Revit, Dynamo, Excel, and so on.

The converter from BHoM objects to OWL/RDF generates an RDF graph serialized in turtle format. For existing BHoM objects, the naming conventions translated to RDF are directly influenced by their original name and namespace. For example, a BHoM polyline translated to RDF is defined as: `:BH.oM.Geometry.Polyline`, where `:BH.oM.Geometry` indicates the namespace the `Polyline` resides in. In a knowledge graph, the terminological part sets the structure, including the schema or ontology. The assertional part fills in the content of a knowledge graph with project-specific data. The following subsections present the terminological translation from BHoM to OWL/RDF, the assertional translation from BHoM to OWL/RDF, and the translation back from OWL/RDF to BHoM.

B.1. Translation of terminologies - BHoM to OWL/RDF

Identifiers. We let the user define a base URI from the Grasshopper component for OWL classes that the generated ontology outputs. The defined URI is the base web address of the ontology, to which a prefix can be introduced from the same component. E.g.:

```
@base <http://bhom.xyz/bhOWL/>.
@prefix : <> .
```

To avoid confusion between native BHoM and non-BHoM objects, we assume that all classes used within one project belong to the same generated ontology. However, future work will store native BHoM objects in a certain URI, using a bhOWL prefix, whereas only non-native objects URI and base address will be customized from the Grasshopper components.

Classes. Every BHoM class (e.g. BHoM_Room) is mapped to a class in OWL. BHoM interfaces are ubiquitous and often used to represent concepts in a taxonomic manner. Since BHoM C# interfaces include properties that each class that implements the interface has, we considered all BHoM C# interfaces (such as IElement2D or ICurve.) when translating to OWL/RDF to be mapped to an OWL class. In this case, a single object, such as a "Room" (see Listing 3) becomes an owl:subClassOf OWL classes BHoMObject, IRegion, and IElement2D (including superclasses, and interfaces that help to construct them). The interface IElement1D represents all types with a linear (1 dimensional) representation, e.g., columns, beams, pipes, etc.; in this case, every class with a linear representation becomes a subclass of owl:IElement1D class. So, both BHoM classes and BHoM interfaces are mapped to OWL classes, and class inheritance relations and interface implementation relations are represented with a rdfs:subClassOf predicate. A definition of a Room BHoM class and IRegion interface is in OWL is defined as:

```
:BH.oM.Architecture.Elements.Room rdf:type owl:Class;
                                  rdfs:subClassOf BH.oM.Analytical.Elements#IRegion;
                                  rdfs:subClassOf BH.oM.Base.BHoMObject;
                                  rdfs:label "Room"@en .
```

```
:BH.oM.Analytical.Elements.IRegion rdf:type owl:Class;
                                   rdfs:subClassOf :BH.oM.Base.IBHoMObject;
                                   rdfs:subClassOf :BH.oM.Dimensional.IElement2D;
                                   rdfs:label "IRegion"@en .
```

Custom Classes. BHoM allows the creation of custom classes. To map such classes to OWL classes, we force the usage of an input "Type" in the initial BHoM object. Such labeling and naming conventions help to create ontology classes with newly defined custom classes, which can result in specific re-usable ontologies (for example, timber building ontology). The advantage of such work is that the ontology is generated while designing; it is flexible enough to be extended and guide the design process.

Object Properties and Data Properties. Object properties and data properties are extracted from each C# class, and mapped to their corresponding RDF type. For example, the "Perimeter" property, is a property of the C# interface IRegion which the class Room implement. In this case the object Perimeter in OWL is defined as:

```
:BH.oM.Architecture.Elements.Room.Perimeter  rdf:type owl:ObjectProperty;
                                             rdfs:domain :BH.oM.Analytical.Elements.IRegion;
                                             rdfs:range :BH.oM.Geometry.ICurve;
                                             rdfs:label "Perimeter"@en."
```

On the other hand, datatype properties are expressed with literals following the XSD schema, following the SWTs stack. For example, each BHoM object has a GUID, expressed with a string:

```
:BH.oM.Base.BHoMObject.BHoM_Guid rdf:type owl:DatatypeProperty;
                                 rdfs:domain :BH.oM.Base.BHoMObject;
                                 rdfs:range xsd:string;
                                 rdfs:label "BHoM_Guid"@en .
```

B.2. Translation of assertions - BHoM to OWL/RDF

This subsection delves into the translation process of assertions, starting with unique identifier creation for new assertions by assigning names and IDs to BHoM objects. It outlines the methodology of constructing URIs based on domain names and provides a detailed illustration. The discussion extends to the translation of literals, emphasizing the mapping of primitive C# data types to RDF and addressing the intricacies of handling dictionaries and lists. Additionally, the section explores the representation of RDF ordered collections and elucidates the translation challenges associated with geometrical data in BHoM.

