



Common Data Environments to facilitate information management in HVAC engineering

Seidenschnur, Mikki

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

Citation (APA):
Seidenschnur, M. (2023). *Common Data Environments to facilitate information management in HVAC engineering*. Technical University of Denmark. DCAMM Special Report No. S328

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Common Data Environments to facilitate information management in HVAC engineering

Mikki Seidenschnur

PhD Thesis

Principal supervisor: Associate Professor Christian Anker Hviid, Ph.D.

Co-supervisor: Associate Professor Kevin Michael Smith, Ph.D.

Principal company supervisor: Senior Chief Specialist Christoffer Borgwardt-Stampe, M.Sc.

Company co-supervisor: Senior Chief Specialist Frederik Blum Winther, Ph.D.

DTU Civil & Mechanical Engineering

Department of Civil & Mechanical Engineering

Technical University of Denmark

Nils Koppels Allé

Building 404

2800 Kongens Lyngby, Denmark

Preface

This thesis is submitted to the Department of Civil and Mechanical Engineering at Technical University of Denmark (DTU) in partial fulfillment of the requirements for the degree of Doctor of Philosophy. The study was conducted as an industrial Ph.D.¹ project between March 2020 and March 2023. The principal supervisor was Associate Professor Christian Anker Hviid, DTU Construct; the co-supervisor was Associate Professor Kevin Michael Smith, DTU Construct. The company principal company supervisor was Senior Chief Specialist Christoffer Borgwardt-Stampe, Ramboll; the company co-supervisor was Senior Chief Specialist Frederik Blum Winther, Ramboll. The dissertation is paper-based and consists of the present thesis and the papers created during the Ph.D. thesis (Chapter 6).

Kongens Lyngby, 20th February 2023

A handwritten signature in black ink, appearing to read 'Mikki Seidenschnur', with a stylized flourish extending to the right.

Mikki Seidenschnur

¹An industrial Ph.D. project in Denmark focuses on research in industry-related topics. The organization consists of a hosting company (Rambøll), The Ph.D. student, and the university (DTU)

Acknowledgements

I extend my gratitude to the Ramboll Foundation, Ramboll A/S, and Innovationsfonden Denmark for providing the funding necessary to carry out this industrial Philosophiae Doctor (PhD). Specifically, I would like to thank Ramboll A/S for showing a continuous interest in this research project and working hard to implement it within the organization.

I thank my principal supervisor at DTU, Associate professor Christian Anker Hviid, for helping me initiate the application process for the PhD project. Without your help, I would not have initiated the PhD project to begin with. I also thank you for meeting with me regularly throughout the project to discuss research topics and provide feedback on research output. I want to extend my gratitude to my co-supervisor at DTU, Associate professor Kevin Michael Smith, for providing me with extensive feedback on the project and research output. A special thanks to the mental support you provided in the late stages of writing the thesis. It has been a tremendous effort, and I would not have been able to carry it out without your support.

A special thanks to my principal supervisor in Ramboll, Senior Chief Specialist Christoffer Borgwardt-Stampe, for his continued interest in the project and for helping me to implement the research findings into the organization of Ramboll. Your tenacity and competencies have helped me immensely to generate an impact within the Ramboll organization. I look forward to our continued collaboration. I thank my co-supervisor, Senior Chief Specialist Frederik Blum Winther, for the insights provided to the simulation world and for helping me set the boundaries of the PhD project. Thank you for letting me draw on your experience as a PhD. It helped me get through some tricky writing spells. Finally, I would like to extend my gratitude to Country Market Director Ronni Holm Dam for helping us realize the project and for helping me in succeeding within the Ramboll organization. Your experience with software development in the Architecture, Engineering, Construction, and Operation (AECO) industry has been paramount to scoping the PhD

project to increase organizational impact.

Thanks to Pieter Pauwels for hosting me during my four months external research stay at the Eindhoven University of Technology (TUE). I contacted Pieter because he is a well-known driving force for the digital transformation of the AECO industry. He provided me with valuable feedback on my research articles and contributed directly to those articles. Your feedback has been a massive help in creating quality publications. I also thank my research colleagues at TUE, Ekaterina Petrova, Alex Donkers, Shahryar Sarabi, Julia Kaltenegger, and many more, for valuable discussions, over lunch or otherwise, and in general for making Eindhoven my home for four months.

I want to extend my thanks to my colleagues at Ramboll and DTU, with whom I have enjoyed many cups of coffee, lunches, and social activities in general. Though I have split my time between offices (Ramboll and DTU), my colleagues in both workplaces have worked hard to include me in all the activities, even during the COVID-19 lockdowns. Without this social interaction, I don't know how I would have survived for three years!

I especially want to thank my colleagues, friends, and partners in crime, Ali Küçükavcı, and Esben Visby Fjerbæk. Ali, I have shared this three-year PhD journey with you, and we have produced numerous publications together, and I owe a large part of this success to you. Had it not been for you, this journey would have been impossible. Esben, you started as a master's thesis student under the guidance of Ali and me. After exceeding our expectations, you went on to do an industrial PhD in the same area as me. I greatly appreciate your friendship and look forward to our continued collaboration.

Throughout the research project, I have been the co-supervisor of six master students' theses. I thank Esben Visby Fjerbæk, Jon Martin Tangeraas, Fredrik Seeberg, Gry Haxholm, Camilla Jakobsen, and Oskar Gram Nielsen for using the Common Data Environment (CDE) developed in this PhD project for their projects. This has greatly advanced my research and generated valuable contributions.

I want to thank my friends and family for supporting me through the PhD journey. I hope that I one day can repay the support that you have given me. A special thanks to my twin brother, Mark Seidenschneur, for his mental support, but especially for mentoring me in software development. Through your guidance, and background as a software engineer, I refocused my research to a much broader perspective of development within the AECO industry. You taught me the secrets of software development. Without your mentoring, this project would not have been the same.

Finally, my most profound appreciation goes out to my fiancée, Natasha Maria Lund Andersen. You have been the most essential part of this journey. I

fear that my constant rambling about this project might have been tedious at times - but you have always listened enthusiastically and tried to solve problems with me. At this point, I believe that you deserve a PhD-degree as well. I dedicate this work to you.

Foreword

When I started my first full-time position as a Mechanical engineer, I quickly recognized that changes in building design are inevitable. Therefore, like most of my colleagues, I started to digitalize my calculations in Excel spreadsheets - so I could dynamically recalculate the Heating, Ventilation, and Cooling (HVAC) system when there were design changes. But, after I had developed a multitude of tools based on different projects, I realized that my developments were not useful for others and could rarely be reapplied to another project. Last but not least, I also found that I had increased the amount of manual input that I needed to acquire from different stakeholders in the design project.

I developed excel spreadsheets, while others developed C# plug-ins for Revit, Python for Grasshopper, and so on. We were developing tools that only we could use and where input was provided manually. We simply had no place for storing all project-related information. This reminds me of a quote by George Westerman:

"When digital transformation is done right, it's like a caterpillar turning into a butterfly, but when done wrong, all you have is a really fast caterpillar." — George Westerman, MIT Sloan Initiative on the Digital Economy

I realized that AECO organizations spend countless hours on tool development that is rarely implemented. The tools are too narrowly scoped and aim to solve the simplest issues without addressing some of the underlying problems. One of those problems is that interoperability is poor, and often information is shared manually as documents between stakeholders using emails.

I wanted to take a step back and reassess the process of creating tools and sharing information in the AECO industry. Therefore, when I had the opportunity to research just that, I had to take it.

Summary (English)

Information management in the Architecture, Engineering, Construction, and Operation (AECO) industry is fragmented, which creates an information gap. This can be accredited to stakeholders working in a document-centric/file-based manner. Consequently, stakeholders do not have access to the most recent information on the building project, and there exist several sources of truth.

The information gap significantly affects the Heating, Ventilation, and Cooling (HVAC) discipline. There is no single source of truth, meaning that Heating, Ventilation, and Cooling (HVAC) engineers have to acquire the information manually from all the stakeholders. To minimize the burden of information acquisition, HVAC simulation models are over-simplified and rarely reflect the physical reality of the HVAC system in the building. Due to the over-simplification of HVAC simulation models, most HVAC systems are oversized. Consequently, there is a gap between the predicted and measured performance of buildings, called the performance gap.

The overall objective of this thesis was to provide the HVAC discipline in the AECO industry a place to centralize project-related information in a Common Data Environment (CDE) providing a single source of truth. The purpose of the CDE is to enable the digital transformation of the AECO industry through a platform that can be organically extended using a modular system architecture. Furthermore, the CDE aims to close the information gap for HVAC engineers and, as a result, reduce the performance gap.

This thesis presents a CDE based on a microservice system architecture. The CDE can translate proprietary Building Information Modeling (BIM) models into an object-oriented database that provides a single source of truth for the project stakeholders. The database of the CDE is based on the Flow System Classes (FSC) and Thermal Zone Classes (THERM) object models developed in this thesis. Through the CDE's microservice architecture, several microservices were developed, capable of performing whole-building simula-

tion, hydraulic simulation, dynamic hydraulic calculation, model validation, and serializing proprietary BIM models into object models created in this thesis. Most notably, the CDE contain tools capable of performing whole-building simulation and detailed HVAC calculations of systems.

The developed microservice-based CDE was used to evaluate the performance gap in different use cases. The use case of Frederiksberg school showed that the developed whole-building simulation microservice can simulate the existing BIM model. The results were used with measurements from the school to evaluate the performance gap, which illustrated a large discrepancy between the predicted and measured performance.

The CDE provides the basis for the digital transformation of the AECO industry by replacing file-based BIM with a model-centric approach. Through Flow System Classes (FSC), Thermal Zone Classes (THERM), and Flow Systems Ontology (FSO), it is possible to represent HVAC systems and thermal zones as a single source of truth in the CDE. The microservice architecture in the CDE allows for an organically scalable system architecture. This was illustrated by adding microservices capable of whole-building simulation and detailed HVAC simulation in EnergyPlus (E+) and Modelica, respectively. In conclusion, the developed CDE combined with whole-building simulation and detailed HVAC simulation has the potential to reduce the performance gap by closing the information gap between stakeholders.

Summary (Danish)

Informationshåndtering i Arkitektur, Ingeniør, Entreprise og Drift (AECO) industrien er fragmenteret, hvilket skaber et informationsgab. Det skyldes bl.a. aktører, hvis arbejdsproces er baseret på udveksling af filer. Dette betyder at aktørerne ikke har adgang til den mest opdaterede information relateret til byggeprojektet og at der eksisterer adskillige "sandheder".

Informationsgabets påvirker specielt HVAC disciplinen. Der er ikke en single source of truth, hvilket betyder at HVAC ingeniører manuelt må fremskaffe informationen fra andre aktører. For at minimere byrden om fremskaffelse af information, er HVAC modeller ofte oversimplificeret og reflekterer sjældent den fysiske realitet af HVAC systemet i bygningen. Pga. denne oversimplificering af HVAC simulationsmodellerne, er de fleste HVAC systemer overdimensioneret. Som konsekvens er der en forskel på den simulerede og målte præstation i bygninger, kaldet præstationsgab.

Det overordnede formål med denne afhandling er at give AECO industrien et CDE for at centralisere alt projektrelateret information i en single source of truth. Formålet med CDE'et er at muliggøre den digitale transformation af AECO industrien gennem en platform, som kan blive udvidet organisk vha. en modulær system arkitektur. Ydermere, forsøger CDE'et på at lukke informationsgab for HVAC ingeniører, og som konsekvens, reducere præstationsgab.

Denne afhandling præsenterer et CDE baseret på en microservice arkitektur. CDE'et er i stand til at oversætte proprietære BIM modeller til en objektorienteret database, som giver en single source of truth for alle projektets aktører. Databasen for CDE'et er baseret på FSC og THERM objektmodellerne, der er udviklet i denne afhandling. Gennem CDE'ets microservice arkitektur, blev adskillige microservices udviklet, som er i stand til at udføre "whole-building" simulering, hydrauliske simuleringer, dynamiske hydrauliske beregninger, modelvalidering, og serialisere proprietære BIM modeller til objektmodellerne skabt i denne afhandling. Vigtigst af disse udviklinger er

CDE'et, der indeholder værktøjer, der er i stand til at lave "whole-building" simuleringer og detaljerede HVAC simuleringer på HVAC systemer.

Det udviklede microservicebaseret CDE blev brugt til at evaluere præstationsgab i nogle forskellige brugssager. Bl.a. viste brugsagen fra Frederiksberg Skole at det udviklede "whole-building" simuleringsværktøj kunne simulere en eksisterende BIM model. Simuleringsresultaterne blev brugt til at sammenligne med målte data fra skolen, for at evaluere præstationsgab.

CDE'et ligger fundamentet for den digitale transformation af AECO industrien, ved at udskifte filbaseret BIM med en modelorienteret metode. Gennem FSC, THERM og FSO, var det muligt at representere HVAC systemer og termiske zoner som en single source of truth i et CDE. Microservice arkitekturen i CDE'et gør det muligt at skalere systemarkitekturen organisk. Dette blev illustreret ved additionen af microservices, der er i stand til at udføre "whole-building" simulering og detaljerede HVAC simuleringer, i henholdsvis EnergyPlus (E+) og Modelica. Afslutningsvis, så har det udviklede CDE, i kombination med "whole-building" simulering og detaljerede HVAC simuleringer potentialet til at reducere præstationsgab, ved at lukke informationsgab mellem aktører.

List of publications

This dissertation is paper-based and is based on the **journal papers** listed below. For the full paper, refer to chapter 6.

- I Seidenschnur, M., Küçükavci, A., Fjerbæk, E. V., Smith, K. M., Pauwels, P., & Hviid, C. A. (2022). A common data environment for HVAC design and engineering. *Automation in Construction*, 142, [104500]. doi:10.1016/j.autcon.2022.104500
- II Seidenschnur, M., Küçükavci, A., Fjerbæk, E. V., Smith, K. M., & Hviid, C. A. (2023). A Common Data Environment with an EnergyPlus microservice for Post-occupancy evaluation of the Energy Performance Gap. *Journal of Building Engineering*. Under Review.
- III Kukkonen, V., Küçükavci, A., Seidenschnur, M., Rasmussen, M. H., Smith, K. M., & Hviid, C. A. (2021). An ontology to support flow system descriptions from design to operation of buildings. *Automation in Construction*, 134, [104067]. doi:10.1016/j.autcon.2021.104067.
- IV Fjerbæk, E. V., Seidenschnur, M., Küçükavci, A., Smith, K. M., Hviid, C. A. Coupling Modelica and a Common Data Environment for simulation of HVAC systems. *Journal for Building Performance Simulations*. Under Review.
- V Küçükavci, A., Seidenschnur, M., Pauwels, P., Rasmussen, M. H., & Hviid, C. A. (2023). Efficient management and compliance check of HVAC information in the building design phase using semantic web technologies. *Journal of Building Engineering*. Under Review.
- VI Küçükavci, A., Seidenschnur, M., Rhiger, S. J., Negendahl, K., & Hviid, C. A. (2023). Rightsizing HVAC components using an ontology-driven Common Data Environment. *Journal for Building Engineering*. Under Review

Furthermore, this thesis is also based on the following **conference papers**:

- VII Seidenschnur, M., Küçükavci, A., Smith, K. M., & Hviid, C. A. (2022). A Web-based Common Data Environment for Continuous Commissioning of buildings. *BuildSim Nordic 2022, book of abstracts*
- VIII Fjerbæk, E. V., Seidenschnur, M., Küçükavci, A., Smith, K. M., Hviid, C. A. (2022). From BIM databases to Modelica - Automated simulations of heating systems. *CLIMA 2022 Conference*. <https://doi.org/10.34641/clima.2022.365>
- IX Küçükavci, A., Seidenschnur, M., Kristoffer, N., & Hviid, C. A. (2022). Taking advantage of semantic web ontologies and shape constraints for Heating, Cooling, and Ventilation Systems. *E3S Web of Conferences* 362, 04002 (2022), *BuildSim Nordic*. <https://doi.org/10.1051/e3sconf/202236204002>.
- X Pauen, N., Kukkonen, V., Küçükavci, A., Rasmussen, M. H., Seidenschnur, M., Schlütter, D., Hviid, C. A., & van Treeck, C. (2022). A roadmap toward a unified ontology for building service systems in the AECO industry: TSO and FSO. In P. Pauwels, M. Poveda-Villalón, & W. Terkaj (Eds.), *CEUR Workshop Proceedings (Vol. 3213, pp. 53-64)*. CEUR-WS. *CEUR Workshop Proceedings*.

Acronyms

AECO	Architecture, Engineering, Construction, and Operation. iii, iv, vii, ix–xii, xix, 1–4, 6, 7, 9–16, 18–20, 24, 26, 28–31, 34, 35, 44, 55, 57–61, 64, 66, 67, 69–71, 233
API	Application Programming Interface. 13, 17, 39, 43, 45, 59, 60, 70
BHoM	Buildings and Habitats object Model. 24, 25
BIM	Building Information Modeling. ix–xii, xix, 6, 9, 11–13, 15, 17, 18, 26, 28–31, 34, 35, 41–46, 49, 50, 54, 56, 58–62, 64–67, 69–71
BOT	Building Topology Ontology. 19, 20, 30, 31, 35, 36
BPS	Building Performance Simulation. 5, 9–12, 26, 42, 43
CAD	Computer-Aided Design. 6
CDE	Common Data Environment. iv, ix–xii, 7, 9, 13–18, 24–26, 28–34, 39, 41–52, 55, 56, 58–71, 233
DTU	Technical University of Denmark. i, iii, iv, 20
E+	EnergyPlus. x, xii, 13, 26, 49, 51–54, 56, 59, 61–63, 65–69
EPG	Energy Performance Gap. 5
FMI	Functional Mock-up Interface. 12, 56
FMU	Functional Mock-up Unit. 13, 56
FPO	Flow Properties Ontology. 37–39, 49, 54, 60, 67

FSC	Flow System Classes. ix–xii, 33, 34, 38–44, 46–50, 58–60, 62, 63, 66–69, 71, 233
FSO	Flow Systems Ontology. x, xii, 21, 33–38, 44, 49, 54, 59, 60, 66, 67, 233
GDP	Gross Domestic Product. 2
HTTP	Hypertext Transfer Protocol. 16, 46
HVAC	Heating, Ventilation, and Cooling. vii, ix–xii, 4–7, 9–13, 15, 17, 18, 20, 25, 26, 28–37, 42, 44–46, 48–51, 54–56, 58–71, 233
IBPSA	International Building Performance Simulation Association. 12
IDF	Input Data File. 51, 52, 61, 69
IESVE	Integrated Environmental Solutions Virtual Environment. 26, 61, 66
IFC	Industry Foundation Classes. 13, 15, 18–20, 25, 58–60, 71
ifcOWL	Industry Foundation Classes Web Ontology Language. 20
JSON	JavaScript Object Notation. 17, 18, 22, 30, 39–41, 43, 44, 46, 49, 51, 59–61, 63, 71
JSON-LD	JavaScript Object Notation for Linked Data. 22, 24, 38, 60
LBD	Linked Building Data. 19–21
MEP	Mechanical, Electrical, and Plumbing. 25, 70
OOP	Object Oriented Programming. 22
OWL	Web Ontology Language. 19, 20, 24, 31, 34, 38, 41, 60, 61, 67
PDF	Portable Document Format. 3
PhD	Philosophiae Doctor. iii–v, xix, 33, 58, 64, 66, 68, 71, 73
RDF	Resource Description Framework. 21, 22, 30, 38, 41, 59, 60, 67, 71
REC	Real Estate Core. 20
REST	Representational State Transfer. 16, 45
RQ	Research Question. 58

SAREF	Smart Applications REference. 19
SEAS	Smart Energy Aware Systems. 19
SPARQL	SPARQL Protocol and RDF Query Language. 21, 22, 37, 38, 60
Spawn	Spawn-of-EnergyPlus. 12, 13, 26, 56, 63, 68, 71
THERM	Thermal Zone Classes. ix–xii, 33, 42–44, 49, 51, 56, 59, 61–63, 66, 68, 69, 71, 225, 233
Turtle	Terse RDF Triple Language. 21, 30, 36–38, 60, 67
UML	Unified Modeling Language. 38, 42
URI	Unique Resource Identifiers. 22, 60
W3C	World Wide Web Consortium. 21, 22
XML	Extensible Markup Language. 30

Reader's guide

This thesis is divided into six chapters.

The thesis introduces the main problem in Chapter 1. Chapter 1 provides the basis for understanding the issue of fragmented information management in the Architecture, Engineering, Construction, and Operation (AECO) industry and why it leads to a performance gap in predicted and measured performance.

The background chapter 2 provides a comprehensive literature review of existing information management solutions and tools that allow for advanced whole-building and hydraulic simulation of BIM models. If the reader is unfamiliar with software engineering, specifically object-oriented programming, and web-based programming, the author of this thesis highly recommends reading the background chapter.

The thesis then introduces the research design chapter 3, which is essential for all readers to understand the research contributions made throughout this thesis. The research design presents six research questions based on the introduction and background. It also introduces four research tasks aimed at answering the research questions. Finally, the research design elaborates on the methods used to solve the research tasks.

The results chapter 4 provides a summary of all the research contributions made, in the form of journal papers or conference papers, to the research tasks posed in the research design.

The discussion and conclusion chapter 5 section provides the answers to the research questions asked in the research design. Furthermore, it overviews the specific contributions made to academia and industry. Finally, the future outlook of this thesis is discussed.

The papers chapter 6 provides the reader with all the research articles that have contributed to the work in this paper-based PhD thesis.

Contents

Preface	i
Acknowledgements	iii
Foreword	vii
Summary (English)	ix
Summary (Danish)	xi
List of publications	xiii
List of acronyms	xv
Reader's guide	xix
 Contents	 xxi
 1 Introduction	 1
1.1 Productivity in the AECO industry	2
1.2 The information gap	3
1.3 The performance gap	5
1.4 Digital transformation	6
1.5 Vision	7
 2 Background	 9
2.1 Building performance simulations	10
2.1.1 Idealized building performance simulation models	10
2.1.2 Hydraulic calculation/simulation	11
2.1.3 Detailed HVAC simulations	12
2.2 Web-based Common Data Environments	14
2.3 System architecture of a CDE	15

2.3.1	Monolithic architecture	15
2.3.2	Microservice architecture	16
2.4	Data models	18
2.4.1	Industry Foundation Classes	18
2.4.2	Discipline-specific vocabularies	19
2.5	Summary	26
3	Research Design	27
3.1	Problem statement	28
3.2	Research questions	28
3.3	Research tasks	29
3.4	Methodology	30
3.4.1	Research task 1	30
3.4.2	Research task 2	30
3.4.3	Research task 3	31
3.4.4	Research task 4	31
4	Results	33
4.1	Research tasks and related articles	34
4.2	Research task 1	34
4.2.1	Flow System Ontology	35
4.2.2	Flow System Classes	38
4.2.3	Key findings	41
4.3	Research task 2	42
4.3.1	Thermal Zone Classes	42
4.3.2	Key findings	43
4.4	Research task 3	44
4.4.1	System architecture	44
4.4.2	Microservice developments	45
4.4.3	Key findings	47
4.5	Research task 4	48
4.5.1	Modelica in a CDE	49
4.5.2	EnergyPlus in a CDE	51
4.5.3	Key findings	55
5	Discussion & Conclusion	57
5.1	Research questions revisited	58
5.1.1	RQ1: Replacing file-based BIM with a CDE?	58
5.1.2	RQ2: A CDE for seamless data integration?	59
5.1.3	RQ3: A data model for flow systems	60
5.1.4	RQ4: A data model for thermal zones	61
5.1.5	RQ5: A microservice CDE for advanced building simulation	62
5.1.6	RQ6: Can a CDE reduce the performance gap?	64

5.2	Contributions	66
5.2.1	A system architecture for microservice-based CDE . . .	66
5.2.2	Flow System Ontology	67
5.2.3	Flow System Classes	67
5.2.4	Thermal Zone Classes	68
5.2.5	Microservices	68
5.2.6	Future outlook	69
5.3	Concluding remarks	71
6	Papers	73
6.1	Paper I - A common data environment for HVAC design and engineering	74
6.2	Paper II - A Common Data Environment with an EnergyPlus microservice for Post-occupancy evaluation of the Energy Performance Gap	91
6.3	Paper III - An ontology to support flow system descriptions from design to operation of buildings	105
6.4	Paper IV - Coupling Modelica and a Common Data Environment for simulation of HVAC systems.	121
6.5	Paper V - Introducing a Semantic Web Ontology and Rule-Set to Support Capacity- and Size-Related Property Descriptions and Validation of Heating, Ventilation and Air Conditioning Components in The Design Phase of Buildings	138
6.6	Paper VI - Efficient management and compliance check of HVAC information in the building design phase using semantic web technologies	161
6.7	Paper VII - A Web-based Common Data Environment for Continuous Commissioning of buildings	184
6.8	Paper VIII - From BIM databases to Modelica - Automated simulations of heating systems	191
6.9	Paper IX - Taking advantage of semantic web ontologies and shape constraints for Heating, Cooling and Ventilation Systems	199
6.10	Paper X - A roadmap toward a unified ontology for building service systems in the AECO industry: TSO and FSO	207
	Appendices	221
	Appendix A UML Diagram	223
	A.1 THERM UML diagram	224
	Bibliography	227

CHAPTER 1

Introduction

This Chapter introduces the problem that led to the initiation of this research.

Section 1.1 introduces the reader to the lack of productivity increase in the AECO industry.

Section 1.2 provides the context for why the AECO industry is falling behind on productivity due to the information gap.

Section 1.3 formulates the performance gap, the consequence of the information gap, and how it may be reduced.

Section 1.4 introduces the existing development paradigms in large organizations or industries and provides the bridge to the vision of the following Section.

Finally, Section 1.5 collects all the problems described into a vision of how to bridge the information gap and reduce the performance gap.

1.1 Productivity in the AECO industry

The AECO industry is one of the largest sectors in the world, with a total spending of \$10 trillion on materials and services each year. However, the sector has been trailing in productivity for decades compared to other sectors, as shown in Figure 1.1.

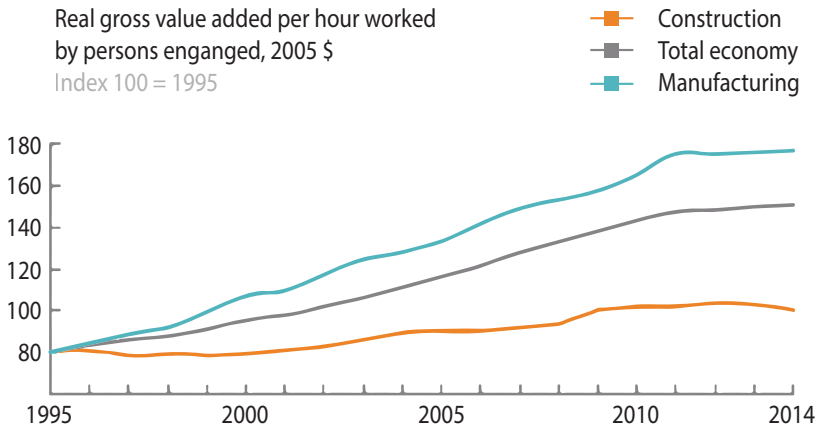


Figure 1.1. Figure adapted from the report published by McKinsey Global Institute [1] "Reinventing construction: A route to higher productivity". The Figure has been created based on 41 countries that contribute to 96% of the global Gross Domestic Product (GDP)

Figure 1.1 shows that for two decades, the growth of labor-productivity in the Architecture, Engineering, Construction, and Operation (AECO) industry has averaged around 1%, compared to the 2.8% for the world economy, and 3.6% in the manufacturing sector [1]. The report estimates seven ways to increase productivity within the AECO industry. Four of them focus mainly on information management:

1. ***Reshape regulation and raise transparency***
By making information about the design process available to all stakeholders, transparency of the project is raised.
2. ***Rethink design and engineering processes***
By implementing new technologies for information management, the design and engineering processes can be revolutionized.
3. ***Improve on-site execution***
On-site execution is often hindered by poor information availability. The same objects might appear differently on different drawings.

4. *Infuse digital technology, new materials, and advanced automation*

By infusing new digital technology, it is possible to improve information management and, as a result, improve the automation behind it.

The Architecture, Engineering, Construction, and Operation (AECO) industry has the potential to increase productivity by focusing on improving information management through digital transformation.

1.2 The information gap

Multiple stakeholders are involved in designing buildings. These stakeholders are responsible for providing information about the building's design, construction, and operation. Often one stakeholder is responsible for delivering information that another stakeholder needs to carry out a task, as shown in Figure 1.2. The Figure shows how building stakeholders work in a file-based and document-centric manner, creating a work environment where each stakeholder works independently of the other [2].

During the design phase, information is shared manually by each stakeholder on a need-to-know basis. For instance, the architect can share information with the energy and indoor climate engineer on request as a document (spreadsheet), often through an email. The current state of that document then provides the basis for energy and indoor climate engineer. But once updates are posed to that document on the architect's local drive, the energy and indoor climate engineer is not notified.

After the design phase, construction documentation is usually provided to the contractor as Portable Document Format (PDF) files. The building's construction phase is initiated with the design phase's conclusion. The construction stage typically results in the contractor re-designing the project using construction documentation in terms of drawings while consulting the architect and engineer to ensure that the original functionality is maintained. The result is that the information available in the design stage is no longer compatible with the information in the construction phase. After redesigning the building, the construction is initiated, and the phase is concluded with commissioning the building. At this point, the architect and engineers are barely involved. Once the building has been commissioned, the building life cycle transfers into the operation phase, where the architect, engineer, and contractor are not involved unless the building users are experiencing significant issues, such as non-operational building components.

Figure 1.2 shows that stakeholders work on different platforms and rely on document-centric documentation of buildings. Tools that have been useful

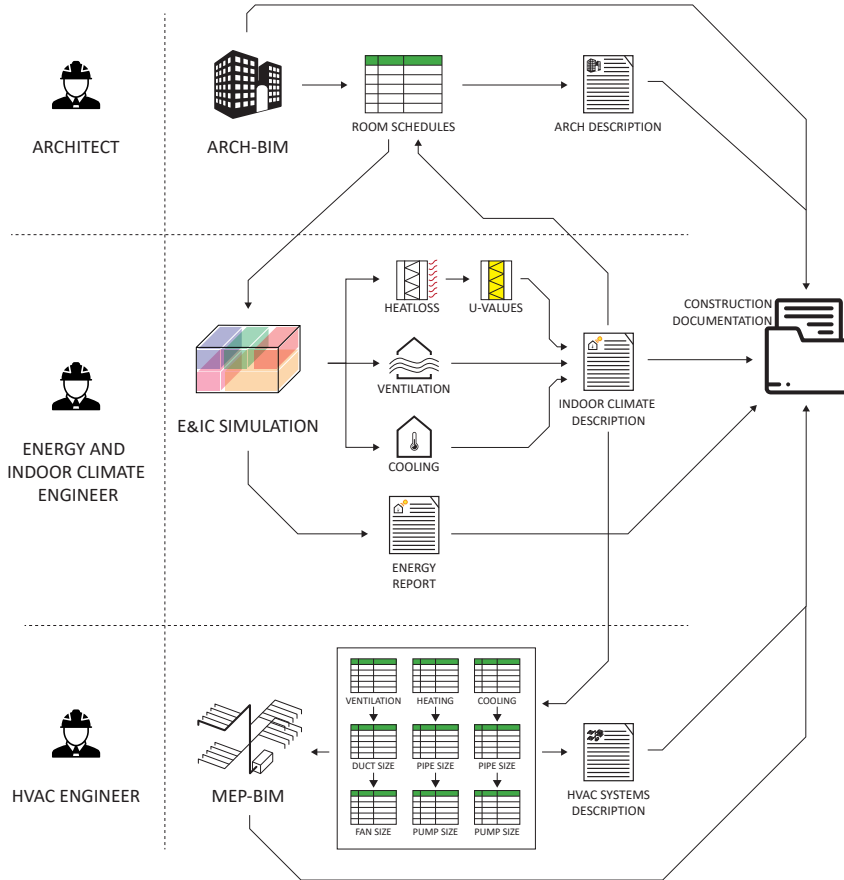


Figure 1.2. The Figure shows how designers often work in silos in a document-centric manner. The final project delivery of the design phase usually results in a construction documentation package.

in the design phase of the building, providing evidence-based design, are no longer used in the construction phase - here, changes are made based on the construction documentation developed in the design process. Changes made to improve the constructability of the Heating, Ventilation, and Cooling (HVAC) system do not result in a recalculation and resizing of the HVAC system. This means the building is not constructed according to the original design intention, leading to an incompatible HVAC system.

Information management between stakeholders in the AECO industry is fragmented and lacks a common strategy/technology, which creates an information gap [3]. This is caused by stakeholders working in a document-centric

manner, meaning they cannot access the most recent information that provides boundary conditions for their specific discipline. This information gap is caused by inadequate information management and poor tool interoperability. The gap in information results in a difference between the predicted and measured performance of buildings called the performance gap [4].

1.3 The performance gap

With the focus on increasing building performance, it has become clear that there is a gap in performance from the predicted building performance to the measured performance. The term Energy Performance Gap (EPG) provided by [4] is expanded to the performance gap in this thesis since it is more than just a discrepancy in the energy performance - it can be the performance of indoor air quality, temperature, noise, etc. Other researchers [5, 6] have identified several reasons for the performance gap from predicted to measured performance:

- **Predicted performance**

- *Design assumptions:* There are wrongful assumptions about the design. This could be occupant behavior, system behavior, or weather predictions, i.e., Often simulation models are simplified to make one room representative of the entire building. This means that the other rooms are usually under- or oversized [7].
- *Modelling tools:* The Building Performance Simulation (BPS) tools can have wrongful and error-prone models or over-simplified equations. One example is that buildings are often modeled as idealized systems, which assumes that the HVAC system is always capable of providing enough heating, cooling, and air to any given room in the building [7].

- **Measured performance**

- *Management and controls:* Poor operation of the building results in excessive energy waste and poor indoor climate performance [8, 9]. Making the building performance transparent for facility management allows for the intervention of the faulty system.
- *Occupant behavior:* It is extremely hard to predict occupant behavior accurately. Occupants greatly influence the building operation. Rooms are used differently than designed; occupants open windows, block air inlets, etc. [10].

For the predicted performance, many design assumptions are based on an estimate by the HVAC engineer performing the simulation. An example is

the information about the occupants using the room. The building owner has information about the building's usage. Still, since the information is not readily available, the engineer cannot use it for their simulation, resulting in the engineer making an (often wrongful) assumption. Moreover, engineers use over-simplified simulation models to simulate HVAC systems, mainly due to the lack of information about the system, like HVAC system components and controls.

For the measured performance, the building systems often operate poorly without the building owner knowing it. This can be addressed by providing the predicted performance and comparing it to the measured performance of the building owner. This will spark the owner to intervene and get the building operating correctly.

Most of the identified causes for the performance gap can be explained by the information gap between stakeholders in the building design, construction, and operation phases. Specifically, the AECO industry can be digitally transformed, which improves information management and interoperability between different tools of the AECO stakeholders and potentially reduces the performance gap.

1.4 Digital transformation

For many years, the AECO industry has focused on improving information interoperability through digitization/digitalization. The industry is in the early stages of switching from digitalization to digital transformation to improve information management. According to [11] the differences are as follows:

- **Digitization** is the process of making an otherwise analog task digital. An example is making hand drawings of buildings digitally using Computer-Aided Design (CAD) software. Another example is to make hand calculations of an HVAC system available in a spreadsheet.
- **Digitalization** is closely related to digitization but is a more ambiguous term used differently in this industry. In our definition, digitalization increases the automation of existing digital technologies. For instance, in the building industry, companies have automated processes for calculating simple key indicators from a building by extracting information from a BIM model. One example is automated quantity take-off from the BIM model.
- **Digital Transformation** is considered an effort to transform the way organizations work in the AECO industry and implement digital technologies. It will often include several digitalization projects and make

core cross-sectional changes to how organizations in the AECO industry work.

The AECO industry has been digitalizing for many years, but productivity is still trailing behind other sectors. It demonstrates that the AECO industry has hit a barrier to taking the next step towards increased productivity and closing the performance gap through improved information management. AECO organizations must start strategizing toward a digital transformation to close the information gap between the different stakeholders of the AECO industry.

1.5 Vision

There is a performance gap between predicted and measured building performance. Most simulations are carried out based on information provided in a document-centric manner, meaning there is no single source of truth. This creates an information gap between stakeholders, causing the performance gap.

A single source of truth model-centric approach can close the information gap by improving information management and interoperability by providing an environment where all the relevant project information resides, called a Common Data Environment (CDE). The vision of this Ph.D. thesis is to provide AECO stakeholders with a CDE that can be used for simulations in the HVAC discipline to reduce the information gap and, as a result, the performance gap. A CDE provides a platform for all future automation within the AECO industry. The CDE has the potential to digitally transform the AECO industry and enable the use of simulation tools without wrongful assumptions and oversimplified simulation models.

CHAPTER 2

Background

This chapter aims to provide a literature review of the state-of-the-art simulation tools for whole-building and HVAC simulation in a CDE. The chapter uses the literature review to frame the key challenges of the thesis. The chapter contains four sections.

Section 2.1 reviews recent efforts within Building Performance Simulation (BPS) tools and the state-of-the-art interoperability developments from BIM to BPS.

Section 2.2 reviews existing CDEs used in the AECO industry.

Section 2.3 reviews existing system architectures and their capabilities of providing a Common Data Environment (CDE).

Section 2.4 reviews existing data models and their serialization methods.

Section 2.5 summarizes the key findings of the literature review into the context of this thesis and provides the basis to carry out the research design in the following chapter.

2.1 Building performance simulations

Building Performance Simulation (BPS) tools are used to evaluate the expected performance of a building under various external and internal conditions. BPS tools can predict how a building responds to factors such as weather and occupancy level. A key component in providing an accurate BPS model is a detailed building model, including the physical characteristics, like constructions, materials, and HVAC systems. More advanced simulation engines have been employed in the AECO industry, meaning that the requirements for information have increased to improve the simulation performance. Therefore, most simulation engines allow simplified models to reduce the need for input information and resources spent on simulations, called idealized models.

2.1.1 Idealized building performance simulation models

Idealized building performance simulations rely on a simplified model that requires less input information [12, 13] than a more detailed and complex model. One example is idealized simulation models for heating, cooling, and ventilation loads. Figure 2.1 illustrates an idealized HVAC system which supplies heating, air, and cooling to the rooms. The purple lines mark the idealized model.

Figure 2.1 shows a case where only the thermal zones are included in the BPS model. Therefore, it requires less input information to simulate the BPS model. To provide a more realistic estimation of the building performance, the complex nature of the HVAC systems should be included in the simulation since their interaction is essential to delivering heating, cooling, and ventilation to the thermal zones.

Idealized simulation models make it easier to provide the input to run BPS model with limited information. They also decrease the computational cost of the simulation. One of the main issues with assuming idealized conditions is that the physical reality is often more complicated than assuming that there always is enough supply of room heating, ventilation, and cooling from the HVAC system. The HVAC system is made of a complicated network of ducts, pipes, fittings, valves, pumps, etc., that control the hydraulic flow of the system. Each simulation step throughout the year provides different boundary conditions in the thermal zones [13] for sizing the HVAC system.

With the increasing amount of cloud-based whole-building simulation tools available today [5, 14–16], the attraction to using idealized models has faded. This is due to the computational cost being less of an issue. It is, however, still relevant since cloud services charge the user for the amount of computation

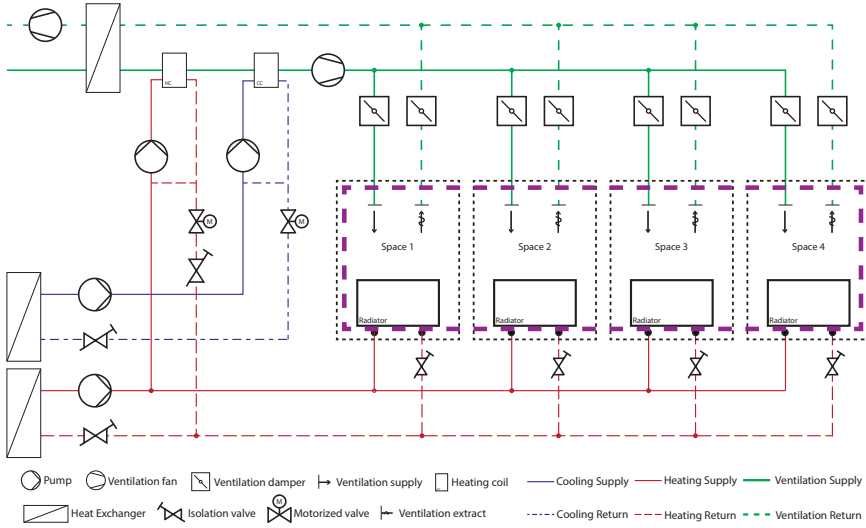


Figure 2.1. Illustration of an idealized BPS model with the ducting and piping network behind it. Purple marks the idealized system. Everything outside the purple marking is the HVAC system.

used.

Most of the AECO industry has implemented BIM [17]. Consequently, information about the HVAC system is already available to the HVAC engineers. Therefore, this information should be used to populate whole-building simulation tools with realistic information instead of calculating HVAC systems under idealized conditions. The use of idealized HVAC system models in whole-building simulation results from the information gap between stakeholders caused by poor information management and data interoperability.

2.1.2 Hydraulic calculation/simulation

As a result of idealized HVAC simulation models, hydraulic simulations in the AECO industry are usually limited to static models with worst-case sizing. The HVAC engineer assumes a "worst-case" scenario based on the local regulation in that country. An example is for sizing a heating system in Denmark. The HVAC engineer calculates the heat loss of the building in the event of -12°C , which is the design condition suggested in Danish regulation [18]. This can lead to under- or oversized HVAC systems [19, 20].

In under- and oversized HVAC systems, it can be difficult for the building management system to control the HVAC system in the building. This is

because the control components (valves, dampers, pumps, fans, etc.) of the HVAC system were selected for a peak load in the heating season, meaning that they have impaired control authority in the majority of the heating season [21, 22]. Another example is if pumps are selected to be 100% utilized during the peak load of the building, they will rarely be used to their full capacity. Consequently, this results in a higher than-necessary investment and inefficient operation, leading to excessive energy use. Without a realistic hydraulic simulation model, it is impossible to predict the actual performance of the HVAC system before commissioning.

