Department of the Built Environment

Integrating geometric validations with semantic web technologies, applied to the Dutch building regulations

7CC40 - Graduation project construction management and engineering - 40 EC

By

G.M.J. van Galen

Graduation Supervision Committee

First Supervisor	dr. ir.-arch Pieter Pauwels
Second Supervisor	dr. ir. Ekaterina Petrova
Third Supervisor	prof. dr. Michel R.V. Chaudron

Graduation

Program	Construction Management and Engineering (CME)
Academic Year	2024-2025
Date of Final Presentation	23-06-2025
Student ID	1502948
Study Load (EC)	40

This thesis has been realized in accordance with the regulations as stated in the TU/e Code of Scientific Conduct.
Disclaimer: The Department of CME of the Eindhoven University of Technology accepts no responsibility for the contents of MSc theses or practical training reports.
The information provided here is publicly available and may be freely accessed, used, and shared without restriction.

This page is intentionally left blank

Table of contents

1 Introduction . . . 1
1.1 Problem definition . . . 2
1.2 Research questions . . . 2
1.3 Research design . . . 3
1.4 Importance of the thesis . . . 3
1.5 Reading guide . . . 4
2 Literature study - automated geometric rule validation using semantic web technologies 5
2.1 Semantic web technologies . . . 5
2.1.1 Resource Description Framework . . . 5
2.1.2 Ontologies . . . 6
2.1.3 SPARQL . . . 8
2.1.4 SHACL . . . 9
2.2 Automated Compliance Checking . . . 9
2.2.1 Building model preparation . . . 10
2.2.2 Rule interpretation . . . 16
2.2.3 Rule execution . . . 18
2.2.4 Reporting of the results . . . 21
2.3 Geometric rule validation . . . 22
2.3.1 Geometric Kernels . . . 22
2.3.2 Geometry in semantics . . . 24
2.3.3 Geometric validation and functions . . . 25
2.4 Conclusion . . . 26
3 Methodology . . . 27
3.1 Developing a framework for geometric SHACL validation . . . 27
3.1.1 Methodological justification . . . 28
3.1.2 Deliverables . . . 29
3.1.3 Software architecture . . . 31
3.1.4 Research validation . . . 32
3.2 Adding geometric capabilities to SHACL . . . 32
3.2.1 Model translator . . . 32
3.2.2 Extracting geometric objects . . . 36
3.2.3 Geometric rule validation . . . 38
3.3 Web application . . . 42
3.3.1 Building model translation . . . 42
3.3.2 Rule interpretation and execution . . . 43
3.3.3 Reporting of the results . . . 45
4 Results . . . 47
4.1 Answering the research question . . . 47
4.2 Proof of concept . . . 48
4.3 Building model preparation . . . 49
4.3.1 Research validation . . . 50
4.3.2 Geometric computations . . . 55

5 Conclusion, discussion and future work 57
5.1 Conclusion 57
5.2 Discussion and future work 58
6 References 60

Summary

In the Architecture, Engineering and Construction (AEC) sector, collaboration between multiple disciplines is needed to develop complex projects. With the implementation of Building Information Modeling (BIM), a lot of this collaboration has been improved using digital processes. A big part of BIM is the use of Industry Foundation Classes (IFC) to share data between modeling software. For improved data collaboration and sharing, recent studies have shown that the use of semantic web technologies could be the next step in data access and sharing.

Within the AEC sector, the use of semantic web technologies is also known as the use of Linked Building Data (LBD). Linked building data uses technologies to represent and share building information in a computer -readable and software-independent way. Instead of relying on the large IFC data file, LBD splits the building data into RDF triples. These triples define relationships between building elements. These triples allow data to be integrated from different sources and support collaboration. Although LBD offers many advantages over the use of the IFC data structure, it currently lacks the support for geometric rule validation. This limitation prevents semantic web technologies from automatic rule validation. Since these spatial relations are not defined in the IFC data structure, an integration of geometric rule validation must be developed. Resent studies have developed tools to add the necessary geometric data in the model preparation phase so that semantic web technologies can use these properties for the validation of geometric rules. A method that is promising, but has some downsides. For instance, all the geometric data that are not added can also not be checked. This might lead to false positives or negatives in the testing results. Another promising study was done, which adds the geometric validation at runtime using new commands in a SPARQL query. This research showed promising results, but was based on older semantic web technologies.

The goal of this thesis is to analyze the current implementations of automated compliance checking and semantic web technologies to add run-time geometric validation. More specifically, the goal is to explore to what extent geometric rule validation can be performed using the geometry available in the IFC data structure without requiring manual enrichment of geometric properties. The literature study has shown that current SHACL implementations do not allow for the validation of geometric rules. The reason for this is that SHACL shapes can only validate the existence of relations and the literal values the represent. It has no understanding of the geometric value the literals represent and therefore cannot validate these rules. To solve this, the SHACL framework needed to be extended with these capabilities. To add this functionality, the thesis analyzes the four phases of automated compliance checking defined by Eastman and his colleagues. These four phases are the building model preparation phase, rule interpretation phase, rule execution phase, and the final phase where the results are reported. The preparation of the building model highlights the use of continuous data directly from the modeling software and the exporting of IFC data. How can the data be stored using semantic web technologies and is this format in accordance with the format needed for geometric rule validation. Technologies such as information delivery specifications are examined on their ability to structure data. For the rule interpretation phase, research is done on current technologies for the translation of rules into a machine-readable format. From manual rule interpretation to natural language processing. The rule execution phase analyzes the methods needed for geometric rule validation and how these rules can be executed. It is ana-

lyzed how semantic web technologies try and validate both within the AEC and other sectors. The geometric foundations are analyzed to determine how libraries are defined in other sectors. The reporting of these results and how these results can be used is analyzed from the export of text documents to computer readable formats for importing in modeling software to correct invalid objects.

To achieve the goal of the thesis, four deliverables have been developed to create a prototype that uses semantic web technologies to add geometric rule validation. Since adding data extraction to the modeling software, the decision has been made to start with the exported IFC data structure since the data is structured, well-defined and is seen as a standard in the AEC sector. The first deliverable parse the IFC data to a class library written in the C# language. The second deliverable extends the class library with a geometric kernel to extract geometric representations from the IfcShapeRepresentations. The third deliverable uses this geometric kernel to allow for validation of the IFC objects based on high-level verb methods (e.g. contains, underneath). The final deliverable is a web application that allows the parsing of IFC models and run-time rule validation based on semantic web technologies. The final deliverable is a full implementation of a web application that allows the user to upload an IFC file. Based on this IFC file an RDF graph is created on which geometric validation request can be done. Based on a validation request sent by the user, the software is able to collect the geometric representations from the IFC object. These objects are used in the validator from the third deliverable to determine compliance with the received request. This results in the answer to the question to what extend geometric rule validation can be added to the semantic web tech stack following current practices. Finally, the research questions are answered followed by a discussion regarding the conclusion and future steps that can be taken to build upon this research.

Samenvatting

In de bouwsector (AEC) is samenwerking tussen verschillende disciplines benodigd om complexe projecten te ontwikkelen. Met de implementatie van Building Information Modeling (BIM) is deze samenwerking sterk verbeterd. Een belangrijk onderdeel van BIM is het gebruik van Industry Foundation Classes (IFC) om data te delen tussen softwarepaketten. Recente studies hebben aangetoond dat het gebruik van semantische webtechnologieën de volgende stap zou kunnen zijn in de toegang tot en het delen van data, wat een verbeterde samenwerking en uitwisseling van data oplevert. Binnen de AEC-sector staat het gebruik van semantische webtechnologieën ook bekend als het gebruik van Linked Building Data (LBD). Linked Building Data maakt gebruik van technologieën om bouwinformatie op een computerleesbare en softwareonafhankelijke manier weer te geven en te delen.

In plaats van het gebruiken van grote IFC-databestand, splitst LBD de bouwdata op in RDF-triples. Deze RDF-triples definiëren relaties tussen bouwelementen. Deze triples maken het mogelijk om data uit verschillende bronnen met elkaar te verbinden. Hoewel LBD veel voordelen biedt ten opzichte van de IFC-datastructuur, ontbreekt momenteel de ondersteuning voor validatie van geometrische regels. Deze beperking maakt automatische validatie van regels door semantische webtechnologieën bijna onmogelijk. Omdat deze ruimtelijke relaties niet gedefinieerd zijn in de IFC-datastructuur, moet een integratie van de validatie van geometrische regels worden ontwikkeld. Recente studies hebben tools ontwikkeld om de benodigde geometrische data toe te voegen in de modelvoorbereidingsfase, zodat semantische webtechnologieën deze eigenschappen kunnen gebruiken voor de validatie van geometrische regels. Een veelbelovende methode, maar heeft enkele nadelen. Zo kunnen bijvoorbeeld alle geometrische data die niet worden toegevoegd, ook niet worden gecontroleerd. Dit kan leiden tot vals-positieve of vals-negatieve resultaten in de testresultaten. Er is nog een veelbelovende studie uitgevoerd, waarbij de geometrische validatie tijdens runtime wordt toegevoegd met behulp van nieuwe commando's in een SPARQL-query. Dit onderzoek leverde veelbelovende resultaten op, maar was gebaseerd op oudere semantische webtechnologieën.

Het doel van dit proefschrift is om de huidige implementaties van geautomatiseerde validatie en semantische webtechnologieën te analyseren om runtime geometrische validatie toe te voegen. Meer specifiek is het doel om te onderzoeken in hoeverre geometrische regelvalidatie kan worden uitgevoerd met behulp van de geometrie die beschikbaar is in de IFC-datastructuur zonder dat handmatige verrijking van geometrische eigenschappen nodig is. Uit literatuuronderzoek is gebleken dat de huidige SHACL-implementaties de validatie van geometrische niet kan verwerken. De reden hiervoor is dat SHACL-vormen alleen het bestaan van relaties en de waarden die ze vertegenwoordigen kunnen valideren. Het begrijpt de geometrische waarde die de letterlijke waarden vertegenwoordigen niet en kan deze regels daarom niet valideren. Om dit op te lossen, moest het SHACL-framework met deze mogelijkheden worden uitgebreid. Om deze functionaliteit toe te voegen, analyseert het proefschrift de vier fasen van geautomatiseerde validatie, zoals gedefinieerd door Eastman en zijn collega's. Dit proefschrift analyseert de vier fasen van geautomatiseerde nalevingscontrole zoals gedefinieerd door Eastman en zijn collega's. Deze vier fasen zijn de voorbereidingsfase van het bouwmodel, de interpretatiefase van de regels, de uitvoeringsfase van de regels en de laatste fase waarin de resultaten worden gerapporteerd. De voorbereiding van het

bouwmodel benadrukt het gebruik van continue data rechtstreeks vanuit de modelleringssoftware en de export van IFC-data. Hoe kunnen de data worden opgeslagen met behulp van semantische webtechnologieën en is dit formaat in overeenstemming met het formaat dat nodig is voor de validatie van geometrische regels? Technologieën zoals informatieleveringsspecificaties worden onderzocht op hun vermogen om data te structureren. Voor de regelinterpretatiefase wordt onderzoek gedaan naar huidige technologieën voor de vertaling van regels naar een computer leesbaar format. Van handmatige regelinterpretatie tot natural language programming. De regeluitvoeringsfase analyseert de methoden die nodig zijn voor de validatie van geometrische regels en hoe deze regels kunnen worden uitgevoerd. Er wordt geanalyseerd hoe semantische webtechnologieën proberen te valideren, zowel binnen de AEC als in andere sectoren. De geometrische fundamenten worden geanalyseerd om te bepalen hoe bibliotheken gedefinieerd zijn in andere sectoren. De rapportage van deze resultaten en hoe deze resultaten gebruikt kunnen worden, wordt geanalyseerd aan de hand van de export van tekstdocumenten naar computerleesbare formaten voor import in modelleringssoftware om ongeldige objecten te corrigeren.

Om het doel van het proefschrift te bereiken, zijn vier producten ontwikkeld om een prototype te creëren dat semantische webtechnologieën gebruikt om geometrische regelvalidatie toe te voegen. Voor deze thesis is besloten om met het IFC data format als inputformaat te beginnen. Dit omdat dit dataformat gestructureerdis , goed gedefinieerd is en als standaard wordt gezien in de AEC-sector. Het eerste product vertaaldt de IFC-data naar een klassenbibliotheek geschreven in C#. het tweede product is een uitbereiding op de klassenbibliotheek door een geometrische kernel toe te voegen om geometrische representaties uit de IfcShapeRepresentations te exporteren. De derde deliverable gebruikt deze geometrische kernel om validatie van de IFC-objecten mogelijk te maken op basis van ”High-level methods” (bijv. contains, below). Het eindproduct is een webapplicatie die het parsen van IFC-modellen en runtime-regelvalidatie mogelijk maakt op basis van semantische webtechnologieën. Het eindproduct is een volledige implementatie van een webapplicatie waarmee de gebruiker een IFC-bestand kan uploaden. Op basis van dit IFC-bestand wordt een RDF-grafiek gemaakt waarop een geometrische validatieaanvraag kan worden uitgevoerd. Op basis van een door de gebruiker verzonden validatieaanvraag kan de software de geometrische representaties van het IFC-object verzamelen. Deze objecten worden in de validator gebruikt om te bepalen of aan de validatie aanvraag is voldaan. Dit resulteert in het antwoord op de vraag in hoeverre geometrische regelvalidatie kan worden toegevoegd aan de technologieen van het semantische web, volgens de huidige werkwijzen. Tot slot worden de onderzoeksvragen beantwoord, gevolgd door een discussie over de conclusie en toekomstige stappen die kunnen worden genomen om voort te bouwen op dit onderzoek.

Abstract

Since the implementation of building information modeling, there is an increased reliance on the exchange of data in the Architecture, Engineering and construction sector. To share data between the different modeling software packages, the Industry Foundation Classes(IFC) have been developed. Although this data format has been regarded the main way of transferring data in the built environment, it is known to have its limitations. Recent studies propose the use of semantic web technologies to represent building data for data exchange. The use of the semantic web technologies allow for more flexible and linkable data that is also better defined and machine-readable. However, the current downside is the lack of geometric rule validation needed for Automated Compliance Checking (ACC). Recent studies have added geometric rule validation to the semantic web technologies by adding the geometric properties as RDF nodes which can later be validated through SHACL shapes. This thesis investigates whether runtime geometric rule validation can be integrated into the Semantic Web technology stack using only the geometric information available in the IFC schema without prior enrichment. This is currently impossible using SHACL shapes since these shapes have no knowledge of the geometric values. The goal of this thesis is to check to what extent geometric rule validation can be added to semantic web technologies in an industry-oriented setting. After the literature study, the decision was made to add functionality to a SHACL shape where it understands high-level method rule validation such as ”Underneath” or ”Contains”. To deliver this proof-of-concept, four deliverables are developed. First, a library of C# classes is developed to parse the IFC file format into C# classes. This library also contains basic functionality for the parsing of classes into RDF nodes and triples. The second deliverable is an extension to the C# class library where a geometric kernel is added capable of extracting and interpreting geometry from the IFC objects. The third deliverable creates validation functions that can be used by the new SHACL functionality. The final deliverable is a web application in which a user can upload an IFC file and send a validation request. Using the proof-of-concept, the validation request is executed, and the results are returned to the user. The results of this proof-of-concept show that it is possible to extend SHACL shapes with geometric functionalities. Using the objects gathered by the SPARQL Queries and using the high-level method verb, SHACL functionality is created where the objects can be used in the validation framework. This workflow is therefore able to use geometric rule validation without enriching the IFC data. The thesis concludes with a discussion on the current implementation and future studies that could enhance the functionality of the proof-of-concept.

Keywords: BIM. IFC, SHACL, SPARQL, Geometry

Acronyms

ACC automated compliance checking

AEC architecture, engineering and construction

API application programming interface

BIM building information management

IFC Industry Foundation Classes

LBD linked building data

NLP natural language processing

OWL Web ontology language

RDBMS Relational database management system

RDF Resource Description Framework

RDFS RDF Schemas

Figures

1 Reading guide . . . 4
2 Semantic Triple . . . 5
3 Four phases of automated compliance checking . . . 10
4 Class and object wall example . . . 12
5 Inheritance of IfcSpace and IfcWall . . . 13
6 IfcRelVoidsElement . . . 13
7 Methods of rule interpretation . . . 16
8 SweptAreaSolid representation . . . 24
9 Design research cycle . . . 28
10 Geometric validation use case . . . 28
11 The four research activities . . . 30
12 Project architecture . . . 31
13 Parts of the step line . . . 33
14 Relation between step line and class structure . . . 35
15 Three extrusion directions for a room definition . . . 38
16 User interface for rule execution . . . 43
17 Ground floor of the building model . . . 48
18 First floor of the building model . . . 49
19 Model view as imported . . . 50
20 Model validation result of missing ventilation point . . . 52
21 Model validation result of missing cooking appliance . . . 53
22 Model validation result of combustion appliance in the bathroom . . . 54
23 Ifc model with a thousand rooms . . . 55

Tables

1 Advantages and disadvantages: continuous data usage 11
2 Advantages and disadvantages: IFC Export 14
3 Advantages and disadvantages: Information delivery specification 15
4 Advantages and disadvantages: Manual rule interpretation 17
5 High-level verb rules 17
6 Advantages and disadvantages: Logic rule-based interpretation 18
7 Advantages and disadvantages: natural language rule processing 18
8 Rule types (Nuyts et al., 2024) 19
9 SQL table for rooms 20
10 Query result for minimum size of verblijfsruimte 20
11 Advantages and disadvantages: NoSQL Database management system 20
12 Advantages and disadvantages: Textual reporting of the results 21
13 Advantages and disadvantages: visual reporting of the results 22
14 Performance evaluation of 1, 10, 100 and 1000 rooms 55

Listings

1 RDFS example . . . 6
2 Examples of ifcOWL (Beetz et al., 2008)) . . . 7
3 Simple SQL querty for gathering living rooms . . . 8
6 IDS example . . . 14
8 SPARQL query with runtime geometric function . . . 25
9 SHACL rule with runtime geometric function . . . 29
10 IfcObject class . . . 33
11 Pseudo code for the step line parser . . . 34
12 Psuedo code for the extraction of a surface from the IfcExtrudedAreaSolid . . . 36
13 Psuedo code for the inside validator . . . 39
14 Pseudocode for the general validator . . . 40
15 winding number algorithm . . . 41
16 SPARQL query for selecting living areas . . . 42
17 Handle validation request pseudocode . . . 43
18 Living area SPARQL query . . . 51
19 Mechanical ventilation point SPARQL query . . . 51

1 Introduction

The building industry involves a lot of teamwork. People from different companies and professions need to share information and work together using different software tools. As buildings and projects become more complicated, the need for fast, accurate, and clear information sharing is more important than ever. To meet this need, the architecture, engineering and construction (AEC) sector started using building information management (BIM). With the use of BIM, the AEC sector found a way to share model data between their (modeling) software by using the Industry Foundation Classes (IFC) data structure. During the BIM revolution, the IFC data structure had proven to be a effective format sharing geometric data in a small file size to use between the different geometric kernels (Pauwels et al., 2011). For many years now, research has tried to implement faster and more flexible data sharing using semantic web technologies. The semantic web, a concept introduced by Tim Berners-lee in the late 90's, aims to make web content more available (Berners-Lee, 1998). It does so by making data understandable to computers. It makes use of ontologies to define the data and the SHACL validation tool to validate the data. Within the AEC sector, the use of semantic web technologies is also known as the use of linked building data (LBD).