Identifiers. To create new assertions, we first need to give them a name and a unique ID. Each BHoM object already has a GUID. Following linked building data principles, IDs need to be URI built on domain names. Therefore, we provide a mechanism within the developed Grasshopper components that builds a unique URI based on a given domain concatenated with the GUID of the instance.

```
<http://www.example.com/3374A3DC31B4FDF6F> rdf:type owl:NamedIndividual,  :BH.oM.Architecture.Elements.Room;
                                           :BH.oM.Base.BHoMObject.BHoM_Guid "aea3f8ad-bc636"^^xsd:string.
```

Literals. The RDF and OWL recommendations use the simple types from XML Schema. Whereas translation of primitive C# datatypes (such as strings, integers, boolean, etc.) are directly mapped to primitive XML datatypes, the translation of dictionaries, enums and lists requires a different approach.

In RDF ordered collections can be represented using the RDF Collection vocabulary and the rdf:Seq type. An RDF list is a series of nodes, each linked to the next using the property rdf:rest, with the final node linked to a blank node (also known as a "bNode") labeled rdf:nil. The actual values in the list are represented as properties of each node, with the property name typically being rdf:first. A rdf:Seq is a specialized collection that maintains the order of its members. Each member in the sequence is assigned a unique integer-based position. On the other hand, since BHoM is a .NET-based software, in BHoM lists can be created, modified and manipulated in various ways, similar to lists in C#. To simplify serialization, we map BHoM lists to rdf:Seq instead of rdf:list. An example of an BHoM polyline which contains three control points in RDF is present below:

```
<http://www.example.com/2FB125F5C85CB3BDF08>  rdf:type owl:NamedIndividual, :BH.oM.Geometry.Polyline, rdf:Seq;
                                          rdf:_1 <http://www.example.com/50FC7044566B5230E41A962C> ;
                                          rdf:_2 <http://www.example.com/DA505DCF960046F6025394F5> ;
                                          rdf:_3 <http://www.example.com/9B9F6FE119128B7E9F3B901C> .
```

Geometry. The translation of geometrical data is dependent on BHoM's geometry computation method. In BHoM, a geometry is represented using the IGeometry interface and its implementing class Geometry_oM. The Geometry_oM class contains a collection of objects that represent the geometry's defining curves, meshes, surfaces, vectors, etc. For example, a line is represented using the ICurve interface and its implementing class Line. The Line class contains information such as the start and end points of the line, which define its geometry, as well as additional properties such as its length and direction. This information is stored in the Line object and can be used to represent a line in a threedimensional model. When translating to OWL/RDF a BHoM line defined by two points is described as it follows:

```
<http://www.example.com/2FB125F55CB3BDF08> rdf:type owl:NamedIndividual, :BH.oM.Geometry.Line;
                                       :BH.oM.Geometry.Line.Start <http://www.uni-stuttgart.de/620E23163FAEFE>;
                                       :BH.oM.Geometry.Line.End <http://www.uni-stuttgart.de/0E40647D8182FC1A>;
                                       :BH.oM.Geometry.Line.Infinite "False" xsd:boolean.
```

```
<http://www.uni-stuttgart.de/620E23163FAEFE> rdf:type owl:NamedIndividual, :BH.oM.Geometry.Point;
                                             :BH.oM.Geometry.Point.X "0"^^xsd:double;
                                             :BH.oM.Geometry.Point.Y "0"^^xsd:double;
                                             :BH.oM.Geometry.Point.Z "0"^^xsd:double.
```

```
<http://www.uni-stuttgart.de/620E23163FAEFE> rdf:type owl:NamedIndividual, :BH.oM.Geometry.Point;
                                             :BH.oM.Geometry.Point.X "8"^^xsd:double;
                                             :BH.oM.Geometry.Point.Y "22"^^xsd:double;
                                             :BH.oM.Geometry.Point.Z "-1"^^xsd:double.
```