Major software vendors have developed tools to allow HVAC engineers to perform simple hydraulic calculations on BIM models in proprietary data, like MagiCad¹ for Revit. These tools provide simple hydraulic calculations of the HVAC system, but they do not incorporate the input from BPS tools. As a result, the calculations are performed in a static case rather than utilizing the dynamic results from the whole-building simulation. Combining whole-building simulation with hydraulic simulation tools is necessary to reduce the performance gap from predicted to measured energy performance.

2.1.3 Detailed HVAC simulations

In the AECO industry, there is an increasing demand for what this thesis defines as detailed HVAC simulation. Detailed HVAC simulation is a simulation model that emulates the actual building as realistically as possible. Therefore, a detailed HVAC simulation is part of a whole-building simulation that includes the entire HVAC system and control sequences to imitate and test control sequence scenarios.

In response to the increasing demand for detailed HVAC simulation, the use of Modelica as a BPS tool has gained momentum within academia [23–28]. Modelica is an object-oriented language based on different libraries. In recent years the *International Building Performance Simulation Association (IBPSA) project 1* [29] has been leading the effort to provide comprehensive libraries for making advanced BPS models. Currently, this involves four libraries; *Buildings* [30], *AixLib* [31], *BuildingSystems* [32], and *IDEAS* [33]. The development of these Modelica-based libraries has made it possible to model many aspects of HVAC systems, including control sequences.

Spawn-of-EnergyPlus (Spawn) was developed as part of the *Buildings* library². Spawn supports coupled simulations with the use of the Functional Mock-up Interface (FMI) standard that Modelica is built on. Spawn takes the original

¹<https://www.magicad.com/>

²<https://www.energy.gov/eere/buildings/articles/its-alive-after-five-years-lab-spawn-energyplus-finally-here>

simulation interface of E+ and packages it into a Functional Mock-up Unit (FMU) so that weather, lighting, envelope, and loads models are packaged together [34,35]. This enables Spawn to run detailed HVAC simulation models, a state-of-the-art endeavor.

HVAC system simulations in a Modelica simulation environment require very detailed information on the HVAC system of a building. This includes geometry, components, the specific use of thermal zones, etc. Therefore, the information gap persists. These aspects can be modeled directly in a Modelica simulation environment but often lead to assumptions and over-simplification of the system. Therefore, research projects have developed tools for automatic translation of BIM models into Modelica object models to be simulated in a Modelica simulation environment [25,36,37].

IFC2Modelica, by Andriamamonjy et al. [25], created a parser to create a Modelica object model based on the Industry Foundation Classes (IFC) format, using the *IDEAS* library. The parser uses model view definitions to check that the IFC model contains sufficient information to create a Modelica model. The parser uses *IfcOpenShell*³ to generate the Modelica object model. The use-case in the paper focuses on HVAC systems modeling, even though some simple thermal zones are included.

BIM2Modelica [36] also uses *IfcOpenShell* to parse IFC files to Modelica object models, using the *BuildingSystems* library. They focus on creating multizone models, computational fluid dynamics, and district heating models. *BIM2Modelica* does not parse HVAC systems.

Revit2Modelica [37] uses a BIM model from Revit to generate the Modelica object model based on the *Buildings* library. This is done through a Revit plugin created with Revit's C# Application Programming Interface (API). The tool creates a Modelica object model for whole-building simulation, excluding HVAC systems.

The creation of tools to generate Modelica object models has contributed positively to the interoperability for detailed HVAC simulations. They provide a means to translate files but no means of storing them as a single source of truth. This means that while it solves the issues related to the over-simplification of simulation models, it does not bridge the information gap caused by working in a file-based environment. To provide a better basis for performing detailed HVAC simulations in the AECO industry, there needs to be a web-based Common Data Environment (CDE) that gathers all information into a single source of truth.

³<https://ifcopenshell.org/>

2.2 Web-based Common Data Environments

A web-based CDE connects all the stakeholders of a building in a common central repository that is used for the design, construction, and operation of a building, as shown in Figure 2.2.

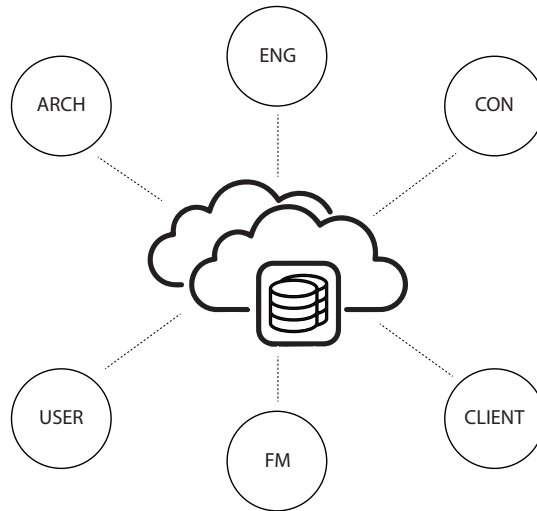


Figure 2.2. Figure showing how all project stakeholders have access to the same project information throughout the building life-cycle.

Since the word CDE has become more prevalent in recent years, there is an ISO standard called 19650, with a definition in 3.3.15, defining that a CDE is an: "agreed source of information (3.3.1) for any given project or asset (3.2.8), for collecting, managing and disseminating each information container (3.3.12) through a managed process." This definition is somewhat ambiguous in meaning. Therefore, this thesis elaborates further: A CDE provides access to a web-based central repository of the building model or asset. The CDE acts as a single source of truth to all project stakeholders. For instance, this means that information from the architectural model can be used for energy and indoor climate simulation. Another example is structural calculations available to the contractor during construction. A CDE eliminates the need for a file-based exchange of information and, as a result, provides transparency to the building design project.

Several research projects have created CDEs for various purposes based on different core languages. For instance, Beetz et al. (2010) [38] created the BIMServer.org to provide an open-sourced web-based platform to allow all stakeholders of an AECO project to work on the same central repository. Although they do not define BIMServer.org as a CDE, the platform's intention

aligns with the definition of a CDE. The BIMServer.org project uses IFC files as a foundation for data storage, which can be problematic; this is discussed further in Section 2.4.1.

Another effort to provide a commercial CDE was carried out by Autodesk, called Autodesk Drive⁴ (earlier called A360). According to Autodesk, it provides permission control, audit trail, document control and versioning, custom metadata, integrated with design workflows, design and office file viewing, 2D and 3D viewing and compare, etc. While Autodesk Drive constitutes a CDE by many of the definitions in the ISO standard 19650, it is still a proprietary tool which is one of the most significant issues regarding interoperability - proprietary tools limit the user to software and file formats developed by the specific vendor. This limitation makes it hard for the project team using the CDE to incorporate their functionalities - for instance, if they want to automate work processes, like automated simulations in an energy and indoor climate simulation tool.

Several efforts try to provide a CDE, and they all meet the requirements differently. Most are proprietary tools without access to the underlying data; some are based on the IFC format. In a comprehensive review by Jaskula et al. [39], they concluded that not a single one of the reviewed CDEs provides a single source of truth.

The existing CDEs have the potential to improve information management in the design, construction, and operation of buildings. However, the existing CDEs for the AECO industry are limited to the IFC format or based on proprietary tools, like Autodesk Revit. As a result, this means that transfer between a CDE and BIM tools is still file-based, meaning there is no single source of truth. Furthermore, the existing CDEs have not been developed with a focus on describing HVAC systems. Consequently, they do not provide sufficient information to describe the complex nature of HVAC systems.

2.3 System architecture of a CDE

To create a web-based CDE and select an appropriate system architecture for the application, this Section reviews the different types of existing system architectures that can be used to create a CDE.

2.3.1 Monolithic architecture

A monolithic architecture is a traditional approach to building applications in the software industry. It is built as one coherent unit containing the entire application in one code base [40].

⁴<https://drive.autodesk.com/>

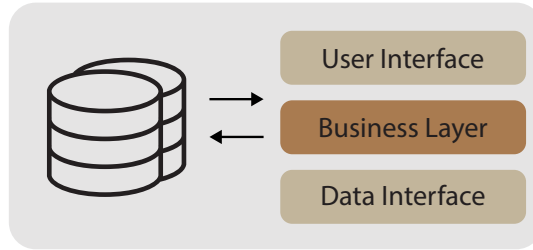


Figure 2.3. Illustration of a monolithic system architecture.

Monoliths are easily created and tested in the early design phase of a code project. However, at a later stage in the project, they become hard to manage and deploy in the operation of a large project. The entire stack must be updated and redeployed every time the developer updates the code base. For instance, when there is a minor update to the project, the developer must deploy a new solution to the entire code base, which can be a resource-demanding task, depending on the code project. This makes monolithic systems architecture ill-suited for large software development projects with many different stakeholders.

Another issue is that monoliths do not scale well in applications with high computational demand - and it does not adopt new technology well. If one part of the application receives a lot of traffic, it can be hard to scale just that part without scaling the entire application. This is because the monolith is challenging to modularize, meaning that the server demands more resources rather than distributing the task on several servers. Similarly, if the application has been created for a certain size of an organization, it is hard to down-scale that application during times of low traffic.

Using a monolithic architecture for a CDE in the AECO, especially if it involves simulation environments, is not well suited. Often, there is a need to perform a large quantity of demanding simulations in parallel. Furthermore, several stakeholders have different interests in a CDE, meaning that developing services for the CDE would require strict code base management for stakeholders not to interfere in each other's area of expertise.

2.3.2 Microservice architecture

In contrast to a monolithic architecture, a microservice architecture is a more modern approach based on modular microservices that perform very simple tasks. Usually, they receive a Hypertext Transfer Protocol (HTTP) request with a body, perform their operation, and then return a response to the application making the HTTP request using Representational State Transfer

(REST)ful APIs. Figure 2.4 shows an example of a microservice architecture. Microservices connect to the main application, but they can also use each other to perform operations.

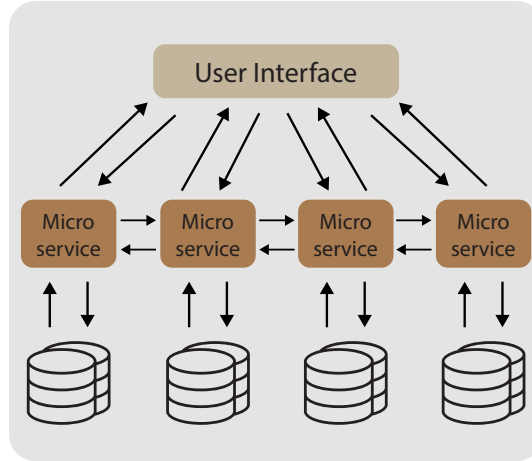


Figure 2.4. Illustration of a microservice architecture.

Microservices are units that run independently of each other. This makes it possible to develop, update, deploy, and scale any microservices without affecting other microservices [40]. Another advantage to microservices is that they are built to use technologies like Kubernetes⁵, making it possible to scale the application horizontally and create more instances when needed. If, for instance, an HVAC designer wants to run a parameter variation study of an energy simulation. This can result in several thousands of simulations. A monolithic architecture does this by scaling the application vertically, adding more threads to the current server rather than creating more servers (horizontal scaling). In a microservice architecture, creating a thousand containers that all simulate each parameter variation is possible - making it a faster process.

Microservices provide the road to creating a scalable web-based CDE that increases interoperability between advanced simulation tools, like detailed HVAC simulations and BIM models. The microservice architecture has the potential to ease the development and deployment of web-based detailed HVAC simulation engines. As a result of using recent technologies, like Kubernetes, the services within the CDE are easily scalable.

Microservices within one ecosystem are restricted to receiving an input, usually in a web-ready format, such as JavaScript Object Notation (JSON). This

⁵<https://kubernetes.io/>

means that microservices are based on a target object model. Therefore, to create a CDE based on a microservice architecture, data models must be developed to represent the relevant disciplines, in this case, thermal zones and HVAC systems.

2.4 Data models

There must be an underlying data model to build a platform based on any system architecture. This section reviews some of the existing data models and their serializations.

Data models were at the core of every digital development from the beginning of the digitalization of the AEEO industry. It was clear from the beginning that designing a super-schema encompassing all perspectives was a near-impossible task without constant modification. Douglas T. Ross and Jorge E. Rodriguez formulated [41]:

The first step in this direction is to recognize once and for all that it is completely impossible to construct a system that will satisfy the requirements immediately and without modification. To postulate the existence of a closed system for Computer-Aided Design, as we mean, is completely contradictory to the very sense of the concept.

The following subsections present the existing data models used in the AEEO industry, with their benefits and drawbacks.

2.4.1 Industry Foundation Classes

The IFC schema is a standardized digital description of everything included in the built environment, like buildings and infrastructure. IFC was adopted as an ISO standard in 2013 (ISO 16739-1:2018) [42]. The main goal of *buildingSMART* is that IFC should provide a standard format that major software vendors can use to exchange information between BIM modeling tools and external applications like HVAC or energy simulation programs. The development of the IFC schema was initiated in the 1990s and developed in the EXPRESS schema language. It has since been developed in several directions.

Since the industry is moving towards a more cloud-based approach, i.e., CDEs, the development of IFC has taken a new direction, trying to retrofit the IFC schema to more web-ready technologies. One effort by Afsari et al. (2017) [43] converted the IFC schema into a web-ready technology called JSON. They provided a translation of the IFC schema into a JSON format [43]. An-

other development, which is open source, is the IFC.js initiative⁶, utilizing the IFC.json format to create a JavaScript library capable of loading, displaying, and editing IFC files in a web browser⁷. While this makes the IFC schema available in a web-ready format, improving interoperability, the main problem is that the IFC schema is a super-schema trying to represent (too) many domains at the same time. Furthermore, several researchers have found that IFC files are error-prone when used to populate whole-building simulations [44–47].

IFC provides the stakeholders with a shared vocabulary that tries to improve interoperability in the AECO industry. However, the cost of the super-schema is flexibility [48]. Discipline-specific vocabularies should be developed to enable a shared vocabulary while obtaining a flexible data model. Furthermore, such discipline-specific vocabularies should be ready for web-based application exchanges. Therefore, other efforts have sought to recreate discipline-specific vocabularies rather than one super-schema [48]. The following sub-section summarizes the existing developments of discipline-specific vocabularies.

2.4.2 Discipline-specific vocabularies

The developments of existing discipline-specific vocabularies emphasize the core issue with the IFC super-schema [49–51].

This thesis has identified two efforts to create discipline-specific vocabularies using Web Ontology Language (OWL) or object model. This thesis distinguishes these for readability and to underline the work carried out. Essentially, both OWL ontologies and object models are object models with different design methodologies and communities behind them. Furthermore, the serialization methods vary between the two types of object models.

The use of OWL for the AECO domain is an effort usually carried out within the world wide web consortium Linked Building Data (LBD) community group⁸. The LBD community group develops ontologies that aim to provide the entire AECO domain discipline-specific vocabularies.

Object models, on the other hand, are not developed by one common entity but by different stakeholders within the AECO industry or academia.

Web Ontology Language

During recent years there have been numerous developments of discipline-specific OWL ontologies, such as Building Topology Ontology (BOT), Brick, Smart Energy Aware Systems (SEAS), Smart Applications REference (SAREF),

⁶<https://ifcjs.io/>

⁷<https://ifcjs.github.io/info/docs/Introduction>

⁸<https://www.w3.org/community/lbd/>

Real Estate Core (REC), and more, by the LBD community group. But what is an ontology? An ontology in computer science is the means to model the semantic nature of a system in a machine-interpretable way [52, 53]. One example is the representation of the professional links to the author of this thesis, as illustrated in Figure 2.5.

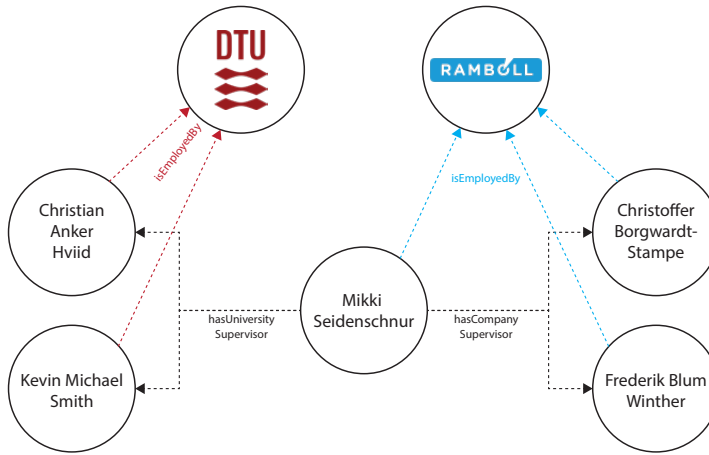


Figure 2.5. Simple knowledge graph illustrating the semantic connection between the author of the thesis and the supervising team. The connections are defined by subject/predicate/object triples, as seen in Figure 2.6.

Figure 2.5 shows that Mikki has a team of university supervisors (Christian and Kevin) and a team of company supervisors (Christoffer and Frederik). Furthermore, it shows that Mikki and his company supervisors are employed in Ramboll and that the university supervisors are employed at the Technical University of Denmark DTU. This knowledge graph is relatively simple, flat, and flexible - making it easy to extend or connect to other knowledge graphs.

Using ontologies to describe the AECO industry is not new. One of the first ontologies developed in the AECO industry was Industry Foundation Classes Web Ontology Language (ifcOWL) [54], which is an OWL representation of the IFC schema. Since it is a direct translation of the IFC schema, it has the same issues described with IFC - it is a super-schema that lacks flexibility [55], which is why there should be discipline-specific ontologies [56].

For instance, Mads Rasmussen created the BOT ontology⁹, to represent only the core topology of a building, including physical and conceptual objects and their relationships [57]. The LBD community group has developed several other ontologies for the AECO industry. An extensive review of ontologies related to HVAC systems is available in **Paper III**.

⁹<https://w3c-lbd-cg.github.io/bot/>

The official language for developing ontologies in the World Wide Web Consortium (W3C) and LBD community group is Resource Description Framework (RDF), which is serialized into Terse RDF Triple Language (Turtle) (.ttl format), and can be queried using SPARQL Protocol and RDF Query Language (SPARQL) [58]. Turtle provides a syntax to simplify the definition of triples, which is made from a subject-predicate-object paradigm, as seen in Figure 2.6.



Figure 2.6. Illustration of a triple composition, from [59].

Furthermore, Figure 2.7 illustrates a simple example, from FSO (See **Paper III**), of how to provide a semantic description of the connection from a pipe to a fitting.

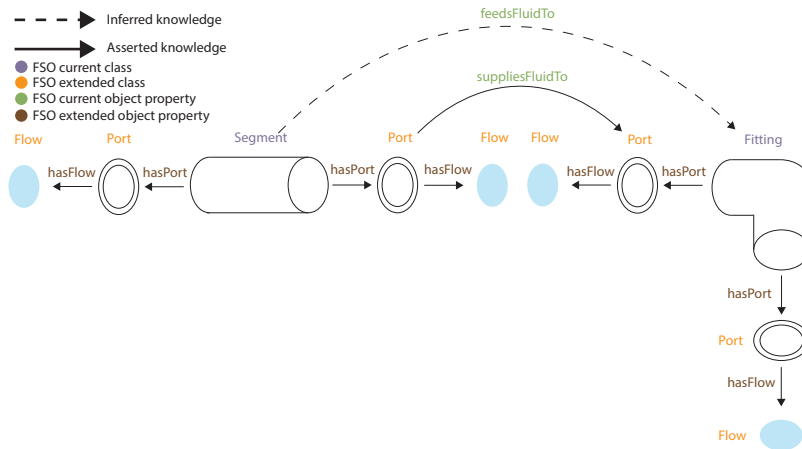


Figure 2.7. Illustration of the semantic description of a pipe segment and a fitting, through FSO, presented in **Paper V**.

Listing 2.1 provides a simple example of a text-string from a .ttl-formatted file, representing the model in Figure 2.7.

```

1 @prefix fpo: <https://w3id.org/fpo#> .
2 @prefix fso: <https://w3id.org/fso#> .
3 @prefix inst: <https://example.com/inst#> .
4
5 inst:Port-2 rdf:type fso:Port ;
6     fso:hasFlow inst:Flow-2 ;
7     fso:suppliesFluidTo inst:Port-3 .
8

```

```

9 inst:Flow-3 rdf:type fso:Flow .
10
11 inst:Flow-1 rdf:type fso:Flow .
12
13 inst:Pipe-1 rdf:type fso:Pipe ;
14     fso:feedsFluidTo inst:Elbow-1 ;
15     fso:hasPort inst:Port-2 , inst:Port-1 .
16
17 inst:Elbow-1 rdf:type fso:Elbow ;
18     fso:hasPort inst:Port-4 , inst:Port-3 .
19
20 inst:Port-3 rdf:type fso:Port ;
21     fso:hasFlow inst:Flow-3 .
22
23 inst:Flow-4 rdf:type fso:Flow .
24
25 inst:Port-1 rdf:type fso:Port ;
26     fso:hasFlow inst:Flow-1 .
27
28 inst:Flow-2 rdf:type fso:Flow .
29
30 inst:Port-4 rdf:type fso:Port ;
31     fso:hasFlow inst:Flow-4 .

```

Listing 2.1. Listing illustrating the serialized turtle file that represents the model illustrated in figure 2.7.

However, the downside to the RDF graph in Listing 2.1 in contrast to JSON is that the RDF graph is poorly enabled for Object Oriented Programming (OOP) languages, such as C#. Therefore, the user needs to use a SPARQL query to access the data in an RDF data store or use RDFlib or other tools.

Since formats like JSON are classically used for web-based applications, W3C has developed a data format called JavaScript Object Notation for Linked Data (JSON-LD). JSON-LD is another serialization of data that serializes the RDF structured files into the JSON format. JSON-LD aims to provide developers using RDF-structured data with a more commonly used format for web applications [48]. Several more serialized formats exist, like N-Triples, N-Quads, Trig, and XML.

Listing 2.2 shows an example of the same model from Figure 2.7 expressed in Listing 2.1, now in JSON-LD. Listing 2.2 also illustrates that JSON-LD is not very human-readable. JSON-LD provides a more web-ready version of an RDF graph, which does make RDF graphs easier to work within a web-application. It links components through Unique Resource Identifiers (URI)s, making it a unique component or property which does not exist in several places.

```

1 {
2   "@graph" : [ {

```



```

3      "@id" : "inst:Elbow-1",
4      "@type" : "fso:Elbow",
5      "hasPort" : [ "inst:Port-4", "inst:Port-3" ]
6    }, {
7      "@id" : "inst:Flow-1",
8      "@type" : "fso:Flow"
9    }, {
10     "@id" : "inst:Flow-2",
11     "@type" : "fso:Flow"
12   }, {
13     "@id" : "inst:Flow-3",
14     "@type" : "fso:Flow"
15   }, {
16     "@id" : "inst:Flow-4",
17     "@type" : "fso:Flow"
18   }, {
19     "@id" : "inst:Pipe-1",
20     "@type" : "fso:Pipe",
21     "feedsFluidTo" : "inst:Elbow-1",
22     "hasPort" : [ "inst:Port-2", "inst:Port-1" ]
23   }, {
24     "@id" : "inst:Port-1",
25     "@type" : "fso:Port",
26     "hasFlow" : "inst:Flow-1"
27   }, {
28     "@id" : "inst:Port-2",
29     "@type" : "fso:Port",
30     "hasFlow" : "inst:Flow-2",
31     "suppliesFluidTo" : "inst:Port-3"
32   }, {
33     "@id" : "inst:Port-3",
34     "@type" : "fso:Port",
35     "hasFlow" : "inst:Flow-3"
36   }, {
37     "@id" : "inst:Port-4",
38     "@type" : "fso:Port",
39     "hasFlow" : "inst:Flow-4"
40   } ],
41   "@context" : {
42     "hasFlow" : {
43       "@id" : "https://w3id.org/fso#hasFlow",
44       "@type" : "@id"
45     },
46     "suppliesFluidTo" : {
47       "@id" : "https://w3id.org/fso#suppliesFluidTo",
48       "@type" : "@id"
49     },
50     "hasPort" : {
51       "@id" : "https://w3id.org/fso#hasPort",
52       "@type" : "@id"
53     },
54     "feedsFluidTo" : {
55       "@id" : "https://w3id.org/fso#feedsFluidTo",
56       "@type" : "@id"
57     },

```

```

58     "fso" : "https://w3id.org/fso#",
59     "fpo" : "https://w3id.org/fpo#",
60     "inst" : "https://example.com/inst#"
61   }
62 }

```

Listing 2.2. The listing shows an example of the system shown in figure 2.7, serialized in JSON-LD.

One problem for OWL ontologies is that they can be complex and challenging to understand by developers unfamiliar with the underlying concepts and formalisms, like most of the self-made software developers in the AECO industry. Implementing and using OWL for web applications in large organizations can be a massive undertaking. Developing and adding new components to existing ontologies is not flexible, making it harder to utilize for a platform built on a microservice architecture.

Discipline-specific data models should encourage web-ready flexible data structures that can easily be understood by the software developers in the AECO industry and be edited or extended to suit the use of the specific developer.

Object Models

In recent years, the AECO industry has seen a development of object models, trying to provide a common vocabulary for all stakeholders in the building project [60]. At the time of writing, there are two open-sourced object models: The Buildings and Habitats object Model (BHoM) and Speckle.

According to the BHoM¹⁰ GitHub, BHoM is:

"The BHoM is a collaborative computational development project for the built environment. It is a collective effort to share code and standardize the data that we use to design, everyday – across all activities and all disciplines."

BHoM provides a basis for transferring data from one application to another. It means that most transfer of data will happen between the specific applications, without a CDE to connect them, as shown in Figure 2.8, into a file serialized based on the BHoM.

BHoM provides a fundamental data structure for all things in the built environment related to data. It is not a single source of truth since it is file-based. One could use the data structure and parsers to create a database based on

¹⁰<https://bhom.xyz/>

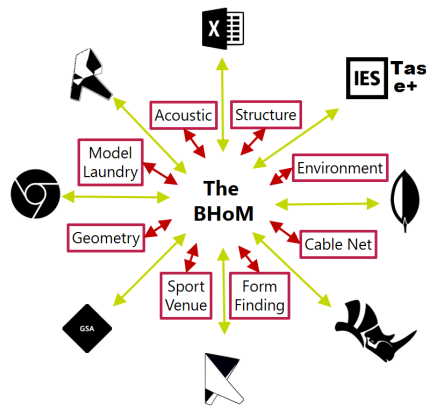


Figure 2.8. Illustration of the BHoM from the GitHub wiki page.

the BHoM. However, upon investigating the BHoM GitHub repository¹¹, it is clear that BHoM lacks some standardization, even though it is fully flexible.

Another issue is that the mission of BHoM is to represent everything in the built environment, just like the IFC schema. Finally, the object models for the Mechanical, Electrical, and Plumbing (MEP) models are relatively simple and need further development. It is currently not possible to use BHoM to semantically describe the connection between components in an HVAC model. Therefore, the object model should be developed further.

Another partially open-source development is Speckle. According to Speckle¹², they are an

".. open-source data platform for architecture, engineering, and construction. It liberates your data from proprietary file formats and closed source software and puts it back into your hands."

Like the BHoM, they have a central object model that provides the basis of the database. Furthermore, Speckle provides a platform that connects different tools, like Revit to Speckle, Speckle to Rhino, etc. While the general idea of Speckle follows the definition of a CDE, one problem is that the object models for the HVAC discipline are rather sparse. Like BHoM, the Speckle object model contains HVAC components but does not have a semantic description

¹¹<https://github.com/BHoM/BHoM>

¹²<https://speckle.systems/about/>

of their relationship. This may be because the mission of Speckle is to create storage for large 3D models.¹³

Object models are widely used in different industries to develop software applications. The AECO industry already uses them at the center of proprietary and open-source tools like Autodesk Revit, Integrated Environmental Solutions Virtual Environment (IESVE), E+, Modelica, and several others. However, the few open object models that describe the disciplines within AECO domain with no central application are insufficient to describe HVAC systems in a web-ready format.

2.5 Summary

As a result of poor data interoperability for BIM models in the HVAC domain, HVAC engineers are using idealized simulation models that do not capture the complete physics of the HVAC system in the building. This means that HVAC systems are oversized by the HVAC engineer to ensure that they can deliver according to the peak demand of the thermal zones. Tools for coupled HVAC and BPS should be utilized to lower the performance gap by closing the information gap.

Researchers have developed tools such as *IFC2Modelica*, *BIM2Modelica*, and *Revit2Modelica* for automated transfer of BIM model information to perform detailed HVAC simulations in Modelica simulation environment. However, these tools are academic-only tools that have seen no industry adoption. It takes a lot of effort to augment BIM models with the necessary information to perform detailed HVAC simulations. As long as the file-based exchange is the industry standard, it will be hard and not prosperous to implement tools like Spawn and Modelica in the everyday workflow. Consequently, there are limited means to performing whole-building simulations that also contain a detailed HVAC simulation.

Implementing the existing tools in a CDE based on a microservice architecture has the potential to allow co-simulation of whole-building simulation and detailed HVAC simulation generated based on a BIM model. To facilitate detailed HVAC simulation in a CDE, data models are needed that can represent HVAC systems, thermal zones, and their connection.

¹³<https://github.com/specklesystems>

CHAPTER 3

Research Design

This chapter presents the research design based on the challenges identified in the introduction (1) and background (2) Chapters.

Section 3.1 formulates the problem statement based on the work of Chapters 1 and 2.

Section 3.2 presents the research questions.

Section 3.3 poses the research tasks created to answer the research questions.

Section 3.4 describes the methodology for carrying out the research tasks.

3.1 Problem statement

Productivity has barely increased in the AECO industry for the past decade. This is mainly because the information management in the AECO industry is fragmented and lacks strategies/technologies to close the information gap between stakeholders. Consequently, simulation tools have poor interoperability and rarely use the same information.

The information gap between stakeholders leads to a gap in buildings' predicted and measured performance, called the performance gap. Predictions of the performance of the HVAC systems are carried out using advanced whole-building simulation tools. However, since there is a lack of information for the HVAC system, often simulations of the HVAC system are carried out using idealized HVAC systems or using wrongful assumptions. This leads to a simulation model that does not represent the complex nature of the HVAC system and its connection to the thermal zones of the building.

Tools exist to simulate the dynamic HVAC system coupled with the thermal zones through a Modelica simulation environment, using recently developed object libraries, like *Buildings*, *AixLib* *BuildingSystems*, and *IDEAS*. Furthermore, several research projects (*IFC2Modelica*, *BIM2Modelica*, *Revit2Modelica*) have worked on creating tools to generate Modelica object models semi-automatically using BIM models. While these tools increase interoperability, they provide a file-based exchange of information, meaning there is no single source of truth for the information.

To provide a single source of truth, there should be a Common Data Environment (CDE) that contains discipline-specific object models. Such a CDE should be created using a web-based microservice architecture to allow AECO organizations to scale the CDE as necessary. The single source of truth CDE is a tool that will digitally transform the AECO industry, and reduce the performance gap, as a consequence of eliminating the information gap.

3.2 Research questions

The research questions are based on a vision to provide a single source of truth using a web-based Common Data Environment (CDE) that centralizes all project data and drives a modular development environment that lowers the barrier for the digital transformation of the HVAC discipline, in AECO industry.

The research questions are:

1. Can a CDE serve as a single source of truth, replacing file-based BIM,

and consequently close the information gap in the building design process?

2. Can a CDE enable seamless data integration between proprietary and open data formats?
3. Is it possible to formulate and implement a discipline-specific data model to represent (HVAC) flow systems and their properties in a web-based CDE?
4. Is it possible to formulate and implement a discipline-specific data model for the representation of thermal zones and their properties in a web-based CDE?
5. Does a technology-agnostic CDE based on a microservice architecture facilitate the digital transformation by providing a platform for organic additions of automated design services in the building design process?
6. Is a microservice CDE for whole-building simulation with detailed HVAC simulation a tool that can reduce the performance gap in a practical HVAC design setting?

3.3 Research tasks

This thesis intends to create the infrastructure for a technology-agnostic Common Data Environment (CDE) easily adoptable by the HVAC discipline in the AECO industry. Furthermore, it focuses on how the created CDE can evaluate and reduce the performance gap from predicted to measured energy performance in the HVAC discipline. To achieve this, the following research tasks are planned:

1. To create a flexible data model capable of representing flow systems that facilitate seamless data integration from BIM models into a web-based CDE;
2. To create a flexible data model capable of representing thermal zones and their connection to flow systems that facilitate seamless data integration from BIM models into a web-based CDE;
3. To create a technology-agnostic CDE system architecture capable of scaling with the AECO industry demands;
4. To develop microservices for whole-building simulation with detailed HVAC simulation that can be used to evaluate the performance gap from predicted to measured performance.

3.4 Methodology

This Section proposes the methodology for carrying out each of the research tasks listed in Section 3.3.

3.4.1 Research task 1

To create a flexible data model capable of representing flow systems that facilitate seamless data integration from BIM models into a web-based CDE.

Flexible data models are vividly and enthusiastically discussed in the AECO industry. Section 2.4 introduces part of the existing body of data models with their benefits and drawbacks. Recent academia tends to focus on the serialization of data formats rather than getting the common vocabulary ready to represent the core concepts of a discipline. Whether data models are serialized in JSON, Turtle, Extensible Markup Language (XML), or any other available format, there needs to be an object model behind it. The literature study in Chapter 2 showed a lack of discipline-specific object models to represent flow systems.

Research task 1 is formulated to create a vocabulary for describing the energy and mass flow relationship of systems, their components, and the composition of such systems. This thesis does not make any assumptions about data formats. Therefore, it creates a semantic web ontology based in an RDF format and an object model for the representation of flow systems in a web-ready JSON format. This thesis intends to use the vocabularies for advanced hydraulic simulation to later allow for detailed HVAC simulations, specified in Section 3.4.4. Use case demonstrations are provided to demonstrate the functionality of the vocabulary. The use cases intend to showcase that it is possible to serialize the created object models from a proprietary BIM model and store them in a web-based object-oriented database. The object model repository is publicly available on GitHub.

3.4.2 Research task 2

To create a flexible data model capable of representing thermal zones and their connection to flow systems that facilitate seamless data integration from BIM models into a web-based CDE.

Based on research task 1 and the literature study in Chapter 2, there is a need for a flexible object model to represent thermal zones in a building to perform whole-building simulation. Rasmussen et al. [56] created the BOT ontology to provide a common core vocabulary for the description of the build-

ing in a simplified and extendable way. Therefore, the BOT ontology does not contain the definitions to describe thermal zones, meaning that only a simple whole-building simulation can be performed using the BOT ontology.

Research task 2 creates a vocabulary for the description of the thermal zones of a building. It uses the knowledge obtained from research task 1 and is constructed as an object model instead of an OWL ontology. The vocabulary enables whole-building simulation coupled with detailed HVAC simulation using the CDE developed in research task 3. Furthermore, it allows for whole-building simulations isolated from the detailed HVAC simulation for more straightforward use cases. Use case demonstrations are provided to demonstrate the functionality of the developed vocabulary. The use cases intend to showcase that it is possible to serialize the created object model from a proprietary BIM model and transfer it into a web-based object-oriented database. The object model is publicly available on GitHub.

3.4.3 Research task 3

To create a technology-agnostic CDE system architecture capable of scaling with the AECO industry demands.

Chapter 1 illustrates that the digital transformation of the AECO industry is halted by silo-based developments. Currently, tools are being developed in the industry without linking them to a greater context. While this seems like an easy and decentralized solution to develop tools for simple tasks, it also means that one tool usually only works for one BIM data model. Chapter 2 illustrates that a solution to this problem is the web-based Common Data Environment (CDE). The CDE is built on a microservice architecture, allowing for a scalable solution.

Research task 3 incorporates the data models and their serialization tools developed in research tasks 1 and 2 and links them to a CDE. Furthermore, the CDE contains simple microservices, consuming the data in the CDE, to prove that several tools can consume the same data and therefore be connected in the CDE. The microservices can: perform a model validation, perform a simple airflow calculation, and give basic statistics about the HVAC model transferred to the CDE from the proprietary BIM format. The object model repository is publicly available on GitHub.

3.4.4 Research task 4

To develop microservices for whole-building simulation with detailed HVAC simulation that can be used to evaluate the performance gap from predicted to measured performance.

One of the main problems with the most commonly used whole-building simulation tools is that they are over-simplified. The HVAC engineer usually assumes that if they simulate the rooms with the highest heating/cooling/air demand in the building, they can extrapolate that information to the rest of the building. This usually generates a case where the actual heating, cooling, and air demand is poorly estimated, leading to under- or oversized HVAC systems. Furthermore, whole-building simulation tools contain assumptions of idealized HVAC systems. Idealized HVAC systems are modeled to consistently deliver the heating, cooling, or ventilation demand to all of the rooms in the building. There is a need for coupled whole-building simulation and detailed HVAC simulation to realistically predict the performance of the building and HVAC system.

To provide whole-building simulation coupled with detailed HVAC simulation, there is a need to develop several microservices. The literature study in Chapter 2 makes it clear that several open-source tools are available for whole-building simulation and detailed HVAC simulation of the HVAC system. Therefore, this research task aims to develop microservices building on the CDE created in research task 3. The CDE creates a single source of truth based on the object models introduced in research tasks 1 and 2 for all microservices to base their calculation, closing the information gap. Finally, several use cases are conducted to prove that the microservices can be used to calculate the performance gap.

CHAPTER 4

Results

This Chapter uses the articles produced during this Ph.D. to provide the research findings related to the research tasks formulated in the research design Chapter.

Section 4.1 provides an overview of the articles developed during this PhD thesis and shows which research tasks they were developed for.

Section 4.2 introduces Flow Systems Ontology (FSO) and Flow System Classes (FSC) object model to represent flow systems.

Section 4.3 summarizes the THERM object model to represent thermal zones.

Section 4.4 presents the system architecture developed for the CDE and validates the developed object models from the previous Sections.

Section 4.5 exemplifies how the system architecture developed in Section 4.4 can be used to perform automated whole-building simulations, including detailed HVAC simulations.

4.1 Research tasks and related articles

This Chapter summarizes the main results of the research tasks presented in Chapter 3. The research tasks were performed to answer the research questions provided in the research design, in Chapter 3. Chapter 6 contains the appended research papers discussed in the results.

Table 4.1. Table of research tasks, and in which papers they are carried out.

	Research tasks	Appended papers
1	To create a flexible data model capable of representing flow systems that facilitate seamless data integration from BIM models into a web-based CDE	I, III - V, VII - X
2	To create a flexible data model capable of representing thermal zones and their connection to flow systems that facilitate seamless data integration from BIM models into a web-based CDE	I, II
3	To create a technology-agnostic CDE system architecture capable of scaling with the AECO industry demands	I, II, V, VII
4	To develop microservices for whole-building simulation with detailed HVAC simulation that can be used to evaluate the performance gap from predicted to measured performance	I, II, IV, VI, VIII

The following subsections are organized according to research tasks and provide a summary of each research task and the papers that answer them. For simplification, only the main contributions of the research papers have been included in this Chapter. Each research paper contains a detailed description of the relevance and contribution to research and industry.

4.2 Research task 1

In research task 1, two data models were developed, called:

1. Flow Systems Ontology (FSO) - based on OWL ontologies and
2. Flow System Classes (FSC) - based on object models

The purpose of research task 1 was to create flexible data models to represent flow systems. Developing an OWL and object model helps to understand the

benefits and drawbacks of each of the principles. Furthermore, this research task aims to create a flexible discipline-specific common vocabulary representing a flow system with the BIM model. The following sub-sections present the developed data models.

4.2.1 Flow System Ontology

Paper III presents FSO which was initially conceptualized by Rasmussen (2019) [61], as an extension to the BOT ontology. The general idea proposed by Rasmussen was that a flow system consists of consumers, sources, and distribution systems. However, the development was conceptual and was not ready to be used in the AECO industry. The purpose of FSO is to provide a common vocabulary for describing the energy and mass flow relationships between systems and their components in a minimal extendable way. FSO is an open-source development¹. FSO contains 14 classes and 23 object properties.

The main idea for FSO is that it makes up the discipline-specific description needed to describe flow systems. If the BOT ontology is linked with FSO, it also allows for flow system descriptions in the context of a building. Figure 4.1 provides an example of FSO used with BOT to conceptualize a building and its systems.

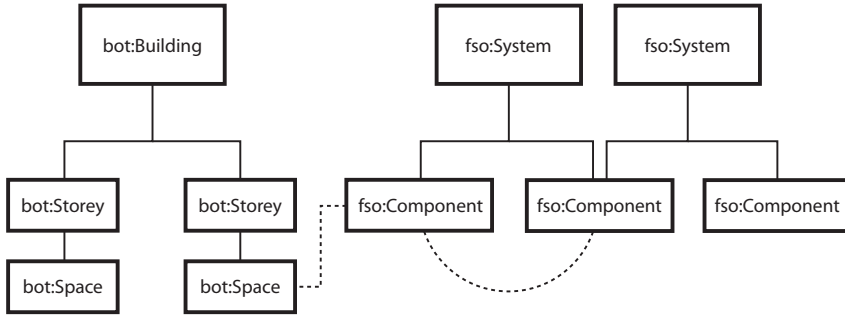


Figure 4.1. Illustration of FSO as an extension to bot. Original Figure from **Paper III** [59].

Figure 4.1 illustrates that buildings are conceptualized as two parallel tree hierarchies, which can be described using BOT and FSO. In this example, *fso:Component* transfers heat to (*fso:transfersHeatTo*) the *bot:Space* on the left-hand side. Furthermore, the Figure shows one *fso:Component* that is connected with (*fso:connectedWith*) another *fso:Component*. An example of that could be a duct that supplies fluid to (*fso:suppliesFluidTo*) an air terminal.

With the established core terminology, it is possible to describe an HVAC

¹<https://alikucaavci.github.io/FSO/>

system, as shown in Figure 4.2. Figure 4.2 shows that FSO has the expressiveness to describe an entire HVAC system and the connections between all of the components and sub-systems. It also exemplifies the interface between FSO and BOT, showing a chilled beam in a room. It shows that *<Room-1>* *transferHeatTo* the *<Beam-1-HeatExchanger>* and the *<ReturnVent-1>*. The Figure shows the connection on a conceptual level. To formalize this, the system was created in a Turtle file, as shown in Listing 4.1.