LBD started in the semantic web domain based on standards defined by the World Wide Web Consortium such as Resource Description Framework (RDF) (Bonduel et al., 2018). LBD helps to create a network where related information can be easily linked and accessed (Pauwels and McGlinn, 2023). Recent research by van den Bersselaar et al. (2024) studied the integration of geometric properties using semantic web technologies, showing that it is possible to represent spatial information semantically. However, research (van den Bersselaar et al., 2024, Nuyts et al., 2024) states that semantic web technologies still lack the ability to perform the required geometric rule validation. Different reasons are given for the lack of this ability. Katsigarakis et al. (2019) state that the IFC data structure lacks support for geometric validation in the semantic web context, and Amor and Dimyadi (2021) state that there is no complete understanding for the full range of functions needed for compliance checking. Research on adding geometric rule validation by Zhang et al. (2018) states that IFC models are too complex and not query-friendly and geometric and spatial relationships needed for compliance checking are not directly accessible without additional computations.

There is a need for these geometric rule validations in the AEC sector. In recent years, the number of rules and building codes has increased drastically within the AEC sector (van den Bersselaar et al., 2024). To address this, automated compliance checking (ACC) has been developed to automate the validation process of BIM models. This development started with Eastman et al. when he proposed a method in 2009. This research proposed a process divided into four phases: the building model preparation phase, rule interpretation phase, rule execution phase and the final phase for the reporting of the results. Later research acknowledges that these phases defined by Eastman et al. help achieve better quality, productivity gain, and cost savings in the AEC industry when implemented correctly (Z. Zhang et al., 2022). The ACC process cannot be fully functional without the ability to validate geometric functionality. This thesis will investigate to what extent geometric rule validation can be added to the semantic web tech stack in an industry-oriented setting.

1.1 Problem definition

Research has shown that semantic web technologies have many advantages over current standards such as the IFC file structure (Pauwels et al., 2011). Technologies like ontologies can replace hard coded class structures and SHACL can validate data objects on property values and ontology compliance. However, without the ability to add geometric functionality and validation, only the basic properties and relations can be validated. These validations are deemed insufficient (Guo et al., 2021) and therefore will prevent full adaptation of LBD. Adapting the semantic web principles in the AEC sector would allow the ability to connect to various domains (Pauwels, Zhang, and Lee, 2017), better support building performance assessments (Corry et al., 2013) and lower the technical barrier for smaller companies (Jeroen et al., 2019). Because of all the advantages that are missed out on, there is a need for geometric rule validation to be added to semantic web technologies.

1.2 Research questions

Past research often tries to solve one specific section of ACC. However, the objective of this research is to fill the research gap and to create a framework capable of rule validation using linked building data and geometric computations. Studying the research gaps and their implementation on existing technologies will be combined to create this ACC framework. For this, the following main research question will be answered:

'How can geometrical computations be added to the current state of automated compliance checking using linked building data to validate a building model of the IFC format?'

To answer this question, a better understanding of the aspects of this research question must be created. To achieve this better understanding, the following subquestions will be answered:

Sub-question 1: What is the current state of linked building data technologies?
To begin understanding the limiting geometric validation in semantic web technologies, there must be a understanding of the current limiting factor. This research question will examine what tools are used in semantic web technologies. With this knowledge, it can be determined why there currently is a lack of geometric rule validation and how a solution can be derived.

Sub-question 2: How can a model data be translated into computer-readable code without loss of data?
To add geometric validation to semantic web technologies, it must be known what type of data is used and how it is translated into the semantic web technologies database. The limitations that might be present during the translation phase must be understood to get a full picture. It must be known how much data is lost in the translation process, and it must be determined if this is a limiting factor in geometric validation.

Sub-question 3: What geometric validations need to be applied?
When there is a understanding of semantic web technologies and the data that is used, it must be defined what type of geometric validations are needed. These results might give insight into the workings of rule validation and give new angles to a solution. The resulting functions can also be used after a proof-of-concept is designed to check if these geometric validations can actually be done.

Sub-question 4: How can the resulting framework be implemented to satisfy the user's needs?
For this thesis, it is expected that a proof-of-concept will be developed that implements geometric rule validation functionality to semantic web technologies. Therefore, it needs to be known that the proof-of-concept can be implemented and made available to users of semantic web technologies to validate their models based on geometric rules. Therefore, it must be known how this should be implemented and how automated compliance checking works.

1.3 Research design

To answer the research question on to what extent geometric rule validation can be added to the semantic web technologies, the study will be divided into three main phases. The first phase is the literature study, which is divided into two parts. It starts with a study of ACC with respect to the semantic web tech stack. Here, the current ways of working such as the IFC data structure are weighted against technologies such as RDF, OWL and SHACL. The second part of the literature study examines the geometric aspect of rule validation. The study focuses on how geometric validation is currently done and places this in the semantic context. The second represents the methodology of the paper. It starts by defining the goal of the application, how it will achieve the goals and how the result can be validated. For this process, the development of a web application that implements geometric validation to semantic web technologies is presented. The final phase is the result and conclusion of the thesis. This phase validates how successful the web application is in adding geometric rule validation to the semantic web. Based on these results, conclusions will be drawn and improvements will be stated.

1.4 Importance of the thesis

The current state of geometric rule validation in semantic web technologies is insufficient. Without this functionality, the AEC industry can never fully implement LBD principles without losing needed functionality in ACC. As a result, current processes are less efficient and more prone to error. Research has shown that future implementations of semantic web technologies can validate and even correct building models, but these are not yet implementable. Finding connections between current implementations and semantic web validations is believed to lead to results where geometric validation can be added and applications can move towards LBD. It is important to note that this thesis only investigates to what extent it is possible to add geometric validation to the semantic web tech static. Unfortunately, it is unrealistic that this thesis can produce a market-ready result in which all problems regarding geometric data validation are solved. It is still a valuable step to add the needed functionality.

1.5 Reading guide

The thesis begins with a study of the literature on automated compliance checking and geometry in Chapter 2. This literature study shows summaries of information needed to answer the research questions. These summaries are based on recent studies which share a common ground with this thesis. The literature study is followed by the methodology in Chapter 3. The methodology starts by explaining the goal of the thesis based on the literature found. The methodology will also state the choices made for the creation of the rule validation framework. After the development of a proof of concept the results will be shown in section 4. Both positive and negative results of the thesis will be shown. Based on these results, a conclusion will be drawn and limitations will be mentioned. These are shown in Chapter 5. Figure 1 shows a visual representation of this thesis.

Figure 1: Reading guide

2 Literature study - automated geometric rule validation using semantic web technologies

The introduction has shown the lack of geometric rule validation in semantic web technologies which is needed for automatic compliance checking. The goal of this thesis is to check to what extend this geometric rule validation can be added. This Chapter is a literature study on the current state of geometric rule validation in the semantic web context. It starts with a study on the semantic web and its development in Section 2.1. This is followed by a study on automated compliance checking in Section 2.2. Finally, a study is done on geometric rule validation and the difficulties it entails (Section 2.3). At the end of this literature study a conclusion is drawn on what the literature has shown with regards to the goal of this thesis.

2.1 Semantic web technologies

Semantics is the study of what a word means in a given context (Pauwels and McGlinn, 2023). For a structural engineer, adjacent has a different meaning than to a spatial developer and for a computer it is all just ones and zeros. The goal of semantic web technologies is to structure the data to make everyone understand what each object represents. A difficult task given the large amount of specialized knowledge and complexity of natural languages (Pauwels and McGlinn, 2023).

To achieve data understood by everyone, the world wide web consortium developed technologies making the semantic web tech stack (van den Bersselaar et al., 2024). The tech stack starts with the use of the Resource Descriptive Framework (RDF). Within this framework, data is structured using ontology languages. Finally, using SPARQL and SHACL, the shapes can be collected and validated. The semantic web tech stack contains many more tools to structure data and make it more accessible. However, for the aim of this thesis, this will cover all technologies needed for Automated Compliance Checking (ACC).

2.1.1 Resource Description Framework

The Resource Description Framework (RDF) is the foundation of the semantic web technologies. The framework provides an easy way to describe data using data triples. An RDF-triple consists of three parts: (1) a subject, (2) a predicate, and (3) an object. Using this triple a fact is stated about a subject, the predicate is the relationship and the object is the value it is connected to (Pauwels and McGlinn, 2023). This way, a relation between two resources can be identified (Klyne et al., 2014). An example of a triple is shown in Figure 2 where a (1) wall is defined that (2) contains a (3) window.

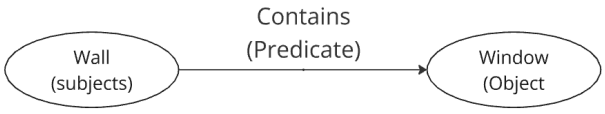

Figure 2: Semantic Triple

On top of the RDF, the Resource Description Framework Schema (RDFS) is developed. The RDFS allows better organization and structuring of data. It not only uses the single statements of RDF-triples, but also provides additional functionality such as defining types (rdfs:types), hierarchies (rdfs:subClass), domains (rdfs:domain) and ranges (rdfs:range). An example of the RDFS is shown in Listing 1 where a ”wall” class is created, which is of kind ”buildingElement”. This wall class has the ”hasHeight” property, which is of a decimal value. Using this schema allows for the enrichment of RDF data and gives the user a better understanding of the data.

```
:Wall
    a       rdfs:Class .
:BuildingElement
    a       rdfs:Class .
:Wall
    rdfs:subClassOf :BuildingElement .

:hasHeight a rdf:Property ;
       rdfs:domain :Wall ;
       rdfs:range xsd:decimal .
```

Listing 1: RDFS example

2.1.2 Ontologies

To make sense of the objects and subjects used in these RDF triples, they must be defined and structured. RDF does this using ontologies. Ontologies catalog the details of types or things and the relationship between those things that are relevant for the domain (Jacob E.K., 2003). Famous ontologies used for RDF are RDF Schemas (RDFS) and Web ontology language (OWL). These ontologies define classification hierarchies and constraints (Pauwels and McGlinn, 2023). To validate compliance of the data in a triple with these defined classifications, tools such as SHACL can be used.

The current standard for exchanging building model data is the IFC file. This data structure, developed by buildingSMART, is used to share data between different parties in the built environment in the Netherlands. To enable modern web and data technologies, to better access and use the information contained in IFC models, buildingSMART also developed the ifcOWL ontology (data building group, 2017). IfcOWL is a direct translation of IFC data classes to RDF schema classes to translate all data efficiently. Listing 2 shows three examples of a well-used ontology, this being ifcOWL.

```
:IfcElement
    a       owl:Class ;
    rdfs:subClassOf owl:Thing.

:IfcBuildingElement
    a       owl:class ;
    rdfs:subClassOf :IfcElement ;
    owl:disjointWith :IfcFurnishingElement

:IfcDoor
    a       owl:class ;
    rdfs:subClassOf :IfcBuildingElement ;
    owl:disjointWith :IfcWall :IfcWindow
```

Listing 2: Examples of ifcOWL (Beetz et al., 2008))

An important aspect of the use of RDF in the built environment is the translation of model data to RDF triples. During this translation, the model (mostly IFC models) needs to be translated to the ifcOwl ontologie in triple format. Using tooling such as the IFC-to-RDF converter, an IFC file can be translated to RDF triples making use of the ifcOWL ontology. However, this is the first step in translating the IFC data into the needed data. Often times, other calculations are needed for the right queries to be executed. Another tooling in the RDF is the shape constraint language known as SHACL. SHACL can validate data sets based on constraints by comparing two RDF graphs (van den Bersselaar et al., 2024). It does so by comparing the data graph against the shapes graph where it is validated if the objects are correctly defined.
Outside of the built environment, different geometric ontologies and SPARQL extensions are developed. GeoSPARQL is developed by the Open Geospatial Consortium to provide a standardized exchange basis for geospatial rdf data. It makes use of 2D geometry to determine if 2D polygons are disjoint, touch, overlap, equal, within, or contain each other. GeoSPARQL makes use of the WKT (well-known text representation of geometry) ontology. Research has already been done on the implementation of this ontology in combination with ifcOwl by Pauwels et al (2017). They concluded that challenges arise when handling ordered lists, making it difficult to use the data efficiently.

2.1.3 SPARQL

A method for the execution of rules is through the use of queries on a database. Queries are commands that request data from a database. This command can request all data from the database or ask for data with a filter. A long-standing standard for database querying is SQL with its respective SQL queries and Relational database management system (RDBMS)(Codd, 1970). Within this relational database, data objects like users or buildings are stored in tables. Using a simple SQL query as seen in Listing 3 can retrieve all the rooms and their respective information from the database where the type is a living room.

```
SELECT * FROM rooms WHERE Type = 'Living room'
```

Listing 3: Simple SQL querty for gathering living rooms

In the AEC sector, recent studies focus on the RDF framework. The RDF framework makes use of querying tools like SPARQL. This framework has the potential to link data and query rules. Rule querying, however, has a downside. Only data directly in the database can be validated. The second problem is that currently, validations requiring geometric or relational computations are not able to be validated (Nuyts et al., 2024). SPARQL queries use a structure similar to SQL queries. A SPARQL query as seen in listing 4 can retrieve all rooms and their respective information from the database where the room is of type living room.

```
SELECT ?space
WHERE {
    ?space rdf:type ifc:IfcSpace .
    ?space ifc:hasType ?type .
    ?type rdf:type ifc:IfcType .
    ?type ifc:name "Living Room" .
}
```

Listing 4: Simple SPARQL query for gathering living rooms

2.1.4 SHACL

To validate the data used in RDF, the Shapes Constraint Language (SHACL) can be used. SHACL can be used to validate RDF graphs against a set of conditions (n.d.). This validation is done by creating a SHACL shape. These shapes are rules and constraints that describe what correct data should look like. This shape can describe what properties an object must have and what value types and values are allowed. This way, data is not only machine readable but also understood by everyone and ready for automated compliance checking. In Listing 5 an example is shown for a SHACL shape of a wall. This shape defines that a wall must have at least a name of type string and can have a property of OverallHeight which is of the decimal type.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ifc: <https://standards.buildingsmart.org/IFC/DEV/IFC4_OWL#> .
@prefix ex: <http://example.org/ns#> .