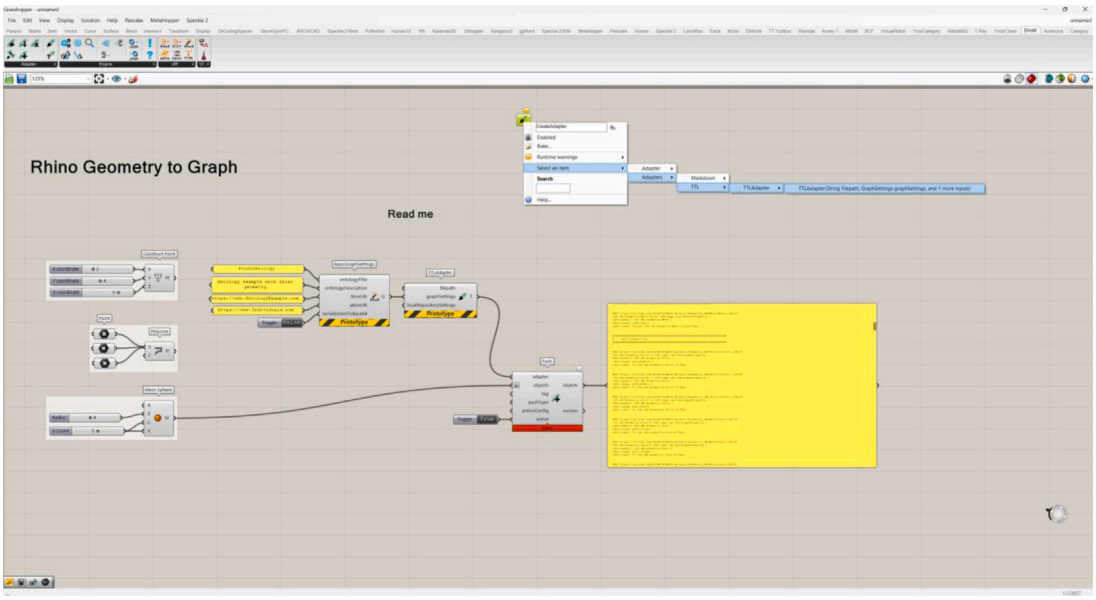

Fig. 15. SWT within Grasshopper through BHoM user interface.

While complex meshes might end up very large in RDF, we have also developed a way to serialize the geometry and store it as a string in Base64 encoding. This means geometrical types would not be ontology classes, and there will be only one geometrical node in the graph for each object (e.g., column). This method still allows for deserialization to BHoM objects and design software; however, such storage would not allow AEC designers to query geometric information alongside semantic ones in the graph. Therefore, we leave the option to the designers to decide when to serialize the geometry and when not to.

B.3. Translation back from OWL/RDF to BHoM

The translation from the serialized OWL/RDF format back to BHoM (deserialization) takes the serialized file in the target format and reads it with a C# library called DotNetRDF. Using this library allows to support different file formats upon deserialization, although only TTL is tested and verified, being the only format currently supported in the serialization from BHoM. The DotNetRDF read returns the individual nodes' data. The destination schema is derived from the local installation of BHoM. The BHoM objects corresponding to individuals are instantiated using the appropriate BHoM C# types, whose name is extrapolated from the serialized text. The deserialization supports every type supported in the serialization, including Custom Types and the supported collections (lists/Seq).

Appendix C. SWT within Grasshopper

SWT within Grasshopper through BHoM User Interface is illustrated in Fig. 15.

Appendix D. Installation and update instructions

To ensure full functionality of the BHoM framework and RDF conversion tools, follow these installation steps:

1. Download and run the BHoM Installer from BHoM Website.
2. Download the latest release from the "Assets" section and unzip the file.
3. Run `Run_installer.bat` to fetch additional files from GitHub and finalize the installation.
4. A command window will confirm the progress and success of the installation.

For ontology development and additional functionality:

- Install GitHub Desktop and clone the BHoM repository.
- Open Grasshopper example files from the repository.
- If encountering "could not find local repository" errors, set the `GitRootPath` in the `LocalRepositorySettings` component to the parent directory of the BHoM repository.
- If BHoM components do not appear in Grasshopper, uninstall and reinstall BHoM.

To update RDF_Prototypes to the latest commit:

- Repeat steps 2 and 3 from the installation process.

For more details, visit the BHoM RDF GitHub Wiki.

Data availability

The data supporting this study is stored at https://doi.org/10.18419/darus-4360.

References

[1] D. Elshani, T. Wortmann, S. Staab, Towards better co-design with disciplinary ontologies: Review and evaluation of data interoperability in the AEC industry, in: P. Pauwels, M. a Poveda-Villalón, W. Terkaj (Eds.), Proceedings of the 10th Linked Data in Architecture and Construction Workshop, LDAC 2022, 3213, CEUR Workshop Proceedings, 2022, pp. 43–52, URL: http://ceur-ws.org/Vol-3213/ urn:nbn:de:0074-3213-6.