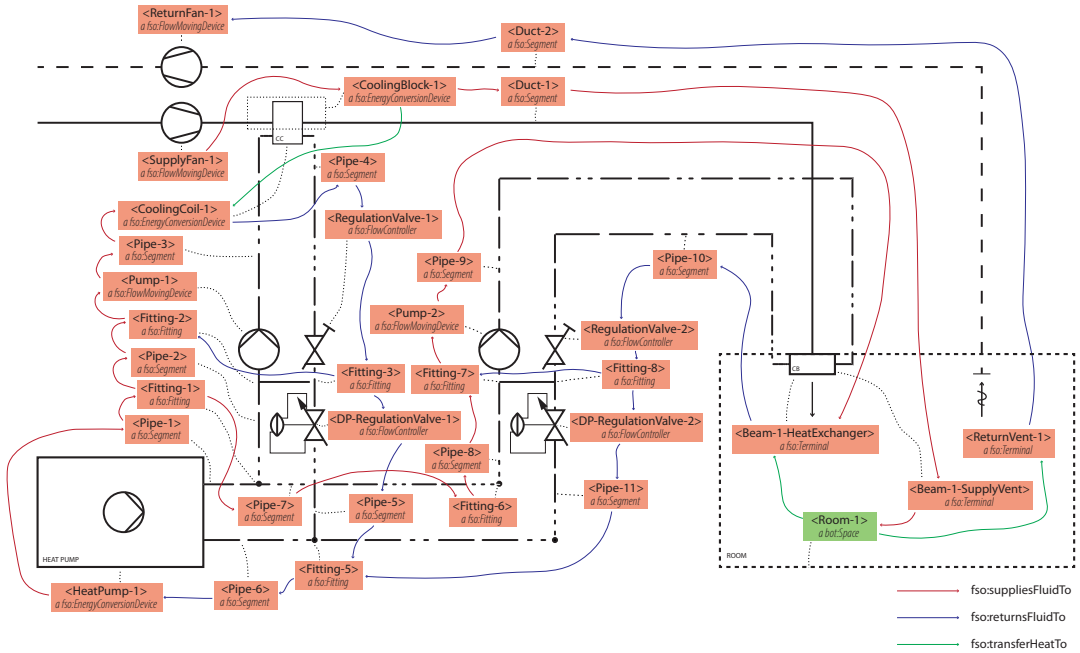


Figure 4.2. Illustration of the FSO terms used to describe an HVAC system, which contains a heating, ventilation, and cooling system. Original Figure from **Paper III** [59].

```

1 @prefix fpo: <https://w3id.org/fpo#> .
2 @prefix fso: <https://w3id.org/fso#> .
3 @prefix inst: <https://example.com/inst#> .
4
5 inst:CoolingCoil-1 a fso:System ;
6   fso:hasComponent
7     inst:CoolingCoil-1-LiquidSide ,
8     inst:CoolingCoil-1-AirSide .
9
10 inst:CoolingSystem-1 a fso:System ;
11   fso:hasSubSystem inst:Beam-1 , inst:CoolingCoil-1 ;
12   fso:hasComponent
13     inst:HeatPump-1 ,
14     inst:Pump-1 ,
15     inst:Pump-2 ,

```

```

16     inst:RegulationValve-1 ,
17     inst:RegulationValve-2 ,
18     inst:DP-RegulationValve-1 ,
19     inst:DP-RegulationValve-2 ,
20     inst:Pipe-1 ,
21     inst:Fitting-8 .
22
23 inst:VentilationSystem-1 a fso:System ;
24     fso:hasSubSystem inst:Beam-1 , inst:CoolingCoil-1 ;
25     fso:hasComponent
26         inst:CoolingCoil-1-LiquidSide ,
27         inst:SupplyFan-1 ,
28         inst:CoolingCoil-1-AirSide ,
29         inst:Duct-1 ,
30         inst:ReturnVent-1 ,
31         inst:Duct-2 ,
32         inst:ReturnFan-1 .

```

Listing 4.1. Listing illustrating a partial data model that represents the HVAC system illustrated in Figure 4.2, adapted from **Paper III** [59].

After the initial development of FSO, it was clear that if it should gain adoption for practical use, there need to be further use cases illustrated. **Paper X** provides a use-case for FSO, in which an HVAC system containing a complicated six-way valve is conceptualized. **Paper X** also shows how FSO aligns with parallel developments of ontologies to represent flow systems with different use cases.

FSO describes the core characteristics of HVAC systems. However, **Paper III** posed a limitation of FSO, which is that it does not contain the terminology to describe the properties of components for HVAC systems. FSO is intentionally kept lightweight to expand the application and does not contain the information necessary to perform simulations in the HVAC domain.

FSO Extension: Flow Properties Ontology

Paper V illustrates the Flow Properties Ontology (FPO). The purpose of Flow Properties Ontology (FPO) is to extend FSO so that components in FSO are augmented with capacity and size-related properties to perform simulations of HVAC systems. **Paper V** extends FSO with FPO, sub-dividing the existing 9 components into 19 medium-level components to explicate the type of component. For instance, the *EnergyConversionDevice* is split into four medium-level components: *Boiler*, *Chiller*, *HeatExchanger*, and *Heat-Pump*. Finally, FPO also contains port descriptions from one component to another, as shown in Figure 4.3.

Paper III, **Paper V**, and **Paper IX** exemplifies how FSO and FPO can be queried using SPARQL queries. **Paper III** shows that it is possible to perform simple HVAC calculations on the system described in the Turtle

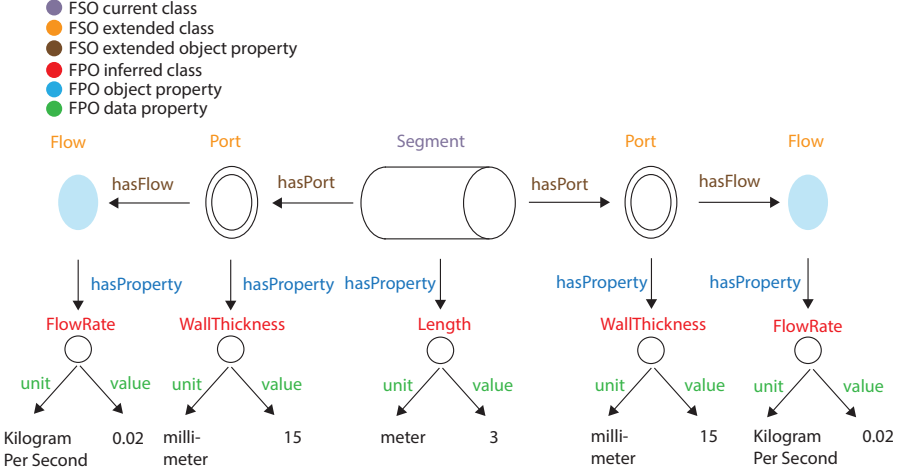


Figure 4.3. Illustration of an *fso:Segment* and its FPO classes, objects, and data properties, from **Paper V**.

syntax. The result returned from a SPARQL query are in the form of a knowledge graph or a table. Using a SPARQL query, it is then possible to inspect the resulting graph from the query, as shown in Figure 4.4.

FSO and FPO represents flow systems in OWL and therefore makes it possible to serialize it in an RDF-based model. These include Terse RDF Triple Language (Turtle) and JavaScript Object Notation for Linked Data (JSON-LD). However, as described in Section 2.4, these formats are hard to consume for web applications and hard to understand for developers not used to the RDF-based formats. Furthermore, the results returned from a SPARQL visualizer is usually in the form of a graph, as shown in Figure 4.4, or as a table.

4.2.2 Flow System Classes

Paper I proposed the Flow System Classes (FSC) object model. The purpose of FSC was to represent the entire flow system for sub-systems, components, and their properties. In contrast to the development of FSO, a Unified Modeling Language (UML) class diagram was used to generate the FSC object model. Figure 4.5 illustrates a simplified overview of the FSC object model.

Figure 4.5 shows that the *Component* class is the super-class for all other types of components, which includes *FlowTerminal*, *Fitting*, *EnergyConversionDevice*, *FlowController*, *FlowMovingDevice*, and *FlowSegment*. Those sub-classes are then divided further to provide the level of detail needed

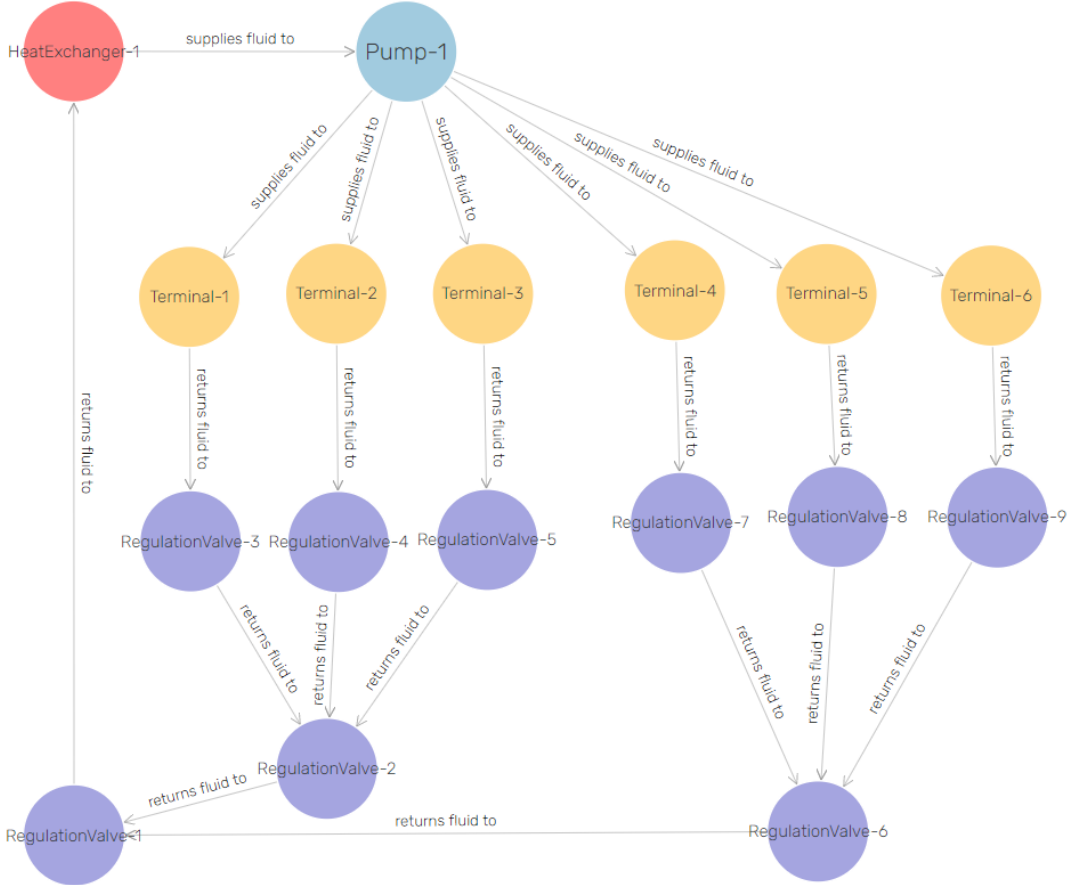


Figure 4.4. Illustration of an *fso:Segment* and its FPO classes, objects, and data properties, from **Paper III** [59].

to perform flow system calculations/simulations. For instance, the *Fitting* component is divided into *Bend*, *Cross*, *Reduction*, *Tee*, and *Cap*. The FSC object model consists of 37 classes and 54 methods. Figure 4.6 shows a simple use-case of FSC.

In **Paper I**, the FSC object model was used to serialize a JSON file based on the proprietary format of Revit, using the Revit C# API. To illustrate the functionality of the FSC exporter, two example models were serialized to a JSON file with all their components. To prove that the serialized JSON files are suitable to provide seamless integration to a CDE, three simple microservices were created in the CDE. The microservices were developed in a

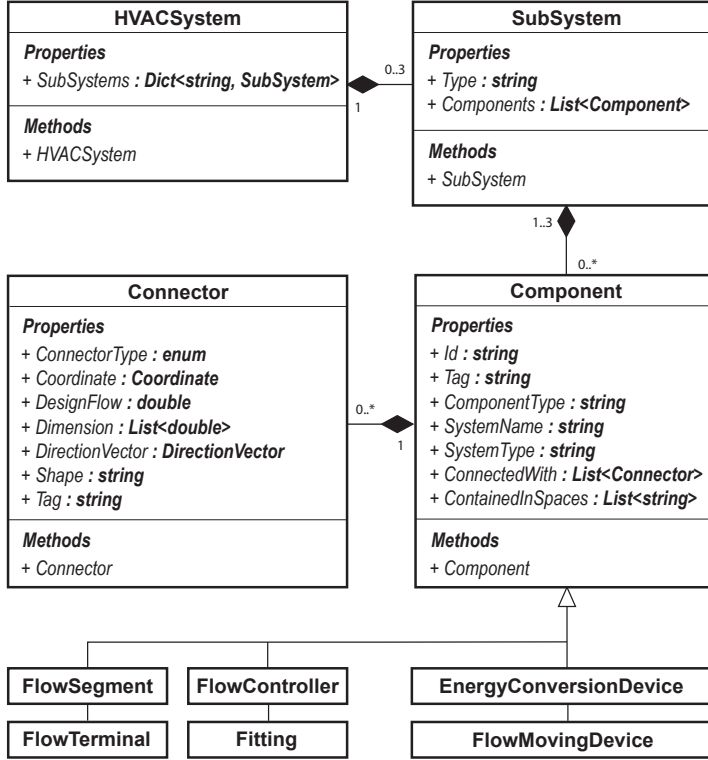


Figure 4.5. Illustration of the FSC object model from **Paper I** [62].

commonly used framework called Flask². The microservices are all available on GitHub³.

The developed microservices take the JSON file serialized from the FSC object model as the input. Once they have received the JSON file, they perform their intended operation. Upon finishing the operation, they return a result JSON. Based on Python, the microservices are developed in the Flask framework, a simple way to perform calculations on a JSON. Python is one of the most popular programming languages, which makes it widely available to even entry-level developers. While **Paper I** provided an example of microservices written in Python, they can be created using any programming language or framework that allows for microservices, like .NET Core, Node.js, etc.

²<https://flask.palletsprojects.com/en/2.2.x/>

³<https://github.com/orgs/Virtual-Commissioning/repositories?type=all>

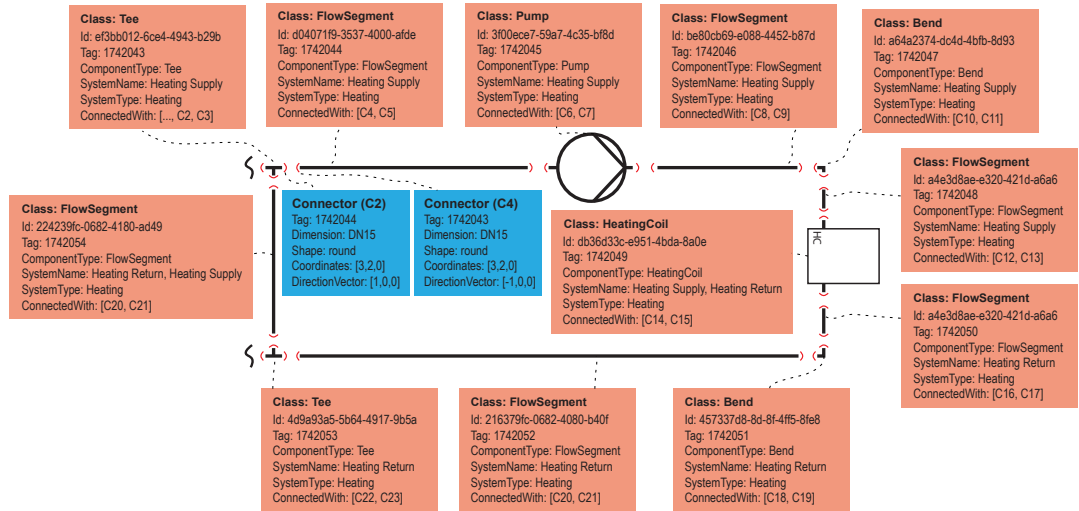


Figure 4.6. Illustration of the FSC object model from **Paper I** [62] used to describe a mixing loop for heating.

4.2.3 Key findings

Research task 1 aims to create a flexible data model capable of representing flow systems and facilitating seamless data integration from BIM models into a web-based CDE. Two data models were created to answer research task 1: one in a Web Ontology Language (OWL) and one as an object model. The development of both data models and their use cases showed that to enable seamless data integration from BIM to a CDE, it is more suitable to use an object model for this thesis. By using the object model, it is possible to serialize a JSON directly from the BIM tool and feed it into an object-oriented database, such as mongoDB, in a CDE. Furthermore, the data stored in the object-oriented database can be consumed directly by microservices created in Flask and return a response. However, OWL ontologies offer a level of standardization for schema development that does not exist in a regular object model. An object model provides flexibility and a more technology-agnostic approach but lacks standardization. The object model can be adapted to be serialized in an RDF-based format.

4.3 Research task 2

4.3.1 Thermal Zone Classes

Iteration 1 - THERM

Section 4.2 shows that object models have the potential to provide seamless integration from BIM to CDE to microservices. The first iteration of the THERM object model was introduced in **Paper I**, as shown in Figure 4.7.

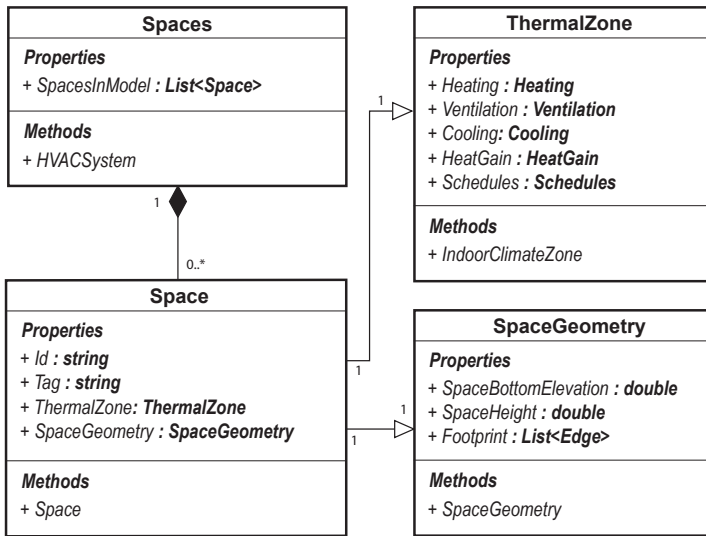


Figure 4.7. UML-diagram of the first iteration of THERM, from **Paper I** [62].

The first iteration of THERM conceptualizes the connection of spaces and flow systems. The space contains an *Id* and a *Tag* to relate it to the FSC object model. Furthermore, it contains a *ThermalZone* and *SpaceGeometry*. The *ThermalZone* contains all information needed for to perform thermal simulations. The *SpaceGeometry* contains the information that allows for the 3D description of the room.

The first iteration of THERM enables the connection of the HVAC system and the thermal zones, but it is too simple to describe the needed information for a BPS. **Paper I** also describes a way to connect flow systems with rooms semantically. In FSC, a property called *ContainedInSpaces* is added to the *Component* super-class, as shown in Figure 4.5. This allows for the semantic connection of HVAC systems and thermal zones.

Additionally, **Paper I** introduces a microservice capable of calculating the dimensioning airflow of a ventilation system using the airflow demand of the

thermal zones. This is possible because all components that are terminal units are mapped to be contained within the space that exists in the BIM model and because the spaces contain an airflow demand property.

Iteration 2 - THERM

Paper II introduces the next iteration of THERM. The purpose of the second iteration of THERM is to enable BPS in a web-based CDE. Like FSC, THERM describes the building as a hierarchy, as shown in Figure A.1. The THERM hierarchy contains a *Site* which contains a *Building*, which contains *Zones*, which contains *Surfaces*, *InternalGains*, *HVACSystems*, *Infiltration*, and *ZoneShadings*. Furthermore, THERM also describes the geometry of the given thermal zone.

To be able to use the THERM object model in a CDE, a THERM exporter is created in the Revit C# API. The THERM exporter allows for the thermal zones of a BIM model to be serialized in a JSON file. Figure 4.8 illustrates a use case serialized from Revit with the THERM exporter.

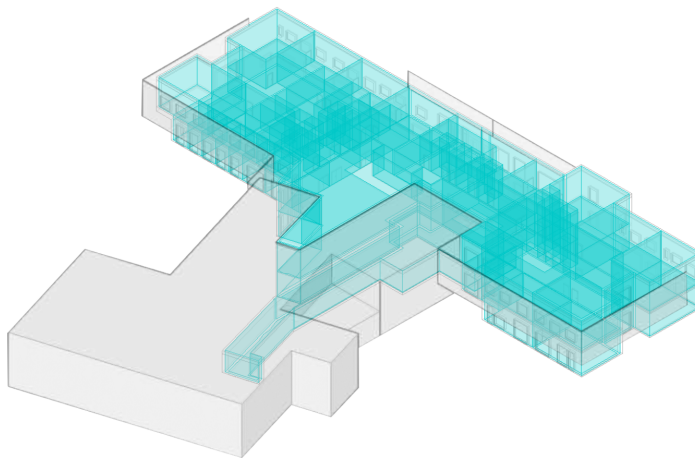


Figure 4.8. Illustration of the Frederiksberg School Revit serialized as a THERM JSON file, from **Paper II**.

4.3.2 Key findings

Research task 2 aims to create a flexible data model capable of representing thermal zones and their connection to flow systems, facilitating seamless data integration from BIM models into a web-based CDE. With the creation of THERM, it is possible to transfer data from a BIM model into a CDE, and from there, use microservices to perform simulations. Furthermore, with the connection of FSC and THERM, it is now possible to describe a flow system's

connection with a given thermal zone. This creates the outset for performing whole-building simulation with detailed HVAC simulation, which is described in Section 4.5.

4.4 Research task 3

Research task 3 aims to create the system architecture for a scalable and technology-agnostic CDE for the AECO industry. Furthermore, it aims to solve the interoperability issues in the AECO industry by providing a single source of truth in a technology-agnostic platform, using common data formats, such as FSO, FSC, and THERM described in Sections 4.2 and 4.3.

4.4.1 System architecture

The following provides a list of requirements of a CDE, which is addressed in the system architecture. A CDE should:

1. Provide a single source of truth data storage using discipline-specific data models and thereby enable seamless data integration
2. Provide model validation tools to ensure data integrity
3. Provide an application that is scalable as it increases or decreases in size and complexity
4. Enable easy development and deployment of technology-agnostic microservices (Python, JavaScript, C#, etc.)
5. Enable the development of tools that connect existing simulation tools

Paper I describes the first iteration of the system architecture for a CDE, capable of living up to the above requirements.

Figure 4.9 shows the first iteration of the system architecture. First, the HVAC database is generated using the BIM HVAC model. This is done using FSC object model and the FSC object model exporter, both presented in **Paper I**. The BIM HVAC model is serialized as a JSON file, which is loaded into the HVAC database. After the first step, the data is contained within the CDE and can be used by any microservice that receives the FSC object model serialized as a JSON file. The second step is to develop tools to use the data in the CDE. Therefore, three microservices are developed as part of **Paper I**.

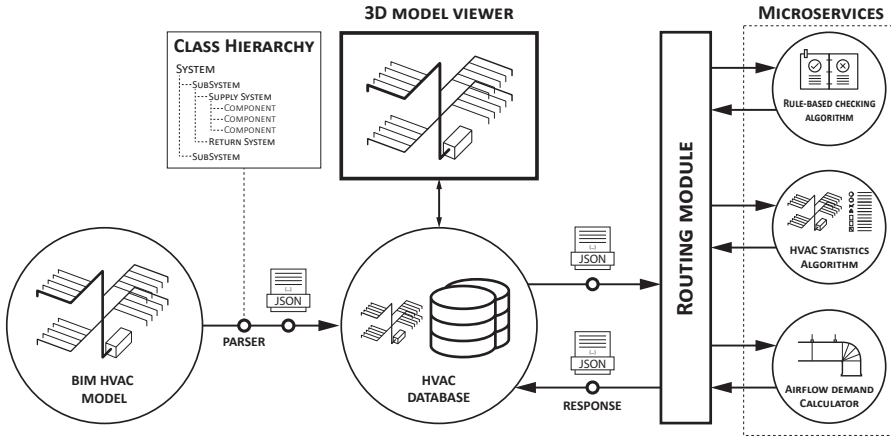


Figure 4.9. Illustration of the system architecture in **Paper I** [62]

4.4.2 Microservice developments

Figure 4.9 shows the microservices developed for the CDE in **Paper I**. The microservices include tools that provide statistics of the HVAC system, model validation of the HVAC system, and a tool for calculating the airflow demand of the ventilation system. The microservices are developed to prove that it is simple to develop and deploy microservices for the CDE. All the microservices for the CDE are developed in Flask, a tool allowing easy development and deployment of Application Programming Interface (API)s using Representational State Transfer (REST).

To illustrate the functionality of the microservice developments, an example model was developed, as shown in Figure 4.10. The example model contains three subsystems: One for heating, ventilation, and cooling. It also contains four thermal zones, which have a given airflow. The example model contains all types of subsystems that can be represented in an HVAC system.

For simplicity, the following paragraphs only provide examples of the HVAC statistics and airflow calculation microservices. The model validation microservice is used to validate the HVAC system in **Paper I** but is not described further in this Section.

HVAC statistics

Large HVAC system BIM models are often complex to manage, and it is hard to provide an overview of the different components in the HVAC system. The HVAC statistics microservices aim to provide the CDE with an overview of the different types of components that have been transferred from the HVAC BIM model. It also provides an overview of the BIM model's length of ducts

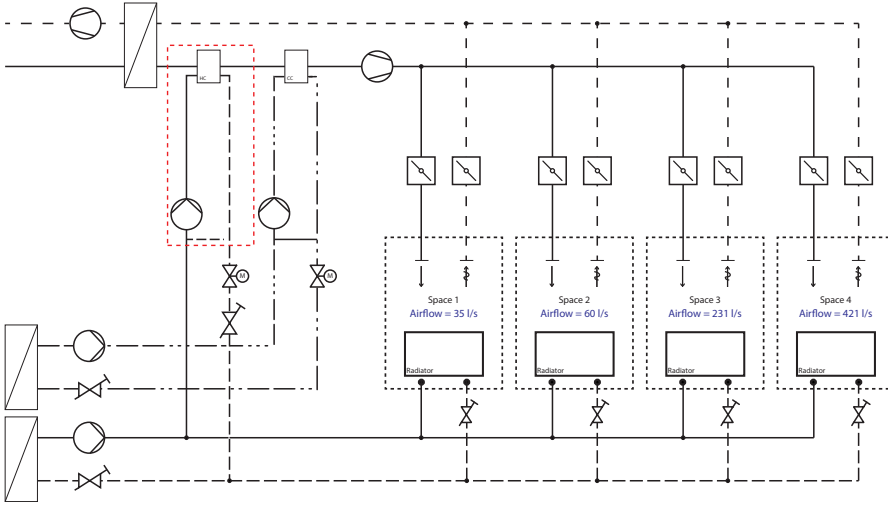


Figure 4.10. Illustration of the example model used to check the microservices. Illustration from **Paper I** [62].

and pipes, which can be used for quantity take-off.

The statistics microservice receives the FSC object model from the CDE as an HTTP post request, performs the calculation, and returns the result to the CDE. Table 4.2 shows the result from the initial BIM model and the FSC object model in the CDE. It shows that for most components, the amount counted is the same in Revit as in the CDE. It also illustrates that the *HeatExchanger* was counted four times in the CDE compared to two times in the Revit model. This is caused by the nature of FSC - it is counted twice more because heat exchangers are the interface between two sub-systems. Consequently, they are each serialized twice and provide a connection of sub-systems.

Airflow calculation

The *ConnectedWith* attribute of the FSC object model allows for traversing the HVAC system, as shown in Figure 4.11.

The recursive algorithm traverses a ventilation system described using the FSC object model, serialized in JSON. Since the algorithm is recursive, it can traverse the HVAC system components down to the *Fan* that supplies the air or to another desired stop condition. The algorithm was written to provide evidence that the data models developed in research tasks 1 and 2 can represent the semantic nature of any HVAC system.

Table 4.2. The Table illustrates the components reported in Revit and the components reported in the CDE. Table from **Paper I** [62].

Components	Amount Revit	Amount CDE
<i>AirTerminal</i>	8	8
<i>MotorizedDamper</i>	8	8
<i>Bend</i>	30	30
<i>Reduction</i>	40	40
<i>Tee</i>	14	14
<i>BalancingValve</i>	8	8
<i>MotorizedValve</i>	2	2
<i>HeatExchanger</i>	2	4
<i>Fan</i>	2	2
<i>Pump</i>	4	4
<i>Shunt Valve</i>	2	2
<i>PressureSensor</i>	2	2
<i>TemperatureSensor</i>	2	2
<i>Radiator</i>	4	4
<i>FlowSegment</i>	96	96
Total components	223	225

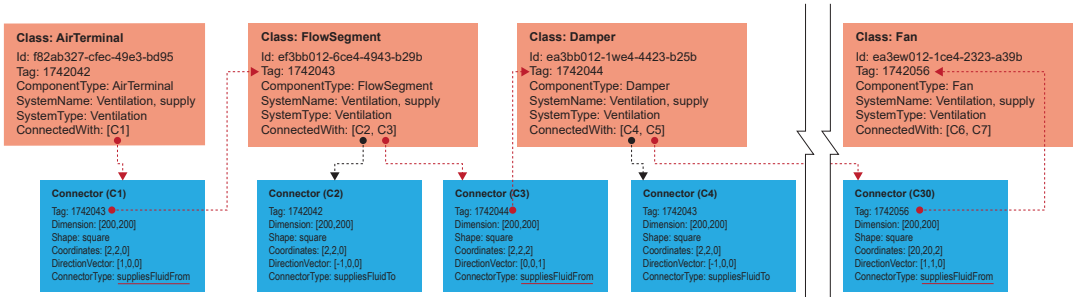


Figure 4.11. Illustration of an algorithm capable of traversing a ventilation system described using the FSC object model. From **Paper I** [62].

4.4.3 Key findings

The system architecture of the CDE proposed in **Paper I** provides a single source of truth using the FSC schema developed in **Paper I**. The CDE is created using a microservice architecture that allows for microservices development.

Several simple microservices are developed in **Paper I** to show the ease of development and deployment. Consequently, the CDE is easily scalable as more services are developed. This enables the development of technology-

agnostic microservices that containerize existing simulation tools.

4.5 Research task 4

The first iteration of the CDE system architecture, presented in **Paper I**, allows microservices to perform simple operations on the FSC object model. This research task aims to enable whole-building simulation with detailed HVAC simulation. Therefore, the system architecture was extended with additional microservices, as shown in Figure 4.12.

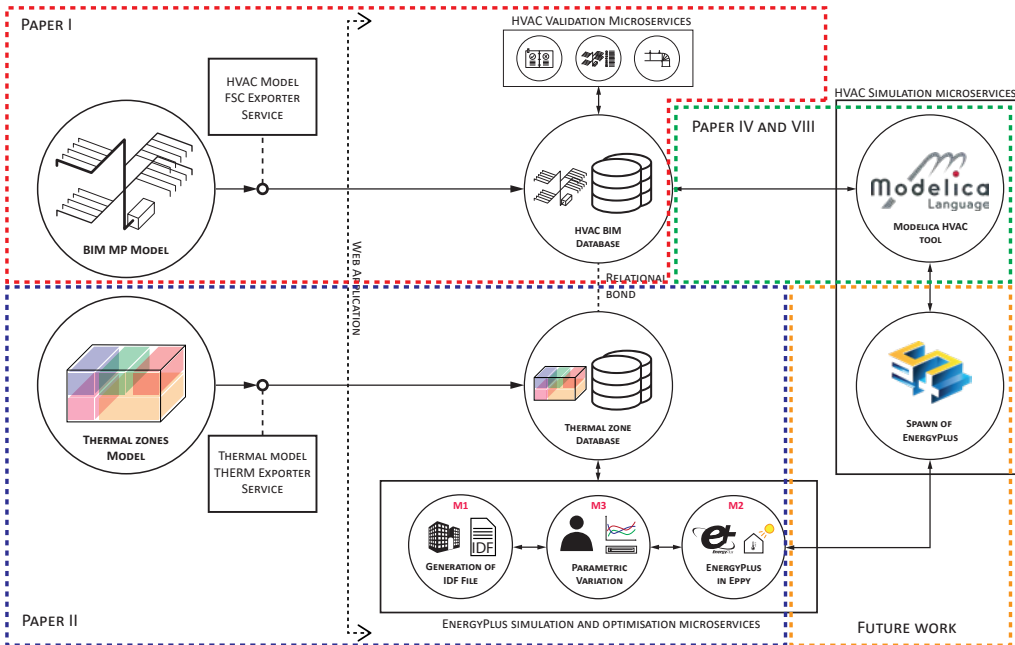


Figure 4.12. Illustration of the system architecture in **Paper II**, original Figure updated to show which papers feature the specific developments.

Paper I introduces the first iteration of the system architecture, shown in research task 3. The contributions of this research task aim to provide microservices in the CDE to enable whole-building simulation with detailed HVAC simulation to evaluate the performance gap from predicted to measured performance.

Paper IV and **Paper VIII** provided a parser that uses the FSC object model to generate a Modelica object model and is introduced in subsection 4.5.1. Consequently, **Paper IV** and **Paper VIII** contributes to improving

interoperability from BIM to detailed HVAC simulation and, therefore, allow isolated detailed HVAC simulation.

Paper II presents a system architecture capable of transferring the data from the proprietary BIM model in a CDE to perform whole-building simulations. The paper introduces several microservices capable of performing a whole-building simulation using E+, using the THERM object model introduced in research task 2. The microservices are described in detail in sub-section 4.5.2. Furthermore, the developed microservices are used to evaluate the performance gap of an existing school building.

Finally, **Paper VI** illustrates how an external application can use the microservices developed in **Paper II**. The application uses the developed ontologies FSO and FPO to perform a sizing of the HVAC system based on the results of a whole-building simulation, described in sub-section 4.5.2.

4.5.1 Modelica in a CDE

Paper VIII [63] conceptualize the first iteration of the FSC to Modelica parser. **Paper IV** introduces the second iteration of an automated toolchain that can perform a detailed HVAC simulation, using Modelica, in the CDE developed in **Paper I** and **Paper II**. Furthermore, the FSC to Modelica parser is created to simulate the HVAC system in a Modelica simulation environment. Using this parser, the FSC object model, which is serialized as a JSON, is translated into a .mo file, the input file used to simulate in the Modelica simulation environment Dymola. Figure 4.13 shows the mapping between the FSC and Modelica object models.

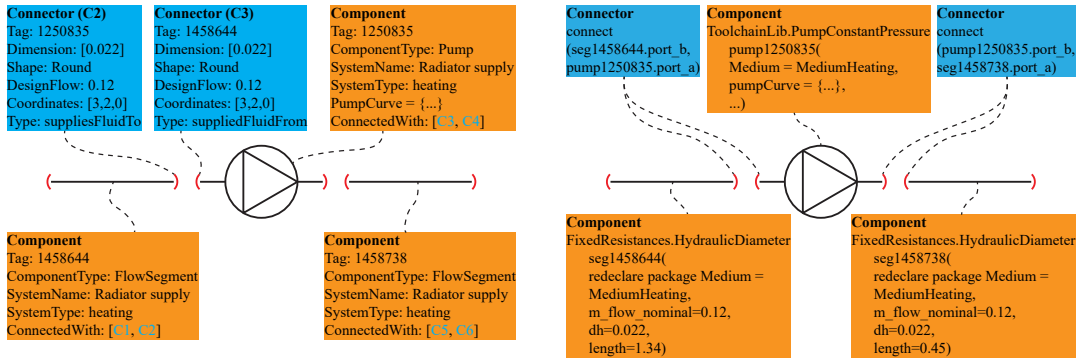


Figure 4.13. Illustration of three different components translated from FSC to Modelica, from **Paper IV** [64].

Detailed HVAC simulations in Modelica

To illustrate that the microservice developed to perform simulations in Mod-

elica works as intended, an example model was created, shown in Figure 4.14. The example model is modeled in 3D, using Autodesk Revit, and contains heating, ventilation, and cooling systems. The BIM model has all the components shown in Figure 4.14.

Simulating and comparing the effects of two different control strategies in a typical design situation in an HVAC system is challenging in a routine design situation. Few tools are capable of simulating the effects of different control strategies. **Paper IV** simulated two different weather compensation curves for the example model shown in Figure 4.14, during faulty and ideal⁴ operation.

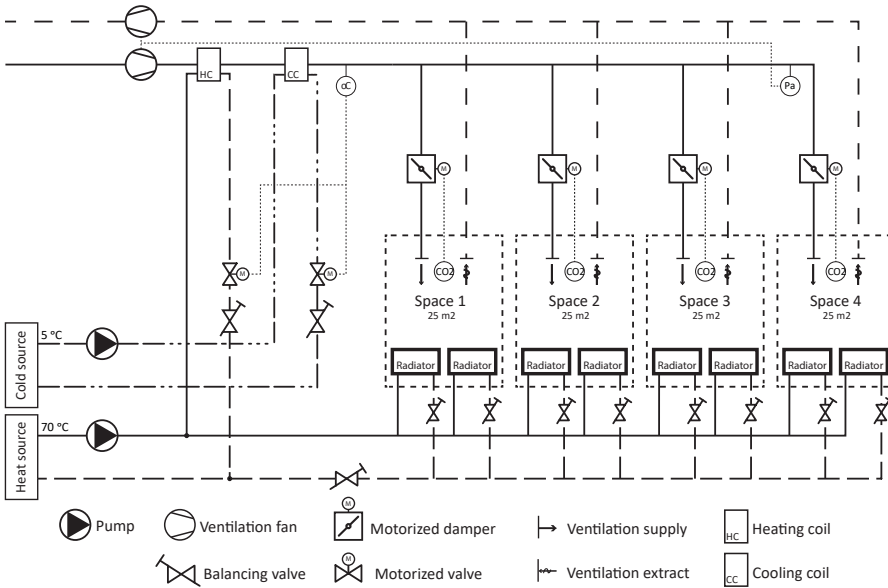


Figure 4.14. Schematic of 3D example model from Revit used to simulate HVAC system with different control strategies using Modelica, from **Paper IV** [64].

After simulating the example model in Dymola, the results showed the model in Figure 4.14 under faulty and ideal operation, as seen in Figure 4.15. In summary, the simulations showed that using a low-temperature heating curve reduces errors when operating a faulty heating system for the specific use case.

The simulation shown in Figure 4.15 represents a simple use case of the developed FSC to Modelica toolchain. It shows that the toolchain can be used to perform an isolated detailed HVAC simulation of a BIM HVAC model described in the FSC object model in a CDE. It contributes to HVAC engineers

⁴Ideal operation means that it operates as it is intended to. Not to be mistaken with *idealized*

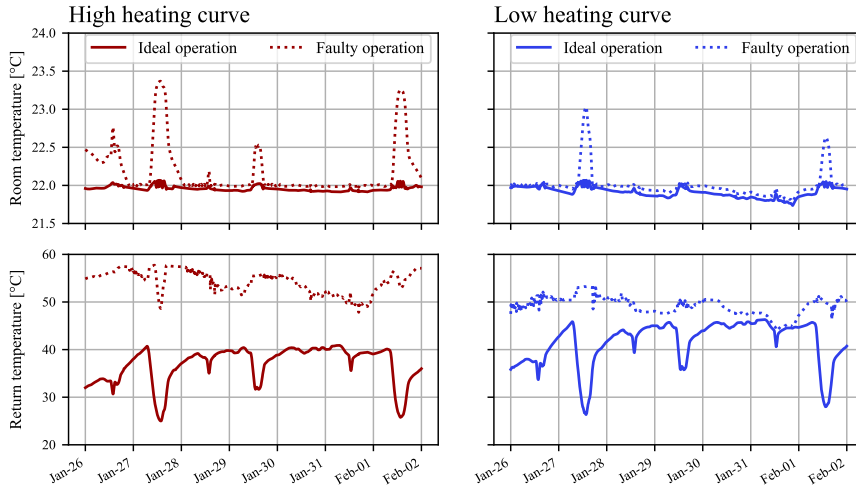


Figure 4.15. The left plots show the simulated temperature using a high-temperature heating curve for weather compensation in the room and heating system return. The right plots show the simulated temperature using a low-temperature heating curve for weather compensation in the room and heating system return. From **Paper IV** [64].

being able to use Modelica as a simulation tool for virtual prototyping of the HVAC system, which includes advanced system controls. It enables HVAC engineers to make decisions based on evidence rather than over-simplified HVAC calculations.

4.5.2 EnergyPlus in a CDE

Paper II introduced microservices for whole-building simulation in EnergyPlus (E+), as shown in the blue dotted box of Figure 4.12. The microservices consisted of three interdependent microservices called M1, M2, and M3. The microservices utilize the THERM object model developed earlier in **Paper II** (see Section 4.3.1) to perform a whole building simulation in E+.

Microservice M1 generates the native Input Data File (IDF), the input file to simulate in E+. The Input Data File (IDF) is generated using the THERM object model, which is stored as a JSON file in the CDE.

Microservice M2 is a container for the eppy⁵ Python package used to perform E+ simulations. Microservice M2 takes an IDF as input and, after running the E+ simulation, returns the E+ result files.

⁵<https://eppy.readthedocs.io/>

Microservice M3 creates a parameter variation study based on an existing IDF and the desired variation of the user. Figure 4.16 shows the process of running a variation study. For further technical details on the developed microservices, refer to GitHub⁶, or the detailed description in **Paper II**.

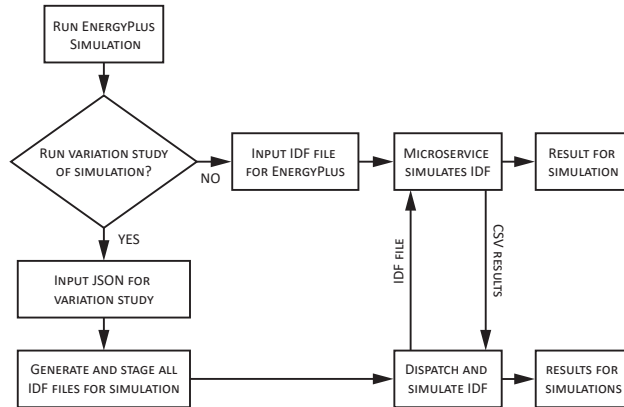


Figure 4.16. Process of running E+ simulations in the CDE, as shown in Figure 4.12. From **Paper II**.

Evaluating the performance gap

To evaluate the performance gap, **Paper II** includes a simulation using the three microservices and measurements from Frederiksberg school. Figure 4.17 shows the IDF generated from the Revit model in Figure 4.8.