ex:WallShape
  a sh:NodeShape ;
  sh:targetClass ifc:IfcWall ;
  sh:property [
    sh:path ifc:Name ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
  ] ;
  sh:property [
    sh:path ifc:OverallHeight ;
    sh:datatype xsd:decimal ;
  ] .
```

Listing 5: Simple SHACL shape of a wall

2.2 Automated Compliance Checking

Automated compliance checking (ACC) is the process of using technology to ensure that organizations follow legal, regulatory and internal rules, reducing the need for a lot of manual work. Traditionally, compliance checks are based on manual inspections done by experts during the execution of their work. Unfortunately, this process is prone to errors and depends on the knowledge and alertness of the expert. To prevent these manual compliance checks and to speed up the process, the AEC sector started implementing automated compliance checks. Within the AEC sector, ACC is the process of automatically checking if a (building) model complies with specific rules. The use of ACC can reduce the time and effort required to verify compliance and prevent mistakes due to complex engineering requirements (Eastman et al., 2009). The process makes it easier to ensure the built environment is fit for purpose, constructed according to the design, safe to use and functional, cost effective and sustainable throughout its service life (Amor and Dimyadi, 2021). In 2009 Eastman and his colleagues researched automated compliance checking in the built environment. They concluded that the ACC process can be divided into four stages. The building model preparation phase, the rule interpretation phase, the rule execution phase and the reporting of the results phase (Eastman et al., 2009). The correct implementation of the four phases helps to

achieve better quality, productivity gains, and cost savings in the AEC industry when implemented correctly (Z. Zhang et al., 2022).

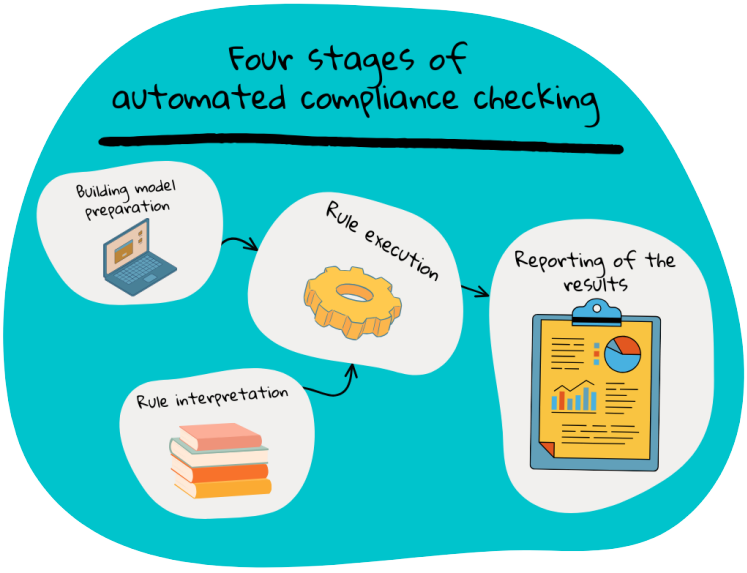

Figure 3: Four phases of automated compliance checking

2.2.1 Building model preparation

The first phase of Eastman's automated compliance checking is the building model preparation phase. The goal of this phase is, as the name suggests, to prepare the data model for rule validation. Leaving data out of the preparation phase makes the entire validation process obsolete. A waste of both time and effort. The preparation of the model data consists of two steps. First, the input data must be defined. Second, the format of the computer-readable code should be defined.

Following current trends, there are two types of inputs for model translation. The first method is the usage of data directly from modeling software. This method is also known as using continuous data. Creating extensions for applications like Revit or Tekla allows for continuous compliance checking and processing of feedback. The second method is to export the data from the application, usually done using the IFC data structure, to be imported into other software. How the data format for the computer readable code should look like is defined by the input. When using the continuous data method, the data objects are similar to the respective modeling software. When exporting the data, it will reflect this exported data format. Both methods come with their own strengths and weaknesses based on their intended usage. Studies have been done on how to better manage the process of managing and sharing data.

Continuous data

The first input type for automated compliance checking is the use of model data directly from the modeling software. Using data from the modeling software (e.g. Revit, Tekla) allows the usage of continuous live data. By extending the existing modeling software functionality with the needed functionality, the user can check data and apply changes . This extension is done through the use

of an application programming interface (API). In simple terms, an API exposes functions of the modeling software for programmers to use and make their own functions. These APIs work in a modular way, allowing for all sorts of functionality. This allows for continuous use of real-time data without steps in between the modeling software. This increases the validation speed and allows for user feedback within the model. It even allows the code to modify the model to correct for invalid objects. For example, a validation of the required distance between two objects is expected to return true or false. Using continuous validation, it is possible to automatically correct the position of the validated object. This automated process, even though it is hard to implement, saves time and minimizes mistakes. The second advantage is that the data does not need to be translated before it is used. Therefore, there is no chance that data is lost before validating the model. The use of continuous data in automated compliance checking is seen primarily in the commercial sector (Amor and Dimyadi, 2021). Integrated REVIT tools such as SMARTreview APR have been developed to validate models within the REVIT application. The reason for this being commercial is that most AEC companies, such as small architectural firms, do not have the resources to develop these extensions.

However, directly implementing model preparation and model validation in modeling software also introduces limitations. The biggest downside of this is the hindered scalability of the developed software. When the software is directly implemented on the modeling software, it becomes specific to only that tool (e.g. Revit or Tekla). Although this code can then be made publicly available, it is written using the APIs of the modeling software that are incompatible with other modeling software making reuse more difficult (Pauwels, Zhang, and Lee, 2017). The research by Zhang et al. (2018) states that this fragmentation reduces the ability to standardized validation strategies between disciplines. This fragmentation can further reduce long-term adaptability and hinder broader compliance checking goals. All while being a time-consuming effort for development of these independent software implementations.

Advantages	**Disadvantages**
+ Speed of feedback	- Specific validation methods
+ No translation needed	- Development is time consuming
+ Possibility for automated corrections	- Less scalable

Table 1: Advantages and disadvantages: continuous data usage

Industry foundation classes

The second trend on the input for model translation is the export of data. The current market standard for the export of model data is using Industry Foundation Classes as an import and export method. The modeler can choose which data is needed for the validation and export this data using the IFC file format. This data is then imported into the validation software for further processing. This way of working has multiple advantages. The biggest advantage is that all software packages with the option to export the data format can be used as input for the validation. The downside is that exporting and importing data has the certainty that data is being lost. The first version of the IFC format dates back to 1996 and was developed by the Industry Alliance for Interoperability. An initiative started by Autodesk and now known as the organization BuildingSMART. Since then, IFC has made significant improvements, the important ones being the release of version IFC2x3

and IFC4 released in 2006 and 2013 respectively (Grutters, 2022). IFC 4 became an international standard, becoming the main format for data exchange used in building information management within ISO 16739: 2013 (Buildingsmart, 2024).

The IFC data format and its implementations have many advantages. The big advantage of the IFC data format is the amount of data that can be stored while still conforming to its base rules. IFC files and its objects contain information about the model, including not only the shapes of the objects but also details about the materials, performance, and how the building will be used over time. A big downside of the IFC data format is data that is lost or changed while exporting and importing the data (Pauwels et al., 2011). Research by Katsigarakis et al. (2019) states that the quality of this exported and imported data is usually insufficient for complicated validation. Exported models may lack precise geometric representations and relationships may be lost making advanced simulations or rule validations more difficult. The second downside is the file size and the snapshot it represents. When a small adjustment is made to the model (e.g. moving a wall), the entire model has to be exported and imported again. This causes a lot of data traffic and does not adhere to the semantic web principles. A final downside to the IFC data structure is the versioning. Since the IFC file format is a universal standard, compatibility is of great importance. However, as the market evolves, the data needed within the IFC format changes. When BuildingSMART releases a new mayor IFC version, for instance, the change from version 2x3 to version 4, a lot of data objects were added and others were made obsolete. Every application making use of the IFC format had to comply to these changes, with backward compatibility being needed as well.

The IFC data structure and its objects are a framework developed with the intention of being very data efficient. These objects, found in the documentation by buildingSMART, make use of an object-oriented approach to define its data structure. Each object represents a specific entity within a building model (ISO 16739-1:2018). The IFC data structure works with classes by which objects can be defined. Simply put, it first defines what parameters an object can have after which multiple objects can be made defined by these parameters. An example of this is shown in Figure 4. Here, a wall class is shown. Based on this class, two walls are instantiated. This structure is known in software development as object-oriented programming.

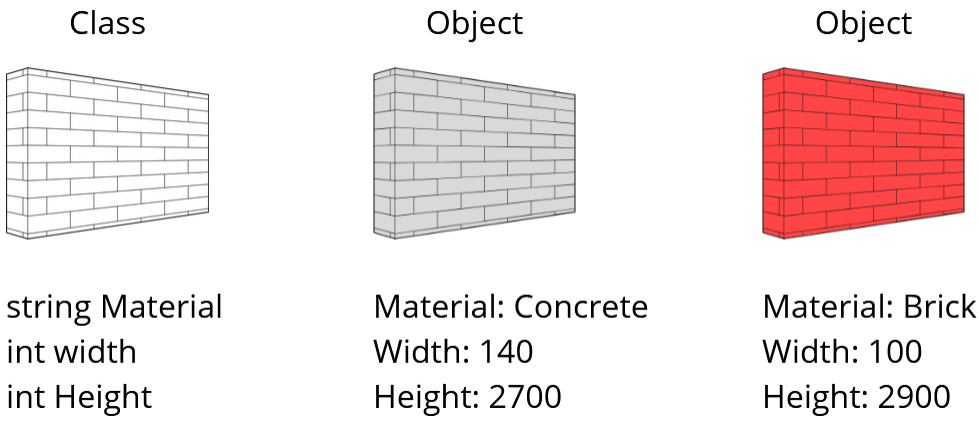

Figure 4: Class and object wall example

The IFC data objects are defined by inheritance and are defined to meet requirements and demand from the market. Inheritance means that some objects can share common features or properties of other objects. It is a way to organize the data in a structured manner. In Figure 5, it is shown that both IfcWall and IfcSpace inherit the same class, namely IfcProduct. IfcProduct contains the location and representation of building objects. Because both a space and a wall have a position and a representation, both objects inherit this information instead of defining these on their own. By making use of this inheritance while saving the data, copies of objects of parents of objects do not have to be saved multiple times, resulting in smaller file sizes.

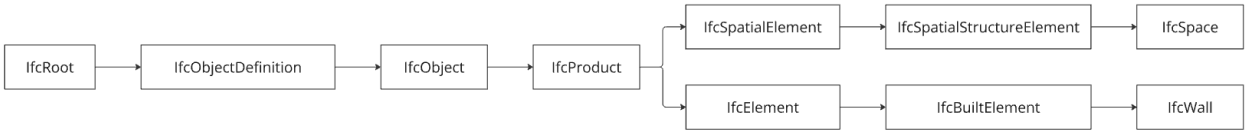

Figure 5: Inheritance of IfcSpace and IfcWall

In addition to inheritance, the IFC data structure uses relations between objects. These relations define how objects or entities relate or interact with each other. All of these inherit from the IFC object IFCRelationship. Examples of these relations are IfcRelConnects, which states that elements connect and IfcRelDecomposes which states which elements compose each other (Buildingsmart, 2024). An implemented example of a relation that inherits the IfcRelDecomposes is the IfcRelVoidsElement. This relationship defines which elements create a void in another object. In simple terms, an element is created by removing the geometric representation of an object with the geometric representation of a second object. Figure 6 shows the working of this relation. The relation connects the building element with the opening element, establishing the relation that the opening is part of the element. When using IFC data, it is important to remember that even though the possibility of adding relations is available, it cannot be assumed that all relations are indicated in this way. That the ability exists that a relation can be created between a wall and a window does not mean that all these relations have been defined. This relation might never have existed or could have been lost during the parsing of the data as shown in the research by Katsigarakis et al. (2019).

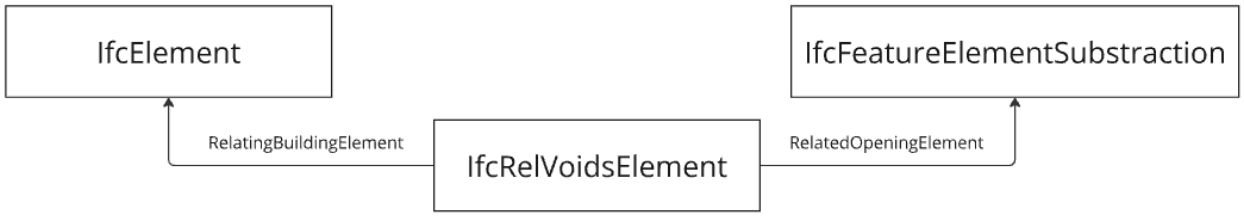

Figure 6: IfcRelVoidsElement

The IFC data format has well-defined objects. However, the format in which this structure is shared has different variations. Currently, there are three formats that are most commonly used in the AEC sector. These are the .ifc(step), .xml and .Json file formats. In addition to these, there are also the .zip, .ttl, .rdf .hdf and .sqlite formats (Buildingsmart, 2024). Currently, the most used file format is the .ifc step file format. It scores the best on file size and is proven to be effective for the

data exchange of 3D represented objects. The major disadvantage of this file format is that it uses file-based operations where data quality is reduced.

Advantages	Disadvantages
+ Compatibility across platforms. + Improves collaboration. + Only one tool per function needed	- Possible loss of information. - No Real-time validation. - Managing IFC schemas and ensuring alignment with regulatory requirements adds complexity. - May require additional effort to map exported data back to the original model for corrections.

Table 2: Advantages and disadvantages: IFC Export

Information delivery specification (IDS)

Within the AEC sector, successful project outcomes depend on well-defined data structures to avoid miscommunication and the resulting errors. For this, an information delivery specification (IDS) can be used. An IDS is a specification designed to standardize and manage the data exchange of model data in BIM projects. An IDS provides a consistent approach to specify what data should be included in a BIM model, how it should be structured, and the format in which it must be delivered. An IDS has the strength of being a computer-interpretable file format (XML) and is based on the IFC file format (Bimcollab, 2024). The IDS consists of a list of requirements defined in the XML format as seen in the Listing 6. A well defined IDS structures the required data in detail by model scope, level of detail, data structure, and other criteria. The IDS specification first states on which object the requirement is applicable (IFCWALL in case of Listing 6). It states that each wall should have the property 'LoadBearing' within the propertyset 'Pset_WallCommon'.

```
<specification name=" Load-Bearing" description="Requires each IFCWALL
   to have the LoadBearing property" ifcVersion="IFC2X3">
  <applicability minOccurs="1">
    <entity>
      <name>
        <simpleValue>IFCWALL</simpleValue>
      </name>
    </entity>
  </applicability>
  <requirements>
    <property>
      <propertySet>
        <simpleValue>Pset_WallCommon</simpleValue>
      </propertySet>
      <baseName>
        <simpleValue>LoadBearing</simpleValue>
      </baseName>
    </property>
  </requirements>
```

```
</specification>
```

Listing 6: IDS example

This computer readability makes it work well with the automated compliance checking process. Because it is verified that a certain data structure is used, automated complice checking validators can, with more accuracy, collect the needed data for rule validation. This advantage is important for interoperability. The proven data structure promotes collaboration between different platforms and tools. This is specifically the case where the agreements on data exchange are made using multiple disciplines.

Although implementing IDS integration for automated compliance checking has many advantages, it has its downsides. The main disadvantage is the ineffectiveness of the IDS when not all data is (properly) defined. If it is not defined in the IDS how a certain object is defined (e.g. a ventilation point) the software is unable to know how to validate the object. No check of the building regulation with respect to ventilation values can be made. Since a lot of data has to be set up, it is a time-consuming effort where lots of validation is needed from every aspect of the AEC sector. Furthermore, while the integration of IDS in the build environment has been promising, the semantic web technologies use ontologies to define their data. The IfcOWL ontology defines all the IFC data objects in the semantic web context.

Advantages	**Disadvantages**
+ Data accuracy and consistency through standardization	- Complete IDS is required
+ Reducing data format errors	- Challenges in implementation
+ Promotes interoperability	- Initial setup is time consuming

Table 3: Advantages and disadvantages: Information delivery specification

Open source parsing libraries

For the building model preparation phase, the AEC sector is helped by open source software packages. Within the context of linked building data, several open source libraries support this phase by converting IFC data into their respective data classes for programming software such as python or C#. Tools such as IfcOpenShell and xBIM provide the functionality needed for parsing IFC data files. Similarly, open source libraries exist where these classes can be easily added to RDF graphs. One of these is the IFCtoRDF library that can create graphs and let classes be added in the form of nodes and triples. The tools allow developers to more easily implement the parsing capabilities to their software without having to program it themselves. The downside to these packages is that there is little to no flexibility when a different method of implementation is needed. Request can be made for these open source software implementations but this takes time and might not align with the idea of the open source software. It is of great importance to determine what is best for the application that is needed and the tooling that fits best.

2.2.2 Rule interpretation

The second phase of Eastman's automated compliance checking framework is the interpretation of the rules. This phase focuses on translating the rules into a digital format that can be used to automatically verify the compliance of the prepared building model. Where this process has traditionally been done by an expert in the respective field, it is now being transformed into a more systematic process. Manual interpretation is being replaced by logic rule-based or natural language processing (NLP) based applications (Zou et al., 2020). Logic rule-based interpretation translates a rule back to verbs and nouns to get high-level rules that can be executed. The NLP allows the system itself to read and interpret human language. Recent literature on rule interpretation makes use of linked building data to interpret (and execute rules). Through the use of data queries on a database, it can be determined if a model complies with the rule that it searches for. Research shows that the current AEC sector still has difficulty in interpreting and interpreting rules accurately and efficiently (Z. Zhang et al., 2022). With a shift from manual experts to machine readable code, edge cases and specific knowledge are hard to all define in single rules. In addition to the edge cases, other issues currently are restrictions by the building model, the understanding of the rules, and the lack of relationships between representations and the type of rules (Z. Zhang et al., 2022). More details on the rule interpretation methods, their advantages and disadvantages, are given in the following sections.

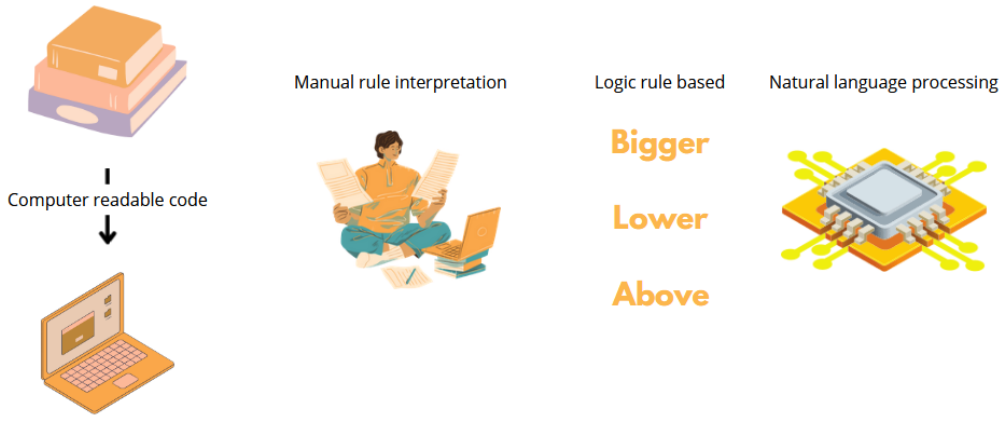

Figure 7: Methods of rule interpretation

Manual rule interpretation

Early development of automated compliance checking involved manual interpretation of rules and regulations. This manual process was of great importance since experts needed to create computer-readable code for rule execution. At the time, manual interpretation of rules was the most reliable method to ensure that regulations were correctly translated. Although this manual approach was successful in handling specific rules, it had some limitations J. Zhang and El-Gohary, 2017. As the AEC sector grew, so did the amount of rules and regulations and their complexity. At the time, rules were needed to be hard coded, which was seen as a black box with limited validation for the interpreted rules (Zheng et al., 2022). The growth in rules and the time needed to implement the code increased the time needed for manual interpretation of rules. Other methods

produced have the potential to be five times as efficient in time, making manual rule interpretations obsolete (Zheng et al., 2022).

Advantages	Disadvantages
+ Reliable translation	- Time consuming
+ Expert involvement	- Inefficient
	- Growing complexity

Table 4: Advantages and disadvantages: Manual rule interpretation

High-level method interpretation

A second way to interpret rules is through the use of high-level method interpretation. This method uses high-level abstractions to interpret rules. The latest studies by Lee and his colleagues (2023) present an approach involving this method to define high-level methods by focusing on verb phrase translation, providing usable techniques that allow software developers to enhance their low-level rule verification processes. By being able to translate rules into verbs using if then statements, functions can be generalized and reused, increasing efficiency in time, effort, and cost (Lee et al., 2023).

The biggest advantage for using this method is the scalability and modularity. By breaking down complex rules into high-level methods, rules can be created more easily to verify rules and regulations. Another advantage is the representation of the rules that are checked. By translating the rules to high-level methods, they are also simplified and easier to understand. This method, however, has some limitations. The first one being the defining of all high-level methods. Studies by Lee et al. (2023) have been done on commonly used relationships shown in Table 5. However, difficulties arise when verb such as ”Adjacent” need to be defined. When high-method rule interpretation is implemented, agreements have to be made on what these verbs mean.

Inside:	True or false value if a object is inside another object
Outside:	The inverse of inside
Above:	True or false value if the lowest point of object A is higher than the highest point of object B
Below:	True or false value if the highest point of object B is lower than the lowest point of object A
Intersect:	True of false value if object A intersects object B

Table 5: High-level verb rules

A second limitation is the data loss on the translation of the rules. By simplifying the rules, they can cause misinterpretation on the meaning of the rule and miss the point of the rule completely. Furthermore, there are also rules and exceptions that are too complex to be translated back to high-level methods. These rules might be falsely interpreted within the translation and might not give the desired results.

Advantages	Disadvantages
+ Automatic scalability	- Complexity
+ Transparency	- Loss of rule data on translation
	- Limited on implemented verbs and adjectives

Table 6: Advantages and disadvantages: Logic rule-based interpretation

Natural language processing (NLP)
With the rise of artificial intelligence, researchers have studied the use of automatic rule translators to translate rules into computer-readable code. One of the promising results following this investigation is the use of NLP. NLP is able to translate entire legal documents, such as building regulations, and translate its contents into a computer-readable format such as logical IF-THEN rules. This automatic process can significantly reduce the time and manual effort required to interpret the rules and regulations. Research on NLP shows that computer-readable codes can be extracted from legal documents. However, there are still many challenges to face.

The greatest advantage of implementing NLP as a method for rule interpretation is the speed with which it can process large amounts of data. It does so by automatically identifying the domain words and matching these to the building elements (R. Zhang and El-Gohary, 2019). A big disadvantage of using NLP rules is that they do not always detect things correctly in the applications. As an example, a framework developed by Zhang and El Gohary (2017) had an impressive detection rate of 87.6 to 98.7%. However, This means that the system is still not reliable to identify all noncompliant rules. Secondly, research shows that the current development of NLP in the built environment lacks the technology needed to interpret between regulations and proposals (Beach et al., 2020). A big limitation for this missing technology is the limited training data, the lack of open data sets, and the disagreement about the requirements of a representation of the building regulations (Fuchs, 2021). Without solving these issues, the rules will not be properly translated and the automated compliance check will not show the desired result.

Advantages	Disadvantages
+ fully automatic extraction	- Complexity
+ Scalability	- Accuracy limitations
+ Speed of interpretation	- Dependency on datasets
	- Black box validation

Table 7: Advantages and disadvantages: natural language rule processing

2.2.3 Rule execution

The third phase of Eastman's automated compliance checking framework is the execution of the rules on the prepared model. In this phase, the interpreted rules are applied to the translated building model from phase one to automatically verify its compliance. Traditionally, this process has been done by experts when building the model. The assumption was made that the expert is familiar with all rules with which the model needs to comply. During or after the creation of the

model, the expert can verify that the model is complying manually. Over time, rule validations that execute rules have been hard coded for specific rules where companies had seen great value. Usually, automated validations correlate with the type of rule being validated and its respective complexity. For instance, a validation based on a word is easier than a geometric validation, especially using the semantic web where geometric validation is still lacking. While some companies have made significant progress in automating these checks, the adoption of semantic web technologies remains limited because of this (Pauwels, Zhang, and Lee, 2017).

Rule types

Within automated compliance checks, there are many different rules that need to be checked. Some are simple rules like the comparison of a word in the data of a wall. However, there are also rules and regulations that need more complex rules, such as the positioning of an object. Tomczak et al. (2022) determined that there are 7 types of rules to be checked. Standardized, applicability, fields, value constraints, content, geometry, and metadata. Based on this study, Nuyts et al. (2024) have investigated 8 approaches to validate these types. Their research on most types shows that these types can be automatically validated. This data results in five data types with examples that cover all the rules and regulations within the Flemish building regulations. The rules and their definition are defined in Table 8.

Type	**Example**
Information availability	Each door should have exactly one door width property
Value	The structural dimensions of doorways must be at least 105 cm
Relational	A railing must be provided on both sides of a the staircase
Mathematical	The sum of two time the riser and once the tread of each step should be between 57 and 63 cm or a multiple thereof
Conditional	The slope is at most 6.25 percent for level differences between 25 cm and 50 cm

Table 8: Rule types (Nuyts et al., 2024)

The study by Nuyts and colleagues (2024) shows that of these types, geometric types are not yet manageable by current semantic technologies. A combination of missing geometric data in the IFC file format, a standardized way of extracting this data, and the current implementations of semantic technologies have limited functionality for geometric validations. While all other rule types can be checked using semantic open data, geometric has a higher level of complexity that is not yet available. These geometric validations require software to understand or at least determine spatial relations between building elements. Currently, the AEC sector relies heavily on hard coded functions or external applications like Solibri to perform these geometric validations. A process that required the translation of data which currently does not support semantic web technologies.

Database queries

With the rise of the use of linked building data in the AEC sector, methods have been implemented to validate rules using database queries. Using a query language, relevant information can be collected from a database. To interpret and execute rules, queries try to extract only data from a database that does not comply with rules and regulations (ali waqas et al., 2021). Any match

means the model is not valid. As an example we can take a representation of a table containing six rooms as seen in Table 9.

Id	Name	Type	Surface
1	Hallway	Traffic area	6
2	Living room	Living space	25
3	Kitchen	Living space	12
4	Landing	Traffic area	4
5	Bedroom	Living space	4
6	Bathroom	Bathroom	6

Table 9: SQL table for rooms

Within this database, we want to validate that no room of type "Living space" is smaller than $5m^2$. For this, we can run the query seen in Listing 7. The result of this query would be that room with ID 5 is selected, and the data in Table 10 is returned. This result determines that the returned room does not comply with the given rule.