[2] J. Knippers, C. Kropp, A. Menges, O. Sawodny, D. Weiskopf, Integrative computational design and construction: Rethinking architecture digitally, Civ. Eng. Des. 3 (4) (2021) 123–135, http://dx.doi.org/10.1002/cend.202100027.

[3] D. Elshani, D. Hernandez, A. Lombardi, L. Siriwardena, T. Schwinn, A. Fisher, S. Staab, A. Menges, T. Wortmann, Building Information Validation and Reasoning Using Semantic Web Technologies, in: M. Turrin, C. Andriotis, A. Rafiee (Eds.), Computer-Aided Architectural Design. INTERCONNECTIONS: Co-Computing beyond Boundaries, 1819, Springer Nature Switzerland, Cham, 2023, pp. 470–484, http://dx.doi.org/10.1007/978-3-031-37189-9_31, Series Title Communications in Computer and Information Science.

[4] N. Lässig, M. Herschel, A. Reichle, C. Ellwein, A. Verl, The ArchIBALD data integration platform: Bridging fragmented processes in the building industry, in: J. De Weerdt, A. Polyvyanyy (Eds.), Intelligent Information Systems, in: Lecture Notes in Business Information Processing, 452, Springer International Publishing, 2022, pp. 45–54, http://dx.doi.org/10.1007/978-3-031-07481-3_6, Series Title: Lecture Notes in Business Information Processing.

[5] C. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers, and Contractors, first, Wiley, 2008, http://dx.doi.org/10.1002/9780470261309.

[6] C.S. Dossick, G. Neff, Organizational Divisions in BIM-Enabled Commercial Construction, J. Constr. Eng. Manag. 136 (4) (2010) 459–467, http://dx.doi.org/10.1061/(ASCE)CO.1943-7862.0000109.

[7] F. Hamzeh, V.A. González, L.F. Alarcon, S. Khalife, Lean construction 4.0: Exploring the challenges of development in the AEC industry, in: Proc. 29th Annual Conference of the International Group for Lean Construction, IGLC, Lima, Peru, 2021, pp. 207–216, http://dx.doi.org/10.24928/2021/0181.

[8] R.K. Soman, J.K. Whyte, Codification Challenges for Data Science in Construction, J. Constr. Eng. Manag. 146 (7) (2020) 04020072, http://dx.doi.org/10.1061/(ASCE)CO.1943-7862.0001846.

[9] B.C. Paulson, Designing to Reduce Construction Costs, J. Constr. Div. 102 (4) (1976) 587–592, http://dx.doi.org/10.1061/JCCEAZ.0000639.

[10] M. Levine, D. Ürge-Vorsatz, K. Blok, L. Geng, D. Harvey, S. Lang, G. Levermore, A. Mongameli Mehlwana, S. Mirasgedis, A. Novikova, J. Rilling, H. Yoshino, Residential and commercial buildings, in: B. Metz, O. Davidson, P. Bosch, R. Dave, L. Meyer (Eds.), Climate Change 2007: Mitigation. Contribution of Working Group III To the Fourth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press, Cambridge, UK and New York, NY, USA, 2007, http://dx.doi.org/10.1017/CBO9780511546013.

[11] A. Borrmann, J. Beetz, C. Koch, T. Liebich, S. Muhic, Industry Foundation Classes: A Standardized Data Model for the Vendor-Neutral Exchange of Digital Building Models, in: A. Borrmann, M. König, C. Koch, J. Beetz (Eds.), Building Information Modeling, Springer International Publishing, Cham, 2018, pp. 81–126, http://dx.doi.org/10.1007/978-3-319-92862-3_5.

[12] G. Sibenik, I. Kovacic, Assessment of model-based data exchange between architectural design and structural analysis, J. Build. Eng. (ISSN: 2352-7102) Vol -32 (2020) 101589, http://dx.doi.org/10.1016/j.jobe.2020.101589.

[13] B. Happold, BHoM: The buildings and habitats object model, 2017, URL: https://bhom.xyz/ (Accessed 16 June 2024) Online.

[14] D. Stefanescu, M. Cominetti, A. Rynne, Speckle systems, 2024, URL: https://speckle.systems/, (Accessed 16 June 2024) Online.

[15] P. Janssen, K.W. Chen, Visual dataflow modelling: A comparison of three systems, in: Computer Aided Architectural Design Futures 2011: Proceedings of the 14th International Conference on Computer Aided Architectural Design Futures, Liège, Belgium, ISBN: 9782874561429, 2011, pp. 801–816.