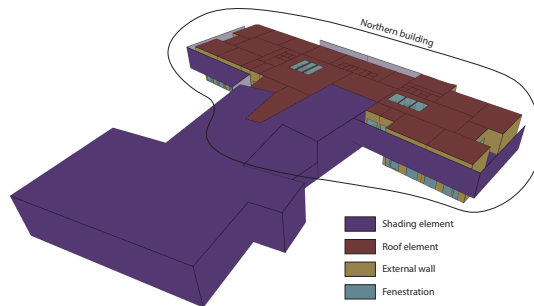


Figure 4.17. E+ IDF of Frederiksberg School northern building, visualized in OpenStudio. From **Paper II**.

To evaluate the performance gap, Frederiksberg school's measured heating energy use was compared to predicted heating energy use using the M1, M2,

⁶<https://github.com/Virtual-Commissioning>

and M3 microservices. Figure 4.18 shows the predicted and measured heating energy use for the north building, in orange and blue, respectively.

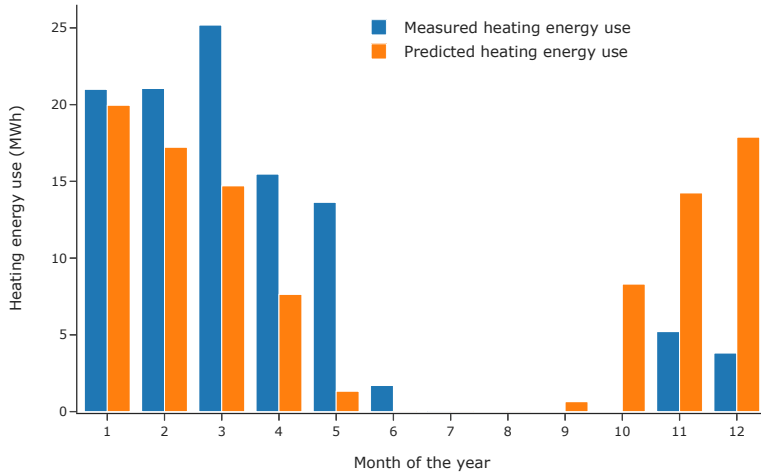


Figure 4.18. Chart showing Frederiksberg school’s predicted heating energy use (orange) compared to the measured heating energy use (blue). From **Paper II**.

Figure 4.18 shows an apparent monthly discrepancy from predicted to measured heating energy use, even though the E+ simulation model was simulated using the weather data from a weather station near Frederiksberg school. **Paper II** theorized that dynamic heating setpoints for the school might cause the discrepancy. Such a dynamic heating setpoint could include a nighttime setback and varying heating setpoints throughout the day. Therefore, microservice M3 was used to perform a parameter variation study, where the nighttime setback setpoint is 17°C and the daytime setpoint ranges, for the five simulations, between 20°C - 22°C with a step of 0.5°C. Figure 4.19 shows the results of the parameter variation study.

Unsurprisingly, the parameter variation study showed that the overall heating energy use decreases when the setpoint is lowered and a nighttime setback is introduced. Though it cannot explain the performance gap, it made it possible to evaluate the performance gap, providing a valuable tool that could potentially reduce the performance gap.

Comparing the predicted and measured performance of Frederiksberg School

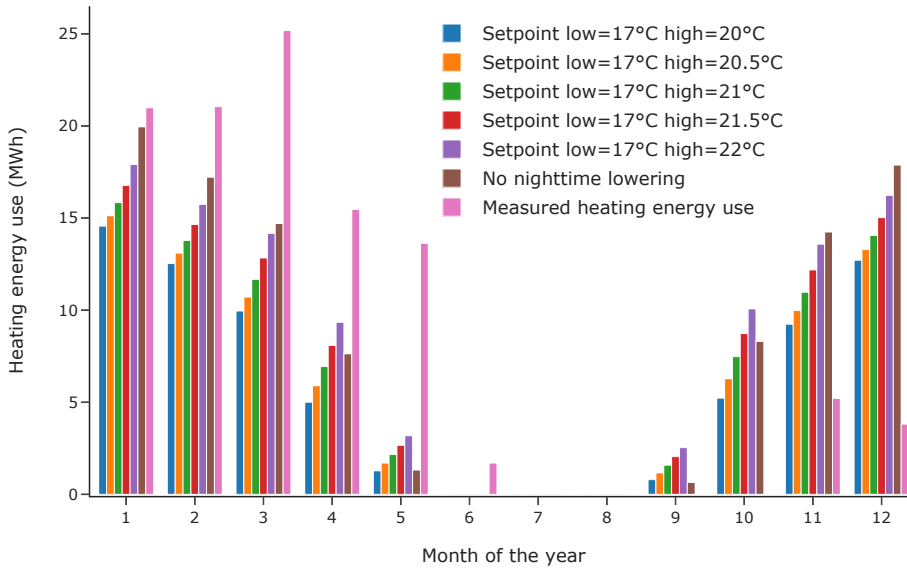


Figure 4.19. Chart showing the predicted heating energy use compared to Fred-eriksberg school’s measured heating energy use. From **Paper II**.

made it clear that the heating setpoint was needed. Specifically, it was necessary since the measurements were carried out during the COVID-19 pandemic. As a result, the government enforced periodical lockdowns on public institutions, including the educational system. Therefore, there is a significant risk that facility management of public institutions has changed the setpoints of HVAC systems, specifically the heating system, since the lockdowns were usually in the winter months.

Whole building simulation with dynamic hydraulic calculation

Paper VI shows an example where the E+ microservices M1, M2, and M3, FSO and its extension FPO were used to perform a whole-building simulation of an existing BIM model with thermal zones. The ventilation system was then sized dynamically based on the dynamic demands of the thermal zones.

Figure 4.20 shows the results of using a full BIM model to perform a whole-building simulation with a hydraulic calculation engine. One of the key find-

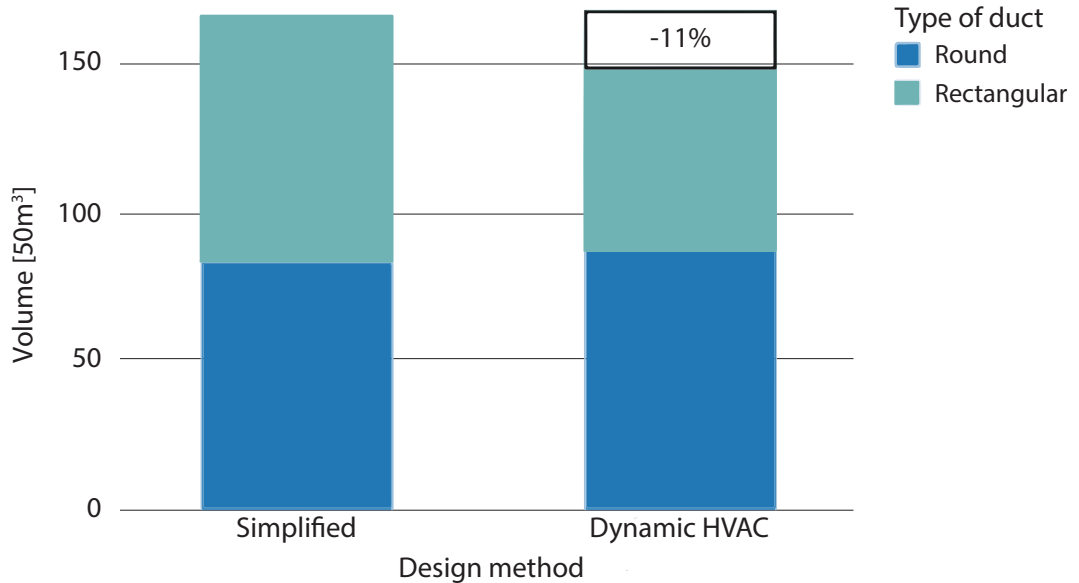


Figure 4.20. Chart showing the difference in using a simplified whole-building to perform a HVAC system sizing, or a full whole-building simulation to perform a dynamic HVAC system sizing. From **Paper VI**.

ings was that it was possible to save 11% on material usage by using this more accurate model rather than an oversimplified one. Furthermore, the full whole-building simulation model allowed for a total energy saving of 23% since it was possible to downsize the entire air-handling unit, compared to the simplified case.

4.5.3 Key findings

Modelica microservice

The development of the Modelica toolchain in **Paper VIII** and **Paper IV** makes simulations in a Modelica simulation environment available to the HVAC designer in the AECO industry. Using the CDE eases the burden of the HVAC design in creating simulations in a Modelica simulation environment by automatically generating the .mo files that go into Modelica. Making Modelica available for the HVAC designer makes it easier to perform detailed HVAC simulations with realistic controls.

The microservice developed to perform Modelica simulations in **Paper IV**, and **Paper VIII** proves that it is possible to perform detailed HVAC simulations using the information in a CDE. This tool is developed as a proof-of-concept, and the use case is simple to ensure that it can be simulated in a

Modelica simulation environment. To ensure that the microservice works for more advanced cases and to provide rigorous testing, future case studies on HVAC systems should be carried out using the Modelica toolchain.

EnergyPlus microservices

This research task shows that using a CDE as a basis for performing whole-building simulations in E+ automates parts of the simulation process for HVAC designers. It facilitates using the existing data in the single source of truth CDE instead of creating new data only expressed in the simulation model. The M1, M2, and M3 microservices enable whole-building simulation in a web-based CDE, using the THERM object model generated from a proprietary BIM model.

However, the whole-building simulation does not include the full HVAC system but an idealized version. With the introduction of a hydraulic simulation engine, it would be possible to simulate the entire building with the realistic HVAC system. Nevertheless, the developments of **Paper II** contribute to generating automated whole-building simulations using a CDE. It enables the HVAC engineer to generate extensive whole-building simulations that use information already available in the BIM model rather than geometrically simplified models. It contributes to increasing the interoperability of whole-building simulation and closing the information gap.

Paper VI shows that it is possible to use the microservices developed for whole-building simulation in E+ to perform a dynamic sizing of the HVAC system rather than basing the sizes on the peak demand of the thermal zones. This tool could be helpful in the existing design process and will help will sizing the ventilation systems according to their actual demand.

This research task introduces a process for comparing an existing building's predicted performance with the measured performance. By comparing predicted and measured performance, it is possible to start investigating scenarios on why the performance gap exists, which has the potential to help HVAC engineer to reduce the performance gap.

A Spawn microservice

The future development of the CDE presented in 4.4 and 4.5 involves connecting the state-of-the-art tool called Spawn-of-EnergyPlus (Spawn). Spawn⁷ is developed as an Functional Mock-up Unit (FMU) in the Functional Mock-up Interface (FMI), which is based on the Modelica language. This enables a whole-building co-simulation with the HVAC controls from the *Buildings*⁸ library. Providing a connection for Spawn is part of this thesis's future outlook for the CDE. See the future outlook Section 5.2.6 for more information.

⁷<https://lbl-srg.github.io/soep/>

⁸<https://simulationresearch.lbl.gov/modelica/>

CHAPTER 5

Discussion & Conclusion

This Chapter discusses and summarizes the research findings.

Section 5.1 revisits the research questions posed in the research design and are answered in the related sub-sections.

Section 5.2 evaluates the contribution made to academia and the AECO industry. It also provides a future outlook.

Finally, Section 5.3 concludes on the findings in this thesis.

5.1 Research questions revisited

This Section revisits the research questions (RQs) posed in Chapter 3 and answers them based on the research tasks carried out in Chapter 4. After revisiting each of the RQs, the following Section concludes on the contributions made to academia and industry in this PhD thesis and provides a future outlook. Finally, the Section provides the concluding remarks for the thesis.

5.1.1 RQ1: Replacing file-based BIM with a CDE?

Can a CDE serve as a single source of truth, replacing file-based BIM, and consequently close the information gap in the building design process?

Research tasks 3 and 4 are working proofs-of-concept for a CDE that provides a single source of truth through an object-oriented database. The database was updated with information extracted from the proprietary BIM software Autodesk Revit through the data models created in sections 4.2 and 4.3. Therefore, the answer to research question 1 is yes; a CDE can serve as a single source of truth, replace file-based BIM, and consequently eliminate the information gap. However, it should be clear that industry adoption takes much more effort than simply introducing a new CDE. Full implementation of the CDE involves many hours of development but also a substantial effort in educating the workforce to use it. One well-known issue in AECO industry is that new tools are notoriously hard to implement, especially if it needs to be widely adopted.

The background Chapter introduced two CDEs. They have both seen limited industry adoption. The reasons for the low industry adoption are discussed in Section 2.2. The CDE developed in this thesis is different since it is open-source but based on discipline-specific object models rather than complex IFC files or proprietary file formats. Using discipline-specific object models to represent each discipline's single source of truth makes it easier to work with them as developers since they are directed at the specific discipline of each stakeholder. For instance, the FSC object model stored in the CDE provides only the single source of truth for the HVAC system and does not involve properties related to other disciplines such as architecture, structures, etc. This makes the CDE introduced in this thesis easier to implement into an existing organization since limited information is needed to populate the database.

Research tasks 3 and 4 show that introducing a CDE based on a microservice architecture makes it possible to scale the CDE with the needed functionalities as new demands arise for the capabilities in simulation/calculation tools. For instance, research task 3 introduces very simple microservices capable of

performing calculations and checks on the FSC object model. This represents the initial proof-of-concept within a company. Research task 4 introduced several microservices, one of which can perform a whole-building simulation using the E+ simulation engine. This type of scalable development is hard to do with other CDEs today, like the BIMServer.org project - since the file structure is based on IFC, every aspect of the project needs to be well-defined to have a model that lives up to the model integrity. Moreover, most proprietary tools serialize IFC files poorly from their internal object model, meaning they cannot represent a single source of truth.

One problem with using a single source of truth CDE is that it requires a higher level of detail when modeling in a BIM modeling tool, such as Autodesk Revit. The data extracted from Revit in FSO, FSC, and THERM represents the 'current state' of the BIM model. If the current state of the BIM model is not very detailed, that will be transferred to the database in the CDE. Since data exists in several places, this problem persists as long as AECO stakeholders model in proprietary tools like Revit. However, the CDE centralizes information in one place. Thus, it eliminates the information gap between AECO stakeholders. Simulations are based on data in the CDE, and the results are stored in the CDE. As a result, the HVAC engineer no longer has to worry about tracking the specified amount of people in the room manually from the architectural BIM model or the room schedule provided by the architect from 5 weeks ago. In the CDE, through data model validation tools and advanced model tracking, this information exists in only one place, making it the single source of truth for all project stakeholders.

5.1.2 RQ2: A CDE for seamless data integration?

Can a Common Data Environment (CDE) enable seamless data integration between proprietary and open data formats?

The CDE created in research tasks 3 and 4 facilitates a seamless data integration between the proprietary BIM tool Revit and the open data formats developed in research tasks 1 and 2. To achieve seamless data integration, exporters were created in the Autodesk Revit Application Programming Interface (API) to serialize a JSON based on the object models created in research tasks 1 and 2. Through research tasks 3 and 4, more than 7 microservices were developed, all based on the different serializations from RDF or JSON. This establishes that it is possible to extract data from a proprietary BIM tool into an open data format and use other tools to use the open data in the CDE.

One problem with working with a proprietary BIM tool like Autodesk Revit is that stakeholders model in that tool, but the single source of truth exists

in the web-based CDE. This means that there is a chance that information gets lost in translation from one object model to another. This problem will likely persist until the AECO industry starts modeling directly in the CDE. While some parts of the industry consider this state-of-the-art, it will likely present significant challenges and might eventually mean that discipline-specific object models converge towards a super-schema, like IFC. Therefore, in this thesis, proprietary BIM tools like Autodesk Revit are considered a microservice used to create and edit geometrically rich BIM models.

5.1.3 RQ3: A data model for flow systems

Is it possible to formulate and implement a discipline-specific data model to represent (HVAC) flow systems and their properties in a web-based CDE?

In research task 1, an OWL called Flow Systems Ontology (FSO) was created with an extension called Flow Properties Ontology (FPO). The papers **Paper III**, **Paper V**, and **Paper X** tested the feasibility of FSO to confirm that it was indeed capable of representing even complex flow systems. It was tested using SPARQL Protocol and RDF Query Language (SPARQL) queries on an example model created manually in a Terse RDF Triple Language (Turtle) file. Furthermore, it was tested on an example model generated using the Autodesk Revit API and the FSO exporter, which is provided in **Paper V**. The exporter serializes the HVAC BIM model into the Resource Description Framework (RDF)-based FSO in a Turtle format, which consisted of more than 6137 HVAC components.

After creating FSO to represent flow systems, it was apparent that RDF graphs are challenging to use in a web application, even with the existing serialization engine in JSON-LD. It is based on the RDF language, but its structure is flat and generally not the most human-readable, as it uses URIs to describe relationships in the same file. This makes it hard to use for developers used to the flexibility in the unstructured JSON files.

Therefore, an object model called Flow System Classes (FSC) was created in research task 1. The FSC object model represents flow systems more hierarchical, rather than in a flat RDF-based manner. By doing this, it is possible to serialize directly to JSON from the Autodesk Revit BIM model, using the C# API. **Paper I** shows that FSC can be used to represent several complex HVAC systems. This is proven through the use of a complex Autodesk Revit HVAC BIM model that consisted of 225 different HVAC components. To test the integrity of the HVAC model exported from Revit to FSC, several microservices were created in research task 3, capable of testing the connectivity of the HVAC system, the HVAC system statistics, and an airflow sizing tool.

Chapter 2 describes that one of the most obvious drawbacks of OWL is that they are complex to serialize in a web-ready format and can be hard to work with for inexperienced developers. However, OWL ontologies efficiently describe HVAC systems. The subject-predicate-object nature of OWL is intuitive when describing the semantic nature of flow systems. Furthermore, it provides a data model based on a stable and standardized schema.

A drawback of object models is that they can become massive nested files that are inefficient. Furthermore, JSON is an unstructured format, meaning that serializing object models in JSON is prone to formatting errors. This also makes object models flexible compared to OWL ontologies.

Research tasks 1 and 2 present two methods to represent HVAC systems: an OWL and an object model. Both can represent complex HVAC systems with benefits and drawbacks. This thesis does not aim to provide a "final" answer on whether OWL ontologies or simple object models are best. Both have benefits and drawbacks and can be used in a CDE. It should be stressed that object models can be the basis of creating an ontology, so in the end, it comes down to which choice of serialization is desired for the specific developer. As long as the object models developed for the HVAC discipline and the AECO industry can represent the properties of the given domain, it does not matter which serialization is used.

5.1.4 RQ4: A data model for thermal zones

Is it possible to formulate and implement a discipline-specific data model for the representation of thermal zones and their properties in a web-based CDE?

Research task 2 creates a discipline-specific data model to represent thermal zones and their properties, called Thermal Zone Classes (THERM). To exemplify that THERM can represent thermal zones and their properties, the use-case was created in **Paper II**. In this paper, THERM was serialized as a JSON file from an Autodesk Revit BIM model. The file contained a total of 84 zones. After transferring the file into the THERM structured JSON, an Input Data File (IDF) for EnergyPlus (E+) was generated and successfully simulated.

The THERM object model was created to perform whole-building simulations in a CDE using E+. Thus, the derived structure of the THERM object model is based on the information needed to perform E+ simulations. Most whole-building simulation tools base their simulation engines on different boundary conditions. Therefore, it might be that the THERM object model in its current state is incapable of providing all the information to run a whole-building simulation in other simulation tools, like IESVE, IDA ICE, etc.

The THERM object model provides a simple geometric model. It merely contains the geometric information needed to represent the thermal zones for a whole-building simulation. If the thermal zone modeled in the BIM tool is complex, like a slanted ceiling or walls, THERM displays a simplified version of that. Furthermore, the architectural models on which thermal zones are usually based are created by architects. This can cause issues since the architect is not checking to see if the walls are facing the right direction, if the wall, ceiling, or floor layers have the right u-value, and so on. This problem can be solved by providing algorithms to check the model integrity of the BIM model and prompt the user in the frontend with any errors that could hinder or cause incorrect simulation.

It can be concluded that it is possible to create a discipline-specific data model to represent thermal zones and their properties.

5.1.5 RQ5: A microservice CDE for advanced building simulation

Does a technology-agnostic CDE based on a microservice architecture facilitate the digital transformation by providing a platform for organic additions of automated design services in the building design process?

HVAC engineers today often design HVAC systems based on rule-of-thumb or through idealized models for hydraulic simulation. There are several reasons for this, but one contributing factor is that acquiring the information to perform whole-building simulation and detailed HVAC simulation is manual and laborious.

Research tasks 3 and 4 show that by centralizing all the project data into a single source of truth CDE based on discipline-specific data models, it is possible to build microservices around that data to perform previously manual operations automatically. Furthermore, research task 4 extends the work of CDE created in research task 3 and presented microservices capable of performing whole-building simulations in E+ and detailed HVAC simulation in a Modelica simulation environment based on the THERM and FSC object models created in research tasks 1 and 2.

Research task 4 (**Paper II** and **Paper IV**) shows that it is possible to create an automatic toolchain that transfers information from a proprietary BIM format into a whole-building or detailed HVAC simulation environment, through a CDE. This process was previously limited to parsers translating directly from proprietary BIM tools like Autodesk Revit into the simulation engine. This creates a "black box" in the middle of the toolchain - the user that exports the data cannot check what data is being transferred to the simulation

engine and therefore has poor control of the assumptions that the simulation model is based on.

By introducing a CDE as the data mediator, the user can inspect data before translation into the whole-building simulation or detailed HVAC simulation tool. It also provides the user with a place to store the results of a given simulation iteration in the CDE. This provides a way of tracking different simulation iterations of the building's predicted performance.

One of the main issues with using a whole-building simulation tool such as E+ is that it automatically assumes idealized HVAC systems, leading to oversimplification of the simulation model, another reason for the performance gap, listed in Section 1.3. There are ways of simulating a simplified (not idealized) version of the HVAC system in the E+ simulation engine. They are, however, still very simplified and may lead to errors by oversimplification.

Paper IV introduces a microservice for detailed HVAC simulation in an Modelica simulation environment, using the FSC object model. Specifically, a parser was created to serialize the JSON based on the FSC object model to generate .mo files for simulation in a Modelica simulation environment. To exemplify that the microservice parses from the FSC object model into the Modelica object model, a simple 3D model was created in Autodesk Revit. The parsing was successful; the simulation was run, and a result was returned from the microservice to the CDE.

Modelica is a language that provides the capability of state-of-the-art hydraulic simulations. In contrast to most whole-building simulation tools, Modelica can simulate realistic rather than idealized HVAC systems. **Paper IV** provided limitations to the developed microservice. Specifically, one of the significant challenges to simulating HVAC systems correctly is that FSC does not contain a detailed description for HVAC controls. To overcome this challenge, FSC was extended to have simple support for PI controllers for valves, dampers, fans, pumps, etc. This addition permits running a use-case simulation. However, HVAC controls are rarely as simple as those described in **Paper IV**. Hence, future work should include a data model that supports a detailed control specification.

FSC and THERM, introduced in research tasks 1 and 2, provide the stepping stone to overcome the final barrier to performing a full simulation that includes a whole-building simulation with detailed HVAC simulation. During this thesis, Spawn-of-EnergyPlus (Spawn) was released as part of the Modelica buildings library¹. The object models for thermal zones (THERM) and HVAC systems (FSC) can be used for simulation in Spawn.

¹<https://www.energy.gov/eere/buildings/articles/its-alive-after-five-years-lab-spawn-energyplus-finally-here>

With the possibility to run a whole-building simulation and detailed HVAC simulation based on the BIM model in the CDE, the requirements for a high level of detail model increase. This challenge has been raised several times by the PhD candidate's host company. This means that designers must model HVAC systems more correctly. Since the introduction of clash detection for BIM models, the level of detail has been discussed vividly in the AECO industry. From an HVAC perspective, usually, designers are not interested in modeling every detail of HVAC system, like valves, dampers, terminals, etc. This presents a challenge in moving to detailed HVAC simulation. Since the details do not exist in the BIM model, they cannot be transferred to the CDE. However, providing a method for automatic detailed HVAC simulation incentivizes designers to model more accurately in the BIM model. Detailed HVAC simulation allows designers to simulate the building as close to reality as possible.

Tools for whole-building simulation and detailed HVAC simulation have existed for decades. Several research projects (*IFC2Modelica*, *BIM2Modelica*, *Revit2Modelica*) have created tools that can increase interoperability by using information available in the existing BIM model to perform whole-building simulation or detailed HVAC simulation. These projects all work file-based, meaning there is no single source of truth for that information. This thesis introduces a CDE as the tool to increase automation by closing the information gap and increasing transparency for all project stakeholders. Using a CDE based on a microservice architecture, this thesis shows that it is possible to add microservices along the way of the project development.

It will take an effort to transcend digitalization and move into the digital transformation of the AECO industry. The CDE introduced in this thesis provides a vessel to digitally transform the AECO industry, one microservice at a time. It increases automation by providing a platform where all automation efforts reside, based on single source of truth discipline-specific data models. In conclusion, the CDE facilitates the digital transformation and, in this thesis, increases the level of automation for whole-building simulation and detailed HVAC simulation.

5.1.6 RQ6: Can a CDE reduce the performance gap?

Is a microservice CDE for whole-building simulation with detailed HVAC simulation a tool that can reduce the performance gap in a practical HVAC design setting?

A CDE used with detailed HVAC simulation can potentially reduce the performance gap from predicted to measured performance. **Papers II** and **IV** shows that the CDE can perform whole-building simulations and detailed

HVAC simulation of existing buildings.

Specifically, **Paper II** shows that it is possible to compare the performance of an existing building with the predicted performance of that building, using an E+ model generated based on the data available in the web-based CDE. **Paper II** provides a proof-of-concept for a method to reduce the performance gap. Still, it should be clear that the whole-building simulation tool in the CDE does not automatically reduce the performance gap.

The CDE provides the means to perform a post-occupancy evaluation after building construction. In this post-occupancy evaluation, the HVAC engineer can compare the prediction of a simulation model based on the as-built BIM model with the measured data from the actual building. One of the primary sources of error when evaluating the performance gap is the sensors of the building. Ensuring that a sensor is calibrated correctly can be extremely hard. Secondly, building sensors and meters collect a lot of data. This can make comparing simulation results with measured results difficult, primarily if the measured results are poorly structured or if the boundary conditions of the actual building are not known. **Paper II** shows an example of this - several hours were spent going through excel spreadsheets to gather relevant information to perform a whole-building simulation of the building.

To solve the problem of fragmented information, the designers of the building should provide a strategy for using data from sensors and meters in the building. For instance, **Paper II** shows that use-case schools did not log room heating and airflow operating setpoints. Consequently, replicating the boundary conditions for the control algorithms in the E+ whole-building simulation was next to impossible. When room temperatures dropped in the rooms during the period of the measured data, no setpoints were logged. Furthermore, most building data in existing buildings are unavailable to the building owner. They are logged in proprietary building management systems and can only be extracted with the help of the building management system vendor, usually at a monetary cost.

One main contributing factor to the performance gap when performing whole-building simulations is that the modeling tools are over-simplified and contain wrongful assumptions about the design due to the information gap between stakeholders. After introducing a whole-building simulation and detailed HVAC simulation tools into the CDE, the simulations are performed using the boundary conditions used in the BIM model. This contrasts with the current design process, where a few representative rooms are often modeled, sized, and extrapolated onto the rest of the building. The same goes for HVAC system, where the heat load of a few rooms in a peak condition is used to size the ducts in a static calculation.

Therefore, it can be concluded that the whole-building simulation and detailed HVAC simulation tools introduced in **Paper II** and **Paper IV** on the CDE have the potential to help HVAC designers in reducing the performance gap in a practical setting.

5.2 Contributions

This Section summarizes the main contributions made to academia and industry throughout this PhD thesis. The main contributions consist of the proposed system architecture for a CDE, the Flow System Classes (FSC), the Thermal Zone Classes (THERM), the Flow Systems Ontology (FSO), and the microservices capable of performing whole-building simulation and detailed HVAC simulation. This Section discusses these contributions and their implication for academia and industry. Finally, it provides a future outlook on the contributions.

5.2.1 A system architecture for microservice-based CDE

The main contribution of this thesis is the CDE system architecture that was proposed in **Paper I** and **Paper II**. The system architecture based on microservices enables developers to create microservices independent of each other, which use the object models (FSO, FSC, THERM) within the single source of truth CDE. This allows for developing a network of microservices that all provide different functionalities. In **Paper I**, this was used to create simple standalone microservices to perform rule-checks on the HVAC system. In **Paper II** and **Paper IV**, this amounted to containerized microservices for whole-building simulation through E+ and for detailed HVAC simulation through a Modelica simulation environment. Furthermore, **Paper II** showed that by using the CDE together with measurements of the heating energy use from a school, it is possible to predict the heating energy use of the school and compare it with the measured heating energy use. Thus, the CDE enables HVAC designers to perform whole-building simulation and detailed HVAC simulation of buildings and compare them to the actual building performance - an essential step in evaluating and, eventually, reducing the performance gap between predicted and measured building performance.

Most companies in the AECO industry focus on developing tools created as add-ins with proprietary BIM tools, such as Autodesk Revit. This approach may seem simple and profitable, but it has several issues. One of them is that it produces a lot of tools that work based on different object models. Some tools work directly with the object model in Revit; others export an object model in the correct format for a specific third-party tool, like IESVE. Consequently, work processes become file-based, as shown in Figure 1.2. In

the ever-evolving design, construction, and operation phases of a building, these files represent different truths, making them incompatible and incomparable. By introducing a CDE where a database represents the single source of truth for a discipline-specific object model, it is now possible to derive all data needed from the common data formats within the CDE.

All developments related to the microservice architecture based CDE is publicly available on GitHub².

5.2.2 Flow System Ontology

Paper III introduces the Flow Systems Ontology (FSO), which is a discipline-specific Web Ontology Language (OWL) to represent flow systems' energy and mass transfer.

Since the purpose of FSO was to create a lightweight ontology that can be extended into different domains, there was a need for an extension to describe the FSO components' capacity and size-related properties. **Paper V** introduces Flow Properties Ontology (FPO), an extension to FSO, to describe components' capacity and size-related properties. **Paper V** illustrates, that by extending FSO with FPO, it is possible to perform advanced flow calculations on HVAC systems in a CDE. Furthermore, **Paper VI** shows that the ontology can be coupled with whole-building simulations in E+ to perform dynamic HVAC calculations that use the results from the whole-building simulation to size the HVAC system.

To allow the AECO industry and academia to apply FSO and FPO, a tool capable of parsing an Autodesk Revit BIM model into FSO and FPO and serializing it into an RDF-based Turtle file was created. The tool is publicly available, in **Paper V** through GitHub³, as an open-source development open to outside contributions. FSO is being used continuously by the authors of **Paper III**.

5.2.3 Flow System Classes

Following the original development of FSO, from **Paper III**, there was a need for an object model that represents flow systems and their components. Therefore, FSC was created in **Paper I**, which represents entire HVAC systems containing heating, cooling, and ventilation system and their components. Furthermore, FSC was extended in **Paper IV** to include simple control specifications to be able to translate the FSC object model into a Modelica object model.

²<https://github.com/Virtual-Commissioning>

³<https://github.com/Semantic-HVAC-Tool/Parser>

FSC is the discipline-specific data model used in the CDE for the representation of HVAC systems. FSC makes out the data model on which most microservices related to flow systems are based. Therefore, it is an integral part of the CDE. The FSC object model is publicly available on GitHub⁴, open to outside contributions.

The FSC object model will be continuously used by the host company of the PhD candidate in the development of the CDE. Furthermore, parallel research projects currently use the FSC object model to create detailed HVAC simulations with high-detail control specifications in Modelica. These developments will be published after the submission of this thesis.

5.2.4 Thermal Zone Classes

Paper IV exposed the need for an object model to represent thermal zones for whole-building simulations. Therefore, we developed the Thermal Zone Classes (THERM), which is a hierarchical description of the thermal zones in the building and their related properties, such as thermal loads, geometry, HVAC conditions, etc.

THERM makes out the data model for thermal zones on which all microservices related to whole-building simulations are based. Therefore, it is an integral part of the CDE. The THERM object model is publicly available on GitHub⁵, open to outside contributions.

The host company has started the development of a database to store all room property information in a CDE to perform whole-building simulations. It uses the THERM object model to allow architects and engineers to access room information through a web application frontend. Furthermore, THERM provides the stepping stone for combining whole-building simulation in E+ with detailed HVAC simulation in a Modelica simulation environment. Therefore, recently started research projects are using the THERM and FSC object models to run simulations in Modelica using Spawn-of-EnergyPlus (Spawn), see Section 5.2.6.

5.2.5 Microservices

The CDE is based on a microservice architecture, which uses the data models developed (FSC, and THERM) to validate, calculate, and simulate the building performance. During this thesis, several microservices were developed to extend the capabilities of the CDE. The developed microservices include tools to validate the HVAC model, calculate the airflow in a ventilation system in

⁴<https://github.com/Virtual-Commissioning/VC-HVACExporter-Service>

⁵https://github.com/Virtual-Commissioning/VC-Analytical_zones_exporter-Service

Paper I, perform whole-building simulation, and detailed HVAC simulation. The following paragraphs describe the contribution of each microservice.

A microservice to perform whole-building simulations in E+ was developed as part of **Paper II**⁶. It uses the THERM object model to automatically generate IDF files to be simulated in E+. This microservice provides an easy-to-deploy solution to perform whole-building simulations in E+.

Furthermore, a microservice to perform detailed HVAC simulation of HVAC systems in a Modelica simulation environment⁷ was created. It uses the FSC object model to automatically generate .mo files that can be simulated in a Modelica simulation environment. This microservice provides a state-of-the-art toolchain for the automatic simulation of HVAC systems modeled initially in BIM to be simulated in Modelica.

All of the microservices developed for the CDE illustrate that a microservice architecture allows for an easily scalable CDE. Furthermore, the microservices are all available to outside developers, meaning they do not have to be accessed through a CDE. Therefore, they can be used through plugins in proprietary tools, such as Autodesk Revit, while the AECO industry matures to the idea of CDEs replacing file-based BIM.

5.2.6 Future outlook

Common Data Environments in the industry

Implementing a microservice-based CDE into the daily work process of the AECO industry will be an extensive task for years to come. Most of the industry agrees that CDEs form the future collaboration on BIM models. To implement CDEs in AECO industry, there needs to be a change in how the industry develops applications. Specifically, it means that AECO organizations must transcend digitalization into digital transformation.

The host company of the author of this thesis has started the development of a CDE using the system architecture proposed in this thesis to use for its design, construction, and operation projects. Some benefits that the host company recognizes in the CDE are:

1. The BIM model information is available to all stakeholders of the project
2. When collecting data from buildings, information is stored in the same place. Consequently, sensor and meter data can be used later in big data analysis

⁶https://github.com/Virtual-Commissioning/VC_EnergyPlus-Simulation_Service

⁷<https://github.com/Virtual-Commissioning/VC-Modelica-Service>

3. Tools that the company develops for analysis and simulations are gathered in one platform, as opposed to being dispersed across excel spreadsheets, python scripts, etc.

These benefits drive the development of the CDE within the host company and provide an incentive to replace file-based BIM with the CDE.

The literature review in the background Chapter made it clear that CDEs have been developed before based on different data models, open or proprietary. Currently, there are commercial products for the centralization of BIM models in CDEs. Still, the AECO industry has seen little adoption of these commercial products. This is caused by the lack of external tools, such as simulation engines, that can be implemented into the CDE. If commercial simulation tools enable access to their simulation engines through an API rather than through their desktop application, using a CDE would be easier to justify. Currently, when stakeholders put their BIM models into a CDE, none use the existing information to populate simulation models for detailed HVAC simulation, whole-building simulation, structural analysis, etc.

One challenge the industry faces with the implementation of CDEs is the transparency that data availability provides. It sets a high standard for building modeling and trust between the stakeholders. For instance, it requires the HVAC engineer to model the entire HVAC system. Otherwise, the single source of truth does not reflect reality. It also requires that all project data is kept up-to-date throughout the entire building life-cycle - which rarely happens in today's design process in AECO industry. Therefore, future work should include a framework for implementing microservice-based CDEs into the design process of buildings in the AECO industry.

The ontologies and object models developed in this thesis provide a vocabulary for describing some parts of the HVAC discipline in the AECO industry. It is, however, clear that they should be developed further to be used for other disciplines related to the HVAC discipline, like the MEP discipline, which includes water systems, electrical systems, gas, etc.

Paper II used historical data extracted using an online building management system. The data were manually compared to the predicted results of a whole-building simulation. To enable HVAC engineers to access building data through the CDE, building management system vendors should focus on developing an API. By creating API access to this data, the building owner can request that HVAC engineers perform regular follow-ups on the building performance, reducing the performance gap with the tools developed in this thesis.

The CDE has the potential to be used for all disciplines of the AECO indus-

try, including structures and architecture. Specifically, research projects have been started during the PhD to include object models to represent structures and architecture so that the CDE can perform a life-cycle assessment of the entire building.

Whole-building and Detailed HVAC Simulations

This thesis has developed the basis for other researchers to apply the FSC and THERM object models to perform whole-building simulations with a realistic HVAC system. In future work, the CDE and object models created in the course of this PhD should be used to enable industry practitioners to simulate buildings as a near-perfect white-box model in open-source tools, such as Spawn-of-EnergyPlus (Spawn). This means that the modeling precision of the HVAC engineers must be substantially increased. With the inclusion of such advanced simulation models, there is an incentive to increase the level of detail in the BIM models.

5.3 Concluding remarks

The CDE developed in this PhD thesis provides a framework for a new way of developing applications for the AECO industry. It catalyzes digital transformation in the AECO industry by closing the information gap, replacing the file-based approach with a model-centric approach.

The CDE can provide a single source of truth for the project data, meaning that all data is stored within a database in the CDE. Even so, it does not provide a complete solution that the industry can use from day one. Many developments are in progress, so different people work in different directions. Consequently, it results in using different technologies for tools trying to solve the same problem. Therefore, the created CDE is technology-agnostic, meaning it does not depend on one technology. This is shown through the use of data models that are RDF-based and JSON-based in the CDE. Furthermore, the CDE also contains numerous programming languages, as seen in the microservices based on C#, Flask, and FastAPI. Furthermore, the proof-of-concept parser can translate a 3D Revit model into the data models created in this thesis. This can also be done for 3D models in IFC, for instance, using IfcOpenShell⁸.

⁸<https://ifcopenshell.org/>

CHAPTER 6

Papers

This Chapter includes the papers written during the course of this PhD project.

6.1 Paper I - A common data environment for HVAC design and engineering



Contents lists available at ScienceDirect

Automation in Construction

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

A common data environment for HVAC design and engineering

Mikki Seidenschur^{a,c,*}, Ali Küçükavcı^a, Esben Visby Fjerbæk^a, Kevin Michael Smith^a, Pieter Pauwels^b, Christian Anker Hviid^a^a Department of Civil and Mechanical Engineering, Technical University of Denmark, Copenhagen, Denmark^b Department of Civil Engineering, Technical University of Eindhoven, Eindhoven, Netherlands^c Ramboll, Copenhagen, Denmark

ARTICLE INFO

Keywords:

Building information modeling
HVAC
Object models
Common data environment
BIM level 3

ABSTRACT

The Architecture, Engineering, and Construction (AEC) industry is transitioning toward using cloud-based Common Data Environments (CDEs) with interlinked BIM models. A CDE that engages all stakeholders of the building's design, construction, and operation phases represents the outset of BIM maturity level 3. This article introduces a CDE called Virtual Commissioning (VC), capable of commissioning an HVAC system before the physical commissioning of the HVAC system. The FSC diagram is introduced, to represent an HVAC BIM model within the VC CDE, and the Revit to FSC exporter, to serialize an HVAC object model from Revit to the FSC diagram. Three microservices were developed to exemplify the ease of developing independently scalable solutions for the VC CDE. Furthermore, the article proves that Modelica simulations can be run, using the microservice architecture of the CDE. To test the robustness of the system architecture for the CDE, two example models were introduced, one simple and one with a high level of complexity. Transferring the example models from Revit to the VC CDE was successful. Finally, in the roadmap for future development, it is proposed that future work should focus on using the CDE for advanced hydraulic simulations, using Modelica and Spawn-of-EnergyPlus.

1. Introduction

Building information modeling (BIM) is the practice of generating and managing well-defined building data [1]. BIM data is typically geometric, spatial, geographic, physical, or quantitative, and it aims to provide a shared repository for stakeholders [2]. BIM can revolutionize the construction sector by streamlining integrated design processes, accurate construction scheduling, and comprehensive error screening [3]. It further can help mitigate climate change and resource depletion by simplifying and enhancing resource- and energy-efficient integrated design processes for new construction [4–6] or renovation [7]. However, most current workflows are manual or semiautomatic [8], often utilizing conventional spreadsheets [9], and most occur too late to impact the design [9] significantly. Most of these workflows do not utilize the full capabilities that BIM can offer if utilized to its full extent.

Succar et al. introduced the BIM maturity levels to describe the BIM-based collaboration between stakeholders [10]. BIM-based collaboration is defined from maturity Level 0 with almost no collaboration to

Level 3 with full integration, in which all stakeholders collaborate using a shared model in a cloud-based common data environment (CDE) [10]. CDEs are applications that connect several services. Several CDEs have been developed for the AEC industry [11–13]. With a CDE, it is possible to create bi-directional links between a database model and simulation services. A CDE enables teams always to have the most updated model to run new simulations or make design decisions. Previously developed CDEs for advanced hydraulic simulations use either the format of gbXML (Green Building XML) or IFC (Industry Foundation Classes). Transforming BIM data to BEM tools using gbXML and IFC formats can introduce extreme errors [14], and the process occurs only once due to the need for manual data entry [15]. More importantly, this BIM to BEM process relies on a file-based exchange mostly, which is more common to BIM Level 2. Therefore, to enable a real-time connection with live building data, building models, and simulation data, an approach needs to be taken that is not file-based, but rather web-based including databases and microservices. That may lead to a simulation-enabled CDE.

* Corresponding author at: Department of Civil and Mechanical Engineering, Technical University of Denmark, Copenhagen, Denmark.