```
SELECT * FROM rooms WHERE Type = 'Verblijfsruimte' AND Surface < 5
```

Listing 7: Rule query for minimum verblijfsruimte size

Id	Name	Type	Surface
5	Bedroom	Living space	4

Table 10: Query result for minimum size of verblijfsruimte

NoSQL Database management systems (DBMS)

A different type of database that has seen an increase in popularity is the NoSQL database management system. Unlike the SQL database known as a relational Database Management System, NoSQL does not rely on fixed schemas in which data is stored. Research shows that this way of data management is better when working with large workloads and large amounts of data where scalability is needed (Kuznetsov and Poskonin, 2014). Another advantage of using NoSQL is the simpler and more flexible way to develop. The downside of this is the weaker consistency of the data making it less reliable. This reliability and uncertainty leads to a greater chance of containing errors.

Advantages	**Disadvantages**
+ Fast and simple development	- Weaker data consistency
+ Scalability	- Prone to error

Table 11: Advantages and disadvantages: NoSQL Database management system

Computational complexity
A known bottleneck in automated compliance checks with respect to geometric and relational validation is the computing power needed for functions. Checking if a toilet is above a living room means that every room and toilet must be collected, then each position needs to be gathered, then it needs to be determined if the toilet is both inside the room and has only a max height difference to the room. Although these checks can be performed rather quickly for a few houses, it becomes complex when the project is an apartment building with multiple living rooms and toilets. Within computer science, this function has a limiting factor defined by the Big O notation as $O(n^2)$. In simple terms, this means that the amount of objects that need validation exponentially increases the computational complexity needed for the validation.

2.2.4 Reporting of the results

The final phase of Eastman's automated compliance checking is the reporting of the results. As the name implies, this is the phase in which the results of the validation are presented to the user. Results of this phase can be a summary of the executed checks, descriptions of the violated rules, or drawings showing the shortcomings. Performing this effectively is important for the user to be able to make the necessary changes effectively. Ways of reporting the results can be divided into three methods, namely: 1) a textual report, 2) a visual report, or 3) a computer readable format. This section shows the advantages and disadvantages of each method.

Textual reports
The first and easiest way to report results back to a user is to generate a textual report of some kind. These reports can range from a simple yes or no value to a detailed structured document. The simplest result, a passed or not passed answer, is a fast and easy method to understand, making it great for quick validation results. However, users usually need to have a better understanding of the rules and objects that have been proven to be invalid. Reports can be generated in a structured way (e.g. per rule or per floor of a building) for the user to easily find and correct the invalid objects.

Advantages	Disadvantages
+ Easy and quick	- Hard to interpret
+ Small format	- Hard to link back to the model

Table 12: Advantages and disadvantages: Textual reporting of the results

Visual reports
For ease or reading and pinpointing the flaws of a model, the choice can be made to export the invalid objects in a visual report. This report will show exactly which object in the model is not passing the validation of a specific rule. Using this report, the user can open the model and correct the objects. Visual reports have proven to be valuable resources in a collaborative setting. Software applications like Solibri built in the option to create slideshows of invalid objects. These reports however, work best when the visual images are explained using textual explanations. For example on what the problem is or the severity of the invalidity.

Advantages	Disadvantages
+ Very detailed result	- Difficult to create
+ Easy to present and share	- Large file sizes

Table 13: Advantages and disadvantages: visual reporting of the results

Computer-readable formats

The export of invalid rules to computer-readable formats is a powerful tool for correcting invalid objects. Using this method, results can be imported into modeling software so that errors can be visually represented and corrections can be applied automatically. For instance, non-compliant elements can be adjusted based on the rules that they should have a minimum distance. This method is especially powerful when working with continuous model validation using REVIT or TEKLA software extensions that can alter the models themselves. However, the challenge with this method is that most of the infrastructure needed for this format is not available. When implementing this continuously, some software has been written to correct very specific rules. But other than that, no software has been added that can read a certain type of data file and automatically correct these errors. The newest current applications let the user import a model with all the errors and visually check each individual error.

2.3 Geometric rule validation

Driven by the usage of digital technologies and BIM, more and more digital representations are being used in the built environment. For a long time (and still to this day), 2D CAD programs were used to create a model of the object that needs to be realized. During this process, the digital representation became more intelligent, and 3D models were created. The switch from 2D to 3D representations came with a challenge to geometrically represent the model, especially with respect to data transfer for cooperation. IFC files have a geometric representation for every type of object. These representations are designed for the kernel-neutral representations for the data to be transferred between modeling software. Currently, it lacks the support for geometric validation to be performed without enrichment of the data. However, the literature on these geometric objects is less available within the AEC sector. Quite a lot of research has been done in other sectors. Outside of the built environment, different geometric ontologies are developed. This section will introduce the literature on geometric objects both inside and outside the AEC sector and dive into the geometric functions that exist and are needed for geometric rule validation.

2.3.1 Geometric Kernels

In computational geometry, a geometric kernel is a software component that deals with geometric objects and operations. Algorithms and geometric objects operate only based on methods and objects defined in a geometric kernel (Hert et al., 2007). It is the core component that makes the computer understand and manipulate geometric entities. The kernel contains the algorithms and data structures necessary to understand and handle geometric objects and computations. Its purpose is to offer a reliable and efficient way to manipulate, analyze, and measure geometric shapes. It does so by making use of geometric objects such as points, lines, vectors, polygons, and curves.

The existence of these geometric kernels allows other users to manipulate geometric representations without fully understanding how it works. They do so by taking away the mathematics complexity behind geometry, making it possible for developers to work with complex shapes, perform calculations, and analyze spatial relationships. These kernels are also used by software packages such as REVIT and TEKLA. The AEC industry uses a wide range of software where stakeholders use different tools, each with its own geometric kernels. (Pauwels and McGlinn, 2023). Within the built environment there are two mainly used kernels that are open source developed and a large number of closed kernels that are of higher quality and are licensed. The two open source developed kernels are Open Cascade Technology and CGAL. Kernels are developed with specific requirements. A kernel can be specialized in large building models or precise small-scale models. Because different kernels exist and they are developed for different programming languages like C++, Python, or C#, they can limit the interoperability.

The main disadvantage of these different geometric kernel and their specialization is that they usually do not work together or use the same functionalities and geometric representations. An example of this is the definition of a circle. A circle can be defined by a position and a radius. It can also be defined by three points. Both implementations have advantages and disadvantages. More importantly, switching between these definitions guarantees in a loss of data and takes computational time and is therefore seen as a limitation. It is therefore of great importance to define a kernel to be used that complies with these base geometric objects.

Geometry in IFC

The main issue of the difference in geometric kernels in the AEC sector was the lack of interoperability between software packages used by involved parties in a project. The architect working in REVIT and a structural engineer working in TEKLA had no way of sharing their building data other than sharing drawings and redrawing the objects. To overcome this issue, the IFC data format was developed. This format allowed the saving of geometric data in a Kernel friendly way while trying to keep the file size low for data sharing. It does so by exporting each object with its respective IfcObjectRepresentation. Each IfcObject has an IfcObjectRepresentation which is the method buildingSMART implemented to represent different objects efficiently. More specifically, the IFCObjectRepresentation has 21 representation where some have even more subclasses. Each of these subclasses defines a shape that represents the IfcObject. BuildingSMART documentation provides the way in which the geometric mesh can be derived for each of these representations. Figure 8 shows the IfcSweptAreaSolid which an representation type. The class itself defined the surface that will be swept using a method. The subclasses ExtrudedAreaSolid and SweptAreaSolid implement this by defining the way the surface is swept.

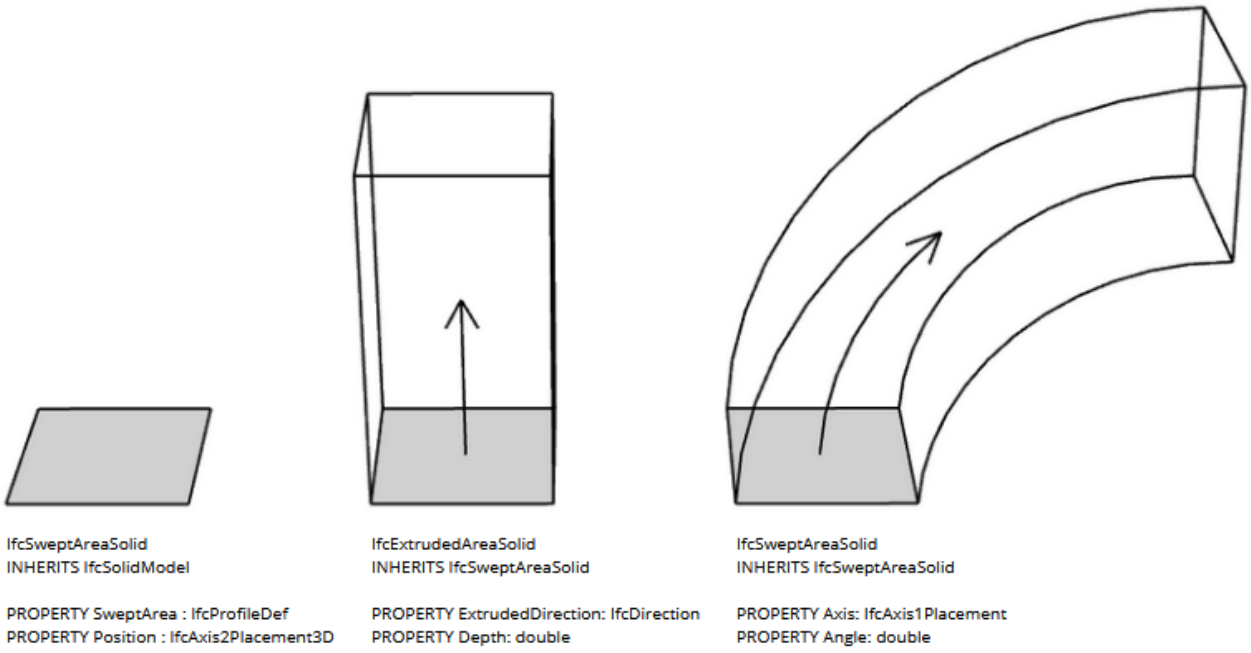

Figure 8: SweptAreaSolid representation

2.3.2 Geometry in semantics

Where the implementation of semantics in the built environment has shown promising results, there is still a limiting factor in the implementation using automated compliance checking. Where the RDF framework and the ifcOwl ontologie have structured the data for semantic rule validation, it has the same limits on geometric validation as mentioned in Section 2.3.1. Due to this lack of support for automated geometric compliance checking, the implementation of semantic technologies is flawed. This lack of geometric validations also comes from the limiting factor of RDF to handle list objects (Pauwels, Krijnen, et al., 2017). The lack of geometric rule validation in semantic web technologies creates several disadvantages. The main reason is that it forces the use of external tools to validate geometric rules.

Papers have been written on the implementation of geometric functionality using semantic technologies in the built environment. The promising bimSPARQL framework has been developed by Zhang et al. (2018). The framework was developed to extend SPARQL functions for querying IFC building data to simplify writing queries and retrieve useful information implied in 3D geometry data according to requirement checking use cases. An example of such a query is shown in Figure 8. According to their research, the framework covers all functions for the properties and quantities of simple data defined in the IFC documentation. The downside of their framework is the complexity of SPARQL queries that need to be developed by the users. The stated example on how a wall can be evaluated on its fire performance is a great addition.

```
SELECT ?space ?terminal
WHERE {
  ?space a ifc:IfcSpace ;
         ifc:predefinedType ifc:LIVINGROOM .
  ?terminal a ifc:IfcFlowTerminal ;
            ifc:predefinedType ifc:MECHANICALVENTILATIONPOINT .