[16] D. Elshani, A. Lombardi, A. Fisher, D. Hernandez, S. Staab, T. Wortmann, Knowledge Graphs for Multidisciplinary Co-Design: Introducing RDF to BHoM, in: P. Pauwels, M. a Poveda-Villalón, W. Terkaj (Eds.), Proceedings of the 10th Linked Data in Architecture and Construction Workshop, LDAC 2022, Vol.3213, CEUR Workshop Proceedings, (ISSN: 1613-0073) 2022, pp. 32–42, URL: http://ceur-ws.org/Vol-3213/ urn:nbn:de:0074-3213-6.

[17] D. Fahland, D. Lubke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, S. Zugal, Declarative versus imperative process modeling languages: The issue of understandability, in: T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, R. Ukor (Eds.), Enterprise, Business-Process and Information Systems Modeling, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 353–366, http://dx.doi.org/10.1007/978-3-642-01862-6_29.

[18] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G.D. Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, A.-C.N. Ngomo, A. Polleres, S.M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge graphs, 54, (4) Association for Computing Machinery, New York, NY, USA, 2021, http://dx.doi.org/10.1145/3447772,

[19] P. Pauwels, K. McGlinn (Eds.), Buildings and Semantics: Data Models and Web Technologies for the Built Environment, First edition, CRC Press/Balkema, Leiden, The Netherlands ; Boca Raton, FL, 2023, http://dx.doi.org/10.1201/9781003204381.

[20] F. Xue, L. Wu, W. Lu, Semantic enrichment of building and city information models: A ten-year review, Adv. Eng. Informatics 47 (2021) 01245, http://dx.doi.org/10.1016/j.aei.2020.101245.

[21] X. han Shen, S.M. Sepasgozar, M.J. Ostwald, Knowledge-based semantic web technologies in the AEC sector, Autom. Constr. 167 (2024) 05686, http://dx.doi.org/10.1016/j.autcon.2024.105686.

[22] D. Elshani, A. Lombardi, A. Fisher, D. Hernandez, S. Staab, T. Wortmann, Inferential reasoning in co-design using semantic web standards alongside BHoM, in: S. Slepicka, L. Kolbeck, S. Esser, K. Forth, F. Noichl, J. Schlenger (Eds.), Proceedings of 33. Forum Bauinformatik, Lehrstuhl für Computergestützte Modellierung und Simulation, Technische Universität München, 2022, pp. 89–97, http://dx.doi.org/10.14459/2022md1686600.

[23] R. Sacks, Z. Wang, B. Ouyang, D. Utkucu, S. Chen, Toward artificially intelligent cloud-based building information modelling for collaborative multidisciplinary design, Adv. Eng. Informatics 53 (2022) 01711, http://dx.doi.org/10.1016/j.aei.2022.101711.

[24] T. Hartmann, A. Trappey, Advanced engineering informatics - philosophical and methodological foundations with examples from civil and construction engineering, Dev. Built Environ. 4 (2020) 00020, http://dx.doi.org/10.1016/j.dibe.2020.100020.

[25] P. Pauwels, D. Shelden, J. Brouwer, D. Sparks, S. Nirvik, T.P. McGinley, Open data standards and BIM on the cloud, in: Buildings and Semantics: Data Models and Web Technologies for the Built Environment, first Edition, CRC Press, London, 2022, http://dx.doi.org/10.1201/9781003204381-6.

[26] A. Dresch, D.P. Lacerda, J.A.V. Antunes Jr., Design Science Research: A Method for Science and Technology Advancement, Springer International Publishing, Cham, 2015, http://dx.doi.org/10.1007/978-3-319-07374-3.

[27] J. Werbrouck, P. Pauwels, J. Beetz, E. Mannens, Data Patterns for the Organisation of Federated Linked Building Data, in: P. Pauwels, M. a Poveda-Villalón (Eds.), Proceedings of the 9th Linked Data in Architecture and Construction Workshop, 3081, CEUR Workshop Proceedings, (ISSN: 1613-0073) 2021, pp. 79–90, URL: https://ceur-ws.org/Vol-3081/ urn:nbn:de:0074-3081-7.

[28] J. Haymaker, C. Kam, M. Fischer, A methodology to plan, communicate and control multidisciplinary design processes, in: R.J. Scherer, P. Katranuschkov, S.-E. Schapke (Eds.), Proceedings of the 22nd CIB W78 Conference on Information Technology in Construction, CIB Publication 304, ISBN: 3-86005-478-3, 2005, pp. 117–124, URL: http://itc.scix.net/paper/w78-2005-p2-1-haymaker.