E-mail addresses: msei@ramboll.dk (M. Seidenschur), alikuc@byg.dtu.dk (A. Küçükavcı), evifj@byg.dtu.dk (E.V. Fjerbæk), kevs@byg.dtu.dk (K.M. Smith), p.pauwels@tue.nl (P. Pauwels), cah@byg.dtu.dk (C.A. Hviid).<https://doi.org/10.1016/j.autcon.2022.104500>

Received 25 March 2022; Received in revised form 16 June 2022; Accepted 24 July 2022

Available online 7 August 2022

0926-5805/© 2022 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

1.1. BEM simulation through a micro-service architecture

Tools for simulation of advanced hydraulic simulations have existed for many years, like HVACSIM+ [16], IDA ICE [17], the Modelica Buildings Library [18], and many more. Running advanced hydraulic simulations can be extremely useful to test, whether a building performs adequately compared to the original design. However, most AEC companies today only perform simple calculations based on rule-of-thumb. This means that most systems are designed based on a static calculation that only applies the peak load for the system. This means that the system can be regulated for the peak load, but cannot regulate the flow for the remainder of the time, due to lacking valve/damper authority. The hydraulic systems are rarely simulated at all in the design phase of a building. This is mainly due to the labour intensive manual work of setting up the boundary conditions needed to run an advanced hydraulic simulation. Efforts should be made to be able to automatically convert from BIM to BEM. The suitability of tools for automatic integration of BIM and BEM varies considerably. Modelica is an equation-based object-oriented modeling language that provides a flexible means for constructing BEM while excelling at HVAC systems and controls [19,20]. Creating Modelica simulations today is a time consuming endeavor, due to the complexity of providing the boundary conditions for a complete solution. However, if the BIM model is used to automatically transfer the boundary conditions, this can eliminate a large part of the manual task, as shown by Fjerbæk et al. [21]. There are substantial efforts to automate the translation from BIM to Modelica-based BEM [2,22,23]. Kim et al. [22] introduced a library with the name of ModelicaBIM. The idea of the library is to be able to perform Modelica simulations, based on a BIM model. Jeong et al. [23] introduced a tool that could export a building modeled in Revit to perform thermal simulations in Modelica. The article from Andriamamonjy et al. [2] directly translated the geometry, systems, and controls which was encapsulated in an IFC4 file and simulated in a Building Energy Performance Simulation (BEPs) model. There are also efforts to use less well-defined IFC files through enrichment and identification [24] and grey-box modeling [25] to generate Modelica-based BEM.

Even with the recent extension of the HVAC domain (Add2TC1) in the current IFC data model (IFC4), it does not provide the necessary structure and attributes to use third-party simulation tools for HVAC design [26] and therefore requires improvements. Many IFC classes do not map well to the (more detailed) classes needed in a BEM tool (e.g. MechanicalEquipment vs. Air Handling Unit). Therefore, a better object model is needed that includes these specialised HVAC classes and properties. This object model ideally serves as a common data format to enable a CDE to run advanced hydraulic simulations. Furthermore, this object model needs to be web-ready to enable a BIM Level 3 CDE approach (e.g. JSON, RDF), that includes a microservice architecture with horizontal scalability.

Hence, this paper investigates the creation and use of such a common web-ready object model, plus its incorporation in a service-oriented CDE. This paper proposes to create a web application named Virtual Commissioning as a CDE. Virtual Commissioning is envisioned by the authors of this article to generate a virtual environment or CDE, that is capable of commissioning the building services, before, and during operation of the building. We do recognize that traditional commissioning is a quality-focused process that delivers the entire building to the owner, according to the owner's objectives and criteria. In future work, we plan to make the VC platform operational for the full commissioning of the building. In this article, the VC CDE revolves solely on the commissioning of the building services, and their performance. The CDE connects a Revit model with an Application Programming Interface (API) endpoint to a MongoDB database. The database is structured based on the data structure of the FSC object model that is introduced in this paper. The FSC object model is generated from a Revit model using a Extract-Transform-Load (ETL) approach. Finally, the VC platform introduces a microservice architecture that makes it possible to

create microservices that can run independently based on the FSC object model in the database. Three microservices are introduced and utilized on two use cases. After testing the VC platform with the creation of microservices, we will test the performance of the FSC exporter tool on a model obtained from a real-world project.

1.2. Aim

The aim of this article is to:

1. Centralize BIM project data so all stakeholders have access to a single source of truth (SSOT) in a web-based CDE based.
2. Create a data structure or object model that can represent a flow system
3. Allow for easy scalability of the CDE using a microservice architecture.

1.3. Outline

Section 2 describes the current state-of-the-art CDEs that raise the BIM maturity to level 3. Section 3 describes in detail the system architecture of the proposed VC platform and the FSC object model (see Fig. 1). Section 4 introduces example models 1 and 2, which we will use to evaluate the performance in Section 5. Section 6 presents the achievements of this paper, together with the limitations. Furthermore, it offers a roadmap for future development. Finally, Section 7 concludes on the contribution of this paper.

2. Background

This section describes the efforts within the development of CDEs for buildings and HVAC systems. This includes the efforts sought to represent HVAC systems with object models. It also describes the state-of-the-art for simulation environments for full building simulation. Finally, the section introduces the state-of-the-art within software development using microservice system architecture.

2.1. Simulation and computation

Andriamamonjy et al. introduced an automated workflow, called IFC2Modelica, for the direct transfer of geometry, system, and control representations encapsulated in an IFC4 file [2]. One issue with this approach is that commercial BIM tools, like Revit, do not serialize all the needed information sufficiently well to carry out the complete data transfer introduced in the IFC2Modelica workflow. After transferring the IFC file from Revit or a similar proprietary BIM tool, the user must manually input the required information to run the simulation in the Modelica environment. Similarly, Jeong et al. created Revit2Modelica to transfer an architectural BIM model into Modelica for an advanced building energy simulation. The Revit2Modelica approach takes a proprietary file format (.rvt) and translates it into the Modelica file format. However, it does not transfer HVAC systems. IFC2Modelica and Revit2Modelica provide a novel approach for simulation of HVAC systems and building energy modeling but based on file-based BIM models. This means that they do not live up to the BIM maturity level 3. A common data environment should be presented to raise the BIM maturity from level 2 to level 3.

2.2. Object models for representation of HVAC models

Efforts have sought to enable a level 3 BIM maturity with automated, flexible data transformation using open standards (e.g., IFC) and semantic web technologies to improve interoperability, data linking, and logical inference [27]. Afsari et al. implemented the IFC schema in a JSON (JavaScript Object Notation) format to facilitate web-based data exchange [28]. Do-Yeop Lee developed a novel framework using BIM

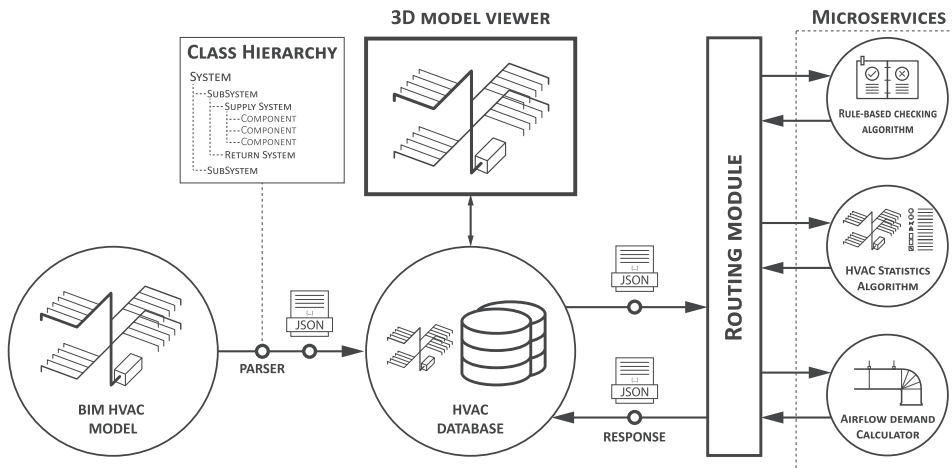


Fig. 1. Proposed System Architecture allows the automated transfer of data from BIM to a database and then follows an automated transfer to a given microservice.

and linked data technologies to share defect data and enhance productivity during construction [29].

Many efforts have specifically concerned building operation, as it is continuous and benefits from a 'live' BIM database. Quinn et al. demonstrated a linked data approach integrating IoT data and BIM [30]. Meanwhile, Tang et al. developed and tested a prototype exchanging building automation system (BAS) data using BACnet and an IFC data model [31]. Similarly, Kim et al. proposed a semantic web-based facilities management approach [32], while Bong et al. developed a BIM-enabled data architecture for fault detection and diagnostics [33]. Furthermore, Mohamed et al. devised an ontology to formalize as-is BIM knowledge for semantic web technologies to improve maintenance [34]. Finally, Balaji et al. created Brick [35] to represent sensors and sub-systems, and the relationship between them. However, while the Brick schema is great at representing data points within the building and HVAC system, it does not represent passive components such as pipes and ducts. Therefore, it is not fit to represent an entire flow system and the aspects thereof. Such developments should help exchange BIM data openly and enable web service applications.

2.3. Common data environments

The development of [BIMServer.org](https://bimserver.org) was an early effort in raising the BIM maturity level from 2 to 3 [?]. The primary purpose of the [BIMServer.org](https://bimserver.org) project was to provide an IFC database that has features like model checking, versioning, project structures, merging, etc. While [BIMServer.org](https://bimserver.org) is an open access open-sourced platform, it is based solely on the IFC schema, introducing serious errors and missing data depending on the tool it is generated by [14]. A proprietary file format approach was carried out by the software vendor Autodesk, with the introduction of the cloud platform A360 and the integration of Forge, which implements an API. For the project team, Forge provides an easy way to share and version Revit models in the cloud; it is still based on the proprietary file format from Autodesk Revit. This introduces a limitation in integrating a link with external applications A360 does not support [36]. Cheng et al. made an online CDE that was based on the gbXML schema [12]. Furthermore, they included an energy modeling approach using the open-sourced tool EnergyPlus. While providing an open platform that eliminates the need for file-based sharing of BIM models, the platform only supports gbXML. The efforts mentioned in this subsection could be specified as CDEs, but none of them introduced a CDE capable of storing an HVAC model with the capability of HVAC simulation. The IBPSA project 1 introduced a CDE based upon IFC, CityGML, and

Modelica [13]. The project seeks to create an open-source tool that allows next-generation computing for the design and operation phases of buildings and district energy and control systems. The IBPSA project 1 integrates the object-oriented modeling language Modelica into their CDE for HVAC simulations. They use IFC as the file format. While the IFC model represents an open data format, it is also known that most proprietary tools, like Revit, have severe errors in parsing from their native format to IFC [14]. The IBPSA project 1 utilizes a classic monolithic architecture, which makes it difficult to scale the application to a cloud computing setup [37].

2.4. Microservice architecture

With the software engineering domain moving toward cloud computing, microservices are becoming more mainstream [37]. Microservices are deployed, tested, and run independently, making it easy to scale an application, especially in a cloud computing setup [38]. This allows several developers to develop/maintain services while the CDE stays in operation.

2.5. Summary

In summary, CDEs have been introduced in earlier works, like [BIMServer.org](https://bimserver.org) and Autodesk Forge. However, they are not capable of representing an HVAC object model. Furthermore, the IBPSA project 1 is a CDE that allows for next-generation computing in Modelica, based on the IFC model of an HVAC system. Though the IFC format is considered open, the parsing from proprietary BIM tools like Revit is error-prone, meaning there is a need to introduce an open format to represent flow systems. Furthermore, none of the CDEs above present a way to incorporate a microservice architecture, allowing for horizontal scaling of the web application. While developments of CDEs have concerned building operation, fewer have enabled BEM and dynamic simulations using web-based BIM. Kukkonen et al. devised a semantic web ontology for flow systems in buildings, which aimed to support web-based design and operation [39]. That ontology inspired the development of an FSC object model capable of handling entire flow systems and the component properties for hydraulic simulation. The data format of a flow system is not openly available from tools like Autodesk Revit, so our implementation of the FSC diagram yields a toolchain for enabling web-based services requiring flow specifications from a shared online BIM database. This builds on the developments integrating BIM and BEMs, but it uses a database to increase the BIM maturity level from 2 to 3.

3. System architecture

Based on Section 2, we propose the creation of a VC platform that serves as a CDE connected to a BIM tool that includes HVAC systems. This section considers three core developments for the VC platform: (1) The FSC object model for flow systems, (2) the implementation of the database, and (3) microservices for containerized and decentralized calculation. The source code for the FSC object model and the database is not shared specifically within this report. However, the class hierarchy is shared in Fig. B.1. The source code for microservices is open-source and has been shared in Section 3.3.

In detail, this section describes the system architecture of the VC platform. The system architecture shows a conceptual model that defines the structure and behavior of the platform. The platform allows for the decentralization of applications with the use of microservices. Fig. 1 shows that the system architecture revolves around a web application with a MongoDB database. The platform provides a link between the BIM model in Revit and the database. The BIM model is transferred using the Revit API by mapping and serializing an FSC object model and sending it to the database in the VC platform. Once the data has been transferred to the database, microservices can be utilized for decentralized calculation. The 3D model viewer is depicted in Fig. 3 to show its placement relative to the system architecture. The 3D model viewer will not be discussed further in this article. Section 3.1 introduces the FSC object model used to describe the flow system and its components. Furthermore, a UML class diagram for relating spaces to HVAC components is presented. Section 3.2 showcases the database setup used for the VC platform. Finally, Section 3.3 shows how a microservice architecture is utilized, enabling several microservices to use the database FSC object model for decentralized calculation.

3.1. The FSC object model

This Section introduces the FSC object model. We used a Unified Modeling Language (UML) class diagram to create the FSC object model. FSC describes the composition of the flow system, with the relationship between the flow system and its components. Moreover, it appends the attributes needed to describe the physics of components in a flow system. For instance, a component is defined by its properties and the relation to any connected components and systems. FSC contains three main features that enable the description of the flow system:

1. The HVAC system is divided into subsystems, creating a system topology.
2. Each component of the system is defined with its physical properties.
3. The connectivity of all components are defined in sufficient detail.

Fig. 2 shows a simple UML class diagram, that describes the HVAC-System, SubSystem, Component, and Connector classes. In total, the FSC diagram contains 37 classes and 54 methods. This article will only describe the core classes in the class diagram and not all of the classes and methods in detail. To see a complete UML class diagram, see Section Appendix B. The FSC UML class diagram has been created using the Modelica Buildings library [40] as inspiration, since the purpose of future work is to be able to simulate in Modelica.

3.1.1. Topology of a flow system

Fig. 3 shows an example of a flow system. The HVACSystem can contain the distribution systems for Heating, Cooling, and Ventilation. Fig. 4 shows that within each of those categories, there are always two different SubSystems: a Supply system (solid lines), and a Return system

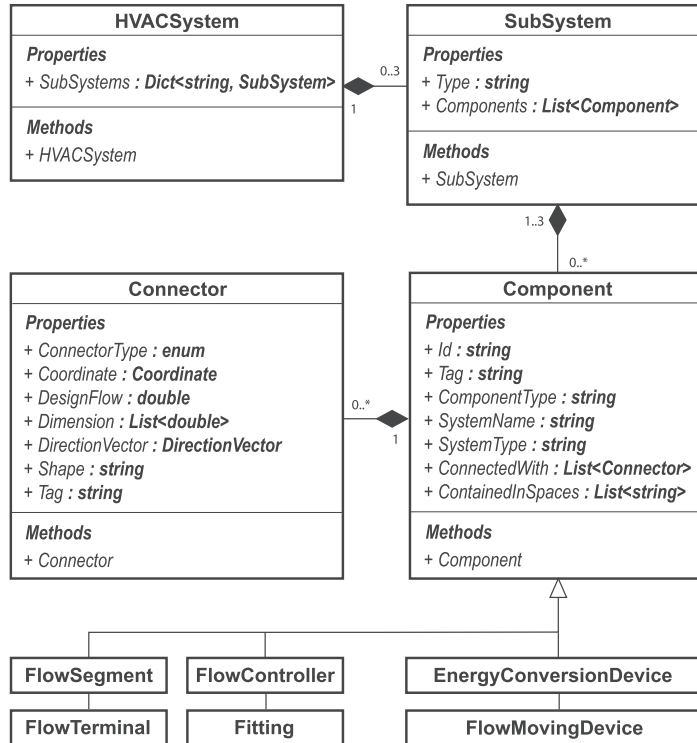


Fig. 2. UML class diagram showing the connection between the four main components of the system. The HVACSystem contains the different SubSystems that is of type SubSystem. In a SubSystem, there is a list of Components.

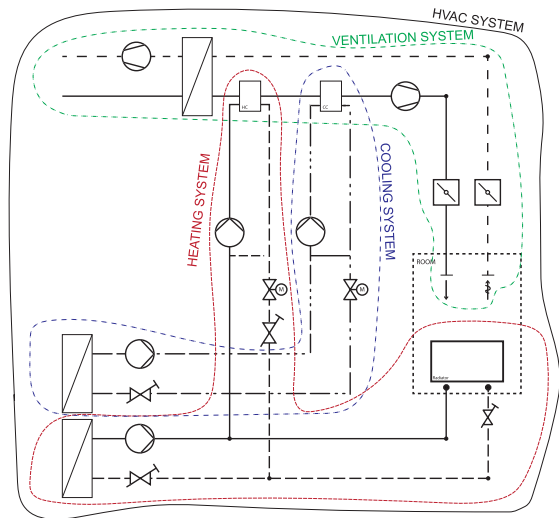


Fig. 3. The overall system topology. An HVAC system can contain a heating, cooling, and ventilation system.

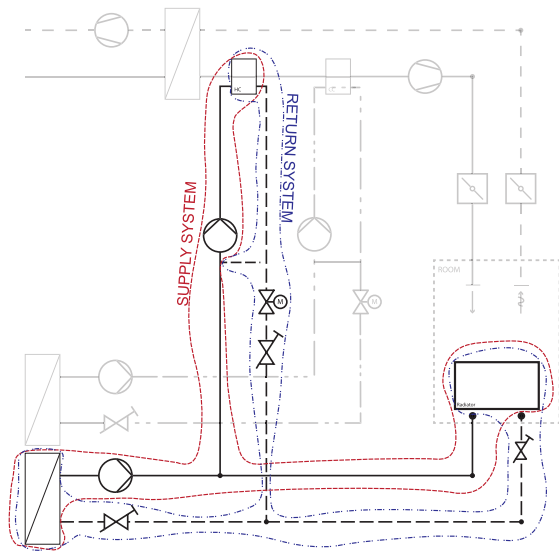


Fig. 4. Every HVAC system contains a supply and return system. As it is illustrated in the figure, the components that bind the supply and return system are both on the supply and return system.

(dashed lines). An end consumer, like a radiator, becomes the interface between the supply and return system. This is reflected by including the same instance of the Radiator in both the supply and return system, with the same Id and Tag. This principle is applied to components that make up the beginning and/or end of a circuit, such as EnergyConversionDevices, and FlowTerminals.

3.1.2. Component properties

In addition to the systems and subsystems, FSC introduces a super-class (Component) that encompasses the properties that exist in all types of components within a system. Fig. 2 shows the properties and

methods contained within the Component class. All FSC subclasses contain the following properties: (1) the Id uniquely identifies the component; (2) the tag identifies the component; (3) the classification of the component type; (4) the system name; (5) the system type; (6) a list of connectors (see Section 3.1.3); (7) the spaces that contain the component. Fig. 5 shows a list of all the subclasses of the Component class, which inherit properties from Component. To see the attributes of each component, see Appendix B.

- EnergyConversionDevice is a device that converts energy from one fluid to another; it includes heating coils, heat exchangers, and radiators.
- Fitting typically describes the connection from one Component to another or several other Component. It includes tees, bends, crosses, reductions and caps.
- FlowController describes a component that controls the flow in a flow system. It includes valves and dampers.
- FlowMovingDevice is a component that moves a fluid, which includes pumps and ventilators.
- FlowSegment is a segment that connects any non-FlowSegment component, which includes pipes and ducts.
- FlowTerminal is the terminal unit of any system, which includes ventilation air terminals.

3.1.3. Component connectivity

A logical description of a flow system must include a module to describe the connectivity of components since the purpose of flow systems is to transfer a fluid from one part of the system to another. Fig. 6 shows that the Components contain a logical description of their relations to each other. Pump-1 is supplied with fluid from Pipe-1, and it supplies fluid to Pipe-2. The description of the connection between one Component and another Component is done with the Connector class. In this example, it means that there will be two Connectors for Pump-1. One connector describes its relationship with Pipe-1

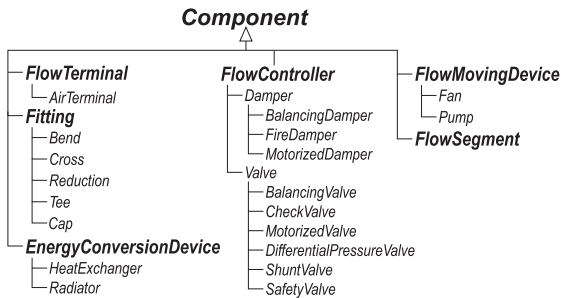


Fig. 5. The tree structure showing all the subclasses to the Component super-class.

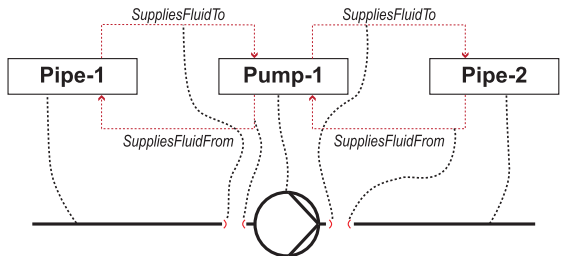


Fig. 6. Example of a topology describing a flow system that consists of a pump connected by a pipe in each connector.

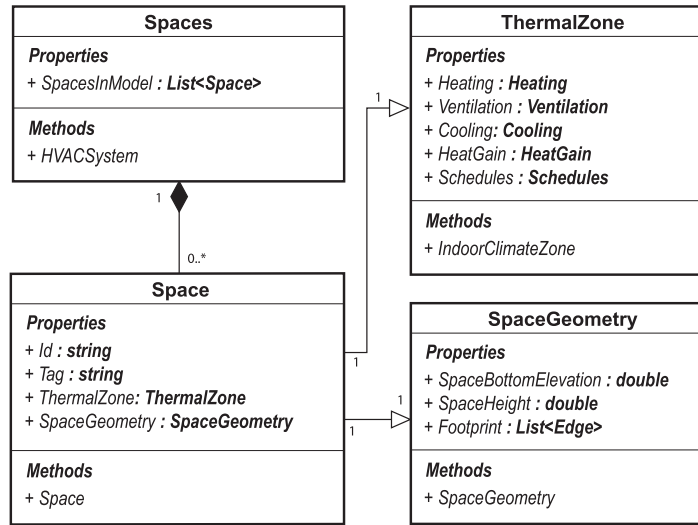


Fig. 7. A part of the UML class diagram for modeling Spaces. For simplicity, the full UML diagram is not shown.

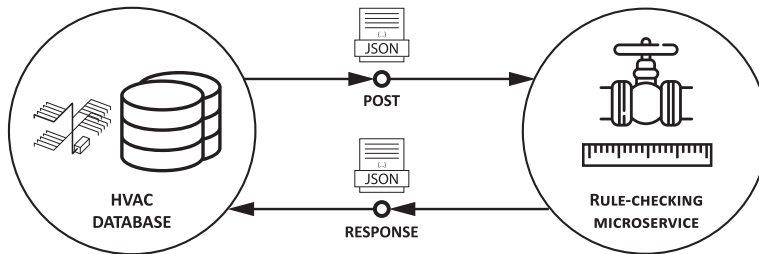


Fig. 8. Illustration of microservice setup. A JSON-file is posted with an HVAC system in the FSC format. Once handled in the microservice, a response JSON is returned.

(SuppliesFluidFrom), and the second Connector describes the relationship with Pipe-2 (SuppliesFluidTo). For consistency and efficiency, every Component describes the relationship to all connected Components, even though the next Component contains a symmetrical connection to the previous Component. “SuppliesFluidTo” describes the forward direction of the flow, “SuppliesFluidFrom” describes from where the flow is coming.

3.1.4. Spaces related to the flow system

In addition to the system representation explained above, we created a class diagram for the properties of spaces to act as boundary conditions for the flow system (see Fig. 7). Such boundary conditions allow for calculating the airflow demand of a ventilation system and sizing the heating system. The ContainedInSpaces property in the Component class is used to describe the relation between spaces and components. The ContainedInSpaces property describes which spaces a component is contained within.

3.1.5. Serialization to JSON data exchange format

Section 3.1 introduced the FSC object model, making it possible to define a flow system and its components, including the spaces of a building. A data exchange mechanism is needed to exchange the FSC object model between platforms. While there are several options to implement such a data exchange (e.g. RDF graphs, XML, CSV, dedicated formats), we chose to focus on a serialization to the JavaScript Object Notation (JSON) format, as nearly all microservices and web service

developments use this format. Listing 1 shows a JSON sample for an example model introduced later in this article.

Listing 1: A JSON object example taken from Fig. 6. The listing shows a three component system. Only the most basic attributes from the base class of Component have been included for all the components. The “...” notation indicates that more components are present but not shown.

Listing 2 The “Spaces” attribute is explained in Section 3.1.4. For simplicity, only the most basic attributes from the base class of Space have been included. The “...” notation indicates that more attributes are present but not shown in this example.

3.2. Database implementation

A mongoDB database stores the FSC object model. mongoDB is an object-oriented database (OOD). The objects created with the FSC object model are stored directly into the database. Representing the data in an OOD is so close to the programming objects that the code is simple to implement. In our implementation, the mongoDB database is instantiated with the use of the serialized FSC object model, as seen in Listing 1.

3.3. Microservice implementations

With the object model and database infrastructure in place, the last element in the system architecture comprises of microservices that operate with the data, as shown in Fig. 1. The microservices developed for this article were all created as Python-Flask API endpoints. In our

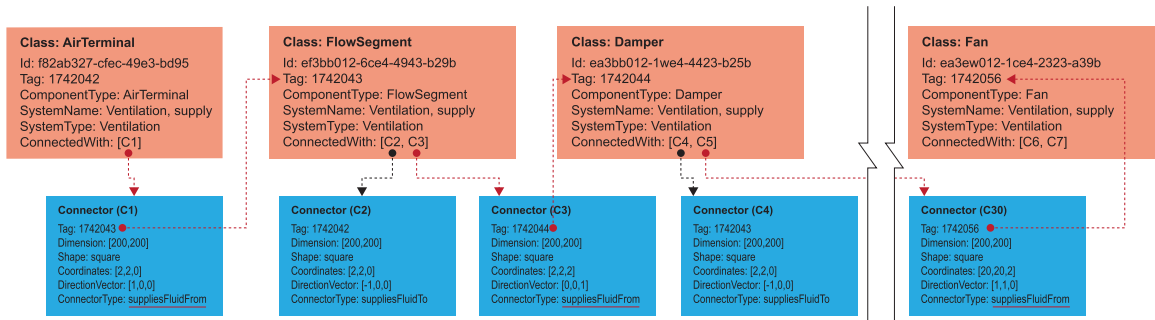


Fig. 9. The Figure shows how to traverse the ventilation system with a recursive function. The recursive function has a stop condition: the ComponentType == "Fan". The red arrows show the path from the air terminal to the fan. In the supply system, the ConnectorType "suppliesFluidFrom" is used as a keyword to find the next component that will lead to the supplying fan. When done for the return system, the ConnectorType "suppliesFluidTo" is used as a keyword to find the next return component that will lead to the returning fan. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

case, we implemented three microservices of use in the HVAC engineering domain:

1. Rule-based checking of system integrity
2. Airflow calculation of ventilation system
3. Calculation of HVAC statistics

Overall, these microservices rely on the infrastructure shown in Fig. 8. The figure illustrates how the microservices work using the first micro-service as an example (rule-based checking). The web application backend makes a POST request to the Flask microservice. The body of the request contains a JSON file that represents the entire flow system in a JSON format, as illustrated in Listing 1. The microservice then checks if all components are in compliance with the requirements described in Table A.1.1. Once it has checked the components, it returns a JSON to the backend, storing it in the mongoDB database.

3.3.1. Microservice for rule-based checking of system integrity

HVAC systems can be complex depending on the size. When handing over a BIM project of the HVAC system, it can be hard to uphold the level of detail, as promised in the Information, Communication, and Technology (ICT) contract of any building design phase. Therefore, it is highly beneficial to have a way to check that the system's integrity holds up. This microservice aims to provide a rule-based checking algorithm with a rule-set to check the FSC object model. The rule-set is shown in Table A.1.1. The source code is made available on GitHub.¹

The functionality of this microservice was already briefly explained in Fig. 8, as a combination of HTTP POST requests, JSON file exchange, microservice computations, and storing of results. The microservice returns a JSON file to the database that describes whether each component lives up to the rules. The result of each component is returned as a Boolean value. The return values are then stored in the database.

3.3.2. Microservice for airflow calculation of ventilation system

With the creation of the object model in the central database, it is possible to traverse through the given HVAC system from one point to another. We created a microservice for airflow calculation of the ventilation system² to exemplify that the system can be traversed. The

algorithm within the microservice starts by resetting all flows on the ventilation system. After resetting the flows, the algorithm takes the airflow demand available in each space and applies them to the air terminals contained in that space. The property ContainedInSpaces is used to find the connection between each AirTerminal and space. Fig. 9 shows an example of the next step to the algorithm. The algorithm takes the airflow of the terminal and then applies it to the next component's connector.

3.3.3. Microservice to calculate HVAC statistics

The primary purpose of the HVAC statistics microservice is to make the VC platform capable of displaying statistics on the HVAC system, including the number of components in the system. Furthermore, it summarizes the total meters of duct/pipe in the model. In summary, the HVAC statistics microservice allows for validation of the FSC object model or even makes it possible to calculate the material usage. Python-Flask was used to create the HVAC statistics calculator with an endpoint that the VC platform can utilize.³ Listing 3 shows an example response from the microservice.

Listing 3: The listing shows an example of a response JSON from the microservice presented in Fig. 8. Each component in the system is counted, and the length of all FlowSegments are summarized into the given cross-sectional dimension.

4. Example models

This Section introduces two example models for showcasing the VC platform and the FSC Object Model in particular.

4.1. Example model 1

4.1.1. The schematic / principle model

Fig. 10 shows the first example model created by the authors of this article. The model contains a heating, cooling, and ventilation system, all connected. The heating system starts with a heat exchanger that converts the heat from the primary heating system (not depicted) to the secondary system, then branches out to a mixing loop for a heating coil (HeatingCoil) in the ventilation system and the radiators of each room. Each radiator is adjustable with a balancing valve (BalancingValve). The motorized valve (MotorizedValve) controls the mixing loop of the heating system.

¹ https://github.com/Virtual-Commissioning/VC-HVAC_rule-checking-Service

² https://github.com/Virtual-Commissioning/VC-Ventilation_dimensioning-service

³ https://github.com/Virtual-Commissioning/VC-HVAC_statistics-Service

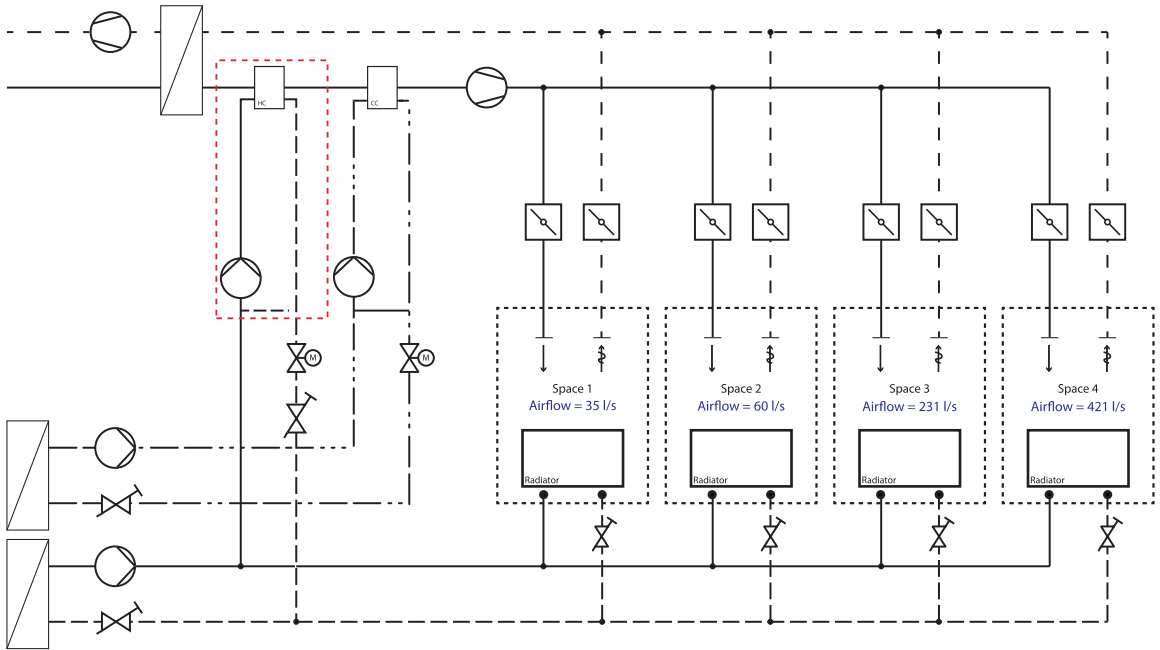


Fig. 10. Example model 1 mechanical schematic. The Figure contains three subsystems: heating, ventilation, and cooling. Furthermore, it includes four spaces that have an airflow. The radiators and the heating coil of the ventilation system provides heating to the room. Similarly, the cooling coil provides cooling by air to the room.

The cooling system is on the secondary side of the heat exchanger (HeatExchanger). The cooling liquid supplies the mixing loop provided by a pump. A mixing loop with another pump may seem excessive in this case, but was included as an example. The shunt is controlled with a motorized valve (MotorizedValve) and a pump (Pump).

The ventilation system contains a ventilation fan that takes the air from the air intake through a heat exchanger (HeatExchanger) and then a heating coil (HeatingCoil) and cooling coil (CoolingCoil) respectively. The air is supplied to space 1, 2, 3, and 4 with the use of air terminals (AirTerminal) controlled by a regulation damper. After supplying the air to the room, the air is extracted through the air terminal (AirTerminal). The air is then exhausted with the ventilation fan (Fan) after it has gone through the heat exchanger, exchanging any excess heat to the supply air.⁴

The example contains spaces, to exemplify the connection between the systems and spaces as seen in Section 3.3.2. All of the spaces are heated by ventilation and radiators, and are cooled by ventilation.

4.1.2. Instantiating the object model

This subsection visualizes the instantiated FSC object model for the Revit model shown in Fig. 11. The serialized JSON which represents the FSC object model is provided for the reader.⁵ Fig. 10 shows a call-out with a red-dotted line. Fig. 12 visualizes part of the instantiation of the object model within the previously mentioned call-out of a system and then serializes it into JSON. Fig. 12 also shows how each component is instantiated with a relationship to the attached connector. For

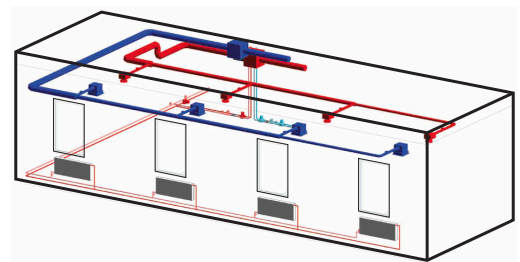


Fig. 11. The example model is shown in Fig. 10, modeled in Revit. The model contains the exact components and rooms shown in Fig. 10, except from the heat exchangers from the primary to the secondary system. The 3D model was modeled in Revit.

instance, the Tee (Tag: 1742043) is instantiated with Connector C2 and C3. This means that Connector C2 and C3 are instantiated within the ConnectedWith attribute of the Tee component. Connector C2, displayed in a blue box of Fig. 12, connects component 1742043 with component 1742044. Furthermore, the Connector class contains the physical properties of the connection port that interfaces with the adjacent component. Such physical properties include the dimension, shape, coordinates, and direction vector of the connector. The direction vector of the port will always orient away from the component. Finally, the ConnectorType displays whether another component supplies the connector or if it supplies another component. For instance, C2 suppliesFluidTo component 1742044, and C4 suppliesFluidFrom component 1742043.

⁴ https://github.com/Virtual-Commissioning/VC-HVAC_rule_checking-Service/blob/main/app/ressources/example_model_1.json

⁵ https://github.com/Virtual-Commissioning/VC-HVAC_rule_checking-Service/blob/main/app/ressources/example_model_1.json

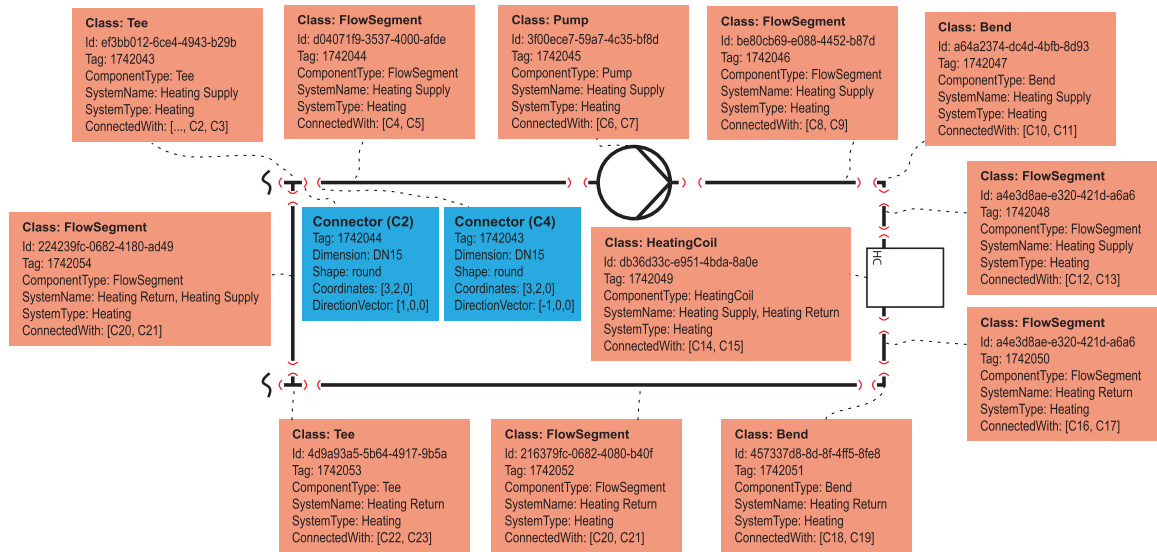


Fig. 12. The figure shows a schematic a small flow system. The red callout from Fig. 10 makes up the example seen in this figure. For simplicity, only the connectors (in the blue boxes) C2 and C4 are shown. Orange boxes make out a component, and blue boxes show the connectors related to a given component. The figure does not contain all properties for all components. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

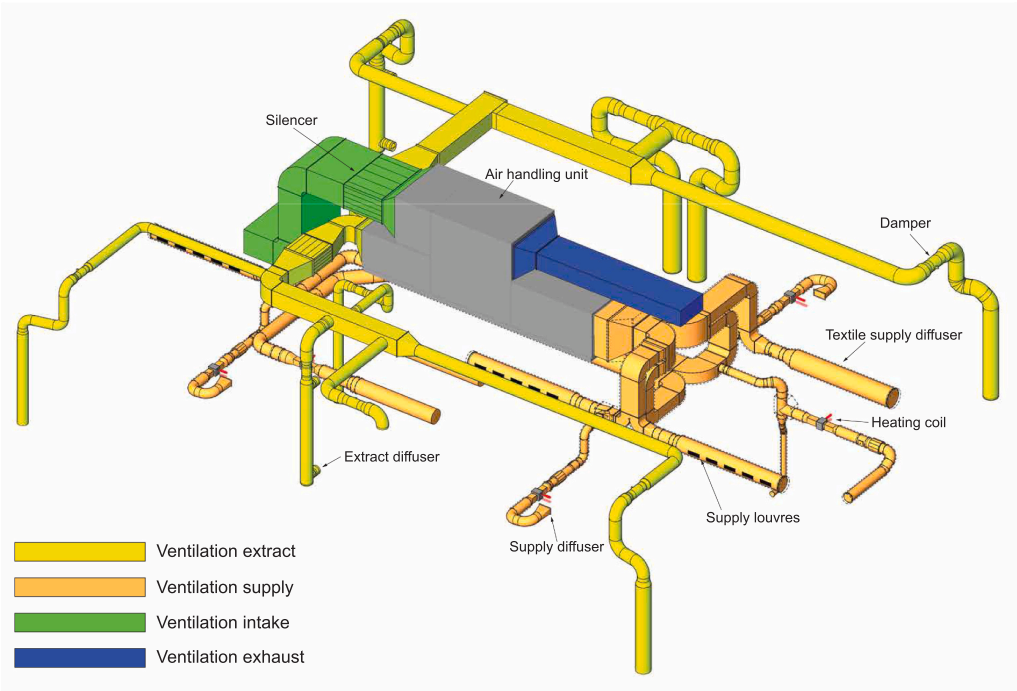


Fig. 13. Example model 2. Revit example model obtained from TU Eindhoven. The model shows a ventilation system with subsystems such as supply, extract, exhaust, and intake air.

4.2. Example model 2

4.2.1. The schematic/principle model

Fig. 13 illustrates the second example model. The Eindhoven University of Technology provided a real-case example model. The model contains a complex ventilation system with extract, supply, exhaust, and intake systems. Furthermore, it has heating coils attached to the supply-side of the ventilation system. The heating system is simplified and contains only the heating coil and the connected pipes. In general, this example aims to show the performance of the VC platform on an “imperfect” model.

The air handling unit (AHU) provides air to the ventilation system. The AHU was modeled as a box with four connectors in this example. This differs from the example shown in Section 4.1 by not specifying the components inside the AHU. A typical AHU consists of fans for supplying and extracting air, heating and cooling coils, silencers, and filters, i.e. it illustrates a “real world problem” in which not all modeling standards are the same - some designers would model all the components within the AHU while some designers (depending on company standards, and the design phase) would model the AHU as a box. The supply and extract system was modeled with variable air volume (VAV) dampers. This means that the ventilation system can vary the airflow in specific rooms. This example model does not contain any spaces, as these are contained in the architectural BIM model.

4.2.2. Instantiating the object model

We used the FSC exporter to generate the FSC object model, based on the Revit model, seen in Fig. 13. Next, the FSC object model was serialized into JSON and imported to the VC platform.