  FILTER(geo:contains(?space, ?terminal))
}
```

Listing 8: SPARQL query with runtime geometric function

The SPARQL query shown in Listing 8 is based on the research and resulting implementation of BimSparql. This research suggested as further research to implement this functionality in a SHACL rule and to simplify the knowledge engineering processes. Instead of SHACL, the paper uses the SPARQL interfencing notation (SPIN). Although valuable at the time, implementing this functionality in SHACL is seen as the next step. Furthermore, BimSparql makes use of the prior enrichment of the data. During the translation of IFC data to RDF data, additional triples are generated needed for geometric validation. Much can be learned about this paper, and its recommendation will be used during the run-time implementation of geometric rule validation in SHACL rules.

2.3.3 Geometric validation and functions

Geometric functions are the functions required for the automated compliance checking of building models in the third phase of Eastmans automated compliance checking, the rule execution phase. These functions are needed to validate the rule and rule types shown in Section 2.2.3. Over the last years, research has been done on the functionality needed for geometric compliance checks. Understanding which (high-level) functions are needed as a basis for other rules is of great importance for a successful geometric compliance checker. The research by Borrmann and Rank (2009) shows geometric functions that come back consistently in rules and regulations. These functions are classified into geometric and spatial functions. The geometric functions are height, width, area, and volume. The spatial functions are contains, within, cross, disjoint, equals, intersect, touch, and overlaps. Although this research has been of great value, other research states that there is no complete understanding of the full range of functions needed for clear automated compliance checking (Amor and Dimyadi, 2021). An understandable conclusion given the many rules and needs of the AEC sector. Many of the geometric functions needed have been studied, and geometric formulas have been created and optimized. Some of these functions have been added to open source libraries that implement these functions (Pauwels, Krijnen, et al., 2017). However, these libraries usually come with their own geometric objects and software implementations. Looking at the geometric functions used in semantic web technologies, research shows that current technologies lack support for the implementation of these functions.

Spatial relations
Understanding spatial relations between objects is essential for geometric rule validation such as collision detection. It is also important for easier validations. The important types for spatial relations are topological and directional relations. For topological relations, operators such as **within, touch, and overlap** can be used (Borrmann and Rank, 2009). If an object is **below, above, or within** another object, there are well-defined terms by which spatial relations can be communicated (Göbels and Beetz, 2024). For geometric models, the definition of **in front of** can be derived when making clear agreements. This variety and quantity of possible relationships in spatial relations is the main challenge (Heise and Borrmann, 2024).

2.4 Conclusion

This Chapter has shown that the process of Automated Compliance Checking is of great importance to ensure that building models comply with rules and regulations. The automation reduces the need for manual rule checking and is less prone to error. With the implementation of BIM and the IFC file format, the implementation of Automatic Compliance Checking became more accessible because (geometric) data could be shared between modeling software and geometric kernels. Although valuable at the time, a newer shift in digital data is the use of semantic web technologies. A framework that better supports data sharing and collaboration. Unfortunately, the semantic web technology currently does not process the required geometric rule validation for Automated Compliance Checking.

Recent studies have provided proof of concepts that add geometric rule validation one way or another. The study by van den Bersselaar et al. added a geometric engine that added the geometric properties in the rule preparation phase so that it can be used in the rule execution phase. However, This method comes with an important downside, if not all the data is added in the preparation phase, it can never be validated. Secondly, adding every aspect to be validated to every possible object can create a huge amount of data of which only a small aspect is used. A method that applied geometric rule validation on runtime is the BimSPARQL study. However, that is based on older technologies. Based on the literature it is believed that a lot of issues mentioned can be solved using newer technologies when making small adaptions.

3 Methodology

The following Chapter shows the methodology of the research project. Based on the literature found in Chapter 2, a method will be proposed to add geometric rule validation to the semantic web tech stack. The Chapter starts with defining the goal of the application in Section 3.1. After these activities, the methodology for each of the four deliverables is presented. Firstly, the methodology for the parsing of the IFC data is provided in Section 3.2.2. This is followed by the methodology for adding geometric functionality and validation functions in Sections 3.2.2 and 3.2.3. Finally, the methodological development of the web application is shown in Section 3.3. The methodology developed is used in Chapter 4 to help answer the research question stated in Chapter 1.

3.1 Developing a framework for geometric SHACL validation

As mentioned in the Introduction, the goal of this thesis was to validate to what extent geometric rule validation can be added to current semantic web technologies. The literature study showed that current validation methods using semantic web technologies are done through the use of SHACL shapes. These shapes define the properties and values that an object in the database should have and validate the RDF graph for compliance. The ability to validate the existence of properties and their values allows a wide range of rules to be validated. Checking if a wall is load bearing or has a minimum fire resistance can all be done with a simple SHACL shape. However, the values of the properties are not enough for a geometric rule to be validated. A SHACL shape has no understanding of the geometric meaning of a list of strings representing the points of the surface. Because of this, it is not possible to validate whether a ventilation point is located inside a living area.

Adding geometric data during the model preparation phase is a method that recent studies have used and have proven to be capable of adding. The research by van den Bersselaar et al. (2024) showed that the geometric values (e.g. the distance between the elements) can be added to the RDF graph before validation to successfully validate the geometric requirement and ensure quality control. In addition to adding data in the model preparation phase, Zhang et al. (2018) did a promising study in which objects are gathered using SPARQL query filters. These filters, combined with a wrapping querying engine allowed for the data to be gathered and additional geometric validation to be added. A method that requires no additional data preparation. This method has two main advantages. The first is that there is no need to add the data in the model preparation phase. Not only is the data not needed, it also prevents false data from being entered where the property value does not match the actual geometric validity. Secondly, validations can be done on geometric representations that where later though of and therefore not added in the model enrichment phase. For this thesis, the choice was made to take the research from this thesis and try to add a similar implementation for SHACL shapes.

The aim of this thesis was to add geometric capabilities to the SHACL shape to add geometric rule validation to semantic web technologies. Following the research, it is possible to add a geometric understanding to SPARQL queries. As stated in the research, the next step was to use this method of implementation and use it for SHACL shapes as well. In this methodology, a proof-of-concept

was designed to add geometric capabilities to current semantic web technologies. A design is shown for each step needed to add functionality to the SHACL shape.

3.1.1 Methodological justification

For the development of a proof-of-concept, the decision was made to go for a solution-oriented iterative design process. Therefore, the design research cycle has been chosen. This cycle is defined, as seen in Figure 9, by four phases, the read, plan, solve, and check phase. The problem was first analyses in the literature study (Read). Following the literature study and highlighted in this methodology, a plan was created to develop geometric validation (plan). This validator was then developed and tested (Solve) and then validated in the following result Chapter (Check). This cycle follows the iterative and goal-oriented approach that is needed for the four deliverables to be developed successfully.

Figure 9: Design research cycle

Starting the planning of the proof of concept, a use case was defined on the functionality expected of the geometric functionality. The first certainty for this use case was that a user must be able to initiate the first step of Eastman's automated compliance checking method. This being the preparation of the building model. A process in which the decision was made about the kind of input data that will be used. The literature study has shown that there were two possibilities to consider. Using continuous data within the modeling software or using the RDF data structure to import the data. In an ideal situation, information could be directly exported from modeling software such as REVIT to be directly used in RDF Graphs for validation. However, this method has two downsides compared to the use of IFC data. Firstly, this way is not supported by modeling software. Where a lot of effort has been put into the export of IFC data, no wide-scale implementations are developed to save the data using semantics. Secondly, the format of the data is also not developed. Where the ifcOWL ontology has been developed that covers the entire IFC data structure, there is no widely accepted ontology for direct modeling data. For this reason, the decision was made to use IFC data as the start of the use case. The full use case is shown in Figure 10.

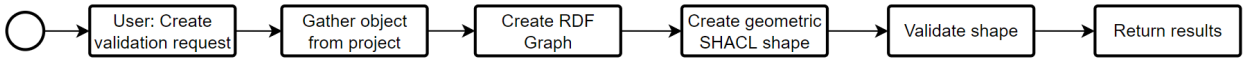

Figure 10: Geometric validation use case

As mentioned in the literature, this use case had a major problem that needed to be solved. There was no possibility of creating a SHACL shape that can do geometric validations based on IFC objects.

A SHACL shape can validate relations defined by triples and the values they contain. For example, it can be checked if a wall has a fire rating and what value types are allowed. This functionality can be extended using the IF-THEN-ELSE functionality, where the fire rating can be different depending on the type of wall. However, no functionality can be created where the literal value of object geometry can be used in geometric functions.

Adding the functionality to the semantic web tech stack was previously attempted by Zhang et al. as shown in the literature. This 2018 study showed that the functionality could be implemented directly in the SPARQL query to collect data from the database. Based on this research, on the run-time implementation of geometric rule validation. For this thesis, the idea is to implement the same kind of implementation in a SHACL by developing the functionality for the SHACL shape seen in Figure 9.

```
ex:LivingRoomShape
  a sh:NodeShape;
  sh:targetClass ifc:IfcSpace;
  sh:property [
    sh:path ifc:contains;
    sh:class ifc:IfcFlowTerminal;
    sh:message "Living rooms must contain a mechanical ventilation
       point.";
    sh:js:filterFunction geo:contains;
  ] .
```

Listing 9: SHACL rule with runtime geometric function

3.1.2 Deliverables

During the planning phase of the proof-of-concept for the SHACL shape, it had been decided that to fully implement and validate the functionality, four deliverables had to be developed. The first deliverable is an IFC class library capable of importing IFC data formats and exporting it to RDF graphs. Currently, there are open source packages to export IFC data to computer readable code and software packages to translate the data to RDF graphs. However, these packages were developed for the purpose of known semantic web implementations. Based on my personal software development abilities, the assumption was made that there would be much added value in developing this class library. These advantages are elaborated on in Section 3.2.1. The second deliverable is the function library with the underlying geometric kernel that adds geometric functionality to the class library. For this thesis, this functionality will be written in a project that extends the class library, split functionality for better structure, and separated responsibility. The third deliverable is the geometric validators needed for rule validation. This validator will be able to transform a user validation request and generate a SHACL shape. Because the validator expects the geometric request, it will understand that it needs to collect the geometric representation of the objects and use the geometric functions for validation. The final deliverable will be a web application that implements the rule-validation framework. The web application will be able to accept IFC files uploaded by the used. These IFC files are translated into objects represented in the IFC class library. Using the interface, the user can then validate these objects using the validators. When the validators return

the correct results without adding additional data, it is proved that the framework can add runtime geometric validations.

Figure 11: The four research activities

Given the accessibility of the SPARQL engine, the complexity of SHACL frameworks and the time constraint of this thesis, it is unrealistic to expect native integration of geometric functions within SHACL. Therefore, the decision has been made to add this geometric validation through custom functionality that validates based on the RDF Graph using SPARQL queries. Proving that a SHACL shape can be created that can be validated based on the geometric functionality will prove that this can be implemented in a full-scale SHACL framework.

3.1.3 Software architecture

For the software architecture of the four deliverables, the first decision made is the choice to develop the framework using the Microsoft tech stack (or free alternatives when needed). This decision is made based on the authors knowledge on this tech stack and its strong support for object-oriented design process. The architecture will make use of Microsoft's coding best practices and the SOLID principle. Especially to preserve the shared responsibility and engineering simplicity. The first deliverable, responsible for providing the data classes and IFC parser, will be represented as a (NuGet) package named IfcSharp. The second and third deliverable share responsibility. These deliverables extend the functionality of the first deliverable with a geometric kernel and validation methods. This extension will be represented in a (NuGet) package that extends the IfcSharp package. This package will be IfcSharp.Geometry.

The final deliverable, the web application, requires three parts. A front end, back end, and a database. The front-end is developed as a single-page application using the React framework. React is chosen over Microsoft's Blazor because of its better support for the chosen rendering technology Three.js. The front end communicates with a C# backend via RESTful API endpoints. This back-end handles the file-parsing, validation logic, and interaction with the MongoDB database. The solution diagram for this final deliverable is shown in Figure 12 separates responsibility, supports scalability, and aligns with proven design patterns for software engineering. It added the geometric validator that can intersect the SHACL shape validation when a geometric validation request is received.

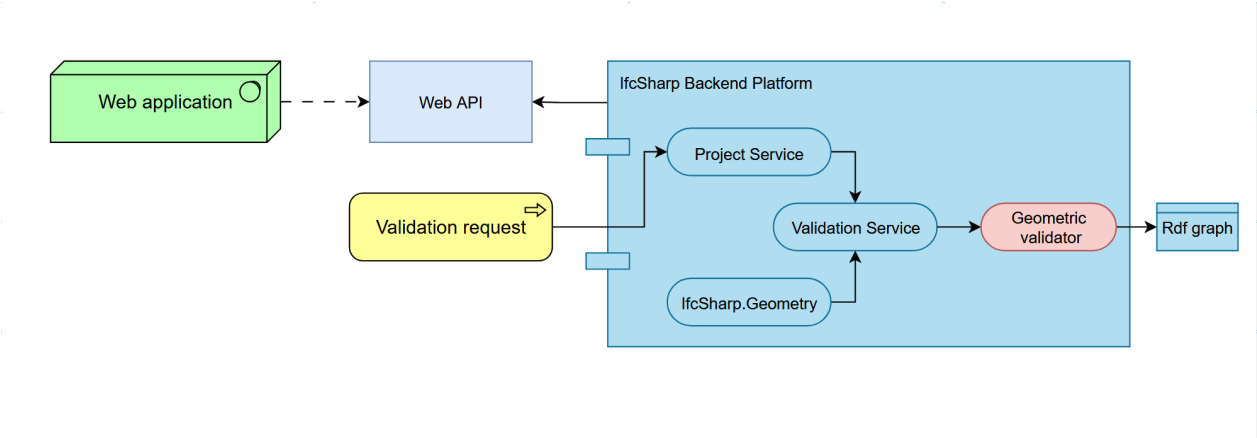

Figure 12: Project architecture

3.1.4 Research validation

To validate the research goals and the framework, the web application must be able to validate geometric rules without further enriching the IFC building model. For this validation, three rules have been collected from the Dutch Building Regulations. These three rules are the following:

1) Within a mechanically ventilated dwelling, each living area must include a ventilation point (Art. 4.122).
2) Each residential function must have a designated space for a cooking appliance (Art. 4.176).
3) An open combustion appliance is not placed in a toilet or bathroom (Art. 3.75).

Rule 1 and 2 implement the geometric functionality for the contains function and rule three implements the adjacent verb. When the user can use the web application to define terms such as a dwelling, living area and mechanical ventilation point, it can run the validation requests. Based on the result of these validations, it is proven to what extent geometric rule validation can be added to semantic web technologies at runtime without additional enrichment other than the IFC data structure.

3.2 Adding geometric capabilities to SHACL

With the literature study completed, there was a better understanding of why geometric rule validation was not possible in current semantic web technologies. The reason for this was that the SHACL shapes could only use literal values for the representation of values. With just this value, the SHACL engine has no understanding of the geometric meaning of literal values. To add this functionality, the idea was to extend the SHACL engine functionality with geometric capabilities. To add this functionality, it was important that there is a good understanding of the data inside the graph and that it is well structured. For this, the decision was made to base the solution on the ifcOWL ontology. This ontology represents all IFC classes and therefore the IfcShapreRepresentations that defined the geometric meshes of the data. It is believed that when SHACL can use these representations, geometric rules can be validated. For the design of this process, the four phases of Eastman's compliance checking are followed. First, the design was made to translate the data from IFC files to RDF graphs. Second, the geometric extensions were added to extract the meshes from this class. Finally, the validation functions where added which the SHACL shape could use. At this time it was believed that by implementing these three functionalities, a SHACL shape could be extended with geometric functionality.

3.2.1 Model translator

The first developed deliverable was the IfcSharp class library capable of translating the IFC data format into its respective classes. This package has three requirements. First, it must contain all the classes represented in the IFC data format as defined by BuildingSMART. Second, it must be able to translate the most commonly used IFC file format (the STEP file format) to these classes in the class library. Finally, it should support the functionality to export the classes to RDF Graph nodes and triples.

As noted in Section 2.2.1, the IFC data structure provides a standardized format to describe the building elements. Even though there are pre-existing libraries that have the required class library and converter. These libraries have been used by many studies and have proven their worth. However, when looking at future deliverables, it is believed that more flexibility is needed than the libraries provide. Using software principles such as reflection, it is of great value to have total control over the class library and its structure. To start the prototype, the class library that contains all the IFC data types was developed. An example of these classes is the IfcObject shown in Listing 10. Important features to note are the properties that have jsonInclude, XmlAttribute and RdfObject attribute. The jsonInclude attribute and XmlAttribute, as the name implies, lets the software know how to handle the objects during json and xml parsing respectively. The RdfObject, provides functionality based on a predicate URI used in RDF triples to connect the nodes.

```
public abstract partial class IfcObject : IfcObjectDefinition
{
        [JsonInclude]
        [XmlAttribute]
        [RdfObject(PredicateConstants.IfcObjectType)]
        public string? ObjectType { get; set; }

        [JsonInclude]
        [XmlAttribute]
        [StepIgnore]
        [RdfObject(PredicateConstants.IfcIsDefinedBy)]
        public List<IfcRelDefines> IsDefinedBy { get; set; }

        public IfcObject()
        {
        IsDefinedBy = [];
        }
}
```

Listing 10: IfcObject class

With the class library created, the development of the data parser started. As mentioned, this parser will make use of the EXPRESS data modeling language, the language in which a STEP file is written, and translate it to the objects in the class library. An example of a IfcSpace representation in EXPRESS format is shown in Figure 13. In this image, the step line is divided into the three parts needed for the translation. The line number, the object type, and the values that represent the object.

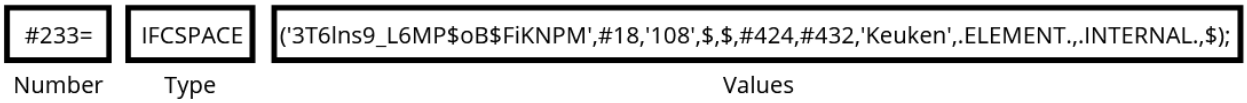

Figure 13: Parts of the step line

To parse the EXPRESS data line into its respective object following the class library, the function shown in Listing 11 was developed. Like the class and object analogy mentioned above, an object needed to be created of the class reflecting the object type. After creating this object, the values can be matched with the parameters to finalize the creation of the object. When all lines in the EXPRESS files are parsed, the IFC file is parsed.

```
FUNCTION GetIfcParentFromIfcLine(T, stepLine, ifcObjects, ifcLines)
    // Extract the type from the IFC line
    type ← GetTypeFromIfcLine(stepLine)

    // Create an instance of the class with the determined type
    returnObject ← CreateInstance(type OR GetTypeFromIfcLine(stepLine))

    // Retrieve properties of the class filtered by inheritance
    properties ← GetPropertiesOfTypeFilteredByInheritance(type)

    // Extract values from the IFC line
    values ← GetValuesFromIfcLine(stepLine)

    // Assign values to corresponding properties
    FOR i FROM 0 TO LENGTH(values) - 1 DO
        SetValueOnObjectByPropertyInfo(returnObject, values[i],
           properties[i], ifcObjects, ifcLines)
    END FOR

    // Return the created object cast to type T
    RETURN returnObject AS T
END FUNCTION
```

Listing 11: Pseudo code for the step line parser

The matching of the values and the parameters of the class can be done using the data structure of the step line and the structure of the IFC classes. As mentioned in section above, the data parser makes use of a method known as reflection. Reflection takes a different approach to programming. Where usually things are predetermined, with reflection, the computer tries to understand what it is handling while running. Reflection can be used for the parsing of the IFC objects because of one reason. This reason it that the EXPRESS data lines are always structured in the same order. This order is defined by BuildingSMART and follows the inheritance of the classes. As shown in Figure 14 the parameters of the object are listed in the same order the properties are defined in the class structure. Reflection can be used for each object to get all the order properties of the class and the values of the EXPRESS line. By injecting the data into the properties the object will be created.

```
#233=IFCSPACE('3T6lns9_L6MP$oB$FiKNPM',#18,'108',$,$,#424,#432,'Keuken',.ELEMENT.,.INTERNAL.,$);
                              1               2    3  4 5   6     7      8          9          10    11

CLASS IfcRoot INHERITS IfcParent
  1  PROPERTY GlobalId : string
  2  PROPERTY OwnerHistory : IfcOwnerHistory
  3  PROPERTY Name : OPTIONAL string
  4  PROPERTY Description : OPTIONAL string
END CLASS

CLASS IfcObject INHERITS IfcObjectDefinition
  5  PROPERTY ObjectType : OPTIONAL string
END CLASS

CLASS IfcProduct INHERITS IfcObject
  6  PROPERTY ObjectPlacement : IfcObjectPlacement
  7  PROPERTY Representation : IfcProductRepresentation
END CLASS

CLASS IfcSpatialStructureElement INHERITS IfcProduct
  8  PROPERTY LongName : OPTIONAL string
  9  PROPERTY CompositionType : IfcElementCompositionEnum
END CLASS

CLASS IfcSpace INHERITS IfcSpatialStructureElement
  10 PROPERTY InteriorOrExteriorSpace : IfcInternalOrExternalEnum
  11 PROPERTY ElevationWithFlooring : OPTIONAL double
END CLASS
```

Figure 14: Relation between step line and class structure

3.2.2 Extracting geometric objects

With the part of the framework designed to translate the IFC file to the IFC class library, the geometric data extraction can now be developed. As mentioned in Section 2.3.1, the IFC format does not work with geometric functions used by geometric functions. Therefore, the framework will implement functionality to extend the class library with geometric representations. As mentioned in Section 2.3.1, the IFC data structure provides a data efficient way of exchanging the geometric representation. Every object in the IFC data structure that can represent a geometric object is derived from the IFCProduct. This class implements IFCObjectRepresentation and IFCObjectPlacement. Together, these values represent the position and mesh of each object. The IFCObjectPlacement is defined as a single point. This point is derived from translating over multiple positions which the object can relate to. For example, the position of the object can be relative to the position of the floor on which the object is.

To add the geometric representation to the class library under the NuGet package IfcSharp.Geometry the package must have the geometric representation classes. For this, the class library must be extended with the basic functionality of a geometric kernel. For the development of the prototype, only limited functionality will be required. The geometric representations needed are only points, lines, surfaces, and meshes. Once these are added as classes, they can be used to extract the geometric representation from the IfcObjectRepresentation types mentioned above. For this to work, the class library needs to know how to extract the mesh, surface, and position for each geometric representation. To add this functionality, the framework will extend the representation classes mentioned above with the functions that can extract these geometric representations. An example of this is the pseudocode seen in Listing 12. With this functionality, every time an IfcExtrudedAreaSolid is found as a property of an object. We can use the function ”GetSurface” to extract the Surface of the extruded area solid. The only object needed is the IfcObjectPlacement of which the base position of the object can be determined.