[29] Q. Yang, Y. Zhang, Semantic interoperability in building design: Methods and tools, Computer- Aided Des. 38 (10) (2006) 1099–1112, http://dx.doi.org/10.1016/j.cad.2006.06.003.

[30] A. Borrmann, M. König, C. Koch, J. Beetz (Eds.), Building Information Modeling Technology Foundations and Industry Practice: Technology Foundations and Industry Practice, Springer International Publishing, Cham, Switzerland, 2018, http://dx.doi.org/10.1007/978-3-319-92862-3.

[31] E.M. Sanfilippo, S. Borgo, What are features? An ontology-based review of the literature, Computer- Aided Des. 80 (2016) 9–18, http://dx.doi.org/10.1016/j.cad.2016.07.001.

[32] J.F. Tchouanguem Djuedja, F.H. Abanda, B. Kamsu-Foguem, P. Pauwels, C. Magniont, M.H. Karray, An integrated Linked Building Data system: AEC industry case, Adv. Eng. Softw. 152 (2021) 02930, http://dx.doi.org/10.1016/j.advengsoft.2020.102930.

[33] P. Pauwels, D. Van Deursen, R. Verstraeten, J. De Roo, R. De Meyer, R. Van de Walle, J. Van Campenhout, A semantic rule checking environment for building performance checking, Autom. Constr. 20 (5) (2011) 506–518, http://dx.doi.org/10.1016/j.autcon.2010.11.017.

[34] S. Esser, A. Borrmann, A system architecture ensuring consistency among distributed, heterogeneous information models for civil infrastructure projects, in: V. Semenov, R.J. Scherer (Eds.), ECPPM 2021: EWork and EBusiness in Architecture, Engineering and Construction, CRC Press, 2021, pp. 596–603, http://dx.doi.org/10.1201/9781003191476-8.

[35] International Organization for Standardization, ISO 19650-1:2018: Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) — Information management using building information modelling — Part 1: Concepts and principles, Technical Report ISO, first ed., International Organization for Standardization, Geneva, Switzerland, 2018, http://dx.doi.org/10.3403/30333757.

[36] C. Zhang, J. Beetz, M. Weise, Interoperable validation for IFC building models using open standards, J. Inf. Technol. Constr. (ITcon) 20 (2015) 24–39, http://dx.doi.org/10.36680/j.itcon.2015.002, Special Issue: ECPPM 2014 - 10th European Conference on Product and Process Modelling.