5. Results

This Section first displays the robustness of the format by visually explicating an example of the format. Following, it shows the platform’s scalability with the use cases of a rule-based checker of the FSC object model, a BIM to airflow calculator, and an HVAC statistics tool.

5.1. Example model 1

5.1.1. Rule checking

The rule-based checking algorithm was used to see whether example model 1 (Section 4.1) lives up to the rule-set presented in the rule-based checking algorithm (Section 3.3.1). Table 1 shows that 219 out of 225 components lived up to the rules presented in Table A.1.1. The six components that did not live up to the rule-based check were the “open ended” components placed at the beginning and end of each system, including two from the ventilation system, two from the heating system, and two from the cooling system, which was expected.

5.1.2. HVAC statistics

Table 2 shows the result of running the HVAC statistics microservice from Section 3.3.3 on example model 1. The only component type counted differently by the microservice is the heat exchanger. Since the heat exchanger should always be connected with two systems - in this case the ventilation and the cooling system, and the ventilation and heating system, this is accepted. Even though every heat exchanger is represented twice in the format, it is also annotated with the same tag. Therefore, it is still possible to distinguish whether it appears twice.

Table 1

The table shows the result of running the rule-based checking algorithm on example model 1. The table shows that the FSC object model contains 225 components.

Components checked	True	False
225	219	6

Table 2

This table illustrates the total amount of components reported in Revit and after the transfer to the VC platform.

Components	Amount Revit	Amount VC
AirTerminal	8	8
MotorizedDamper	8	8
Bend	30	30
Reduction	40	40
Tee	14	14
BalancingValve	8	8
MotorizedValve	2	2
HeatExchanger	2	4
Fan	2	2
Pump	4	4
ShuntValve	2	2
PressureSensor	2	2
TemperatureSensor	2	2
Radiator	4	4
FlowSegment	96	96
Total components	223	225

Table 3

The table illustrates the length of the FlowSegments transferred from Revit to the VC platform, before and after the transfer. Each duct or pipe type is reported with its deviation from the Revit model to the VC platform. All measurements are in millimeters.

Duct Size	Length Revit	Length VC	Dev %
<i>Round ducts</i>			
Ø80	14,490	14,490	0
Ø125	15,430	15,430	0
Ø200	20,260	20,260	0
<i>Pipes</i>			
Ø15	24,259	24,280	<0.01
Ø18	7572	7570	<0.01
Ø22	39,613	39,620	<0.01

Table 3 shows that the length of the components was counted to a precision better than 0.01%. The precision is arguably caused by rounding off values in the microservice or Revit, which is normal and acceptable when dealing with geometry in different systems. Since the discrepancy is so small, it is considered insignificant.

5.1.3. Airflow calculation

Fig. 14 shows the airflow calculation microservice from Section 3.3.2 applied to example model 1, to visualize the functionality and result of the tool. If Space-1 has an airflow demand of 35 l/s, the airflow demand is applied to the air terminals that are contained in Space-1. Then it is added to the existing airflow on the ventilation duct up to the ventilation fan. The script was tested by applying it to the example model described in Fig. 4.1.1. By using the microservice to analyze the ventilation system, it was found that the total airflow needed to run the system was 747 l/s

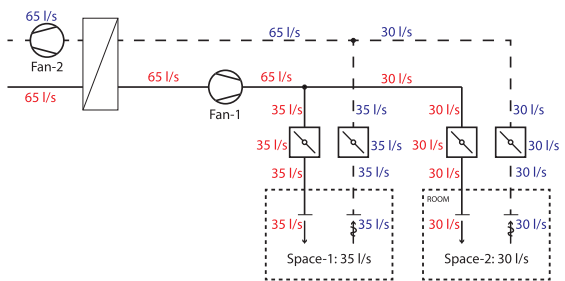


Fig. 14. The ventilation system is traversed and airflows are summed along the supply and return paths.

for the supply and return ventilation fan. The airflow calculation is one of the first steps in choosing a fan that can meet the airflow demand of the system.

5.2. Example model 2

This Section shows a use case test of the HVAC exporter on the example model introduced in Section 4.2. A BIM model has been obtained from the industry to test the performance of the Revit to the VC platform with the use of the FSC exporter. The transfer will be quantified with the HVAC statistics tool (Section 3.3.3) and the HVAC rule checker tool (3.3.1). The purpose is to test the created FSC object model exporter tool with a BIM model that has not been built by the authors of this report. Once the BIM model has been transferred from Revit into the VC platform, the microservices for rule-based checking (Section 3.3.1) and HVAC statistics (Section 3.3.3) will be used to quantify how well the FSC object model based on the BIM model has been transferred into the VC platform.

5.2.1. Rule checking

Table 4 shows the result of the rule-based checking algorithm run on the transfer of the use case from Fig. 13. The table shows that 158 components out of 493 lived up to the rule-check proposed in Table A.1.1. That means that the majority of elements did not pass this check. This will be documented further below in this article, yet the main reason is that the system in the particular building is not as complete and correct as expected by the rules developed in this microservice.

5.2.2. HVAC statistics

Table 5 shows the number of components reported in Revit and the number of components reported after the transfer from Revit to the VC platform with the use of the FSC exporter. The total amount of components for Revit and the VC platform was 487 and 493, respectively. This is explained in the way that FSC divides the flow system. The full system contains both a ventilation and heating system in the model (The heating system only consists of a few pipes connected to the heat exchangers). In the example model, the ventilation and heating systems are connected by heat exchangers. Fig. 3 shows an example of this, near the heat exchangers. This behavior intends to provide an internal connection for each of the systems.

Table 6 shows the length of the FlowSegments reported in Revit and the length of the FlowSegments reported after the transfer from Revit to the VC platform with the use of the FSC exporter. In Table 6 it is reported that not every duct or pipe has the same length after it has been

Table 4

The table shows the result of running the rule-based checking algorithm on Fig. 13. The table shows that the FSC object model for example model 2 contains 493 components.

Components checked	True	False
493	158	335

Table 5

This table illustrates the total amount of components reported in Revit and after the transfer of the FSC object model to the VC platform.

Components	Amount Revit	Amount VC
AirTerminal	33	33
Cap	14	14
Bend	69	69
Reduction	125	125
Tee	13	13
BalancingDamper	29	29
HeatExchanger	6	12
FlowSegment	198	198
Total components	487	493

Table 6

This table illustrates the length of the FlowSegments transferred from Revit to the VC platform, before and after. Each duct or pipe type are reported with their deviation from the Revit model to the VC platform. All measurements are in millimeters.

Duct Size	Length Revit	Length VC	Dev %
<i>Round ducts</i>			
Ø160	2862	340	88
Ø250	7250	540	93
Ø315	24,644	21,680	12
Ø355	3624	3624	0
Ø400	31,443	21,120	33
Ø450	12,653	6430	49
Ø500	33,286	24,980	25
Ø600	8152	8152	0
Ø630	21,916	0	100
<i>Square ducts</i>			
1200 × 600	5600	0	100
200 × 200	3402	3402	0
2100 × 900	4535	3390	25
2178 × 1538	500	500	0
350 × 350	83	0	100
400 × 400	33	0	100
600 × 600	5214	340	93
700 × 350	1868	0	100
700 × 400	2118	0	100
700 × 600	5238	620	88
800 × 400	18,142	0	100
850 × 600	3714	2840	24
<i>Pipes</i>			
Ø32	2016	2016	0

transferred to the VC platform. This behavior is caused by an incorrect Revit model, that does not contain “correct connectivity”. This problem is caused by the microservice not being able to handle specific cases, like it is seen in Fig. 15 where several AirTerminals are placed on the duct. This behavior has not been accounted for in the microservice. It also explains why the Ø630 ducts are not counted a single time in the HVAC statistics microservice. Listing 4 shows the JSON structure in the FSC format behind the object shown in Fig. 15. The Figure shows that five air terminals have been placed directly on the ventilation duct, which is usually not expected. This means that the FlowSegment has a total of 7 connectors.

Listing 4 The listing shows the component mapped from Fig. 15. Not all the connectors are shown, to improve readability of the JSON.

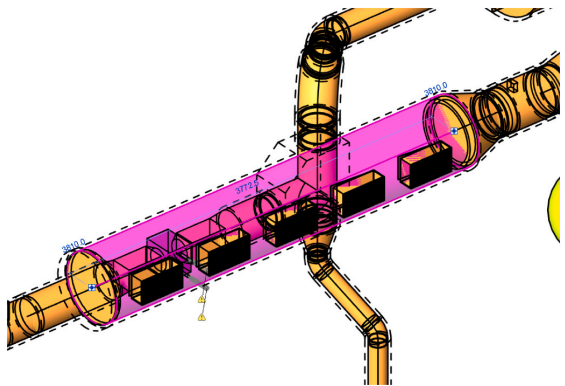


Fig. 15. The ventilation duct in this figure has seven connectors. It has the beginning and the end of the flow segment, but it also has five connectors from the 5 air terminals that have been placed directly on the ventilation duct.

6. Discussion and future work

The VC platform is a common data environment used for HVAC projects during the building's design, construction, and operation phases. This paper describes the process required to develop a CDE and proposes a roadmap for further research and development. The requirements for the VC CDE were that it should expose the proprietary format of a Revit HVAC system in a commonly available platform. Using Python-Flask microservices, we illustrated how the VC platform allows for continuous integration and continuous deployment of functionalities on the platform. The VC platform can scale in any direction with the integration of microservices. Furthermore, it allows for the extension of the FSC object model since the system architecture is based on a microservice architecture. For instance, if the exporter from Revit to the FSC object isn't working, that can be tested and patched independently of touching the code for any other microservices.

6.1. Achievements

We created a CDE with a microservice architecture and showed that the CDE is easily scalable by implementing simple microservices such as the airflow calculator (see Section 3.3.2). A rule-based checker was also implemented to check if the data transfer was successful from Revit to the FSC object model in the VC platform. Furthermore, a microservice was implemented to provide statistics of the data transfer carried out. It offers the user insights into the flow system, such as the length of different ventilation ducts or pipes. The FSC diagram and the FSC exporter allow for the serialization of the FSC object model. A MongoDB OOD stores the FSC object model. The FSC object model creates a connected network of ducts and pipes, making it possible to represent the nature of a flow system. The FSC exporter allows the user to create an FSC object model serialized in JSON to work within the VC platform - or even in external platforms. We used the FSC exporter on an externally developed Revit model and were able to transfer the full FSC object model to the web application to run microservices on it. We proved that the FSC object model is able to link the demand of a space with the airflow system. This can be used to create dimensioning tools for ventilation systems or heating- and cooling systems. In order to enable future use of the VC platform, the authors of this article will continuously maintain and update the platform with new features.

6.2. Limitations of the study

We created BIM model with LOD350 to evaluate the VC platform and the FSC object model, called example model 1. Therefore, the model used to evaluate the VC and FSC was "created to succeed". For instance, all components must have precisely the right amount of connectors. There is a risk of designing the VC platform for "the perfect scenario" where all data is available in the BIM model. It is often a challenge in the AEC industry that BIM models do not contain all the data necessary for a complete model, like component connectivity, component naming, system naming, etc. To handle problems like this, the AEC industry in Denmark has introduced ICT agreements on building projects. It is a contract that obligates the company to deliver building documentation of a certain standard. However, while LOD descriptions are relatively detailed, consulting engineering companies in the HVAC branch still struggle to provide models that live up to the LOD350 standard. We built the VC platform to handle the delivery issues in the HVAC branch by integrating a microservice architecture. For instance, if the HVAC system in the database has an error, it can be fixed later in the VC platform using the microservices. One of the microservices is the rule-based checking algorithm shown in Section 3.3.1. The rule-based checker will let the user know that some information is missing. For instance, such missing information could be that some components are connected incorrectly. Then, it is possible to specify the actual connectivity directly in the VC platform.

The VC platform was tested on a model developed externally to emulate the situation of an imperfect BIM model, as seen in Section 4.2. In Section 5.2 the HVAC Statistics microservice in Section 3.3.3 and the rule-based checking algorithm in Section 3.3.1 was run on example model 2. The results showed that all components were transferred successfully to the VC platform, see Table 5. However, the performance test with the rule-based checking algorithm showed that 158 of 493 components lived up to the rules established in Table A.1.1. The results of this performance test were caused by, for instance, the modeling practice shown in Fig. 15. The figure highlights that the FSC exporter does not always map the components correctly. This behavior is expected for any Revit model that has not been modeled with the intent to transfer it to the database. The user can use the rule-based checking microservice to find which components do not live up to this algorithm. The 3D-viewer of the VC platform visualizes the results and alerts the user which components do not follow the rule-set. Thereby, the user can solve the issues manually.

The test case presented in Section 4 showed the application of the FSC object model together with the VC platform. As part of the platform's deployment, more extensive testing should be carried out on Revit models to ensure the robustness of the FSC exporter and platform.

There are issues that the VC platform cannot solve. For example, the FSC exporter is highly dependent on the template used within the Revit model. If a flow segment like a pipe is modeled as the "Mechanical Equipment" category instead of a "Pipe" category in Revit, then the pipe will be mapped into the wrong category or not mapped at all. Such a fault will cause an exception within the FSC exporter, meaning that an error will cause the exporter to abort the operation and, therefore, fail to export the FSC object model from Revit to the database.

The FSC exporter was designed to link the Revit model and the database. The FSC exporter demonstrates that it is possible to extract information from a BIM tool like Revit to work within a database. For the purpose of this article, Revit families from the Rambøll library was utilized and modified to be able to represent all the information needed to export it into the FSC object model. The work presented in this article does not exclude exporters from other file formats to be made. Such integration could include the open file format IFC. The VC platform is not dependent on the file coming from Revit or any other proprietary format, as long as it follows the FSC diagram.

It will require detailed HVAC models for advanced hydraulic simulations in Modelica. This presents a problem for the AEC industry with the current status of BIM modeling. The AEC industry needs to model buildings more realistically and contain information in the BIM model in an early design phase to provide a correct design based on actual performance rather than rule-of-thumb. However, we believe that with the younger generation coming into the AEC industry, the market is ripe for the digital transformation it will take. Performing these simulations in the early design phase will be more time spent on design than fixing issues during the physical commissioning or operation phase. In the company supporting this article, the method will be employed to do just that - to save time in the long run.

6.3. Roadmap for future development

The VC platform and the FSC object model have been carried out as part of a development to create a CDE for full building simulation using BIM models. This paper introduces the FSC object model, and exemplifies how to use microservices for calculation or even simulations. The VC platform is envisioned as a three-stage development project. The stages are illustrated in Fig. 16. Stage 1 is the work presented in this paper and is the initial development of the VC platform and the FSC object model with the simple calculation microservices. The microservices seek to prove that adding microservices to the VC platform is possible. The authors of this article recognizes that the microservices are very simple, and constitute services that already exist within BIM programs like Revit. The microservices were added to exemplify that tools

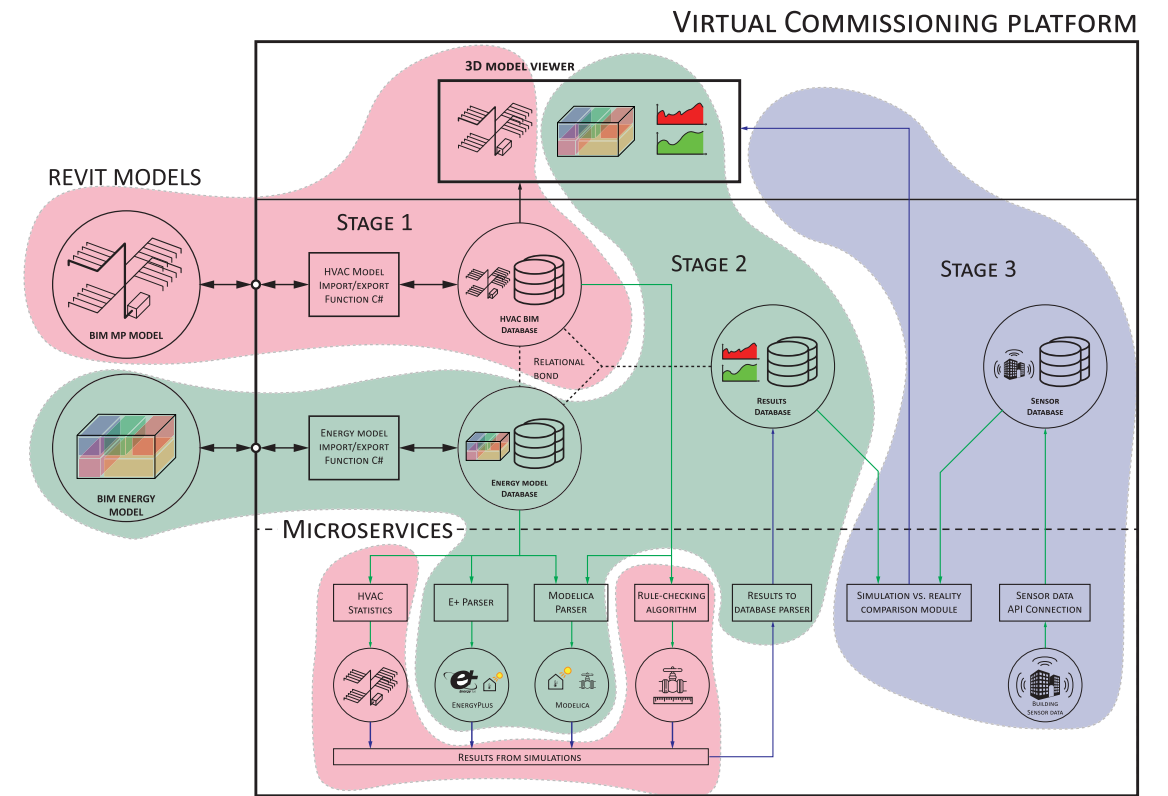


Fig. 16. The Figure shows a roadmap for the future development of the VC platform. The areas marked with red represent the work developed for and presented by this article. The area marked in green represents the next step in developing the VC platform. Finally, the blue area represents the final step, a module that includes sensor data in the VC platform. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

developed within Revit can easily be replicated to a cloud-based CDE based on a non-proprietary format, as opposed to the proprietary format of Revit. The early development of the VC platform and the FSC object model provides a stepping stone for developing a complete CDE for continuous integration of simulation tools through microservices. Stage 2 includes the development of a space class object model to represent a BIM energy model. A simple space class object model was presented in Section 3.1.4. Fig. 16 shows that the “HVAC BIM database” and the “Energy model database” share a relational bond. Thereby, it will be possible to link these two data formats together. Finally, stage 2 introduces the possibility of running whole building simulations through Modelica and EnergyPlus with the use of Spawn Of EnergyPlus in Modelica [41]. Running simulations on indoor climate and HVAC systems simultaneously might help perform more accurate predictive energy models. The work on stage 2 has already begun, and the authors of this article have proved that the link from CDE to Modelica-based Dymola, is possible [21]. Fjærbæk et al. [21] simulated a small heating system in Modelica and was capable of showing the return temperature for each heating loop. The toolchain created by Fjærbæk et al. [21] made it possible to easily initiate Modelica simulations of a heating system - a process that under normal circumstances would be very time consuming, due to the manual labour of creating a Modelica simulation model. Stage 3 will include sensor data and connect it to the BIM model with a relational bond. It will be possible to take data from the operating building and do continuous fault detection on it with sensor data. That

way, the digital twin in the VC platform can inform the Building Management System (BMS) of the actual building to make certain adjustments. An example of an adjustment could be to run with the objective of minimizing energy costs (as opposed to minimal energy usage).

7. Conclusions

The three aims of this paper were to:

- 1. Centralize BIM project data so all stakeholders have access to a single source of truth (SSOT) in a CDE based in a web application
- 2. Create a data structure that can represent a flow system
- 3. Allow for easy scalability of the web application utilizing the principle of microservice architecture.

This article introduces a CDE called the VC platform. The article exemplifies a paradigm shift from a proprietary file-based BIM model to a web-based database BIM model. The VC platform allows for the development of applications in a fully modularized way through microservices. Microservices make it possible to deploy custom applications that run specific tasks independently. We developed the VC platform to enable advanced simulations of HVAC systems that relate to the actual spaces of the building. Future work has been planned to integrate Modelica and Spawn of EnergyPlus as microservices on the VC platform. An externally provided Revit model was used to test the

performance of the VC platform. The performance test proved that the FSC exporter from Revit to the VC platform worked as intended – it transferred all components to the database in the VC platform. After transferring Revit with the FSC exporter to the VC platform, the performance test revealed that not all components were compliant with the rule-based checking algorithm. However, this is not problematic, as the rule-based checking algorithm intends to highlight errors like this so the user can solve them in the VC Platform frontend.

Declaration of Competing Interest

The authors recognize that there is a potential for conflicts of interest via industry affiliations. Mikki Seidenschur is working on a doctoral dissertation in Technical University of Denmark, while also working in Ramboll. Ali Küçükavcı is working on a doctoral dissertation in

Technical University of Denmark, while also working in COWI. Esben Visby Fjerbæk is working as a research assistant at Technical University of Denmark Kevin Michael Smith is working as a researcher at Technical University of Denmark Pieter Pauwels is working as a professor at Technical University of Eindhoven Christian Anker Hviid is working as an associate professor at Technical University of Denmark.

Acknowledgments

Funding: This work was funded by the Ramboll Foundation and the Innovation Fund Denmark (grant 9065-00266A). The use case test model in Section 4.2 was provided by TU Eindhoven. We would like to acknowledge the work of the reviewers of this manuscript for improving the quality of the manuscript with thorough and constructive comments on content and writing.

Appendix A

Table A.1

This table illustrates the rules that exist with all the subtypes of components in the class object model.

Subclass of Component	Rules
FlowSegment	Contains two connectors
HeatExchanger	Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo" Contains 4 connectors Contains two connectors: "suppliesFluidFrom" and two connectors: "suppliesFluidTo" Is contained within two different subsystems
Radiator	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
Bend	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo" Has an angle greater than 0
Cross	Contains 4 connectors Contains at least one connector of "suppliesFluidFrom" and at least one connector of "suppliesFluidTo"
Reduction	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
Tee	Contains 3 connectors Contains at least one connector: "suppliesFluidFrom" and at least one connector: "suppliesFluidTo"
BalancingDamper	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
MotorizedDamper	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
FireDamper	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
BalancingValve	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
CheckValve	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
DifferentialPressureValve	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
MotorizedValve	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
SafetyValve	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
ShuntValve	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
Fan	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
Pump	Contains 2 connectors Contains one connector: "suppliesFluidFrom" and one connector: "suppliesFluidTo"
AirTerminal	Contains 1 connector Contains connector of "suppliesFluidFrom" if supply system Contains connector of "suppliesFluidTo" if return system

Appendix B. UML class diagram FSC

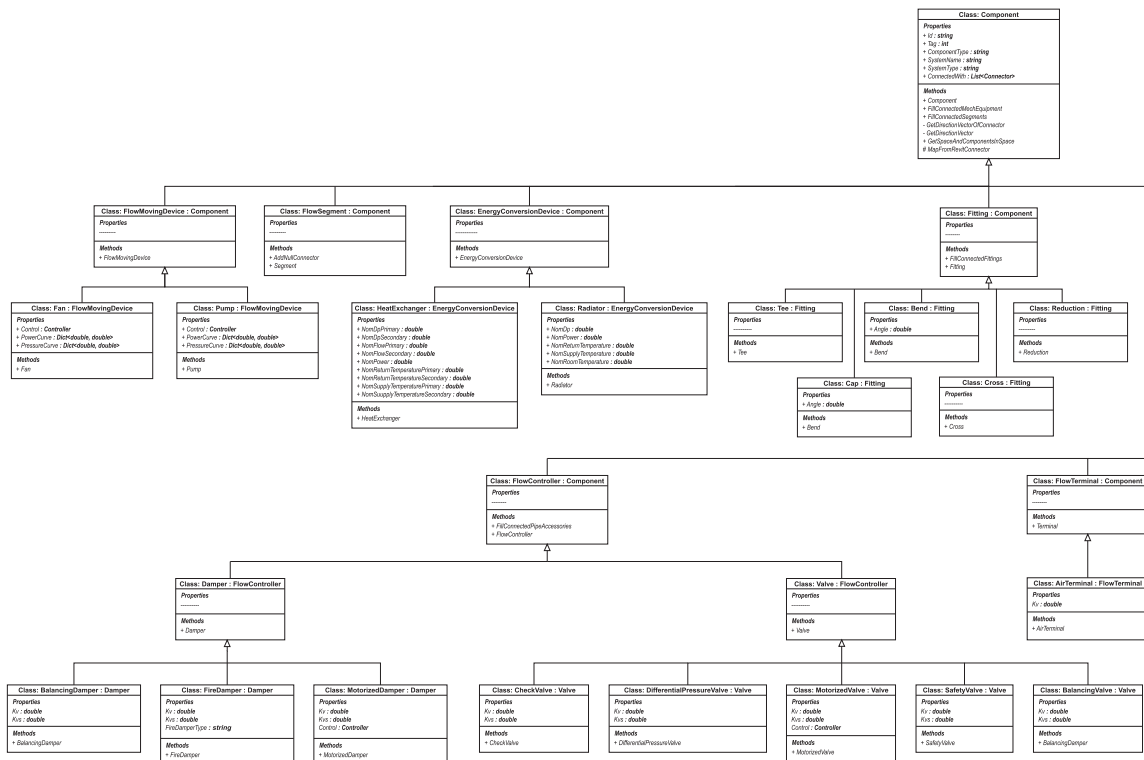


Fig. B.1. Illustration of the full FSC UML diagram. It includes all elements that inherit from component. It extends the diagram displayed in Fig. 2. The methods displayed in the UML are used to create the FSC object model.

References

- [1] K.M. Kensek, Handbook of green building design and construction, Build. Informa. Model. (2014) 1–285, <https://doi.org/10.4324/9781315797076>.
- [2] A. Andriamonjy, D. Saelens, R. Klein, An automated IFC-based workflow for building energy performance simulation with Modelica, Automation in Construction 91 (September 2017), 2018, pp. 166–181, <https://doi.org/10.1016/j.autcon.2018.03.019>.
- [3] B. Hardin, D. McCool, BIM and Construction Management: Proven Tools, Methods, and Workflows, 2nd edition, Sybex/Wiley, 2015.
- [4] F.H. Abanda, L. Byers, An investigation of the impact of building orientation on energy consumption in a domestic building using emerging BIM (Building Information Modelling), Energy 97 (2016) 517–527, <https://doi.org/10.1016/j.energy.2015.12.135>.
- [5] T.O. Olawumi, D.W. Chan, Identifying and prioritizing the benefits of integrating BIM and sustainability practices in construction projects: a Delphi survey of international experts, Sustain. Cities Soc. 40 (February) (2018) 16–27, <https://doi.org/10.1016/j.scs.2018.03.033>.
- [6] J.P. Carvalho, L. Bragança, R. Mateus, Optimising building sustainability assessment using BIM, Autom. Constr. 102 (2018) 170–182, <https://doi.org/10.1016/j.autcon.2019.02.021>. URL 10.1016/j.autcon.2019.02.021.
- [7] R.E. Edwards, E. Lou, A. Bataw, S.N. Kamaruzzaman, C. Johnson, Sustainability-led design: feasibility of incorporating whole-life cycle energy assessment into BIM for refurbishment projects, J. Build. Eng. 24 (February) (2019), 100697, <https://doi.org/10.1016/j.jobe.2019.01.027>. URL 10.1016/j.jobe.2019.01.027.
- [8] K. Safari, H. AzariJafari, Challenges and opportunities for integrating BIM and LCA: methodological choices and framework development, Sustain. Cities Soc. 67 (2020) 102728, <https://doi.org/10.1016/j.scs.2021.102728>.
- [9] T.P. Obrecht, M. Röck, E. Hoxha, A. Passer, BIM and LCA integration: a systematic literature review, Sustainability (Switzerland) 12 (14) (2020) 1–19, <https://doi.org/10.3390/su12145534>.
- [10] B. Succar, W. Sher, A. Williams, Measuring BIM performance: five metrics, Architect. Eng. Des. Manag. 8 (2) (2012) 120–142, <https://doi.org/10.1080/17452007.2012.659506>.
- [11] J. Beetz, N. Gu, BIMserver.org - an open source IFC model server, in: Proceedings of the CIB W78 2010, 2009, pp. 1–9. URL, https://www.academia.edu/1905765/BIMSERVER_ORG_AN_OPEN_SOURCE_IFC_MODEL_SERVER.
- [12] J.C. Cheng, M. Das, A bim-based web service framework for green building energy simulation and code checking, J. Inform. Technol. Construct. 19 (June) (2014) 150–168. URL, <http://www.itcon.org/2014/8>.
- [13] M. Wetter, C.V. Trecek, L. Helsen, A. Maccarini, D. Saelens, D. Robinson, G. Schweiger, IBPSA project 1: BIM / GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges, in: IOP Conf. Ser.: Earth Environ. Sci., 2019, <https://doi.org/10.1088/1755-1315/323/1/012114>.
- [14] G.B. Porsani, K.D.V. de Lersundi, A.S.O. Gutiérrez, C.F. Bandera, Interoperability between building information modelling (Bim) and building energy model (bem), Appl. Sci. 11 (5) (2021) 1–20, <https://doi.org/10.3390/app11052167>.
- [15] K.U. Ahn, Y.J. Kim, C.S. Park, I. Kim, K. Lee, BIM interface for full vs. semi-automated building energy simulation, Energy Build. 68 (PART B) (2014) 671–678, <https://doi.org/10.1016/j.enbuild.2013.08.063>. URL 10.1016/j.enbuild.2013.08.063.
- [16] C. Park, HVACSIM+ User's Guide Update, 2008, <https://doi.org/10.6028/NIST.IR.7514>. URL, <http://www.fire.nist.gov/bfrlpubs/build08/PDF/b08030.pdf>.
- [17] Equa, IDA Indoor Climate and Energy, URL, <https://www.equa.se/en/ida-ice>.
- [18] L. B. N. Laboratory, Modelica Buildings Library, URL, <https://simulationresearch.lbl.gov/modelica/>, 2022.
- [19] M. Wetter, Modelica-based modelling and simulation to support research and development in building energy and control systems, J. Build. Perform. Simul. 2 (2) (2009) 143–161, <https://doi.org/10.1080/19401490902818259>.
- [20] W. Zuo, M. Wetter, W. Tian, D. Li, M. Jin, Q. Chen, Coupling indoor airflow, HVAC, control and building envelope heat transfer in the Modelica Buildings library, J. Build. Perform. Simul. 9 (4) (2016) 366–381, <https://doi.org/10.1080/19401493.2015.1062557>.
- [21] E.V. Fjerbak, M. Seidenschneur, A. Küciükavci, K.M. Smith, C.A. Hviid, From BIM databases to Modelica - automated simulations of heating systems, in: REHVA 14th HVAC World Congress, 2022, pp. 1–7, <https://doi.org/10.34641/clima.2022.365>.
- [22] J.B. Kim, W. Jeong, M.J. Clayton, J.S. Haberl, W. Yan, Developing a physical BIM library for building thermal energy simulation, Autom. Constr. 50 (C) (2015) 16–28, <https://doi.org/10.1016/j.autcon.2014.10.011>.
- [23] W.S. Jeong, J.B. Kim, M.J. Clayton, J.S. Haberl, W. Yan, A framework to integrate object-oriented physical modelling with building information modelling for building thermal simulation, J. Build. Perform. Simul. 9 (1) (2016) 50–69, <https://doi.org/10.1080/19401493.2014.993709>.
- [24] D. Jansen, E. Fichter, V. Richter, A. Barz, J. Brunkhorst, M. Dahncke, P. Jahangiri, C. Warnecke, P. Mehrfeld, M. Dirk, C.V. Trecek, L. Bruno, R. Otto, M. Technik, C. Kg, BIM2SIM - Development of semi-automated methods for the generation of simulation models using Building Information Modeling BIM2SIM – Development of semi-automated methods for the generation of simulation models using Building Information Modeling (September), 2021, pp. 2–4.
- [25] A. Andriamonjy, R. Klein, D. Saelens, Automated grey box model implementation using BIM and Modelica, Energy Build. 188–189 (2019) 209–225, <https://doi.org/10.1016/j.enbuild.2019.01.046>. URL doi:10.1016/j.enbuild.2019.01.046.
- [26] S. Hauer, A. Bres, R. Parti, M. Monsberger, An approach for the extension of openBIM MEP models with metadata focusing on different use cases, Build. Simul. Conf. Proc. 1 (2019) 182–189, <https://doi.org/10.26868/25222708.2019.210932>.
- [27] P. Pauwels, S. Zhang, Y.C. Lee, Semantic web technologies in AEC industry: a literature overview, Autom. Constr. 73 (2017) 145–165, <https://doi.org/10.1016/j.autcon.2016.10.003>. URL 10.1016/j.autcon.2016.10.003.
- [28] K. Afsari, C.M. Eastman, D. Castro-Lacouture, JavaScript object notation (JSON) data serialization for IFC schema in web-based BIM data exchange, Autom. Constr. 77 (2017) 24–51, <https://doi.org/10.1016/j.autcon.2017.01.011>. URL 10.1016/j.autcon.2017.01.011.
- [29] D.Y. Lee, H.L. Chi, J. Wang, X. Wang, C.S. Park, A linked data system framework for sharing construction defect information using ontologies and BIM environments, Autom. Constr. 68 (2016) 102–113, <https://doi.org/10.1016/j.autcon.2016.05.003>.
- [30] C. Quinn, A.Z. Shabestari, T. Misis, S. Gilani, M. Litoiu, J.J. McArthur, Building automation system - BIM integration using a linked data structure, Autom. Constr. 118 (May) (2020), 103257, <https://doi.org/10.1016/j.autcon.2020.103257>.
- [31] S. Tang, D.R. Shelden, C.M. Eastman, P. Pishdad-Bozorgi, X. Gao, BIM assisted building automation system information exchange using BACnet and IFC, Autom. Constr. 110 (2019) 103049, <https://doi.org/10.1016/j.autcon.2019.103049>. URL 10.1016/j.autcon.2019.103049.
- [32] K. Kim, H. Kim, W. Kim, C. Kim, J. Kim, J. Yu, Integration of ifc objects and facility management work information using Semantic Web, Autom. Constr. 87 (2017) 173–187, <https://doi.org/10.1016/j.autcon.2017.12.019>.
- [33] B. Dong, Z. O'Neill, Z. Li, A BIM-enabled information infrastructure for building energy fault detection and diagnostics, Autom. Constr. 44 (2014) 197–211, <https://doi.org/10.1016/j.autcon.2014.04.007>.
- [34] A. Gouda Mohamed, M.R. Abdallah, M. Marzouk, BIM and semantic web-based maintenance information for existing buildings, Autom. Constr. 116 (March) (2020) 103209, <https://doi.org/10.1016/j.autcon.2020.103209>.
- [35] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M.B. Kjærgaard, M. Srivastava, K. Whitehouse, Brick: Towards a unified metadata schema for buildings, in: BuildSys'16: Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments, 2016, pp. 41–50, <https://doi.org/10.1145/2993422.2993577>.
- [36] C. Preidel, A. Borrmann, C. Oberender, M. Tretheway, Seamless integration of common data environment access into BIM authoring applications: the BIM integration framework, eWork and eBusiness in architecture, Eng. Construct. (2016) 119–128, <https://doi.org/10.1016/j.procs.2016.10.1201>. URL 10.1201/9781315386904.
- [37] L.D. Lauretis, From monolithic architecture to microservices architecture, IEEE Int. Symp. Software Reliability Eng. Workshops (ISSREW) (2019) 93–96, <https://doi.org/10.1109/ISSREW.2019.00050>.
- [38] J. Thönes, Microservices, IEEE Softw. 32 (1) (2015), <https://doi.org/10.1109/MS.2015.11>.
- [39] V. Kukkonen, A. Küciükavci, M. Seidenschneur, M.H. Rasmussen, K.M. Smith, C. A. Hviid, An ontology to support flow system descriptions from design to operation of buildings, Autom. Construct. 134 (2020) 104067, <https://doi.org/10.1016/j.autcon.2021.104067>.
- [40] M. Wetter, W. Zuo, T.S. Noudiui, X. Pang, Modelica Buildings library, J. Build. Perform. Simul. 7 (4) (2014) 253–270, <https://doi.org/10.1080/19401493.2013.765506>.
- [41] M. Wetter, T.S. Noudiui, D. Lorenzetti, E.A. Lee, A. Roth, Prototyping the next generation energyplus simulation engine, in: 14th International Conference of IBPSA - Building Simulation 2015, BS 2015, Conference Proceedings, no. April 2016, 2015, pp. 403–410. URL, <https://www.semanticscholar.org/paper/PROTOTYPING-THE-NEXT-GENERATION-ENERGYPLUS-ENGINE-Wetter-Noudiui/8faf8119752b54d456ab6e60ebf204ff7c74fc>.

6.2 Paper II - A Common Data Environment with an EnergyPlus microservice for Post-occupancy evaluation of the Energy Performance Gap

A Common Data Environment with an EnergyPlus microservice for Post-occupancy evaluation of the Energy Performance Gap

Mikki Seidenschneur^{a,b,*}, Ali Küçükavcı^{a,c}, Esben Visby Fjerbæk^a, Kevin Michael Smith^a and Christian Anker Hviid^a

^aDepartment of Civil Engineering, Technical University of Denmark, Brovej 118, 2800 Kgs. Lyngby, Denmark

^bRambøll, Copenhagen, Denmark

^cCOWI, Kgs. Lyngby, Denmark

ARTICLE INFO

Keywords:

Common Data Environments
Energy Performance Gap
Detailed Simulation Models

ABSTRACT

One of the significant challenges of the building sector is the absence of post-occupancy evaluation by the HVAC designers, leading to an energy performance gap between predicted performance and measured performance. This article introduces a common data environment for running detailed simulation models on cloud-based Building Information Modeling (BIM) models to bridge this gap. This article provides a system architecture for a common data environment capable of running Energy Plus simulations using three microservices. Finally, the article offers an application of the developed tools in a real-life use case using meter data provided by Frederiksberg School. The use case exemplified that the common data environment can successfully perform a post-occupancy evaluation of an existing building with a BIM model. Finally, the article presents a future works section describing the roadmap for future development and extensions for the common data environment.

1. Introduction


Researchers and experts in the Architecture, Engineering, Construction, and Operation (AECO) industry have reported a difference in the predicted (simulated) energy performance of buildings and measured energy performance during the operation of buildings, called the Energy Performance Gap (EPG) [1–6]. De Wilde [1] suggest that the main reason for the EPG, is that the designers cannot make realistic assumptions about the occupant behaviour of the building in their simulation models. De Wilde [1] suggests that Post-Occupancy Evaluation (POE) has the potential to minimize the EPG from predicted to measured energy performance by including actual occupant behaviour in the simulation models. Menezes et al. [6] suggests that one way to reduce the EPG is to implement detailed simulation models into the design process and follow it through to the operational phase. The detailed simulation model contains information like the building's occupancy behaviour and control algorithms. Though this means that the model during the design phase can still have wrongful assumptions about the occupant behaviour of the building, it can later be updated to the actual occupant behaviour in the building. This provides a digital twin to monitor and simulate scenarios, introducing a way to perform fault detection and diagnostics. Jradi et al. [7] showed that with the use of EnergyPlus detailed simulation models, they could better predict the energy consumption for the heating, ventilation and lighting in the building after providing more accurate boundary conditions for occupancy and actual weather conditions. With sensor data from the

existing building, detailed simulation models can more accurately predict the energy performance of the building to reduce the EPG.

The complexity of Building Energy Performance Simulation (BEPS) models are ever rising, and researchers are performing more and more advanced simulations of buildings [8, 9]. Wetter et al. [10] introduced IBPSA project 1, an advanced hydraulic simulation engine using the Modelica language. The IBPSA project 1 can be used to run detailed simulation models if the researcher/practitioner has the knowledge to use the libraries developed in the project. However, it does not provide an easy way for an industry practitioner to run detailed simulation models in an ever-changing design process. Other efforts have tried to handle this by creating an automatic transfer from a file-based Industry Foundation Classes (IFC) to a modelica simulation environment [9]. While providing a novel approach for automated data transfer, they are using a file-based approach that lacks data interoperability. De Wilde [1] states that one concern is that researchers in an academic setting carry out detailed simulation models that do not align with simulation models created by practitioners in the industry. In industry, it can be hard to see the monetary value of performing detailed simulation models as they are time-consuming and thus cost more money than simple simulations. Furthermore, as Menezes et al. [6] states, there is no follow-up from the industry experts, after the design phase, mainly caused by the cost of involving the designer in the post-occupancy period.

To make follow-up easier for the designer and client, Seidenschneur et al. [11] introduced a Common Data Environment (CDE) that is capable of transferring data from a BIM model to a centralized database, with the use of the Flow System Classes (FSC) object model. They suggested that future development of the CDE should include a detailed

*Corresponding author

 mse@ramboll.dk (M. Seidenschneur)

ORCID(s): 0000-0002-3881-9586 (M. Seidenschneur);

0000-0001-9883-4633 (A. Küçükavcı); 0000-0001-7694-8373 (E.V. Fjerbæk); 0000-0001-8148-9263 (K.M. Smith); 0000-0002-8340-7222 (C.A. Hviid)

simulation model module that can compare the predicted energy performance with the actual energy performance of the building.

There is a need for a CDE that can be used by the HVAC designer, from design to operation of the building to monitor the building performance and compare it to the predicted performance. The CDE should include a detailed simulation model that represents the physics of the building, i.e., a white-box model. After and during the commissioning of the building, the detailed simulation model will be used to reevaluate the performance of the building, based on the actual use of the building. Furthermore, the CDE should provide a monitoring system for the building once it has been constructed.