```
CLASS IfcExtrudedAreaSolidExtensions

    FUNCTION GetSurface(extrudedAreaSolid, objectPlacement) -> Surface
        // Get the local axis from the object placement
        localAxis ← GetLocalAxisFromObjectPlacement(objectPlacement)
        localAxis.ApplyAxisPlacement3D(extrudedAreaSolid.Position)

        // Check if SweptArea is of type IfcArbitraryClosedProfileDef
        IF extrudedAreaSolid.SweptArea IS TYPE
           IfcArbitraryClosedProfileDef THEN
            acpd ← extrudedAreaSolid.SweptArea AS
               IfcArbitraryClosedProfileDef

            // Check if OuterCurve is of type IfcPolyline
            IF acpd.OuterCurve IS TYPE IfcPolyline THEN
                polyline ← acpd.OuterCurve AS IfcPolyline
                points ← EMPTY LIST

                // Convert polyline points to 3D points
```

```
                FOR EACH point p IN polyline.Points DO
                    newPoint ← NEW Vector3
                    newPoint.X ← p.Coordinates[0]
                    newPoint.Y ← p.Coordinates[1]
                    newPoint.Z ← IF p.Coordinates[2] EXISTS THEN
                       p.Coordinates[2] ELSE 0
                    newPoint ← Translate(newPoint, localAxis)
                    ADD newPoint TO points
                END FOR

                RETURN NEW Surface(points)
            ELSE
                THROW EXCEPTION "OuterCurve type not implemented for
                   surface generation"
            END IF

        // Check if SweptArea is of type IfcRectangleProfileDef
        ELSE IF extrudedAreaSolid.SweptArea IS TYPE
           IfcRectangleProfileDef THEN
            rpd ← extrudedAreaSolid.SweptArea AS IfcRectangleProfileDef
            localAxis.ApplyAxisPlacement2D(rpd.Position)

            depthDirection ← localAxis.XDirection
            widthDirection ← localAxis.YDirection

            // Calculate temporary points for defining the rectangle
            temp1 ← localAxis.Position + (widthDirection * rpd.YDim /
               2)
            temp2 ← localAxis.Position - (widthDirection * rpd.YDim /
               2)

            // Define the four corner points of the rectangle
            point1 ← temp1 - (depthDirection * rpd.XDim / 2)
            point2 ← temp2 - (depthDirection * rpd.XDim / 2)
            point3 ← temp2 + (depthDirection * rpd.XDim / 2)
            point4 ← temp1 + (depthDirection * rpd.XDim / 2)

            points ← [point1, point2, point3, point4]

            RETURN NEW Surface(points)
        ELSE
            THROW EXCEPTION "SweptArea type not implemented for
               surface generation"
        END IF
    END FUNCTION
END CLASS
```

Listing 12: Psuedo code for the extraction of a surface from the IfcExtrudedAreaSolid

The implementation of direct translation of IFC objects to representations other than the mesh come with possible issues. Defining a room using the IfcExtrudeAreaSolid representation assumes that the surface area is drawn and the mesh is pulled upwards from this. However, other ways of defining the mesh are possible. For example, the mesh can start from the ceiling and push the room down. Most of the functionality still works but some errors might occur. This becomes interesting when a (square) room would be defined by the surface area being a wall and the mesh being created by pulling the wall into the space. The three options mentioned are shown in Figure 15. From the perspective of the IFC file data structure, all three options are valid ways of defining and sharing the mesh. Solving this issue will be beyond the scope of this thesis. Mainly because the exporting of IFC data is managed by technical drawing software and the exported models have not shown these errors yet. Therefore, creating a solution for this problem would take a lot of time and little gains to be made.

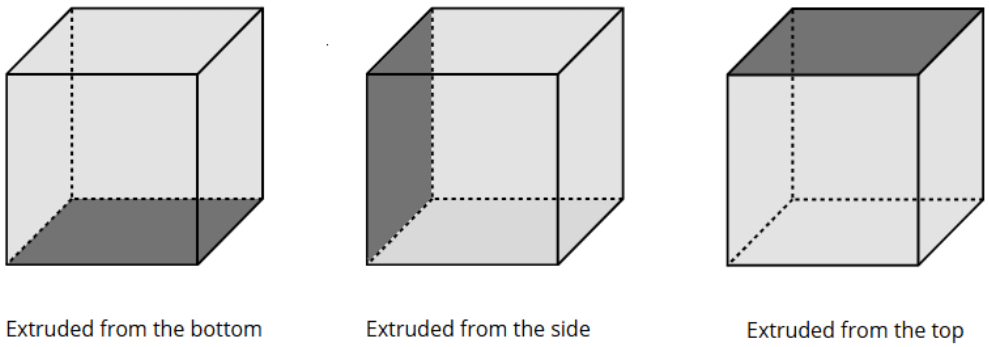

Figure 15: Three extrusion directions for a room definition

3.2.3 Geometric rule validation

With the class library being able to parse IFC files and extract the required geometry, the third deliverable was developed. This is the validation of the geometric rules. This step is a combination of Eastman's second and third phase. The rule interpretation and rule execution phase. The research in section 2.2.2 shows that there are three main approaches to interpret the rules. Manual interpretation, high-level implementable methods, and the use of natural language processing (NLP). From these approaches, the decision has been made to make use of high-level implementable methods for the interpretation and execution of the rules. The high-level functions work well with current semantic web technologies and their use within sparql queries based on the BimSPARQL research. Given that the manual rule interpretation is time consuming, this option was never truly considered. The NLP process will take great steps in the future, but current implementations are still limited and inaccuracies. Research shows that there are two downsides to the use of high-level implementable methods. First, not all methods required and defined, they also differ per discipline. Second, the same function (e.g. contains) can have different implementations when applied to points, surfaces, or meshes. For example, checking if a point is in a surface or a mesh translate to a 2D or 3D function. For both of these downsides it is believed that the solution lies in establishing a way the high-level methods are working. For most of these, extensive research has been done. Secondly, the goal of the prototype is to evaluate to what extent it is possible to

apply geometric rule validation during the runtime of the application, the type of function and the specific workings are therefore less important. Because of this, the decision to go for the high-level methods is still considered the right solution.

The geometric validator framework starts with the validator for each of the methods that need to be validated. These validators contain the functionality needed to receive geometric objects and return if the objects match the validation. Validators can be given two IfcProducts. As mentioned in section 3.2.2, each of these objects has an IfcObjectRepresentation and an IfcObjectPlacement from which the geometric objects can be extracted. The validator makes use of this geometric extraction based on the needs of the validator. An example of this is the inside validator shown in Listing 13. This validator gets the meshes of both objects and validates if object a is inside object b. If so, the validator returns true. If not, it returns false.

```
CLASS InsideValidator IMPLEMENTS IValidator

    FUNCTION Validate(a, b) -> BOOLEAN
        // Get the position of IfcParent a
        success ← TryGetPositionOfIfcParent(a, positionA)
        IF NOT success THEN
            THROW EXCEPTION "Position retrieval not implemented"

        // Get the surface of IfcParent b
        success ← TryGetSurfaceOfIfcParent(b, surfaceB)
        IF NOT success THEN
            THROW EXCEPTION "Surface retrieval not implemented"

        // Check if positionA is inside surfaceB
        RETURN IsPointInSurface(positionA, surfaceB)
    END FUNCTION

END CLASS
```

Listing 13: Psuedo code for the inside validator

The framework adds a general validator for easy single-entry to the validation library. This validator is shown in Listing 14. Instead of having to use a specific validator (e.g. InsideValidator), this validator only needs the verb as defined above. Through the use of reflection, as mentioned in Section 3.2.1, this general validator knows from the verb what specific validator needs to be instantiated and used. This functionality resembles the SHAQL query shown in Listing 9. From the geo:contains the verb can be selected. For each living room and IfcFlowTerminal selected, the validating function can be called.

```
CLASS Validator
    FUNCTION Validate(validationType, a, b) -> BOOLEAN
        // Get the validator name from the enum
        memberInfo ← GetMemberInfo(validationType)
        validatorNameAttribute ← GetCustomAttribute(memberInfo[0],
           ValidatorNameAttribute)
        validatorName ← validatorNameAttribute.ValidatorName

        // Get the assembly containing the validators
        assembly ← GetAssembly(Validator)

        // Find the correct validator type
        types ← GetTypesFromAssembly(assembly)
        type ← FindFirstTypeWithName(types, validatorName)

        // Create an instance of the validator
        validatorInstance ← CreateInstance(type)

        // Retrieve and invoke the "Validate" method
        method ← GetMethod(type, "Validate")
        RETURN InvokeMethod(method, validatorInstance, [a, b])
    END FUNCTION
END CLASS
```

Listing 14: Pseudocode for the general validator

Based on the validation type, by reflection, the validation framework understands what function to call. One of the validation types implemented is the inside validator. This function validates whether the second object is within the first object. The decision had been made to get the position of the IfcLocalPlacement and check if it is inside the surface of the first object. This function makes use of the winding number algorithm which checks if a point is within a polygon. This functionality is shown in Listing 15. Important to note from this functionality is that the way the validators are defined, the proof-of-concept can be easily extended or changed with different functionality. In this way, the proof of concept can be extended with different SHACL shape high-level methods for geometric rule validation. For example, once it is defined how a surface and a mesh can be compared for the verb adjacent. The AdjacentValidator has to be added with the functionality, and then the SHACL shape can handle the adjacent keyword for validation. The downside of this decision was the validation speed of the application. Reflection is a slower process than selecting the specific validator. Since the purpose of this thesis is to extend the functionality, the

decision was made to go for the more flexible option over the fastest solution.

```
public static bool IsPointInSurface(Vector3 p, Surface s)
{
    if (s.Edges.Count < 3) { return false; }

    int windingNumber = 0;

    for (int i = 0; i < s.Edges.Count; i++) {
        Vector3 start = s.Edges[i].StartPoint;
        Vector3 end = s.Edges[i].EndPoint;

        // Check if the point is on the upward edge
        if (start.Y <= p.Y) {
            if (end.Y > p.Y && IsLeft(start, end, p) > 0) {
                windingNumber++;
            }
        }
        // Check if the point is on the downward edge
        else {
            if (end.Y <= p.Y && IsLeft(start, end, p) < 0) {
                windingNumber--;
            }
        }
    }

    // The point is inside the polygon if the winding number is not
       zero
    return windingNumber != 0;
}
```

Listing 15: winding number algorithm

3.3 Web application

The final result of this thesis was the development of a web application that implements the validation prototype in a user-friendly interface. This application allows the user to upload the IFC file to perform geometric rule validation at run-time using the semantic web workflow. The general architecture of this web application has been described in Section 3.1.3. For the correct implementation, the web application will be broken down into the four phases of Eastman's automated compliance checking. Section 3.3.1 shows the translation of the building model. This is the part where the user uploads the IFC file and defines the modeled objects and how they can be defined. Section 3.3.2 shows how the user can make use of the tools available to interpret and implement the rules using the validation process and execute these rules to obtain the result. The way in which the prototype returns these results is shown in Section 3.3.3

3.3.1 Building model translation

Before the building model can be prepared, the web application needs to understand some things about the user for authorization. This is a simple user login system that for this thesis does not need further elaboration. With the user logged in, a project can be created. Upon the creation of this project, the user can add an IFC file upon which the translation of the building model starts on the API. It does so by adding the IFC file data to the project in the database. This way, the data can be used when a validation request is requested for the project.

With the project validation being performed at runtime, the building model translation phase is the phase in which the application must learn about the information available in the model. Without this knowledge, the application is never able to know which objects in the model need validation. Looking at the first rule that will be validated which is "Within a mechanically ventilated dwelling, each living area must include a ventilation point", the application needs to know what both a "living area" and a "ventilation point" is. Looking at the tradition SPARQL query, this is done by using a sub-query or a construct. To select all "ventilation points" a query like the one in Listing 16 can be used.

```
PREFIX ifc: <https://standards.buildingsmart.org/IFC/DEV/IFC4_OWL#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?flowTerminal ?nameValue
WHERE {
  ?flowTerminal rdf:type ifc:IfcFlowTerminal ;
                ifc:name ?name .

  ?name ifc:hasString ?nameValue .

  FILTER(STRSTARTS(?nameValue, "Exhaust Grill"))
}
```

Listing 16: SPARQL query for selecting living areas

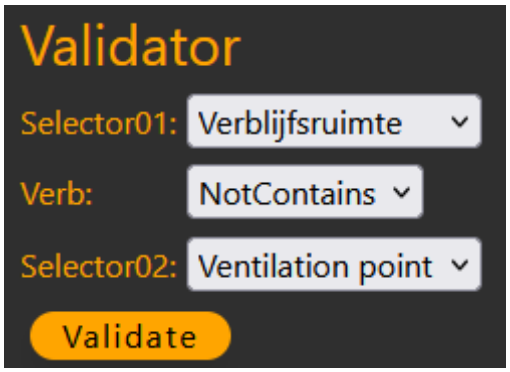

Figure 16: User interface for rule execution

For the application to understand how to select a ”ventilation point” it must implement a similar pattern. For this, the application implements a similar functionality to the SPARQL Selector and is therefore given the same name. The selector allows the user to define how objects are called and by filtering on which parameter and value the objects can be found. In this case: ”Ventilation point IS IfcFlowTherminal WHERE name STARTSWITH ”ExhaustGrill”.

3.3.2 Rule interpretation and execution

With an application that can define how objects can be collected, the user must be able to use these in the creation and validation of rules. For the workings, we look back at the first rule that must be validated: ”Within a mechanically ventilated dwelling, each living area must include a ventilation point”. Translating this rule to high-level methods, it should report on the following rule: ”LivingRoom does not CONTAIN Ventilation point”. The user can create this rule shown in the UI in Figure 16.
When receiving this request in the application layer of the API, the software will use a SPARQL query to collect the requested object the same way a SHACL shape would. Based on the exported nodes and triples from the project’s IFC file, a SHACL graph will be created to validate the models. The geometric representations of each object are collected and validated using the verb provided by the user. This validation functionality is shown in the pseudocode shown in Listing 17 .

```
FUNCTION HandleValidationRequest(request, cancellationToken)
   RETURNS ValidationRequestResponse
response ← NEW ValidationRequestResponse()

 // STEP 1: Get selectors
userIdFilter ← Filter where userId == currentUserId
s1NameFilter ← Filter where name == request.Selector01
selector1 ← Get selector1 from DB with filters

s2NameFilter ← Filter where name == request.Selector02
selector2 ← Get selector2 from DB with filters

// STEP 2: Get project and IFC file
project ← Get project by ID
```

```
ifc ← Get IFC file by project.IfcId

// STEP 3: Parse IFC file
ifcFile ← Parse bytes into IfcFile object

// STEP 4: Filter objects by selectors
objects1 ← Filter ifcFile.Components by selector1
response.Logs.Add(count of objects1 found)

objects2 ← Filter ifcFile.Components by selector2

// STEP 5: Enrich graph
g ← NEW Graph()
allObjects ← Concatenate objects1 and objects2

FOR EACH object IN allObjects DO
    baseUri ← Construct URI from object type, project ID, and
       GlobalId
    mainUri ← Get RDF object by GlobalId from DB
    IF mainUri IS NULL
        Log warning
    Add all RDF triples related to baseUri to graph g
END FOR

store ← NEW TripleStore()
store.Add(g)
processor ← NEW SparqlQueryProcessor(store)

// STEP 6: Run SPARQL queries (for selector1 and selector2)
query1 ← Build and parse SPARQL query using selector1
results1 ← Execute query1 on processor

query2 ← Build and parse SPARQL query using selector2
results2 ← Execute query2 on processor

// STEP 7: Reconstruct objects from graph
objects1b ← []
FOR EACH object IN objects1 DO
    baseUri ← Construct URI from object
    enriched ← Get enriched object from graph g
    Add enriched to objects1b
END FOR

objects2b ← []
FOR EACH object IN objects2 DO
    baseUri ← Construct URI from object
    enriched ← Get enriched object from graph g
    Add enriched to objects2b
```