[37] P. Bourreau, N. Charbel, J. Werbrouck, M. Senthilvel, P. Pauwels, J. Beetz, Multiple inheritance for a modular BIM, in: S. Marques, R. Teulier (Eds.), Le BIM Et L'Évolution Des Pratiques: IngÉnierie Et Architecture, Enseignement Et Recherche, Éditions Eyrolles, ISBN: 978-2-212-67989-2, 2020, pp. 63–82.
[38] G.B.N.C. Framework, Exchange Information Requirements (EIR) Guidance, Technical Report,, Centre for Digital Built Britain (CDBB), 2021, URL: https://www.cdbb.cam.ac.uk/files/eir_guidance.pdf Available from the Global BIM Network Collaborative Framework.
[39] D.I. für Normung, Building Information Modeling (BIM) - Classification according to STLB-Bau, DIN SPEC 91400, Berlin, Germany, 2017, http://dx.doi.org/10.31030/2608112.
[40] B. Toth, P. Janssen, R. Stouffs, A. Chaszar, S. Boeykens, Custom digital workflows: A new framework for design analysis integration, Int. J. Archit. Comput. 10 (2012) 481–500, http://dx.doi.org/10.1260/1478-0771.10.4.481.
[41] buildingSMART International, Industry foundation classes (IFC), 2018, URL: https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/ ISO 16739-1:2018 certified.
[42] M. Boon, The role of disciplinary perspectives in an epistemology of scientific models, Eur. J. Philos. Sci. 10 (2020) http://dx.doi.org/10.1007/s13194-020-00295-9.
[43] J. Oraskari, O. Schulz, J. Werbrouck, J. Beetz, Enabling Federated Interoperable Issue Management in a Building and Construction Sector, in: Proceedings of the 29th EG-ICE International Workshop on Intelligent Computing in Engineering, EG-ICE, 2022, pp. 92–101, http://dx.doi.org/10.7146/aul.455.c200.
[44] J. Werbrouck, O. Schulz, J. Oraskari, E. Mannens, P. Pauwels, J. Beetz, A generic framework for federated CDEs applied to Issue Management, Adv. Eng. Informatics 58 (2023) 102136, http://dx.doi.org/10.1016/j.aei.2023.102136, URL: https://linkinghub.elsevier.com/retrieve/pii/S1474034623002641.
[45] Y.-C. Lee, C.M. Eastman, J.-K. Lee, Validations for ensuring the interoperability of data exchange of a building information model, Autom. Constr. 58 (2015) 176–195, http://dx.doi.org/10.1016/j.autcon.2015.07.010.
[46] T. Wortmann, J. Cichocka, C. Waibel, Simulation-based optimization in architecture and building engineering—Results from an international user survey in practice and research, Energy Build. 259 (2022) 111863, http://dx.doi.org/10.1016/j.enbuild.2022.111863.
[47] M. Senthilvel, J. Beetz, A visual programming approach for validating linked building data, in: Proceedings of the EG-ICE 2020 Workshop on Intelligent Computing in Engineering, Berlin, Germany, 2020, http://dx.doi.org/10.14279/depositonce-9977.
[48] S. Reiss, M. Renieris, Encoding program executions, in: Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001, IEEE Comput. Soc, Toronto, Ont., Canada, 2001, pp. 221–230, http://dx.doi.org/10.1109/ICSE.2001.919096.
[49] M. Bolpagni, R. Gavina, D. Ribeiro (Eds.), Industry 4.0 for the Built Environment: Methodologies, Technologies and Skills, Structural Integrity, Springer, Cham, 2022, number 20 in Structural Integrity, ISBN: 978-3-030-82430-3 978-3-030-82429-7.I.
[50] A. Lombardi, Interoperability challenges: Exploring trends, patterns, practices and possible futures for enhanced collaboration and efficiency in the aec industry, in: P. Ruttico (Ed.), Coding Architecture: Designing Toolkits, Workflows, Industry, Springer Nature Switzerland, Cham, 2024, pp. 49–72, http://dx.doi.org/10.1007/978-3-031-47913-7_3.
[51] G.D. Giacomo, M. Lenzerini, TBox and ABox reasoning in expressive description logics, in: L.C. Aiello, J. Doyle, S.C. Shapiro (Eds.), Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning, KR'96, Morgan Kaufmann, Cambridge, Massachusetts, USA, ISBN: 1-55860-421-9, 1996, pp. 316–327.
[52] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 concepts and abstract syntax, 2014, URL: https://www.w3.org/TR/rdf11-concepts/ World Wide Web Consortium, Recommendation REC-RDF11-Concepts-20140225.
[53] S. Harris, A. Seaborne, SPARQL 1.1 query language for RDF, 2013, URL: https://www.w3.org/TR/sparql11-query/ World Wide Web Consortium, Recommendation REC-sparql11-query-20130321.
[54] J. Beetz, J. van Leeuwen, B. de Vries, IfcOWL: A case of transforming EXPRESS schemas into ontologies, Artif. Intell. Eng. Des. Anal. Manuf. 23 (1) (2009) 89–101, http://dx.doi.org/10.1017/S0890060409000122.
[55] C. Anumba, R. Issa, J. Pan, I. Mutis, Ontology-based information and knowledge management in construction, Constr. Innov.: Inf. Process. Manag. 8 (2008) 218–239, http://dx.doi.org/10.1108/14714170810888976.