1.1. Aim

This article aims to provide a tool to aid the Heating, Ventilation, and Cooling (HVAC) engineer to perform POE of an existing building. The tool also allows the HVAC engineer to perform a detailed simulation model of the actual BIM model from the detailed design phase to the operational phase. Finally, we apply the created tools to a real-life use-case with gathered meter data to showcase that the tool can be used to perform a POE of buildings. All of this is novel because:

- we provide a reliable and fully modularized CDE, based on Microservice Architecture (MSA), allowing for Continuous Integration and Continuous Deployment of the developed microservices
- we provide the basis for an application that HVAC practitioners can use to perform POE, rather than being an academic-only tool
- we apply the provided tool to a real-life use case called Frederiksberg school

1.2. Paper outline

Section 2 lists the current state-of-the-art for POE and the use of CDEs in providing a platform for the automatic simulation of detailed simulation models. Section 3 provides a detailed description of the software developments offered in this article and an application of said developments in a real-life use case. Section 4 emulates a case where the HVAC practitioner is carrying out a POE of Frederiksberg School, intending to find the total EPG from predicted to measured energy performance. Section 5 provides a roadmap for future development of the CDE presented in this paper. Section 6 concludes on the contribution of this paper, and provides a roadmap for future development of the CDE

2. Background

2.1. Post-occupancy evaluation

AECO practitioners use POE to obtain feedback on the performance of a building, including energy and indoor climate performance [12]. POE uses three categories to improve the performance of the building [13]:

1. As a design tool: To improve building procurement, using data obtained to feed-forward in the next design of buildings. Menezes et al. [6] states that there is a lack of feedback since there rarely is any feedback to the HVAC designer once the building has been constructed and occupied. This means that the design team rarely benefits from the lessons learned in the building.
2. As a management tool: To measure the building performance in real-time to make changes in the operation. This is currently being performed to some extent in most buildings as a part of the Building Management System (BMS) installed in the building.
3. As a benchmarking tool: To have a benchmark to measure progress in sustainable goals such as energy consumption of the building. Finally, (3) using POE as a benchmarking tool to identify when components in the building need upgrading or replacing to improve performance. In this field, several advanced automatic fault detection and diagnostics systems have been proposed in the literature in the last decade [14–16]. Most of them include using Machine Learning (ML) to provide a black-box or grey-box model. Finally, some literature suggests a white-box model approach to accurately reflect the physical model [17–19]. Shi and O'Brien [19] reviewed the literature on automated fault detection and diagnostics and recommended some future challenges and possibilities. They proposed to use detailed simulation models as a basis to pre-simulate faults in the building system and use them for training an automated fault detection and diagnostics model grey-box model. Furthermore, they suggested that automatic fault detection and diagnostics programs should be centralized. Otherwise, it is hard to manage all the different types of faults that may occur in a system. Finally, they suggested that there should be sufficient digital infrastructure to handle a large amount of data and correlate it to the BIM model.

2.2. Detailed simulation models of buildings

EnergyPlus was introduced in the early 2000s as an effort to provide a new generation energy simulation program [20, 21]. Since then, several researchers have used EnergyPlus to perform whole building simulations in an effort to more accurately predict the energy performance of buildings [22–25]. However, the EnergyPlus engine was not designed to study the performance of HVAC systems, which can be better simulated with different programs [26] like Modelica. In recent years, the AECO industry has seen an increase in detailed simulation models using the Modelica language as an environment for simulation of HVAC systems [9, 10, 27, 28]. Modelica offers modularity of simulation engines and is open-source¹. Jradi et al. [7] showed that accurate, detailed simulation models have the potential to lower the energy

¹<https://modelica.org/modelicalanguage.html>

performance gap. However, as De Wilde [1] argued, the detailed simulation models are often used by the academic environment but not by practitioners in the AECO industry. This is caused by simulation engines like Modelica being complicated to work with, and it is a laborious task to transfer data manually from a BIM model into the Modelica environment [9].

2.3. Automatic generation of simulation models

A multitude of efforts have been carried out to transfer BIM data directly into simulation models [10, 27, 29–34]. For instance, Andriamamonjy et al. [27] created an automatic IFC-based workflow called IFC2Modelica that can transfer the data from an IFC-model to a Modelica environment. Furthermore, Wetter et al. [10] introduced a BIM/Geographic Information System (GIS) and Modelica framework for building and design, and operation. The main focus of the above efforts is to utilize the existing data formats of IFC or green building Extensible Markup Language (gbXML) to extract information relevant to perform automated or semi-automated BEPS. One problem with the IFC and gbXML format is that they often do not contain the necessary information to be able to perform a detailed simulation model of the energy and indoor climate in the context of the full HVAC system [35]. However, no one intended to store all relevant information for construction processes in IFC files, and no single super schema can support such a task [36, 37]. The AECO industry should move from file-based sharing towards a more network-based integration [38] including a database residing within a CDE to provide full interoperability from BIM to the simulation model.

2.4. Common data environments for design, commissioning, and operation

A CDE has the potential to ease the burden of performing manual file-based data exchange to simulation tools like EnergyPlus (E+). Jradi et al. [7] introduced an online building energy performance monitoring and evaluation tool to reduce the EPG, called Online building energy performance Monitoring and Evaluation (ObepME). They introduced ObepME as a monitoring and evaluation system that enables continuous commissioning into the building's life cycle. Furthermore, they developed an EnergyPlus model to take information from the BMS. Jradi et al. [7] used the implemented system to identify deviations in the expected energy performance of the building and to perform fault detection and diagnostics. Furthermore, ObepME can prevent over-consumption by notifying the operations specialist that some parts of the system measure a higher energy consumption than the dynamic energy simulation model predicted. However, ObepME does not provide a simulation based on a BIM model, making it a laborious task to implement in a construction process. Seidenschnur et al. [11] introduced a CDE based on a MSA principle to allow for easy scalability of the application in a cloud-based web application environment. Furthermore, they introduced an

object model called FSC, which provides the necessary data structure to represent a HVAC system in a BIM model. One microservice suggested by Seidenschnur et al. [11] was a tool that automatically generates a Modelica-based model and simulates it. Such a tool was developed by Fjerbæk et al. [39].

2.5. Summary

Research in POE suggests that a lot of recent efforts try to use black-box, grey-box, or white-box models to accurately predict faults and correct them. They all have potential to accurately predict faults of a HVAC system. However, as suggested by [1], most of these detailed simulation models are not utilized by practitioners in the AECO industry due to the time consumption of manually generated advanced detailed simulation models. Several efforts have tried to tackle the automatic generation of such models [10, 27, 29–34], but they all use the file-based data-exchange to generate said models. Seidenschnur et al. [11] introduced a CDE to move the BIM maturity level from 2 to 3. Furthermore, the contribution was a common way to export data from a proprietary BIM format into an open object-oriented database, like mongoDB. The CDE allows for a MSA and therefore prepares the application for a cloud-based infrastructure which was suggested by Shi and O'Brien [19]. This article continues the work of the CDE suggested by Seidenschnur et al. [11] and developed further by Fjerbæk et al. [39]. We will use the work developed by Seidenschnur et al. [11] to extend the CDE with a module for comparison of sensor data with the simulation results generated in the E+ microservice.

3. Methodology

This article propose an extension of the CDE system architecture created by [11], by adding microservices that includes a simulation engine in E+, an optimization algorithm to stage and dispatch different E+ files. Furthermore, to support the extended system architecture, we developed a data structure containing the thermal zones' necessary properties to perform simulations in E+, called the Thermal Zone Classes hierarchy. Following the detailed description of the extended system architecture, this section provides a use case description of Frederiksberg school to validate that the tools correctly transfer the BIM model into the database. Furthermore, the use case subsection will detail the use case selected for validating the toolchain.

3.1. System architecture

Seidenschnur et al. [11] created a CDE capable of taking a BIM HVAC model and centralizing it into a database. They showed that it was possible to incorporate a microservice architecture to create many services based on the user's specific needs. Figure 1 shows the revised system architecture. Seidenschnur et al. [11] also created a class hierarchy called FSC. FSC formulates a data structure to represent HVAC systems in a CDE. Furthermore, they introduced a preliminary data structure for spaces. However, this data

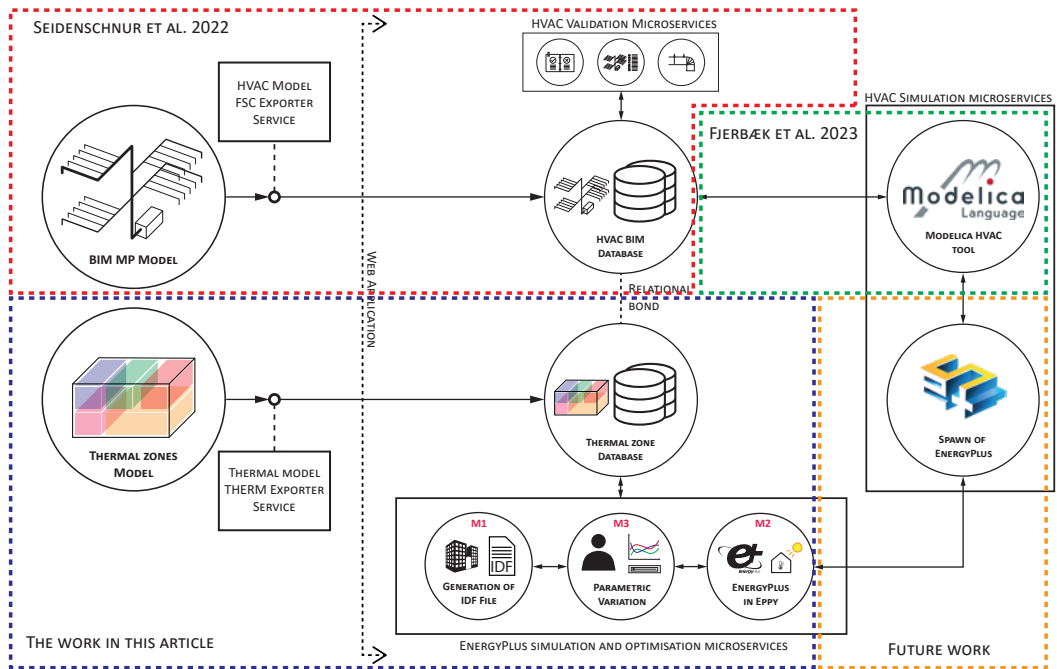


Figure 1: Original figure adapted from Seidenschnur et al. [11], marked with red. Blue marks the work developed in this article. Green marks the work developed by [39]. Orange marks the future work

1 structure was insufficient to represent thermal zones for 25
2 energy and indoor climate simulation in E+. Therefore, 26
3 section 3.1.1 introduces the Thermal Zone Classes hierarchy, 27
4 for the representation of thermal zones, with the boundary 28
5 conditions of rooms. 29

6 This section suggests two core developments to extend 30
7 the system architecture CDE developed by Seidenschnur 31
8 et al. [11]. The first core development is creating a data 32
9 structure capable of representing thermal zones in a build- 33
10 ing for simulation in energy and indoor climate simulation 34
11 tools. Section 3.1.1 introduces the development of Ther- 35
12 mal Zone Classes. The second core development involves 36
13 three microservices: Microservice M1 generates an Input 37
14 Data File (IDF) on which microservice M2 can perform an 38
15 E+ simulation. Furthermore, microservice M3 receives an 39
16 input that determines which parameter variation to run in 40
17 microservice M2. It then uses microservice M2 to run tens, 41
18 hundreds, or even thousands of simulations, for instance, 42
19 perform a parameter variation study for sensitivity analysis. 43
20 The development of the isolated microservices is presented 44
21 accordingly in sections 3.1.3, 3.1.4, and 3.1.5. The microser- 45
22 vices were developed in a framework called Flask, which is 46
23 referred to as a micro-framework. Flask provides an easy-to- 47
24 extend microservice template for future applications². 48

²<https://pythonbasics.org/what-is-flask-python/>

3.1.1. Thermal Zone Classes hierarchy

Figure 10 shows the proposed data structure for Thermal Zone Classes to perform indoor climate and thermal simulations. Figure 10 shows that there are six main pieces of information needed to perform an indoor climate simulation in E+: *Simulation parameters* represents the settings required to run a simulation. *Output parameters* includes what information should be output and in which format. *Schedules* defines varying loads, for instance, people load, lighting, activity level, ventilation, heating, varying setpoints in the building automation system, etc., over time. *Materials* represents the composition of materials in the construction. For instance, a window generally comprises several layers of material - two to three layers of varying glass and 1-2 air gaps. *Construction* represents the different types of wall layers - a wall consists of materials defined and ordered by its layers. Finally, the *Site* represents the actual building. The site is the container for the information generated in the previous five categories. The Thermal Zone Classes represent a tree structure used to describe the hierarchy of the building. A site contains a building, and site shading. The building contains zones and building shading. The zones contain surfaces, internal gains, HVAC systems, infiltration, and zone shading. The appendix includes a detailed Unified Modeling Language (UML) class diagram of all the classes created for the Thermal Zone Classes hierarchy. 49
50

3.1.2. Thermal Zone Classes hierarchy exporter

Often, commercial BIM software like Revit, contains proprietary data. Therefore, we have used Revit's C# Application Programming Interface (API) to map the Thermal Zone Classes hierarchy and then create an open Javascript Object Notation (JSON) file. To create an exporter for the Thermal Zone Classes from Revit to the CDE, the Revit API was used. The script used to export the Thermal Zone Classes hierarchy is found on GitHub³.

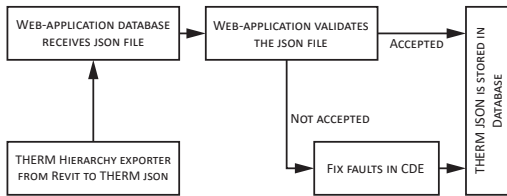


Figure 2: Process to get information from a Revit model of thermal zones into the CDE

Figure 2 shows a three-step concept allowing an exporter that maintains data integrity within the CDE. First, the Thermal Zone Classes exporter maps the data from the Revit model into the Thermal Zone Classes hierarchy described in section 3.1.1. The exporter uses Revit's C# API to loop through all analytical zones in the Revit model and maps them into the Thermal Zone Classes hierarchy. After mapping all thermal zones, surfaces, constructions, etc., in the Revit model and structuring them into the Thermal Zone Classes hierarchy, the tool serializes the data into a JSON-based model and sent to the CDE database. A rule set in the CDE web application then validates the integrity of the model file. If the model file is not compliant with the rule set, the CDE asks the user to fix the faults manually in the Revit model or the CDE. Finally, once the model is compliant, the CDE stores the it in the database.

3.1.3. Microservice M1 - Generation of IDF

The M1 microservice receives the JSON created by the Thermal Zone Classes hierarchy exporter, which is stored in the database and generates an IDF file from it. It uses the EnergyPlus Python (eppy)⁴ python package to structure the IDF file based on the rooms. Four functional parts divide the microservice:

- Identify missing parameters
- Populate missing parameters with default values
- Generate IDF-based E+ model with eppy
- Return IDF as result

First, the M1 microservice receives a Hypertext Transfer Protocol (http) *POST* request from the CDE, with a body

containing the Thermal Zone Classes JSON file. Then, the service identifies (1) if any parameters are missing to perform an eppy simulation. Such missing parameters could be schedules, simulation input parameters, etc. For this article, we generated a JSON file containing all of the default values needed if the Thermal Zone Classes JSON had missing information to generate a working IDF. Once the service has identified the missing parameters, it populates (2) them with default values. Then, it uses eppy to generate (3) an IDF. Then the microservice responds to the *POST* request by returning (4) the generated IDF.

3.1.4. Microservice M2 - E+ simulation

The M2 microservice uses the mentioned Python package called eppy and containerizes in a microservice. The M2 microservice takes; an IDF and an EnergyPlus Weather (EPW) file corresponding to the site in the body of the http *POST* request. The microservice then simulates the IDF and returns the result files of the E+ simulation. The microservice returns a JSON containing all the requested output files from E+. In this article, the "<filename>_hourly.json" and "<filename>_Table.csv" was returned. Several output files are available; please refer to the Output File list from the E+ documentation⁵. Furthermore, the M2 microservices also receives which version of E+ applies, via an Input Data Dictionary (IDD) file. The M2 microservice is found on Github⁶.

3.1.5. Microservice M3 - Parameter variation algorithm

The M3 microservice allows for users to generate new IDFs, based on the user demand. The M3 microservice generates a sensitivity analysis through a parameter variation of the detailed simulation model. Microservice M3 takes in a JSON file. Listing 1 shows the structure of the JSON. The "source_file" value determines the path of the IDF to perform a parameter variation on which the microservice should perform a parameter variation study. The "type_of_object" determines the object type. In this case, the object type is the material - but it might also be schedule, constructions, or any other object within the IDF. Figure 10 shows the objects within the IDF. The "name_of_object" is the ID of the objects to make the parameter optimization. The ID can be a list of several objects or a string containing the word "all." The "parameter_to_optimise" is the specific parameter within each object, like in this example, the thickness of the material, of which the algorithm should create a parameter variation. Finally, the "list_of_inputs" is the actual variations with which the user wants to run the simulation. Figure 3 shows that when the JSON is received by the http request to the microservice, it stages the IDF(s) and simulates them one-by-one in the M2 microservice presented in the previous paragraph. It then receives a result file from the EnergyPlus

⁵<https://bigladdersoftware.com/epx/docs/8-3/output-details-and-examples/output-file-list.html>

⁶<https://github.com/Virtual-Commissioning/VC-EnergyPlus-Simulation-Service>

³https://github.com/Virtual-Commissioning/VC-Analytical_zones_exporter-Service.git

⁴<https://pypi.org/project/eppy/>

microservice, which it stores on the server, to be extracted by the user.

```
{
  "source_file": "app/ressources/example.idf",
  "type_of_object": "Material",
  "name_of_object": ["BR01"],
  "parameter_to_optimise": "Thickness",
  "list_of_inputs": [0.25, 0.30, 0.40]
}
```

Listing 1: The listing shows the body of the HTTP request made to start the optimization algorithm

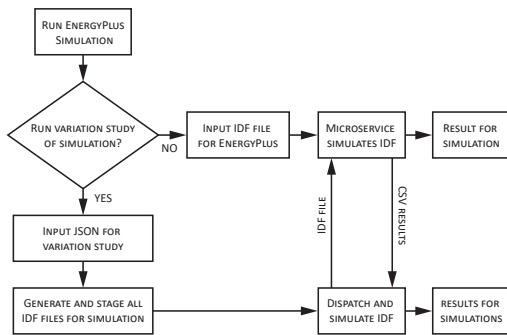


Figure 3: Process to run EnergyPlus simulation in microservice architecture presented in figure 1

3.2. Use case setup

The use case includes a real-life use case from Sorø, called Frederiksberg School. The sensor data collected from the school allows for a comparison of the actual energy performance of the school with the dynamically predicted performance of the school. Provided that the school does not perform according to the predicted performance, we will try to use the E+ microservices, described in section 3.1 to investigate the energy performance of the system and compare it to the actual performance of the building

3.2.1. Frederiksberg School

The case study includes a school in Sorø, Denmark, called "Frederiksberg Skole". Figure 6 shows that the school has three floors: a basement, a ground floor, and a first floor.

After a few years of operation, the facility management of the school has found out that the overall energy consumption of the school is much higher than anticipated. Furthermore, it shows that two wings divide the school, the northern wing and the southern wing. A Variable Air Volume (VAV)- and water-based heating system serves the north and south wings separately. Facility management has installed several meters to individually monitor the energy performance of all the major systems. Therefore, this article uses the northern building as a case study to allow for simulation of only the north wing in E+. The northern building consists of a total of 84 zones, with 16 classrooms, 6 group rooms, ten offices, 23 toilets, two large common areas, 27 smaller rooms

for technical storage, hallways, etc. The north wing zones amount to a total conditioned area of 3664 m²

3.2.2. From BIM to E+

This subsection presents the export of the Frederiksberg school BIM model. The subsection shows snippets from the JSON file exported with the use of the Thermal Zone Classes exporter. Figure 4 shows the Frederiksberg School BIM model, exported using the Thermal Zone Classes exporter tool presented in section 3.1.2. The turquoise zones marked in the figure represents the 84 zones exported from the BIM model. The shaded gray elements provides shading on the facade of the north wing, but also represents the south wing of the building.

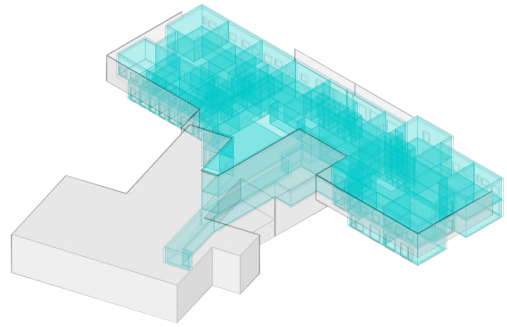


Figure 4: Screenshot of the analytical model exported from Revit.

Figure 4 also shows that only the rooms from the north wing are modeled in Revit and exported. The south wing provides shading for the north wing, and all walls neighboring the southern building are adiabatic. Figure 5 shows the three steps necessary to transfer the BIM model to the Thermal Zone Classes JSON and then into the IDF used for E+. The process involves the developed Thermal Zone Classes hierarchy, its exporter, and the M1 microservice.

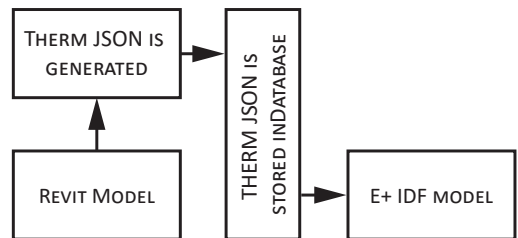


Figure 5: Illustration of the transfer from Revit to Thermal Zone Classes JSON and to an E+ IDF

The transfer of the BIM model into the Thermal Zone Classes JSON was successful, using the Thermal Zone Classes hierarchy exporter. The Thermal Zone Classes hierarchy exporter modeled 241 materials, 210 constructions, and 84 zones. Listing 2 shows an example of a material exported to the Thermal Zone Classes JSON.

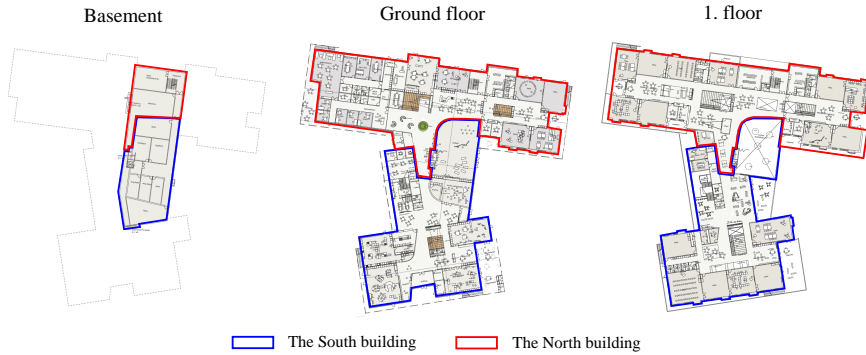


Figure 6: The plan drawings of "Frederiksberg Skole" in Sorø, Denmark. Blue marks the south wing, and red marks the northern wing.

```

1 {
2   "7577371_0.01":{
3     "ReadableName":"StoTherm_facadeputs",
4     "Name":"7577371_0.01",
5     "Roughness":null,
6     "Thickness":0.01,
7     "Conductivity":1.046,
8     "Density":2300.0,
9     "Specific_Heat":657.0,
10    "Thermal_Absorptance":null,
11    "Solar_Absorptance":null,
12    "Visible_Absorptance":null
13  }

```

Listing 2: The listing shows an example of a material expressed in the Thermal Zone Classes JSON

As listing 2 shows, not all values could be extracted from the Revit model into the Thermal Zone Classes JSON. The CDE will automatically provide default values for the roughness, thermal absorptance, solar absorptance, and visible absorptance. Listing 3 shows an example of a construction modeled in the Thermal Zone Classes JSON.

```

22 {
23   "7529784":{
24     "Name":"7529784",
25     "Layers":[
26       {
27         "Layer1":"5237675_0.005"
28       },
29       {
30         "Layer2":"134479_0.25"
31       },
32       {
33         "Layer3":"5237675_0.005"
34       },
35       {
36         "Layer4":"4503105_0.22"
37       }
38     ]
39   }
40 }

```

Listing 3: The listing shows an example of a construction expressed in the Thermal Zone Classes JSON

Listing 3 shows a construction labeled 7529784. 7529784 contains 4 layers in total. The layers are

5237675_0.005, 134479_0.25, 5237675_0.005, and 4503105_0.22. Those layers are roofing, insulation, roofing, and a concrete slab. Finally, listing 4 shows an example of a zone mapped in the Thermal Zone Classes class hierarchy.

```

48 {
49   "Name":"7602777",
50   "X-Origin":0.0,
51   "Y-Origin":0.0,
52   "Z-Origin":0.0,
53   "Type":"NoSpaceType",
54   "Ceiling_Height":2.438,
55   "Floor_Area":13.011,
56   "Volume":31.720818,
57   "Zone_Inside_Convection_Algorithm":"",
58   "Zone_Outside_Convection_Algorithm":"",
59   "Part_of_Total_Floor_Area":true,
60   "Surfaces":[...],
61   "InternalGains":[...],
62   "HVAC":[...],
63   "Infiltration":[...],
64   "ZoneShadings":[...]
65 }

```

Listing 4: The listing shows an example of a zone expressed in the Thermal Zone Classes JSON. The "..." notes that more information is available, but not shown for simplicity

Listing 4 shows the main information contained within the zone model of the Thermal Zone Classes JSON. The most important properties of the zone model are the *Surfaces*, *InternalGains*, *HVAC*, *Infiltration*, and *ZoneShadings*. Listing 5 illustrates an example of what builds the geometry of the zone, but also the construction of the walls.

```

73 {
74   "7602778":{
75     "Name":"7602778",
76     "Surface_Type":"Wall",
77     "Construction_Name":"7490160",
78     "Zone_Name":"7602777",
79     "Outside_Boundary_Condition":"Outdoors",
80     "Outside_Boundary_Condition_Object":"",
81     "Sun_Exposure":true,
82     "Wind_Exposure":true,
83     "View_Factor_to_Ground":"",
84     "VertexCoordinates":[
85       {

```

```

14      "X": -146.3753,
15      "Y": -1600.5608,
16      "Z": 50.74
17    },
18    { ... },
19    { ... },
20    { ... }
21  ]
22 }

```

Listing 5: The listing shows an example of a surface expressed in the Thermal Zone Classes JSON

Listing 5 shows that every surface contains information on: *Name*, *Surface Type*, *Construction Name*, *Zone Name*, etc. Most importantly, it contains information that relates the geometry of the surfaces to the zone. Listing 5 illustrates the vertex coordinates of the specific surface 7602778. After the Thermal Zone Classes hierarchy exporter has exported the Revit model, the M1 microservice was employed to generate an IDF based on the Thermal Zone Classes JSON. To see the full JSON generated by the M1 microservice, refer to GitHub⁷

3.2.3. Inputs for E+ simulation

After the Thermal Zone Classes hierarchy exporter created the JSON in section 3.2.2, we used the M1 microservice to generate an IDF. The IDF generated by the M2 microservice is provided on GitHub⁸ and shown in figure 7.

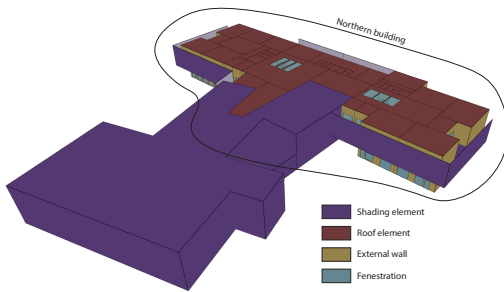


Figure 7: The generated E+ model visualized in Open Studio

The simulation input data is provided in the form of the original IDF used to run the E+ simulation. To ensure similar inputs, the data provided by the sensors from Frederiksberg School, was used to generate setpoint schedules, lighting schedules, internal heat gain schedules, etc. Furthermore, Danmarks Meteorologiske Institut at the Flakkebjerg, Denmark weather station provided the weather data from Sorø. The weather data is available in the GitHub repository for microservice M2 in the resources folder. The weather file represents the weather recorded from January 1, 2020, to December 31, 2020.

⁷<https://github.com/Virtual-Commissioning/>

VC-EnergyPlus-Opt-Service/tree/main/app/ressources

⁸<https://github.com/Virtual-Commissioning/>

VC-EnergyPlus-Opt-Service/tree/main/app/ressources

3.2.4. BMS and sensor data

In terms of a data dump of all loggers set-up for the HVAC system, the sensor data was provided by Frederiksberg School. The data was collected in the period from July 1, 2020, to June 30, 2021. Due to the time period, the data contains one heating season for validation of the heating system. The data has the heating energy consumed by all sub-systems. Such heating consumption includes heating for the ventilation heating coils for the north- and south wings, the mixing loop for the north and south radiator systems, and the heating coil for the hot water production. The data dump also contains the CO₂-level, temperature, damper opening, and thermostatic valve opening for 30 rooms. The room sensors have been randomly selected from the entire building, containing rooms from both the south- and north wings. The room sensor data is not be used further in this article.

4. Results

This section means to go through the process of performing a simple POE of Frederiksberg school with the use of the CDE developed in this article, with the coherent E+ microservices introduced in Section 3.1. The purpose is to show that the CDE is capable of creating a connection from BIM to E+ simulation and then later make the data comparable to measured data. First, we compare the predicted energy performance, based on a simulation model derived from the existing BIM model, to the measured meter data of Frederiksberg school. After illustrating the EPG of this building, we use the microservices created in Section 3.1 to perform a simple sensitivity analysis to try and identify if specific parameters have a higher effect on the EPG than others. Finally, we will not provide a detailed POE of the sensor and meter data from Frederiksberg school, since it is beyond the scope. The purpose is to show provide a proof-of-concept tool that can later be used to perform POE of the EPG.

4.1. Predicted vs. Measured energy performance

The M1 and M2 microservices were used on Frederiksberg School, as shown in section 3.2. Figure 8 shows the predicted heating energy use, calculated with the M2 microservice, compared to the actual measured heating energy use of Frederiksberg school. The figure shows that the predicted heating energy use is lower than the measured heating energy use of the actual building. In total, the predicted energy consumption was 5.16 MWh less than the actual measured energy consumption of Frederiksberg School.

Figure 8 also shows that the predicted heating energy use varies only slightly in the year's first two months. However, as soon as March comes around, the measured heating energy use differs significantly from the predicted heating energy use. For instance, the heating energy use for March is 10 MWh higher than predicted by the E+ simulation model. For most of the summer, there was no heating energy use on for heating, except for the measured heating consumed in June of 1.7 MWh. There is no measured heating energy

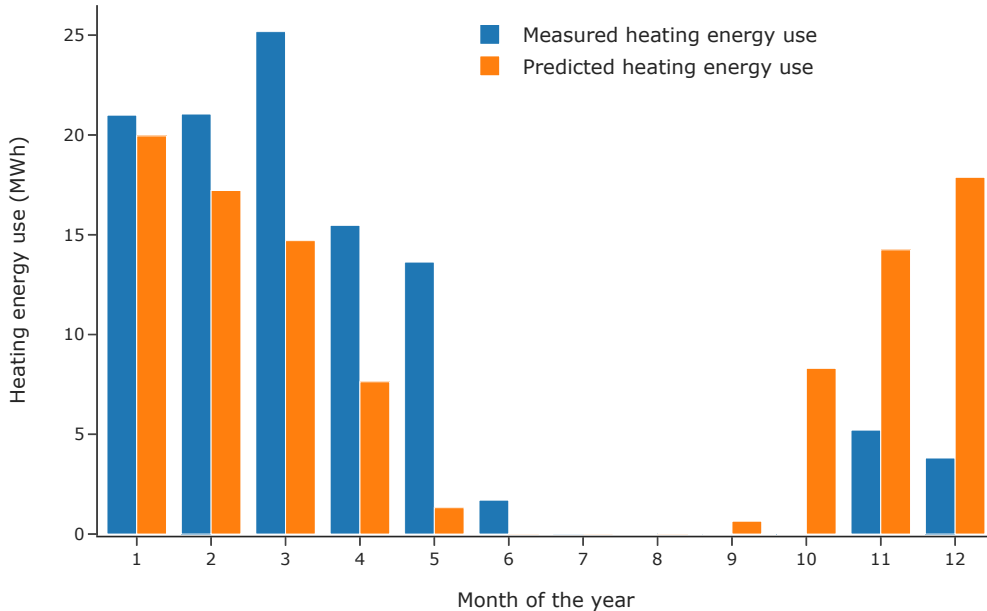


Figure 8: Bar chart showing the predicted heating energy use vs. measured heating energy use of Frederiksberg School

use for the building until November. Figure 8 shows that the predicted heating energy use was much higher than the simulation model for October, November, and December.

4.2. Evaluating the EPG through microservices

Figure 8 shows an EPG of 5.16 MWh, amounting to a gap of 5%. Though the yearly aggregated sum represents a small EPG, figure 8 shows a significant variation monthly. The variation means several parameters can have an underlying wrong assumption throughout the simulations. For instance, the setpoints obtained from the measurements of Frederiksberg school are likely dynamic - facility management might have changed them along the way. The system, however, did not log the setpoints for the zones. Not having the logged setpoints is highly problematic when analyzing how the HVAC system works. A manual inspection of the BMS of the school shows a current setpoint (reading: 09/02/2022) for the zones varying between 21-22°C. In the nighttime, the heating setpoint is lowered to 17°C to save heating energy. We do not have a specific schedule for lowering the heating setpoint. Therefore, we can assume some variation in the heating setpoints. Consequently, we used

microservice M3 to understand how the heating setpoint for the zone changes the predicted heating energy use.

Figure 9 shows that the heating system lowers the heating setpoint for the zones in the building, and there is a potential to save energy. However, the simulation does not explain the large discrepancy from prediction to measurements in March to May or November to December. In this case, facility management needs to log the heating setpoints for the building need to provide a more accurate simulation. Since the period is during a partial coronavirus lockdown, facility management likely changed the heating setpoints dynamically during the measurement period, making it hard for the researchers to predict the heating setpoints. Furthermore, another reason for the sudden increase in heating for February through May might be caused by the return of the school students after the coronavirus lockdown. One of the official strategies from the government of Denmark was to make sure that public institutions, like schools, had a high supply of fresh air. This meant that most schools opened the windows fully in every classroom, when students were present. This could contribute to the excessive heating demand seen from February (where the students

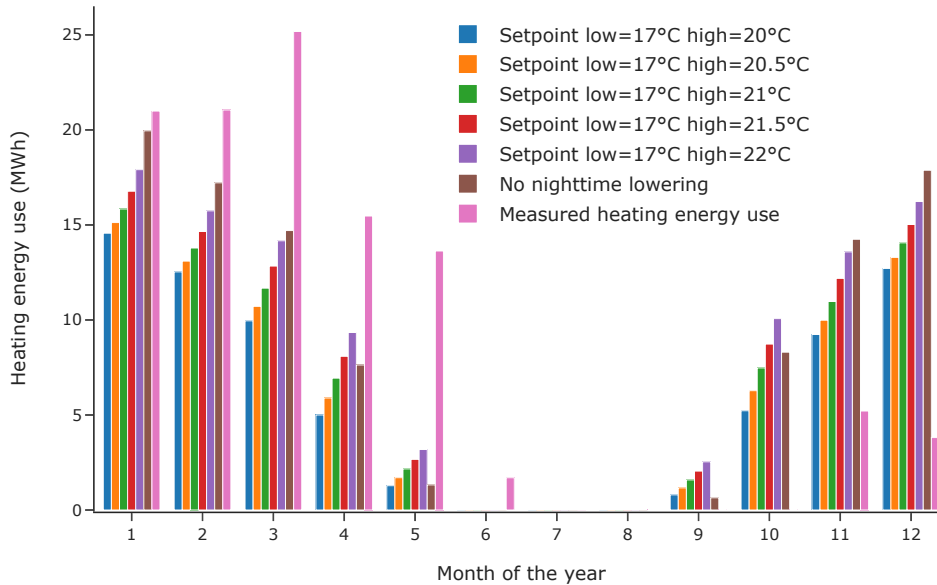


Figure 9: Illustration showing the energy consumption of where the heating setpoint varies

returned to school) until May, where the heating season ended. Finally, the lower than predicted heating energy use in October through December is believed to have been caused by coronavirus lockdown - meaning that the school likely lowered the heating setpoint for the rooms, while few or no students were coming to class in the period, due to schools providing online/hybrid classes.

5. Discussion

5.1. Contribution

The main contribution of this article, is that through setting up a CDE that controls the data flow from predicted to measured energy performance, HVAC engineers can efficiently perform a form of POE or continuous commissioning of any given building. A simple use case is that the meter data will be readily available for facility management. While this sounds like a simple contribution, often, the problem is that the building owners do not know that the building is not performing according to the original intent. The unknown performance usually means that after several years of poor performance, and a large amount of wasted energy, someone notices it due to expensive energy bills. By simply making this data available from the initial commissioning

of the building, poor performance can be flagged from the beginning and lower energy consumption significantly. The CDE presented in this paper shows how a modularized system architecture provides a workflow to develop and deploy microservices in support of other microservices. The Thermal Zone Classes hierarchy and the Thermal Zone Classes hierarchy exporter made in .NET Framework, using the Revit API has been made publicly available and open-source for other contributions⁹. Furthermore, the microservices to perform an E+ simulation and parameter variation has been made available. We suggest that the microservices presented in this paper is used in a CDE, but they can be used autonomously.

5.2. Future work

Figure 1 shows the planned future work of the CDE developed in this paper and by Seidenschur et al. [11] and Fjerbæk et al. [39]. It builds on the work created in Spawn-Of-EnergyPlus (SOEP) project, where Functional Mock-up Unit (FMU)s are used to co-simulate an HVAC system together with whole-building simulation. The current stage of the CDE does not support HVAC system generation within

⁹https://github.com/Virtual-Commissioning/VC-Analytical_zones_exporter-Service

E+, which means that the E+ simulation model is currently created with idealized HVAC systems. For future work, the HVAC system generated with the use of the FSC hierarchy from Seidenschur et al. [11] could be used to create HVAC systems in E+, as presented in the documentation of E+¹⁰. While the Thermal Zone Classes hierarchy presents a relatively generic data structure for representing thermal zone data for whole-building simulation, it is also a data structure mapped based on the parameters needed to perform simulations in E+. For future purposes, the Thermal Zone Classes hierarchy can easily be extended for other simulation engines, such as ESP-r. This article presented a simple parameter variation algorithm in the M3 microservice. It exemplified that such microservices are feasible. However, future work should implement a generic optimization algorithm, like GenOpt [40]. Furthermore, for the deployment of the system architecture requires a refactoring of the code base. This code presents the proof-of-concept of a CDE but is not robust and tested enough to be directly deployed. This article revised the system architecture suggested by Seidenschur et al. [11]. Energy simulations are generally computationally costly to run (i.e., processing time). This CDE indicates a solution to such a problem. Since the system architecture is represented in a microservice architecture, it is possible to scale the application horizontally when running an extensive optimization. For future work, this tool should be deployed in a cloud environment to allow for faster simulation times on large optimization projects. A data management strategy should be in place for every construction project, from the beginning to enable optimized use of the CDE. Generally, one issue with collecting data from buildings without a data management strategy is that there is either not enough data, not the right data points, or even so much data that there is no chance of finding the stakeholder with the information to link all the data. Furthermore, we suggest developing a standard data structure to address the gap between the digital model and the physical model's sensors if we hope these should be generically linked together. We manually found the room IDs in the project material for this project and connected them with the room IDs in the BIM model.

6. Conclusion

The aim of the article was to provide a tool to aid the HVAC engineer in performing POE of existing buildings. We have provided a CDE where a BIM model can be converted from a Revit format into the Thermal Zone Classes hierarchy and later serialized into a JSON that can be stored in a database. By storing the information in a CDE, we have enabled access between all of the project stakeholders. The aim was also to enable the HVAC engineer to perform a detailed simulation model of an existing building, already modeled in a BIM format. By providing the Thermal Zone Classes hierarchy, Thermal Zone Classes hierarchy exporter microservice M1, microservice M2, and microservice M3, we have enabled the HVAC engineer to transfer a BIM

model into an E+ simulation. Furthermore, microservice M3 provides the basis to perform a parameter variation, illustrating that an advanced generic algorithm could be employed to perform higher-level optimizations of the E+ models for future work. Finally, we applied the tools created in this article to a real-life use case with measured data to showcase that the system architecture developed in this article holds the potential to provide a POE tool for HVAC engineers.

7. Acknowledgements

This work was carried out while the main author was employed in Ramboll Denmark. The work is funded by the Ramboll Foundation and the Innovation Fund Denmark (grant 9065-00266A). A proof-of-concept was developed as a part of Jon Martin Tangeraas and Frederik Seeborg's Master thesis project in 2022. We would like to acknowledge the Frederiksberg school and the municipality of Sorø, for providing the data-set for Frederiksberg school.

A. Thermal Zone Classes hierarchy UML class diagram

References

- [1] P. De Wilde, The gap between predicted and measured energy performance of buildings : A framework for investigation, *Automation in Construction* 41 (2014) 40–49.
- [2] P. X. Zou, X. Xu, J. Sanjayan, J. Wang, Review of 10 years research on building energy performance gap: Life-cycle and stakeholder perspectives, *Energy and Buildings* 178 (2018) 165–181.
- [3] R. Galvin, Making the 'rebound effect' more useful for performance evaluation of thermal retrofits of existing homes: Defining the 'energy savings deficit' and the 'energy performance gap', *Energy and Buildings* 69 (2014) 515–524.
- [4] D. Cali, T. Osterhage, R. Streblow, D. Müller, Energy performance gap in refurbished German dwellings: Lesson learned from a field test, *Energy and Buildings* 127 (2016) 1146–1158.
- [5] J. Liang, Y. Qiu, M. Hu, Mind the energy performance gap: Evidence from green commercial buildings, *Resources, Conservation and Recycling* 141 (2019) 364–377.
- [6] A. Menezes, Carolina, A. Cripps, D. Bouchlaghem, R. Buswell, Predicted vs. actual energy performance of non-domestic buildings : Using post-occupancy evaluation data to reduce the performance gap, *Applied Energy* 97 (2012) 355–364.
- [7] M. Jradi, K. Arendt, F. C. Sangogboye, C. G. Mattera, E. Markoska, M. B. Kjærgaard, C. T. Veje, B. N. Jørgensen, ObepME: An online building energy performance monitoring and evaluation tool to reduce energy performance gaps, *Energy and Buildings* 166 (2018) 196–209.
- [8] M. Wetter, Modelica-based modelling and simulation to support research and development in building energy and control systems, *Journal of Building Performance Simulation* 2 (2009) 143–161.
- [9] A. Andriamamonjy, D. Saelens, R. Klein, An automated IFC-based workflow for building energy performance simulation with Modelica, *Automation in Construction* 91 (2018) 166–181.
- [10] M. Wetter, C. v. Treeck, L. Helsen, A. Maccarini, D. Saelens, D. Robinson, G. Schweiger, IBPSA Project 1: BIM / GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges, in: *IOP Conf. Ser.: Earth Environ. Sci.*, 2019. doi:10.1088/1755-1315/323/1/012114.