```
    END FOR

    // STEP 8: Validate combinations and collect results
    FOR EACH object1 IN objects1b DO
        FOR EACH object2 IN objects2b DO
            isValid ←
               Validator.Validate(request.ValidationRequest.ValidationType,
               object1, object2)
            IF isValid THEN
                response.ValidatedPairs.Add((object1, object2))
            END IF
        END FOR
    END FOR
END FUNCTION
```

Listing 17: Handle validation request pseudocode

3.3.3 Reporting of the results

With the ability to upload the model, select the object in the model and send requests to validate the model, the final part is to return the results and show them to the user. For this, a web model viewer is implemented using the THREE.js package. The API endpoint returns the validation results showing the invalid objects (shown in red), the objects that make the selector invalid (shown in orange), and a list of messages explaining the validation results. Based on the tests conducted during development, the application should be able to validate the rules defined in Section 3.1.4. To verify this, these rules are validated in the following Chapter.

This page is intentionally left blank

4 Results

The following chapter shows the results of the framework implementation. First, the research question defined in the introduction is answered. After which, the answer to the research question is proven by using the proof-of-concept to validate a building model based on the three rules shown in the methodology. For each of these rules, the process is shown and any positive or negative results are shown. Finally, the computational time of the application is shown using a second model.

4.1 Answering the research question

As mentioned in the introduction, the aim of this thesis was to answer the following research question: *'How can geometrical computations be added to the current state of automated compliance checking using linked building data to validate a building model of the IFC format?'* The first subquestion sought to delve deeper into the current workings and semantic web technologies and give an understanding of why geometric rule validation cannot be done. The answer to this question turned out to be the current implementation of SHACL shapes. These shapes only allow for the validation of existence and value. Current SHACL engines are not capable of understanding geometric validations, since they have not been defined. The main reason for this is that a SHACL shape currently has no geometric understanding of the values that are defined in the literals. To add this understanding to a SHACL engine, it needs to understand how to collect the geometric values from the RDF graph and know which validators to use upon a validation request. The logic semantic way to do this, is to make use of ontologies that are readily available and define how data is parsed. For this, the ifcOWL ontology was used. This ontology defines all the IFC data classes and therefore all geometric representations. By having the SHACL engine understand how this geometry can be parsed to the kernel.

The answer to extract this data was found in the second research question on how to parse the IFC data format into an RDF graph. The literature had shown that there are open source libraries available for both the parsing of IFC files to data classes and a library to turn classes into rdf graphs where data loss can be kept to a minimum. However, for this thesis the decision was made to create a custom class library capable of both parsing objects from the IFC file and export these to RDF graphs using the ifcowl ontology. This flexibility of a custom class library gave the flexibility to extend the library with the required functionality to extract the geometric values using a custom geometric kernel. This kernel allowed for IfcShapeRepresentations to be translated to points, surfaces and a mesh used for geometric rule validation.

The third research question tried to answer the question of what validation methods had to be made available for successful geometric rule validation. Unfortunately, a full answer to this question could not be found within the goal and time limit of this thesis. Where earlier research had stated that all rules had been defined, later studies showed that, due to a growing industry, new types of validation were constantly found. Research into the rule types following the automated compliance checking phases and the earlier bimSPARQL study had shown that the best approach to adding validation would be through high-level verb methods, which can be implemented in SPARQL queries and possibly in SHACL shapes. These high-level methods such as ”Underneath”

and ”Contains” were deemed sufficient for the goal of this thesis and no further research was done on the subject.

The final research question combines all the subquestions and steered towards the proof-of-concept that has been designed. With the answered research questions, it was concluded that SHACL shape functionality could be extended with geometric functionality by making the SHACL shape understand high-level verbs for geometric rule processing. This understanding could be added by following the ifcOWL ontology and its defined geometric functionality. This was done in the proof-of-concept which proves the claim that by extending the SHACL shape with this functionality, geometric rule validation is possible using semantic web technologies.

4.2 Proof of concept

Answering the research question has led to the development of a proof-of-concept that allowed the geometric validation of automated compliance checking. This proof of concept was believed to be capable of validating the rules defined in Section 3.1.4 of the methodology. When the web application is capable of validating these rules without additional building model preparation, it is believed that the SHACL shape can be extended with geometric functionality. To validate this, a BIM model is created that can be validated on these rules. This model has been developed where rule 1 and 3 are not met. To prevent false negatives in the results, the model will comply with rule number 2. To comply with the rules, the model needs the information defining the living areas, mechanical ventilation points and appliances. Where the mechanical ventilation point and appliances can be filtered by name, the living areas need more data to be found. For this, the property ”isVerblijfsgebied” is added to define if an IfcSpace is a living area or not. This model is designed in Revit and exported using the exported IFC file. The ground and first floor of this model are shown in Figure 17 and 18 respectively.

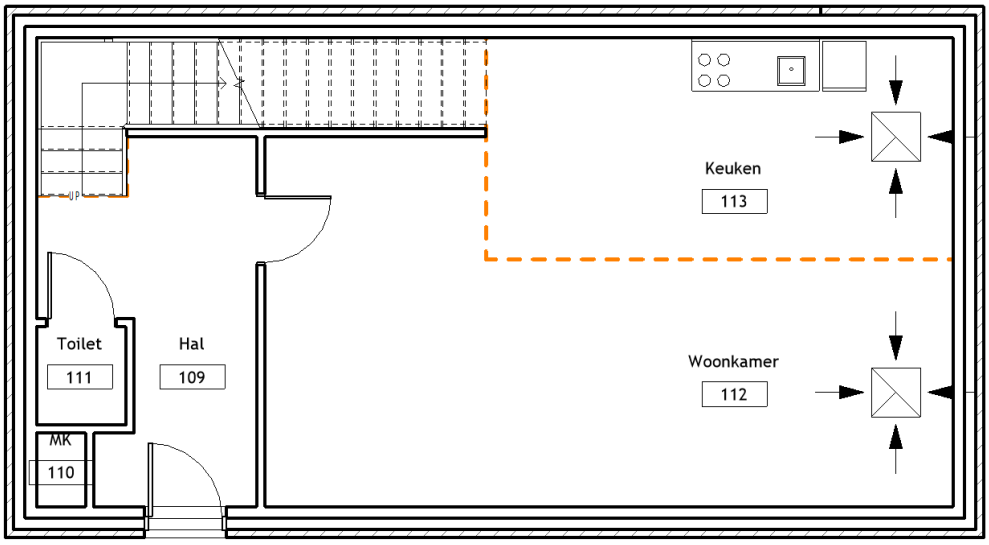

Figure 17: Ground floor of the building model

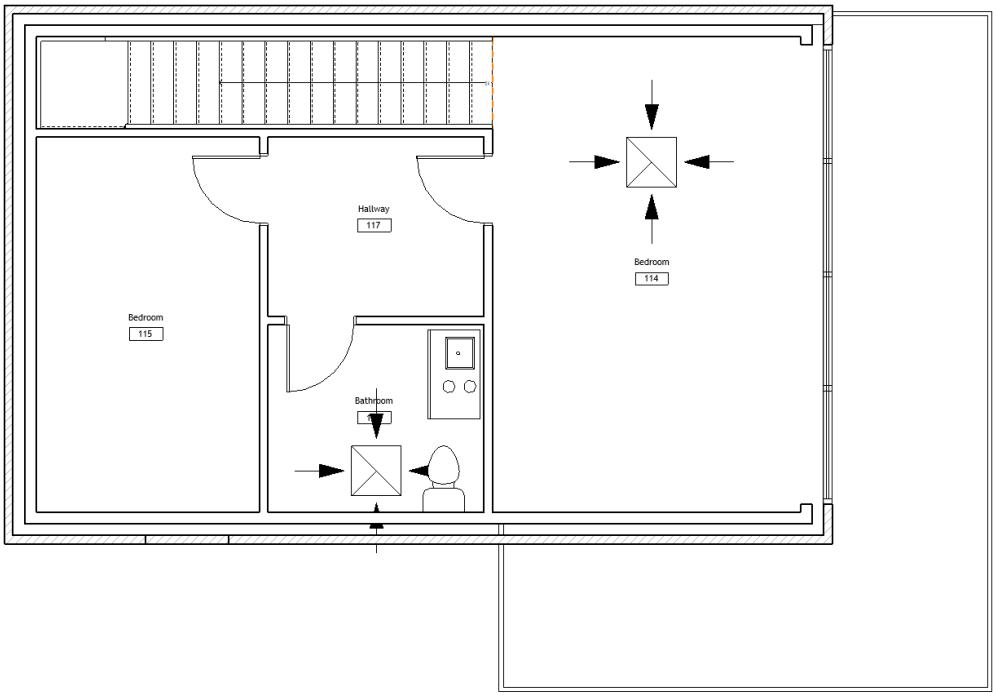

Figure 18: First floor of the building model

4.3 Building model preparation

The first step of the proof of concept is to parse the exported IFC file into the RDF graph. For this thesis, only IfcObjects are enriched with the RDF predicates for parsing to RDF triples. This is done by first translating the IFC file to the classes from the class library. Upon manual inspection, each of the lines in the IFC file format is accounted for in the parsed objects. A result to note is that there is the possibility that not all IFC data objects are available in the class library. When a class representation of an IFC object does not exist in the class library, the parsing still continues, and an error is given. This data is then lost during translation and cannot be used in rule validation. Since this was not the case for the created IFC model, we can proceed and check the results of the parsing to the RDF graph. With the IFC file created, a visual check can be done on both the parsing of the classes and the model viewer created in the web application. This is done by uploading the IFC file using the web API and receiving the objects to be loaded into the viewport. Doing this for the IFC file results in the viewport shown in Figure 19. From this visual check, no missing objects are found. However, one downside of the custom geometric extractor is shown. As shown in the literature study, the IFC classes have not only their own geometric representation but also a relation. This relation is used for the geometric representation of a window to be removed from a wall. Given the complexity of boolean results and the time constraints to add this to the custom geometric engine, the decision was made not to implement the geometric relations. This result is shown in the figure where there are no holes in the walls for the windows and doors. This could have a direct result on the validation process since the geometric representations are not one-to-one when void relations are used. Since this is of no influence on the given rules, adding geometric rules to SHACL shapes and the boolean operations is not required. Therefore, the process continues, and the validations are processed.

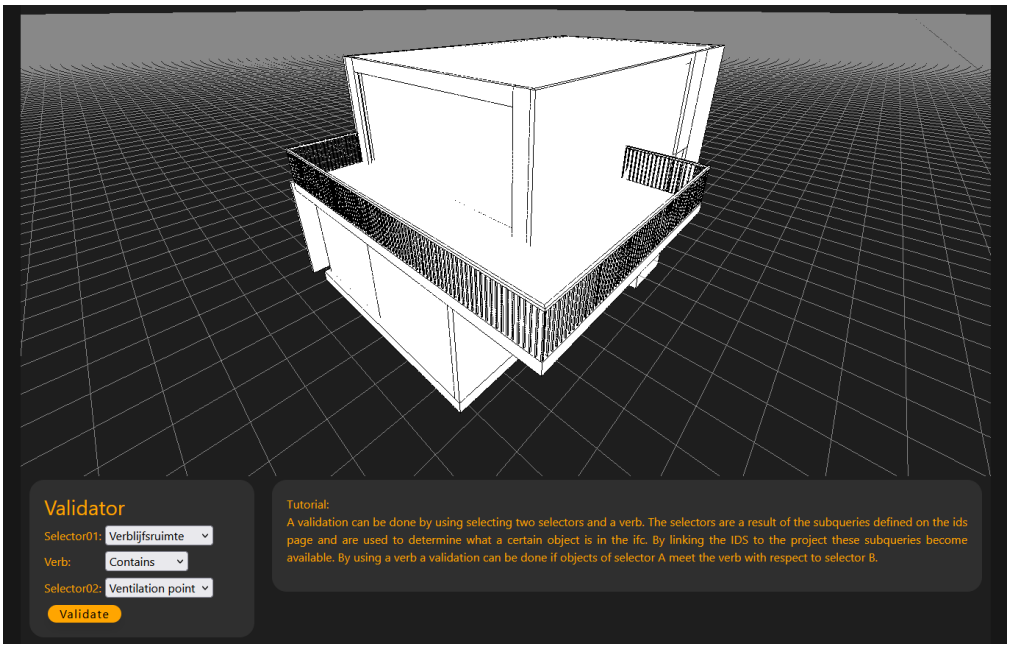

Figure 19: Model view as imported

After the IFC file was parsed into the class library, all classes inheriting IfcObject are collected to be added to the RDF graph. For the IFC file, 103 objects were added to the RDF graph. Within these objects are the 10 spaces and 7 flow terminals needed for the validation of the three rules. All these IfcShapeRepresentations are exported as triples needed for the extraction of the geometric objects. With the manual validation completed, it is accepted that the model can parse all the objects needed from the IFC file to the RDF graph with a parsing time of 36 seconds. With this RDF graph filled with the needed triples, the next step is to validate the three rules.

4.3.1 Research validation

For the validation of the result, the rules defined in Section 3.1.4 are checked. For the first rule ”Within a mechanically ventilated dwelling, each living area must include a ventilation point” the validator ”Verblijfsruimte” ”NotContains” and ”Ventilation point” can be requested. For this rule, the expected result is the return of the bedroom with number 115. For the second rule, ”Each residential function must have a designated space for a cooking appliance”, the validator ”Residential area”, ”Notcontains”, and ”cooking appliance” can be requested. For this rule, the expected result is that no objects will be returned. For the third and final rule, ”An open combustion appliance is not placed in a bathroom”, the validator ”bathroom” ”Contains” and ”cooking appliance” can be requested. For this rule, the expected result is that the bathroom is returned.

Rule 1: Within a mechanically ventilated house, each living area must include a ventilation point (Art. 4.122)

The first validation request checks if each living area includes a mechanical ventilation point. For this, the SHACL shape must understand how to collect both the living areas and ventilation points. As mentioned above, the living areas can be found by filtering on the IfcSpaces where the boolean properties ”isverblijfsgebied” has to be true. To collect these objects from the RDF graph, the SHACL shape uses the SPARQL query shown in Listing 18. Running this query on the RDF graph results in the collection of the four expected living areas.

```
PREFIX ifc:
   <https://identifier.buildingsmart.org/uri/buildingsmart/ifc/4.3/class/>
SELECT DISTINCT ?subject ?propertyName ?propertyValue WHERE {{
    ?subject ifc:IsDefinedBy ?definedBy .
    ?definedBy ifc:PropertySetDefinition ?definition .
    ?definition ifc:PropertySingleValue ?singleValue .
    ?singleValue ifc:HasName ?propertyName .
    ?singleValue ifc:NominalValue ?nominalValue .
    ?nominalValue ifc:HasValue ?propertyValue .

    FILTER(CONTAINS(LCASE(STR(?propertyName)), "isVerblijfsruimte"))
    FILTER(CONTAINS(LCASE(STR(?propertyValue)), "true"))
```

Listing 18: Living area SPARQL query

For gathering of mechanical ventilation points, the same SPARQL query is used but with different property and name in the filter. This query is shown in Listing 19 and shows that this time, the objects are filtered on the objects where the name contains ”mechanical ventilation point”. Running this query on the RDF graph results in the collection of the four expected mechanical ventilation points.

```
PREFIX ifc:
   <https://identifier.buildingsmart.org/uri/buildingsmart/ifc/4.3/class/>
SELECT DISTINCT ?subject ?propertyName ?propertyValue WHERE {{
    ?subject ifc:IsDefinedBy ?definedBy .
    ?definedBy ifc:PropertySetDefinition ?definition .
    ?definition ifc:PropertySingleValue ?singleValue .
    ?singleValue ifc:HasName ?propertyName .
    ?singleValue ifc:NominalValue ?nominalValue .
    ?nominalValue ifc:HasValue ?propertyValue .