[56] M.H. Rasmussen, M. Lefrançois, P. Pauwels, C.A. Hviid, J. Karlshøj, Managing interrelated project information in AEC knowledge graphs, Autom. Constr. 108 (2019) 102956, http://dx.doi.org/10.1016/j.autcon.2019.102956.
[57] M.H. Rasmussen, P. Pauwels, M. Lefrançois, G.F. Schneider, C.A. Hviid, J. Karlshøj, Recent changes in the building topology ontology, in: Proceedings of the 5th Linked Data in Architecture and Construction Workshop (LDAC 2017), 2017, http://dx.doi.org/10.13140/RG.2.2.32365.28647.
[58] A. Wagner, W. Sprenger, C. Maurer, T.E. Kuhn, U. Rüppel, Building product ontology: Core ontology for linked building product data, Autom. Constr. 133 (2022) 103927, http://dx.doi.org/10.1016/j.autcon.2021.103927.
[59] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M.B. Kjæ rgaard, M. Srivastava, K. Whitehouse, Brick: Towards a Unified Metadata Schema For Buildings, in: Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments, ACM, Palo Alto CA USA, 2016, pp. 41–50, http://dx.doi.org/10.1145/2993422.2993577.
[60] C. Peng, F. Xia, M. Naseriparsa, F. Osborne, Knowledge Graphs: Opportunities and Challenges, Artif. Intell. Rev. 56 (11) (2023) 13071–13102, http://dx.doi.org/10.1007/s10462-023-10465-9.
[61] J. Cao, E. Vakaj, R.K. Soman, D.M. Hall, Ontology-based manufacturability analysis automation for industrialized construction, Autom. Constr. 139 (2022) 104277, http://dx.doi.org/10.1016/j.autcon.2022.104277.
[62] P. Pauwels, S. Zhang, Y.-C. Lee, Semantic web technologies in AEC industry: A literature overview, Autom. Constr. 73 (2017) 145–165, http://dx.doi.org/10.1016/j.autcon.2016.10.003.
[63] A. Perisic, M. Lazic, B. Perisic, The Extensible Orchestration Framework approach to collaborative design in architectural, urban and construction engineering, Autom. Constr. 71 (2016) 210–225, http://dx.doi.org/10.1016/j.autcon.2016.08.005.
[64] D. Simeone, S. Cursi, M. Acierno, BIM semantic-enrichment for built heritage representation, Autom. Constr. 97 (2019) 122–137, http://dx.doi.org/10.1016/j.autcon.2018.11.004.
[65] Z. Wang, B. Ouyang, R. Sacks, Graph-based inter-domain consistency maintenance for BIM models, Autom. Constr. 154 (2023) 104979, http://dx.doi.org/10.1016/j.autcon.2023.104979.
[66] A. Donkers, D. Yang, d. Vries, N. Baken, A visual support tool for decision-making over federated building information, 2023, pp. 485–500, http://dx.doi.org/10.1007/978-3-031-37189-9_32.
[67] Q. Tong, Mapping Object-Oriented Database Models Into RDF(s), IEEE Access 6 (2018) 47125–47130, http://dx.doi.org/10.1109/ACCESS.2018.2867152.
[68] P. Jezek, R. Mouček, Semantic framework for mapping object-oriented model to semantic web languages, Front. Neuroinformatics 9 (2015) 3, http://dx.doi.org/10.3389/fninf.2015.00003.
[69] D. Elshani, A. Lombardi, A. Wagner, Bhom_rdf_prototypes, 2021, URL: https://github.com/BHoM/RDF_Prototypes.
[70] D. Elshani, A. Lombardi, D. Hernández, S. Staab, A. Fisher, T. Wortmann, Linked RDF graphs of an architectural and structural representation of a timber structure, 2024, http://dx.doi.org/10.18419/darus-4360.
[71] P.R. Aryan, F.J. Ekaputra, Sparqlgpviz: SPARQL graph pattern visualization, 2020, http://dx.doi.org/10.31219/osf.io/9sxe4.
[72] M. Bonduel, A. Wagner, P. Pauwels, M. Vergauwen, R. Klein, Including widespread geometry formats in semantic graphs using rdf literals, in: 2019 European Conference on Computing in Construction, European Council on Computing in Construction, 2019, pp. 341–350, http://dx.doi.org/10.35490/EC3.2019.
[73] E.C.D.-G. for Informatics, New European Interoperability Framework: Promoting Seamless Services and Data Flows for European Public Administrations, Publications Office of the European Union, 2017, http://dx.doi.org/10.2799/78681.
[74] V.E. Wolfengagen, L.Y. Ismailova, S.V. Kosikov, Model of conversion of data objects for defining the object-relation mapping, Procedia Comput. Sci. 123 (2018) 541–546, http://dx.doi.org/10.1016/j.procs.2018.01.082, 8th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2017 (Eighth Annual Meeting of the BICA Society), held August 1-6, 2017 in Moscow, Russia.
[75] P. Pareti, G. Konstantinidis, A review of SHACL: From data validation to schema reasoning for rdf graphs, in: Reasoning Web. Declarative Artificial Intelligence, in: Lecture Notes in Computer Science, vol. 13100 (2021) 115–144, http://dx.doi.org/10.1007/978-3-030-95481-9_6.
[76] L. Aw, L. Livermore, I. Kaplan, A Semantic Graph Query Language A Semantic Graph Query Language 1, Technical Report UCRL-TR- 225447, Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), 2006, http://dx.doi.org/10.13140/2.1.3429.0568.
[77] Ontotext, Graphdb, 2024, URL: https://graphdb.ontotext.com/documentation/10.4/ Version 10.4 Ontotext.