    FILTER(CONTAINS(LCASE(STR(?propertyName)), "name"))
    FILTER(CONTAINS(LCASE(STR(?propertyValue)), "mechanical
       ventilation point"))
```

Listing 19: Mechanical ventilation point SPARQL query

With these parts of the result validated, the final stage of this validation request is reached. As mentioned in the methodology, the direct implementation in the SHACL engine is out of reach for this thesis. Using these two queries and the validation verb allows the web API to mimic the functionality of the SHACL shape and validate the model using semantic web technologies. The mimicking SHACL shape loops through the living areas and thereafter through the mechanical ventilation points. For each of the ventilation points, it is validated if this point is in the living area. Following the first check performed using the contain validator, the living area and mechanical ventilation point are used as parameters for the contain validator function. This function first collects the position of the mechanical ventilation point using the IfcLocalPlacement. For the living area, the entire mesh needs to be collected. This is done using the IfcProductDefinitionExtension which collects the mesh using the IfcExtrudedAreaSolidExtension. The first validation done results in the mechanical ventilation point not being in the IfcSpace which is the expected result. Requesting this validation by the user returns the results shown in Figure 20 where the validation took 2.45 seconds.

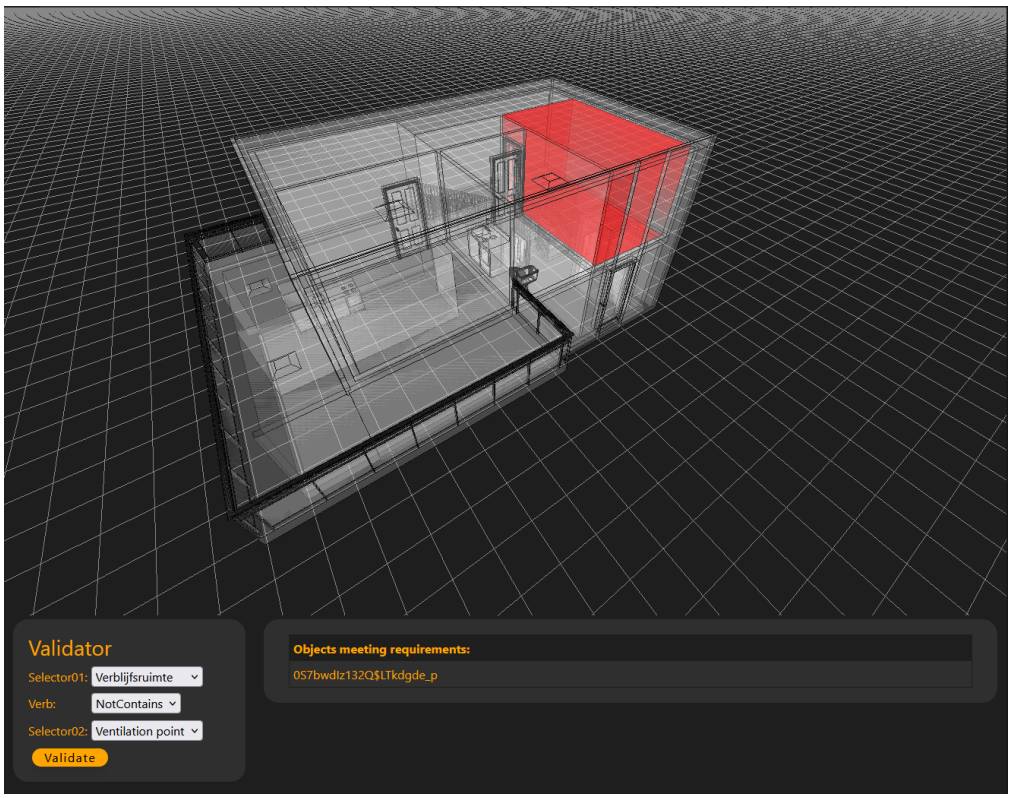

Figure 20: Model validation result of missing ventilation point

Rule 2: Each residential function must have a designated space for a cooking appliance (Art. 4.176)

The second rule was chosen mainly to prevent false negatives from passing the test. The residential function has space for the cooking appliance and should therefore not be flagged as invalid. By selecting both the residential area and cooking appliances in the same query types as the first rule. These queries collected the expected residential area and two cooking appliances. The same as in the first rule, the collection of the mesh of the residential area collects the mesh using the IfcExtrudedAreaSolidExtensions. By collecting the position of the cooking appliances using the IfcLocalPlacement, the same validation can be used as the first rule. Running this validation result returns the results shown in Figure 21 where the validation took 2.0 seconds.

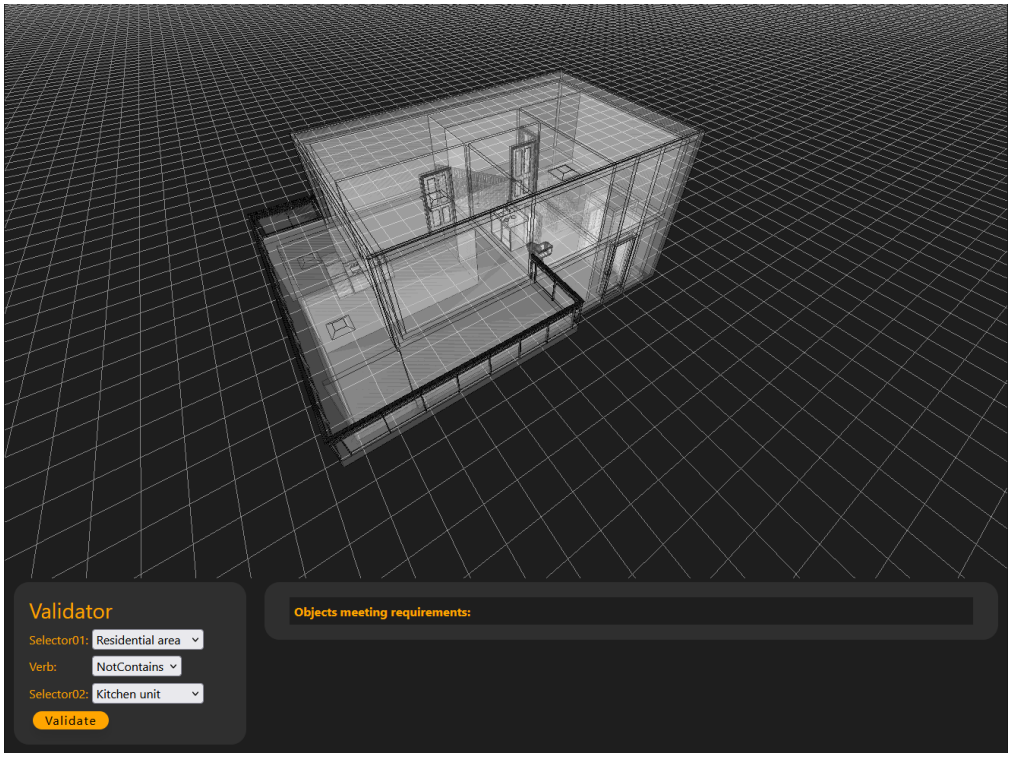

Figure 21: Model validation result of missing cooking appliance

Rule 3: An open combustion appliance is not placed in a toilet or bathroom (Art. 3.75)
The final rule does not make use of the not contains functionality but has to check if a object is contained. This reverses the functionality, which also needs to be supported. It does, however, start with the same functionality if a position is within a mesh and therefore uses the same workflow as the previous two rules. By collecting the bathroom and the combustion appliances, the final validation for the three rules is requested. The results of this validation are shown in Figure 22 and done in 1.6 seconds.

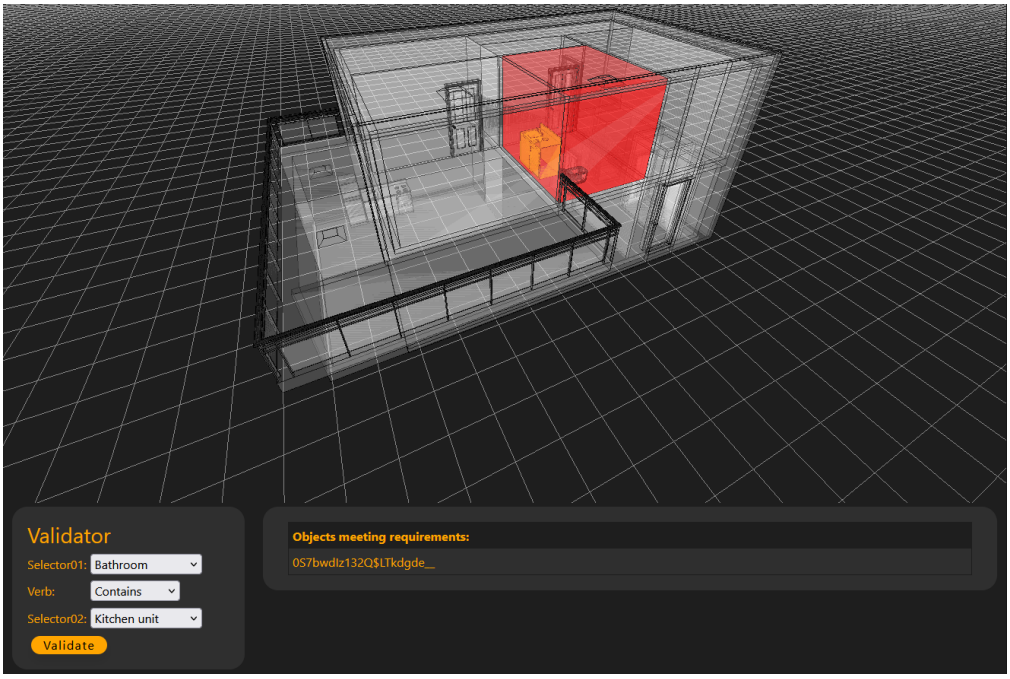

Figure 22: Model validation result of combustion appliance in the bathroom

4.3.2 Geometric computations

The validation of the results following the rules and the IFC file resulted in the validation that all were done in 3 seconds. It is deemed important that the results of a larger validations are also mapped. Especially since the validation is expected to increase exponentially in time when adding more objects to validate in a single request. To test this assumption, a model is created with 1, 10, 100 and 1000 rooms with the final model being shown in figure 23. Each of these rooms contains a single ventilation point.

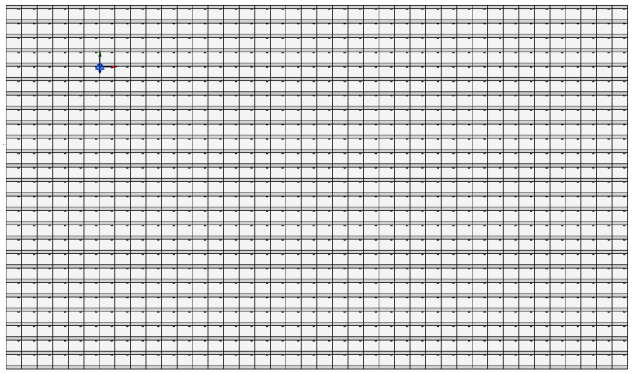

Figure 23: Ifc model with a thousand rooms

For each model, it was checked if the room contained a mechanical ventilation point. During each validation, the total run-time in milliseconds was recorded for evaluation. The results are shown in Table 14. The results show that, as expected on the basis of the literature, the run time grows exponentially based on the objects selected. Where 100 rooms with 100 ventilation points take roughly 0.15 seconds to validate, validating 1000 rooms with 1000 ventilation points takes 13.7 seconds.

Rooms	Time in milliseconds
1	1
10	2
100	151
1000	13706

Table 14: Performance evaluation of 1, 10, 100 and 1000 rooms

This page is intentionally left blank

5 Conclusion, discussion and future work

The conclusion is the final chapter of this thesis. Based on the literature study, methodology, and results, an answer is produced for the main and sub-questions. A reflection is done on the research and the contributions this thesis has made. In addition, limitations and future work are discussed. These aspects allow for further research on the subject to better add geometric validation to the framework.

5.1 Conclusion

The literature had shown that geometric rule validation was not possible using current semantic web technologies. The only workaround to this solution was to add geometric properties as literal values during the building model preparation phase. Since geometric rule validation is needed for automated compliance checking and to make better use of linked building data, an answer was needed to the research question which was: *'How can geometrical computations be added to the current state of automated compliance checking using linked building data to validate a building model of the IFC format?'* To tackle this research question, four subquestions had to be answered starting with the question: *What is the current state of linked building data technologies?*. The research on this question elaborated on what semantic web technologies are, how they are used, and most importantly why they do not support geometric rule validation. The answer to this question was about the functionality of a SHACL shape. This shape was only capable of validating the existence and value of literals (and URIs). To add geometric rule validation, the capabilities of SHACL engines have to be extended with geometric capabilities. For the second research question, an answer was needed to the question *'How can a model data be translated into computer-readable code without loss of data?'* Following this research, it was found that multiple open source software packages exist that export IFC data to computer-readable classes and that packages exist for the parsing of these classes to RDF triples and graphs. However, for the purpose of this thesis and the flexibility needed to make the SHACL shape understand geometry, the decision was made to create custom class library. This library combines the functionality of IFC to classes functionality and classes to RDF functionality. For the validation of these RDF graphs, it was needed to be known what type of rules where going to be validated. To get this answer the following question was asked: *'What geometric validations need to be applied?'*. Research into this question showed that this question currently has no real answer. Where older studies made a claim to have defined all functions needed, more recent studies have shown that the AEC sector is too diverse, new validation types are needed constantly and these validations can have different meanings for different professions. For this thesis, the decision was made to develop a framework where validation types could be added when they were properly defined. Because of this decision the framework could be proven by implementing simple validations such as 'Inside' and 'Underneath'. With an answer to the first three subquestions, the final sub-question needed an answer. This question was: *'How can the resulting framework be implemented to satisfy the user's needs?'* With a better understanding of the problem, data translation and functionality needed, information could be gathered on how the validation could be implemented to add value to the user. Simply stating that functionality could be added to the shape would not suffice. For this question, research into the four stages of automated compliance checking was initiated. Validating that the semantic web

technologies and the new SHACL shape were capable of following this flow gave many insights to the final proof-of-concept web application.

The web application developed as the final deliverable of the proof-of-concept has proven that the SHACL shape can be extended with geometric capabilities. The application allows the user to upload an IFC file and send a validation request based on the properties of the object and a verb to validate. If the geometric representations and how to extract them are known to the class library, it returns the validation result. This validation is done using the SHACL shape method of collecting the two object lists using a SPARQL query and validate the shape using the verb defined by the user. Where previous SHACL implementations did not have the ability to understand geometric requests, the proof-of-concept used the ifcOwl ontology to extend SHACL with the capability to validate geometry. Knowing the literal values and classes of the RDF-represented IFC classes and its respective geometry, the SHACL shape understands how geometry can be collected. This geometry can be used in the geometric validators to add geometric functionality to semantic web technologies. These validators are purposely made using reflection so that the range of high-level verbs that can be validated can be extended. Just as the inside validator was added as shown in the methodology, every verb can be added. Upon adding the verb with the respective functionality for different geometric shapes, the SHACL shape understands the verb and can validate the objects.

5.2 Discussion and future work

Although the results of the research are promising and allow for run-time geometric rule validation using semantic web technologies. There are some limitations that need to be addressed. During the exploration phase of this thesis, the decision was made to opt out of adding geometric validations at the model preparation phase, but to focus on the SHACL shape for runtime validation. It was believed that the runtime validation would have advantages over the other method. This has proven to be true for the size of the RDF graph is not increased based on the initial import and the chance that some geometric relations were not added. However, having to run the geometric extractor every validation request has proven to be expensive with regards to computational power and therefore validation time. The results show that the validation time increases exponentially for each added object in the validation request. For run-time validation, the number of objects to be checked cannot be reduced since each object needs to be checked. However, there is an option for additional optimization with regard to both when functions are called and the functions themselves. For instance, before doing a certain validation, a check can be performed to see if the function is worth checking. If an object contains another object, it does not have to be checked if the objects are not on the same height. The current validation functions are also created based on publicly available functions and can probably be improved.

A second limitation of the study is the degree to which the proof-of-concept is developed. For the geometric extraction, only a few geometric shapes are supported. Even though it is believed that all of these representations can be implemented, these are no guarantees that it is possible and that it will not have a negative effect on the run-time of a validation. The same argument holds for the validation methods implemented. As mentioned in the results, the definition of below and underneath are relatively easy to define. However, the verb adjacent can have a different meaning depending on who is asked. It is therefore not guaranteed that the implemented functionality

is the way it should be defined, and further study is needed to make these verbs semantically accepted.

Finally, the implementation of the proof-of-concept only mimics the workings of a SHACL shape for geometric validation. Currently, the application receivers the validation request with the same parameters as a SHACL shape and handles it the same way. It receives the objects through SPARQL queries and extracts the geometric representations through the extension. This flow is not implemented directly into the SHACL framework and doing so might uncover issues that have not been recognized at this time. For future work, it is believed that a good next step would be to create a direct implementation that, due to time constraints, could not have been done within the limit of this thesis.

A final recommendation that is not based on the addition of geometric capabilities to rule validation is to move away from the IFC data file for the building model preparation. For the purpose of this thesis, the decision was made to use the IFC data structure, since this structure is well defined and could provide the basis for semantic data classes. The downside still remains that the file size is large, as all the data is shared and saved in a single file. Every created IFC file also results in a new project being created since there rarely is a link between old and new files. Shifting towards a solution where the data model is saved directly in a RDF graph is expected to lead to a better workflow.

6 References

(n.d.). https://www.w3.org/TR/shacl/

ali waqas, w., Yao, B., Saleem, M., Hogan, A., & Ngomo, A.-C. N. (2021, April). Survey of rdf stores & sparql engines for querying knowledge graphs. https://doi.org/10.36227/techrxiv.14376884.v1

Amor, R., & Dimyadi, J. (2021). The promise of automated compliance checking. Developments in the Built Environment, 5, 100039. https://doi.org/10.1016/j.dibe.2020.100039

Beach, T. H., Hippolyte, J.-L., & Rezgui, Y. (2020). Towards the adoption of automated regulatory compliance checking in the built environment. Automation in Construction, 118, 103285. https://doi.org/10.1016/j.autcon.2020.103285

Beetz, J., van Leeuwen, J., & de Vries, B. (2008). Ifcowl: A case of transforming express schemas into ontologies. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 23(1), 89–101. https://doi.org/10.1017/s0890060409000122

Berners-Lee, T. (1998). The semantic web road map.

Bimcollab. (2024, August). Ids - information delivery specification. https://www.bimcollab.com/en/products/bimcollab-nexus/ids/

Bonduel, M., Vergauwen, M., Klein, R., Rasmussen, M., & Pauwels, P. (2018). A novel workflow to combine bim and linked data for existing buildings. eWork and eBusiness in Architecture, Engineering and Construction, 347–354. https://doi.org/10.1201/9780429506215-43

Borrmann, A., & Rank, E. (2009). Topological analysis of 3d building models using a spatial query language. Advanced Engineering Informatics, 23(4), 370–385. https://doi.org/10.1016/j.aei.2009.06.001

Buildingsmart. (2024, September). Introduction. https://standards.buildingsmart.org/IFC/RELEASE/IFC4_3/HTML/content/introduction.htm

Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387. https://doi.org/10.1145/362384.362685

Corry, E. J., Coakley, D., O' Donnell, J., Pauwels, P., & Keane, M. M. (2013). The role of linked data and semantic web in building operation. Proceedings of the 13th International Conference for Enhanced Building Operations,

data building group, L. (2017). BuildingSMART Final Standard: The ifcOWL ontology.

Eastman, C., Lee, J.-m., Jeong, Y.-s., & Lee, J.-k. (2009). Automatic rule-based checking of building designs. Automation in Construction, 18(8), 1011–1033. https://doi.org/10.1016/j.autcon.2009.07.002

Fuchs, S. (2021, May). Natural language processing for building code interpretation: Systematic literature review report.

Göbels, A., & Beetz, J. (2024, June). Relative location ontology: An ontological model for representing directional topological relationships between spatial entities in oriented space.

Grutters, B. (2022, July). Ifc release notes. https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/ifc-release-notes/

Guo, D., Onstein, E., & Rosa, A. D. (2021). A semantic approach for automated rule compliance checking in construction industry. IEEE Access, 9, 129648–129660. https://doi.org/10.1109/access.2021.3108226

Heise, I., & Borrmann, A. (2024, June). Infraspatialot: An ontology for the representation of spatial relationships in road infrastructure.

Hert, S., Hoffmann, M., Kettner, L., Pion, S., & Seel, M. (2007). An adaptable and extensible geometry kernel. Computational Geometry, 38(1–2), 16–36. https://doi.org/10.1016/j.comgeo.2006.11.004

Jacob E.K., E. K. (2003). Ontologies and the semantic web. Bulletin of the American Society for Information Science and Technology, 29(4), 19–22. https://doi.org/10.1002/bult.283

Jeroen, W., Pauwels, P., Beetz, J., & Berlo, L. v. (2019). Towards a decentralised common data environment using linked building data and the solid ecosystem.

Katsigarakis, K. I., Giannakis, G. I., Lilis, G. N., & Rovas, D. V. (2019). An ifc data preparation workflow for building energy performance simulation. Computing in Construction, 1, 164–171. https://doi.org/10.35490/ec3.2019.188

Klyne, G., Carroll, J. J., & McBride, B. (2014, February). Rdf 1.1 concepts and abstract syntax. https://www.w3.org/TR/rdf11-concepts/

Kuznetsov, S. D., & Poskonin, A. V. (2014). Nosql data management systems. Programming and Computer Software, 40(6), 323–332. https://doi.org/10.1134/s0361768814060152

Lee, J.-K., Cho, K., Choi, H., Choi, S., Kim, S., & Cha, S. H. (2023). High-level implementable methods for automated building code compliance checking. Developments in the Built Environment, 15, 100174. https://doi.org/10.1016/j.dibe.2023.100174

Nuyts, E., Bonduel, M., & Verstraeten, R. (2024). Comparative analysis of approaches for automated compliance checking of construction data. Advanced Engineering Informatics, 60, 102443. https://doi.org/10.1016/j.aei.2024.102443

Pauwels, P., De Meyer, R., & Van Campenhout, J. (2011). Lecture notes in computer science, 143–158. https://doi.org/10.1007/978-3-642-23017-2_10

Pauwels, P., Krijnen, T., Terkaj, W., & Beetz, J. (2017). Enhancing the ifcowl ontology with an alternative representation for geometric data. Automation in Construction, 80, 77–94. https://doi.org/10.1016/j.autcon.2017.03.001

Pauwels, P., & McGlinn, K. (2023). Buildings and semantics data models and web technologies for the built environment. CRC Press.

Pauwels, P., Zhang, S., & Lee, Y.-C. (2017). Semantic web technologies in aec industry: A literature overview. Automation in Construction, 73, 145–165. https://doi.org/10.1016/j.autcon.2016.10.003

Tomczak, A., Berlo, L. v., Krijnen, T., Borrmann, A., & Bolpagni, M. (2022). A review of methods to specify information requirements in digital construction projects.

IOP Conference Series: Earth and Environmental Science, 1101(9), 092024. https://doi.org/10.1088/1755-1315/1101/9/092024

van den Bersselaar, E., Pauwels, P., & Verhelst, L. (2024). Validation of technical requirements for a bim model using semantic web technologies. Advanced Engineering Informatics, 60, 102426. https://doi.org/10.1016/j.aei.2024.102426

Zhang, C., Beetz, J., & de Vries, B. (2018). Bimsparql: Domain-specific functional sparql extensions for querying rdf building data. Semantic Web, 9(6), 829–855. https://doi.org/10.3233/sw-180297

Zhang, J., & El-Gohary, N. M. (2017). Integrating semantic nlp and logic reasoning into a unified system for fully-automated code checking. Automation in Construction, 73, 45–57. https://doi.org/10.1016/j.autcon.2016.08.027

Zhang, R., & El-Gohary, N. (2019). A machine-learning approach for semantic matching of building codes and building information models (bims) for supporting automated code checking. Sustainable Civil Infrastructures, 64–73. https://doi.org/10.1007/978-3-030-34216-6_5

Zhang, Z., Ma, L., & Broyd, T. (2022). Towards fully-automated code compliance checking of building regulations: Challenges for rule interpretation and representation. Proceedings of the 2022 European Conference on Computing in Construction. https://doi.org/10.35490/ec3.2022.148

Zheng, Z., Zhou, Y.-C., Lu, X.-Z., & Lin, J.-R. (2022). Knowledge-informed semantic alignment and rule interpretation for automated compliance checking. Automation in Construction, 142, 104524. https://doi.org/10.1016/j.autcon.2022.104524

Zou, Y.-C., Zheng, Z., Lin, J.-R., & Lu, X. (2020). Integrating nlp and context-free grammar for complex rule interpretation towards automated compliance checking. https://doi.org/10.13140/RG.2.2.22993.45921