¹⁰https://eppy.readthedocs.io/en/latest/HVAC_Tutorial.html

A web-based approach for Post-Occupancy Evaluation

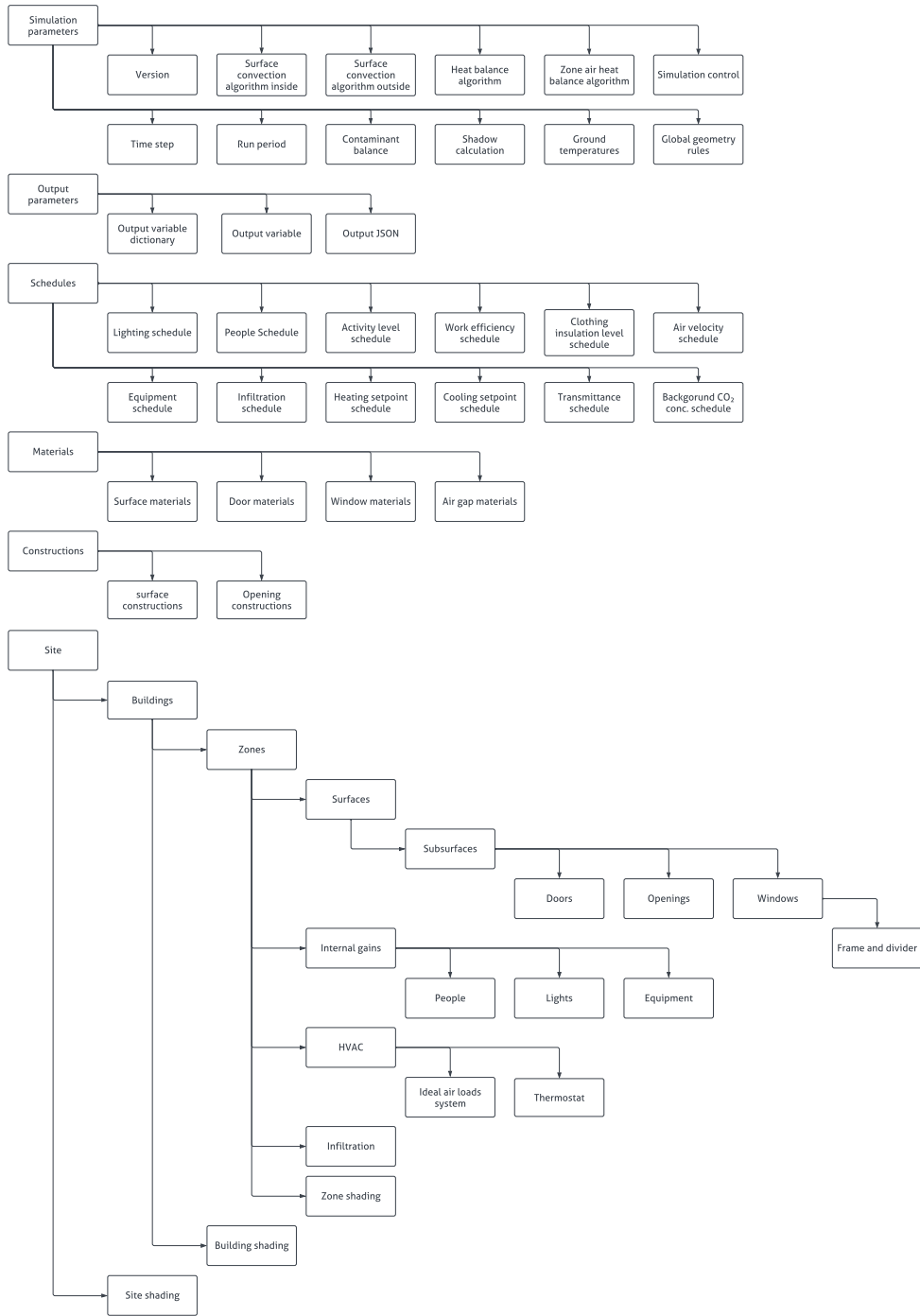


Figure 10: Proposed data structure for the representation of thermal zones prepared for energy, indoor climate, and thermal simulations in EnergyPlus.

- [11] M. Seidenschur, A. Küçükavci, E. Visby, K. Michael, P. Pauwels, C. Anker, A common data environment for HVAC design and engineering, *Automation in Construction* 142 (2022) 104500.
- [12] P. Li, T. M. Froese, G. Brager, Post-occupancy evaluation: State-of-the-art analysis and state-of-the-practice review, *Building and Environment* 133 (2018) 187–202.
- [13] I. Cooper, Post-occupancy evaluation - Where are you?, *Building Research and Information* 29 (2001) 158–163.
- [14] C. H. Lo, P. T. Chan, Y. K. Wong, A. B. Rad, K. L. Cheung, Fuzzy-genetic algorithm for automatic fault detection in HVAC systems, *volume 7*, 2007. doi:10.1016/j.asoc.2006.06.003.
- [15] M. Dey, S. P. Rana, S. Dudley, Smart building creation in large scale HVAC environments through automated fault detection and diagnosis, *Future Generation Computer Systems* 108 (2020) 950–966.
- [16] G. Zimmermann, Y. Lu, G. Lo, Automatic HVAC fault detection and diagnosis system generation based on heat flow models, *HVAC and R Research* 18 (2012) 112–125.
- [17] D. Picard, M. Sourbron, F. Jorissen, J. Cigler, Z. Vána, Comparison of Model Predictive Control Performance Using Grey-Box and White-Box Controller Models of a Multi-zone Office Building, *4th International High Performance Buildings Conference* (2016) 4.
- [18] Filip Jorissen, Damien Picard, Kristoff Six, Lieve Helsén, Detailed White-Box Non-Linear Model Predictive Control for Scalable Building HVAC Control, in: *Proceedings of 14th Modelica Conference 2021*, Linköping, Sweden, September 20–24, 2021, volume 181, 2021, pp. 315–323. doi:10.3384/ecp21181315.
- [19] Z. Shi, W. O'Brien, Development and implementation of automated fault detection and diagnostics for building systems: A review, *Automation in Construction* 104 (2019) 215–229.
- [20] D. B. Crawley, L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, Y. J. Huang, C. O. Pedersen, R. K. Strand, R. J. Liesen, D. E. Fisher, M. J. Witte, J. Glazer, *EnergyPlus* : creating a new-generation building energy simulation program 33 (2001).
- [21] B. Drury, O. Curtis, K. Linda, C. Frederick, *EnergyPlus* : Energy simulation program, *ASHRAE Journal* 42 (2000) 49–56.
- [22] G. Peter, A. Paul, B. Drury, Simulation of energy management systems in energypplus (2014) 2014.
- [23] K. W. C. Dahanayake, C. L. Chow, Studying the potential of energy saving through vertical greenery systems: Using EnergyPlus simulation program, *Energy and Buildings* 138 (2017) 47–59.
- [24] Y. P. Zhou, J. Y. Wu, R. Z. Wang, S. Shiochi, Y. M. Li, Simulation and experimental validation of the variable-refrigerant-volume (VRV) air-conditioning system in EnergyPlus, *Energy and Buildings* 40 (2008) 1041–1047.
- [25] M. Basarkar, X. Pang, L. Wang, P. Haves, T. Hong, Modeling and simulation of HVAC faults in EnergyPlus, *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association* (2011) 2897–2903.
- [26] T. N. Stephane, M. Wetter, W. Zuo, Functional Mock-up Unit for Co-Simulation Import in EnergyPlus, *Journal of Building Performance Simulation* 7 (2014) 192–202.
- [27] A. Andriamamonjy, R. Klein, D. Saelens, Automated grey box model implementation using BIM and Modelica, *Energy and Buildings* 188–189 (2019) 209–225.
- [28] D. Jansen, E. Fichter, V. Richter, A. Barz, J. Brunkhorst, M. Dahncke, P. Jahangiri, C. Warnecke, P. Mehrfeld, M. Dirk, C. V. Treeck, L. Bruno, R. Otto, M. Technik, C. Kg, BIM2SIM -Development of semi-automated methods for the generation of simulation models using Building Information Modeling BIM2SIM – Development of semi-automated methods for the generation of simulation models using Building Information Modeling (2021) 2–4.
- [29] S. Pinheiro, R. Wimmer, J. O'Donnell, S. Muhic, V. Bazjanac, T. Maile, J. Frisch, C. van Treeck, MVD based information exchange between BIM and building energy performance simulation, *Automation in Construction* 90 (2018) 91–103.
- [30] E. Kamel, A. M. Memari, Review of BIM's application in energy simulation: Tools, issues, and solutions, *Automation in Construction* 97 (2019) 164–180.
- [31] F. Jalaei, A. Jade, An Automated BIM Model to Conceptually Design, Analyze, Simulate, and Assess Sustainable Building Projects, *Journal of Construction Engineering* 2014 (2014).
- [32] B. Welle, J. Haymaker, Z. Rogers, ThermalOpt: A methodology for automated BIM-based multidisciplinary thermal simulation for use in optimization environments, *Building Simulation* 4 (2011) 293–313.
- [33] K. U. Ahn, Y. J. Kim, C. S. Park, I. Kim, K. Lee, BIM interface for full vs. semi-automated building energy simulation, *Energy and Buildings* 68 (2014) 671–678.
- [34] I. Kim, J. Kim, J. Seo, Development of an IFC-based IDF converter for supporting energy performance assessment in the early design phase, *Journal of Asian Architecture and Building Engineering* 11 (2012) 313–320.
- [35] G. B. Porsani, K. D. V. de Lersundi, A. S. O. Gutiérrez, C. F. Bandera, Interoperability between building information modelling (Bim) and building energy model (bem), *Applied Sciences (Switzerland)* 11 (2021) 1–20.
- [36] A. Redmond, A. Hore, M. Alshawi, R. West, Exploring how information exchanges can be enhanced through Cloud BIM, *Automation in Construction* 24 (2012) 175–183.
- [37] O. Donnell, R. Sec, C. Rose, T. Maile, V. Bazjanac, P. Haves, SIM-MODEL: A DOMAIN DATA MODEL FOR WHOLE BUILDING ENERGY SIMULATION, in: *SimBuild 2011 IBPSA Conference*, 2012. URL: <https://escholarship.org/uc/item/70c7j74t>.
- [38] B. Succar, W. Sher, A. Williams, Measuring BIM performance: Five metrics, *Architectural Engineering and Design Management* 8 (2012) 120–142.
- [39] E. V. Fjerbæk, M. Seidenschur, A. Küçükavci, K. M. Smith, C. A. Hviid, Coupling Modelica simulations and a Common Data Environment for BIM (2023).
- [40] M. Wetter, GenOpt - A Generic Optimization Program, *Seventh International IBPSA Conference* (2001) 601–608.

6.3 Paper III - An ontology to support flow system descriptions from design to operation of buildings



Contents lists available at ScienceDirect

Automation in Construction

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

An ontology to support flow system descriptions from design to operation of buildings

Ville Kukkonen^{a,b,*}, Ali Küçükavcı^c, Mikki Seidenschur^{c,d}, Mads Holten Rasmussen^e, Kevin Michael Smith^c, Christian Anker Hviid^c

^a Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland

^b Granlund, Helsinki, Finland

^c Department of Civil Engineering, Technical University of Denmark, Copenhagen, Denmark

^d Ramboll, Copenhagen, Denmark

^e NIRAS, Allerød, Denmark

ARTICLE INFO

Keywords:

Building information modeling
HVAC
Semantic web
Ontology
Linked data

ABSTRACT

The interoperability of information from design to operations is an acknowledged challenge in the fields of architecture, engineering and construction (AEC). As a potential solution to the interoperability issues, there has been increasing interest in how linked data and semantic web technologies can be used to establish an extendable data model. Semantic web ontologies have been developed for the AEC domain, but an ontology for describing the energy and mass flow between systems and components is missing. This study proposes the Flow Systems Ontology (FSO) for describing the composition of flow systems, and their mass and energy flows. Two example models are expressed using FSO vocabulary. SPARQL Protocol and RDF Query Language (SPARQL) queries are performed to further demonstrate and validate the ontology. The main contribution consists of developing FSO as an ontology complementary to the existing ontologies. Finally, the paper introduces a roadmap for future developments building on FSO.

1. Introduction

The stakeholders in the architecture, engineering and construction (AEC) industry collaborate on complex, multidisciplinary projects that span years and produce large quantities of information besides the finished physical product. Since the introduction of Building Information Modeling (BIM), the level of information in the models has increased to support even more complex use cases and additional disciplines. While BIM authoring tools widely support the Industry Foundation Classes (IFC) data model, which enables vendor-independent information exchange, it has its challenges in terms of adaptability and extensibility [1]. Further, the information storage and exchange are generally based on static documents, which poses challenges such as propagating changes in dynamically derived information [2].

In addition to the information produced during design and construction, the increasing amount of data collection in building operation and maintenance (O&M) has put pressure on improving data

interoperability between AEC stakeholders. Advancements in computing and sensing technology as well as practical access to real-time data of buildings have made data analytics increasingly applicable in the O&M of buildings [3,4]. Recently, BIM models have been used to inform the configuration and deployment of building management services such as automated fault detection and diagnostics [5–7]. Although building automation system (BAS) metadata has been successfully represented in IFC, there are still concerns as to whether IFC is a suitable data model for representing information such as BAS control and communication [8]. Additionally, as many existing buildings have been designed and built without BIM, the deployment of building services requires other, more lightweight formats for representing information acquired from targeted buildings [9]. The lack of data interoperability in BIM has contributed to a performance gap between the constructed building and the BIM model [10].

The World Wide Web Consortium (W3C) with their academic and industrial partners have published a set of standards that support the

* Corresponding author at: Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland.

E-mail addresses: ville.kukkonen@aalto.fi (V. Kukkonen), alikuc@byg.dtu.dk (A. Küçükavcı), msei@ramboll.dk (M. Seidenschur), mhra@niras.dk (M.H. Rasmussen), kevs@byg.dtu.dk (K.M. Smith), cah@byg.dtu.dk (C.A. Hviid).

implementations of semantic web technologies, with a vision of “web of data” beyond the “web of documents”.¹ The technologies include Resource Description Framework (RDF) [11], SPARQL Protocol and RDF Query Language (SPARQL) [12], and Web Ontology Language (OWL) [13]. RDF is used for describing graphs of data as sets of triples consisting of a subject, a predicate, and an object, as illustrated in Fig. 1. SPARQL defines a language for formulating queries against graphs expressed in RDF. Finally, OWL enables the definition of ontologies, which codify vocabularies and support reasoning over RDF data by defining the semantics of those vocabularies.

The AEC industry has seen an increasing interest in semantic web technologies [1]. The technologies have been proposed as a foundation for solving some of the interoperability issues, and for moving from document-based collaboration towards network-based collaboration [1, 2]. Semantic web technologies, in particular ontology development and their applications, have also been actively studied in the field of industrial manufacturing. Applications range from integrating legacy data sources in the design and operation of manufacturing processes [14], to supporting smart Product-Service Systems integrating information from design, manufacturing, and operation of the delivered products [15]. Product-Service Systems are concerned with integrating service offerings with delivered products in order to guarantee operation. While similar information integration requirements and opportunities for new value creation exist in the AEC industry, one challenge is that the information models are rarely focused on the operation phase. Embracing linked data and ontologies is one approach to enable information models that can support both design and operation [15].

Within the AEC industry, the W3C Linked Building Data (LBD) Community Group² has communicated the benefits of semantic web technologies and linked data by delivering use cases and best practices to the AEC industry since it was established in 2014. Efforts to improve interoperability using the semantic web technologies include the development of ontologies to describe common semantic vocabularies. To this end, the community group has published ontologies for the AEC industry. Other groups and researchers have also proposed ontologies for various subdomains in different levels of abstraction [7,16–19].

While ontologies for describing certain aspects of the AEC domain have been developed and published, a common standardized ontology to describe the energy and mass flow between the components is missing. Such an ontology is needed to support the linked data descriptions of Heating, Ventilation and Air Conditioning (HVAC) systems. This paper will describe a proposed ontology named Flow Systems Ontology (FSO) to represent the composition of flow systems and their energy and mass flow relationships. The proposed ontology is complementary to the ontologies already proposed by the community. FSO enables lightweight, machine-understandable common descriptions of the HVAC components' relationships. The vision is to provide a common foundation for describing flow systems in linked data. This would enable extensions that focus on more specific information perspectives for applications such as hydraulic simulation suites, building energy performance simulation (BEPS), building analytics, and diagnostics.

In this paper, two research questions are considered: do the currently existing ontologies support descriptions of energy and mass flow connections between systems and components such as seen in the building

systems domain; and if not, could such an ontology be created to support industry use cases in design and operation of buildings? The work has been initiated by industrial partners to address identified gaps in information exchange.

The prominent ontologies related to buildings and their systems are briefly described in Section 2. Following that, the proposed Flow Systems Ontology is described in detail in Section 3, including alignments to some of the existing ontologies. In Section 4, example models of a heating system and an active chilled beam system will be expressed using FSO vocabulary to demonstrate that FSO can describe a given flow system. Additionally, examples of queries enabled by FSO will be demonstrated as a means of validating the ontology and to simulate use cases from both design and operation. In Section 5, the results are discussed and the research is placed in the context of a roadmap of transforming existing information sources to linked data and using that linked data to support information interoperability. Finally, conclusions are presented in Section 6.

2. Background

This section describes a selection of the different ontologies developed for buildings and their systems.

An early effort in bridging the gap between BIM and semantic web technologies is the ifcOWL ontology, which maps the IFC schema to OWL [20]. While ifcOWL is expressed in OWL, it is a very direct translation from the EXPRESS schema of IFC, and is thus not optimized for linked data and semantic web applications [21]. Specifically, it covers multiple domains and uses intermediate aggregation objects which are unidiomatic in RDF.

Building Topology Ontology (BOT) was developed by the W3C Linked Building Data Community Group as a lightweight ontology for describing the connections of zones and elements in a building [21]. BOT constitutes a core vocabulary for supporting multiple subdomains within the AEC industries. It is designed to be extendable to more specific domains as needed. In short, BOT consists of a class taxonomy for zones (building sites, buildings, storeys, and spaces), building elements, which may have different spatial relationships to spaces, and interfaces that qualify connections between zones or between zones and elements.

Brick [22] ontology describes building data points, and it originated as a translation of the Haystack tagging framework to semantic web technologies. While some flow relationships between components are present in Brick, it focuses on classifying the different data points. Because the ontology revolves around data points, the scope of the terminology excludes passive components, such as pipes and ducts. The ontology is not intended to describe entire flow systems, including the distribution systems of ducts and pipes. As such, while it is a potential part of a comprehensive description of an operational building, extensions or other ontologies are required for describing the remaining aspects.

The Smart Energy Aware Systems (SEAS) ontology developed in the EUREKA ITEA 12004 SEAS project [23] consists of multiple modules. At the core are several more abstract ontologies, of which the most relevant in this context is the ontology describing systems and their connections. Specifically, the systems and connections module of SEAS contains classes and relationships to describe virtually isolated systems and their composition and connections. However, the requirement that each system may be a subsystem of at most one supersystem, enforced by making `seas:subSystemOf` functional, is incompatible with overlapping systems common in HVAC systems. For instance, a heating coil in an air handling unit can be considered a part of both the heating system and the ventilation system, which cannot be expressed with the SEAS systems module.

The latest version 3.1.1 of the Smart Applications REFERENCE (SAREF) [24] with its extensions for building devices (SAREF4BLDG) version 1.1.2 [25] and system typology patterns (SAREF4SYST) version 1.1.2 [26] published by the European Telecommunications Standards



Fig. 1. An illustration of the RDF data model of subject, predicate, and object.

¹ <https://www.w3.org/standards/semanticweb>

² <https://www.w3.org/community/lbd>

Institute (ETSI) integrates lessons learned from SEAS. As mentioned previously in this section, the SEAS systems are required to have at most one supersystem. This requirement is not present in SAREF4SYST, making it more suitable for describing overlapping systems. However, SAREF4SYST is an upper-level ontology defining systems and their connections, and is not specific enough to describe flow systems. SAREF4BLDG, on the other hand, takes building devices from IFC and describes their taxonomy in OWL [25]. SAREF4BLDG constitutes a subset of the domains described in ifcOWL. The translation incorporates the existing device taxonomy from IFC while avoiding the issues of ifcOWL discussed previously in this section. However, SAREF4BLDG does not consider passive components such as pipes and ducts, nor the component connections.

Real Estate Core (REC) is a modular ontology developed to support data integration for smart buildings [27]. It is developed by a consortium that includes major real estate companies. The purpose of REC is to semantically describe a building in the operation phase from the perspective of the building owner.

In summary, previous work includes several ontologies for describing the devices, spaces, and data points in a building. While a fictive Flow Systems Ontology was mentioned in [21], and an ontology module with the same name was submitted as a pull request to SEAS,³ the work presented here represents a new development from the same ideas. The FSO ontology proposed in this paper is complementary to the outlined ontologies and is not intended to supersede any of them but rather augment some of them. Additionally, SAREF4SYST describes systems and their connections, providing a useful foundation or “upper-level” ontology for describing flow relationships in building systems. While FSO is not a direct extension to SAREF, alignments to SAREF4SYST and SAREF4BLDG are proposed. Conceptually, FSO could be considered a middle-level ontology between SAREF and more domain-specific ontologies, such as REC.

3. Flow Systems ontology

Flow Systems Ontology is an ontology for describing the energy and mass flow relationships between systems and their components, and the composition of such systems. FSO consists of 14 classes and 23 object properties and has a Description Logic expressivity of ALRI. Instances of `fso:System` are virtual collections of components, which can be assigned properties such as design requirements, while instances of `fso:Component` are the tangible objects and devices that are involved in the flow of energy or mass. While this study and the examples focus on building systems, FSO is not intended to be strictly limited to those. While not investigated, it seems feasible that the same abstractions should be applicable in, for example, industrial process systems.

One way to conceptualize a building and its systems is to consider them as two parallel hierarchies with some links between them: one for the spaces, and one for the systems and components. Combining FSO and BOT gives the vocabulary necessary for such a conceptualization, as illustrated in Fig. 2.

The rest of this section provides an overview of the ontology and introduces some of its capabilities. The latest version of the ontology is documented online.⁴ First, in Section 3.1, competency questions are enumerated to help describe the scope of the ontology. Next, the ontology terms are described in two subsections: Section 3.2 introduces system composition and component classification, and Section 3.3 introduces the relationships between systems and components. After that, some examples of the reasoning enabled by the ontology are shown in Section 3.4. Finally, proposed alignments to other ontologies are described in Section 3.5.

3.1. Competency questions

The competency questions (CQ) for FSO are listed below. Their purpose is to illustrate and define the scope of the ontology by defining questions a model should be able to answer when using FSO.

- 1 What components does a flow system contain?
- 2 What subsystems does a flow system contain?
- 3 Given a flow system, which components are sources or consumers of mass or energy?
- 4 What components are in the same fluid loop as a given component?
- 5 What components are up/downstream of a component, and in what order?
- 6 Given a system with components in a fluid loop, which components are on the supply side and which are on the return side?
- 7 What kind of component is a given component?

The competency questions will be referred to when discussing the specifics of the ontology. This will be done using a shorthand, such as (CQ1) referring to the first competency question.

3.2. System composition and component classification

Fig. 3 shows an overview of the primary terminology of the ontology: `fso:System` and `fso:Component` and their relations. Instances of `fso:System` are virtual collections of components, which may have properties assigned to them. Examples of potential system properties include design specifications, such as supply and return water temperature for a heating system. Systems may have subsystems via `fso:hasSubSystem` (CQ2), and a system may be a subsystem of more than one supersystem via the inverse property `fso:isSubSystemOf`. This means that systems may overlap, sharing common subsystems. Sub-properties of the `fso:hasSubSystem` can be used to describe a more specific composition hierarchy (CQ6):

- `fso:hasSupplySystem` links a system to its logical “supply” subsystem. For example, a ventilation system could denote all components between the outside and a ventilated zone as a part of its supply system.
- `fso:hasReturnSystem` links a system to its logical “return” subsystem. For example, a ventilation system could denote all components between the ventilated zone and the outside as a part of its return system.

Additionally, `fso:System` has more specific subclasses that can be used to denote them being the logical supply and return systems:

- `fso:DistributionSystem` is the class of systems that are used to distribute mass and/or energy, linking other systems.
- `fso:SupplySystem`, a subclass of `fso:DistributionSystem`, is the class of systems that are used to supply mass and/or energy to downstream consumers.
- `fso:ReturnSystem`, a subclass of `fso:DistributionSystem`, is the class of systems that are used to return mass and/or energy from downstream consumers.

`fso:Component` and its subclasses are the tangible objects that participate in the flow of mass or energy. A system may have components via the `fso:hasComponent` property (CQ1), which has the inverse property `fso:isComponentOf`. The subclasses of `fso:Component` are based on the IFC taxonomy of `IfcDistributionFlowElement`,⁵ and represent high-level component types applicable

³ <https://github.com/thSMARTenergy/seas/pull/21>

⁴ <https://w3id.org/fso>

⁵ <https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2/HTML/lin/ifcdistributionflowelement.htm>

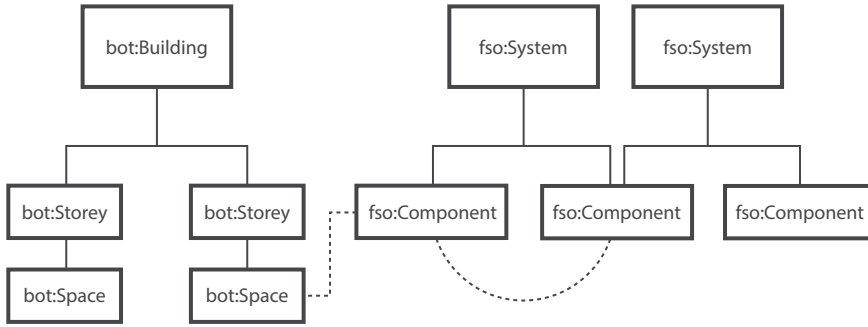


Fig. 2. Buildings can be conceptualized as two parallel tree hierarchies and described using BOT for spaces and FSO for systems. The building can be decomposed into storeys and spaces, while the systems can be decomposed into subsystems and components. Additionally, there exist horizontal relationships between the elements. An example of a horizontal relationship between an `fso:Component` and a `bot:Space` could be the space containing a radiator (`bot:containsElement`), and the radiator transferring heat to the space (`fso:transfersHeatTo`). Two instances of `fso:Component` would be connected with a subproperty of `fso:connectedWith`, such as a duct supplying an air terminal (`fso:suppliesFluidTo`).

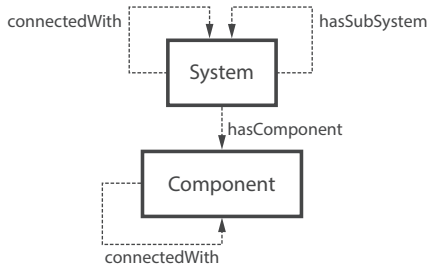


Fig. 3. Illustration of the top-level classes and properties in FSO.

to different contexts. The component classes are defined as follows (CQ7):

- `fso:EnergyConversionDevice` is defined as a “device that is used to convert energy from one form to another, or move it from one system to another”. Examples include devices such as motors and heat exchangers.
- `fso:Fitting` is defined as a “component used to connect segments to other segments, or to other components”, with examples such as pipe junctions.
- `fso:FlowController` is defined as a “device that has the potential to control the flow in a network”. Examples include valves and dampers.
- `fso:FlowMovingDevice` is defined as a “device used to induce movement in a network”, such as pumps and fans.
- `fso:Segment` is defined as a “component used to enable the passage of mass or energy”. That is, pipes, ducts, and cables.
- `fso:StorageDevice` is defined as a “device used to store mass or energy”, such as a battery or a tank for holding water.
- `fso:Terminal` is defined as “a device through which a system interacts with the environment”. Examples include air terminals, heating or cooling terminals, and water taps.
- `fso:TreatmentDevice` is defined as “a device that removes something unwanted from the matter flowing through it”, with examples such as air filters.

These abstract classifications of components together with the flow relationships enable a variety of use cases, which will be demonstrated in Section 4. If gaps in the classifications are identified by future use cases, the component classes can be revised to account for those.

Components can be asserted to be sources or consumers of systems, denoting the relative directions of energy or mass flows (CQ3). Such relationships can be useful for use cases that involve heating or cooling demand calculations. This is achieved with the following properties:

- `fso:hasSourceComponent` links a system to a component that acts as the source of energy or matter into that system. For example, a heating system may contain a heat exchanger that is connected to a district heating loop as an energy source.
- `fso:hasConsumerComponent` links a system to a component that acts as a consumer of energy or matter from that system. For example, a heating system would have radiators that consume energy by transferring heat to the indoor air of a room.

It is worth noting that a component being a source or a consumer is not an intrinsic property of the component, but rather a relation to a system. For example, a heating coil in a ventilation system will be a source of heat for the ventilation system, but a consumer of heat for the heating system. Similarly, a fan in the ventilation system could be considered a consumer of the electrical network, while not having an assigned source or consumer role in the context of the ventilation system.

3.3. Flow relationships

Besides system composition, FSO introduces a vocabulary to describe the flow of energy and matter between the systems and components. A generic `fso:connectedWith` property connects components or systems, and the subproperties can be used to denote more specific relationships in terms of energy and mass flows. An overview of the property hierarchy under `fso:connectedWith` is shown in Fig. 4. The first two layers, or all properties ending in `With`, are symmetrical, while their subproperties ending in `To` or `From` are not.

In particular, the `fso:exchangesFluidWith` subproperty of `fso:connectedWith` has two layers of specificity. The property `fso:feedsFluidTo` and its inverse `fso:hasFluidFedBy` denote the flow order of fluid in a system. `fso:suppliesFluidTo` and `fso:returnsFluidTo` denote the logical supply and return of fluid from a component or system to another (CQ5 & CQ6), and are subproperties of `fso:feedsFluidTo`. These properties can be used to break down a fluid loop of components and systems feeding to another to two sides: a supply and a return side. Note, however, that a `fso:suppliesFluidTo` property path from system A to B to C does not necessarily mean that it is the same fluid flowing through A and C. For systems like water to water heat exchangers, it might be intuitive to denote that the system is being supplied from one side while it itself supplies another

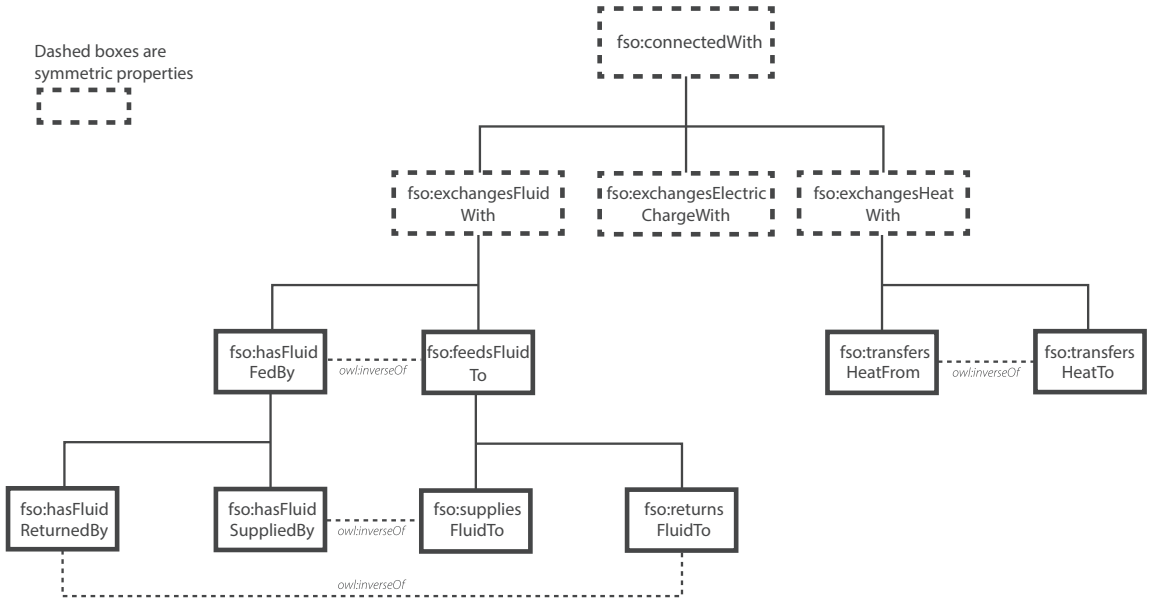


Fig. 4. Illustration of the properties describing fluid and energy flow relationships.

side, even though in reality the fluid is in two separate loops and hence completely separated. However, in order to be able to follow individual fluid loops (CQ4), such systems should be represented as multiple components. Continuing with the water to water heat exchanger example, a heat exchanger has a primary and a secondary side. FSO includes reasoning mechanisms that help inferring the implicit knowledge in these situations. This will be further elaborated in Section 3.4.

Using only `fso:feedsFluidTo` in SPARQL property path expressions with the zero-or-more or one-or-more syntax has the problem that it goes through the whole loop, as illustrated in Fig. 5. For example, in a system with a heat exchanger feeding fluid to two radiators that feed fluid back to the heat exchanger, going either forward or backward through `fso:feedsFluidTo` will connect the radiators. FSO introduces the more specific properties `fso:suppliesFluidTo` and `fso:returnsFluidTo` to break up the loops and accommodate this problem.

The heat transfer connections have a hierarchy as well. `fso:exchangesHeatWith` is a subproperty of `fso:connectedWith`, and in turn has two directional subproperties that are inverses of each other: `fso:transfersHeatTo` and `fso:transfersHeatFrom`. While the

property `fso:exchangesHeatWith` is symmetric, the subproperties are not.

Finally, the ontology defines a third kind of connectivity, `fso:exchangesElectricChargeWith`. While its subproperties have not been detailed at this stage, the property has been included as it is recognized to be similar to, but different from, the two previously defined connections. Like the other direct subproperties of `fso:connectedWith`, it too is defined as a symmetric property.

3.4. Reasoning examples

In addition to explicit assertions, semantic web technologies enable deductive reasoning to produce new assertions. This section shows some examples of the reasoning that FSO enables.

Besides components having connections to other components, system-level connections can be inferred from the system composition and component connections. The flow relationship properties discussed in Section 3.3 utilize property chain axioms to imply the existence of system-level connections. If there exists a component that is connected to another component with e.g. `fso:suppliesFluidTo`, and those

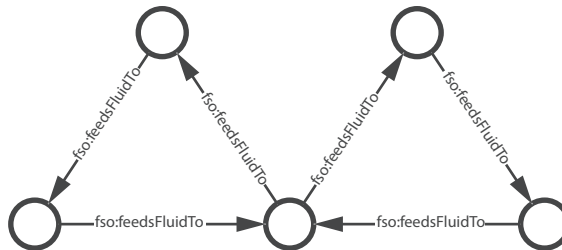


Fig. 5. Illustration of `fso:feedsFluidTo` path connecting two branches of a fluid loop.

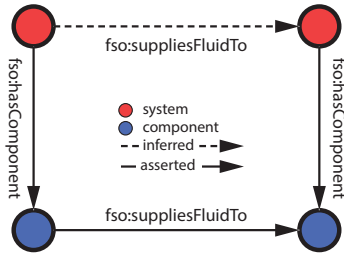


Fig. 6. Illustration of system connections inferred from component connections.

components are part of different systems, then those two systems are also connected by `fso:suppliesFluidTo`, as shown in Fig. 6. This means that if a pipe in a heating system supplies fluid to a heating coil that is also a part of an air handling unit, then the heating system can be inferred to have a `fso:suppliesFluidTo` connection to the air handling unit. Note, that because a component can be a component of many systems, overlapping systems are inferred to have these relationships. Additionally, these property chain axioms make the properties reflexive for the systems, i.e. the system on both ends of the property chain can be the same, leading to a self-referential object property to be inferred.

As was mentioned in Section 3.3, systems such as water to water heat exchangers can be modeled as two components to allow tracking of specific fluid circuits. Modeling the heat exchanger as a single component and connecting it with fluid supply upstream and downstream would result in the model appearing as if all components upstream of the heat exchanger also supply fluid to the components downstream of the heat exchanger. If the heat exchanger is instead modeled as a system with two components, a primary side and a secondary side with an `fso:transfersHeatTo` relationship, this confusion can be avoided. An example of this is illustrated in Fig. 7. The connection between the sides is shown as `fso:exchangesHeatWith`, as it is the symmetrical superproperty of `fso:transfersHeatTo`, and will be asserted in both directions by the reasoner.

With the model as shown in Fig. 7, the reasoning combined with SPARQL property path queries enable tracking the fluid and heat flows.

Finding the components that are in the same fluid circuit as, for example, the component supplying the primary side of the heat exchanger, is achieved by following the path `fso:suppliesFluidTo*/fso:returnsFluidTo*`. The `*` in a path expression denotes zero or more matches, and the `/` denotes a sequence path. If a different view of the network is desired, including those on the other side of such thermal connections, the property `fso:exchangeHeatWith` could be included in the path.

Additionally, with the model as shown in Fig. 7, the reasoner will infer a system-level connection between the heat exchanger system and the systems around it, as discussed earlier in this section. This means that the heat exchanger system can be inferred to e.g. supply fluid to a heating system. Similarly, if the primary side is a component of a district heating system, it is inferred to transfer heat to the heating system.

3.5. Alignments

SAREF4SYST introduces an ontology pattern for modeling systems and their connections. The alignment of FSO to SAREF4SYST is done by asserting the relevant FSO classes and properties as subclasses and subproperties of their more general counterparts in SAREF4SYST, as shown in Table 1. In particular, both `fso:System` and `fso:Component` are aligned as subclasses of `s4syst:System`.

FSO component taxonomy is largely based on the same IFC taxonomy as SAREF4BLDG, so the alignments are straightforward. However, SAREF4BLDG only considers devices and not passive elements, such as segments and fittings. Additionally, SAREF4BLDG devices are specific to buildings, while FSO does not place such requirements. As such, the classes from SAREF4BLDG are subclassed from FSO, as shown in Table 2.

Table 1
Alignments between FSO and SAREF4SYST.

Class / property	Subclass/subproperty of
<code>fso:System</code>	<code>s4syst:System</code>
<code>fso:Component</code>	<code>s4syst:System</code>
<code>fso:connectedWith</code>	<code>s4syst:connectedTo</code>
<code>fso:hasSubSystem</code>	<code>s4syst:hasSubSystem</code>
<code>fso:isSubSystemOf</code>	<code>s4syst:subSystemOf</code>
<code>fso:hasComponent</code>	<code>s4syst:hasSubSystem</code>
<code>fso:isComponentOf</code>	<code>s4syst:subSystemOf</code>

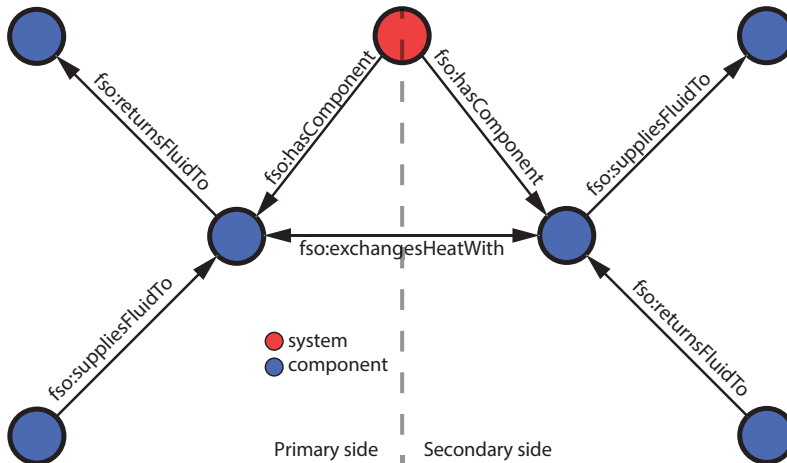


Fig. 7. Example of a heat exchanger modeled as two separate components exchanging heat with each other.

Table 2

Alignments between FSO and SAREF4BLDG.

Class	Subclass of
s4bldg:DistributionFlowDevice	fso:Component
s4bldg:EnergyConversionDevice	fso:EnergyConversionDevice
s4bldg:FlowController	fso:FlowController
s4bldg:FlowMovingDevice	fso:FlowMovingDevice
s4bldg:FlowStorageDevice	fso:StorageDevice
s4bldg:FlowTerminal	fso:Terminal
s4bldg:FlowTreatmentDevice	fso:TreatmentDevice

Table 3

Namespaces for prefixes used in the paper.

Prefix	Namespace
fso	https://w3id.org/fso#
bot	https://w3id.org/bot#
s4bldg	https://saref.etsi.org/saref4bldg#
s4syst	https://saref.etsi.org/saref4syst#
rdfs	http://www.w3.org/2000/01/rdf-schema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
ex	https://example.com/ex#
inst	https://example.com/inst#

4. Example models and use case demonstrations

In this section, example models and use cases are presented to demonstrate the use of FSO. The Turtle and SPARQL files of the examples are openly available in Zenodo [28]. The namespaces for prefixes used throughout the paper are shown in Table 3.

As FSO itself only defines a core vocabulary for describing flow systems in terms of abstract component classes and their relationships,

the use cases depend on the use of other ontologies. This demonstrates that while FSO itself does not define all the vocabulary for the use cases, semantic models of the flow systems form structural skeletons that can be augmented with further information.

First, Section 4.1 introduces an example model of an active chilled beam system for one room in order to demonstrate overlapping systems. Section 4.2 describes an example model of a radiator heating system for several rooms that is used later in the use case demonstrations in Sections 4.3 and 4.4. Section 4.3 presents use cases related to the design of flow systems, followed by use cases from the operation of such systems in Section 4.4.

4.1. Active chilled beam system

This section introduces an active chilled beam system example model. Active chilled beam systems combine heating and cooling with ventilation. In this example model, the interface between cooling and ventilation in an active chilled beam has been illustrated to demonstrate the expressiveness of FSO. In addition to FSO, BOT is used with the prefix bot:.

Fig. 8 illustrates the system in a schematic annotated with FSO terminology. An active chilled beam has a heat exchanger and an air supply, and is modeled as an fso:System consisting of two instances of fso:Terminal. The heat exchanger is a terminal connected to the cooling system, and the air supply is a terminal connected to the ventilation system. The heat exchanger transfers heat from the room, and the air supply supplies fluid, i.e. air, to the room. The cooling coil in the ventilation supply is modeled as two components with a heat transfer connection: the cooling coil itself is supplied with fluid by the cooling system piping, and the cooling coil “air side” is supplied with fluid by the ventilation system.

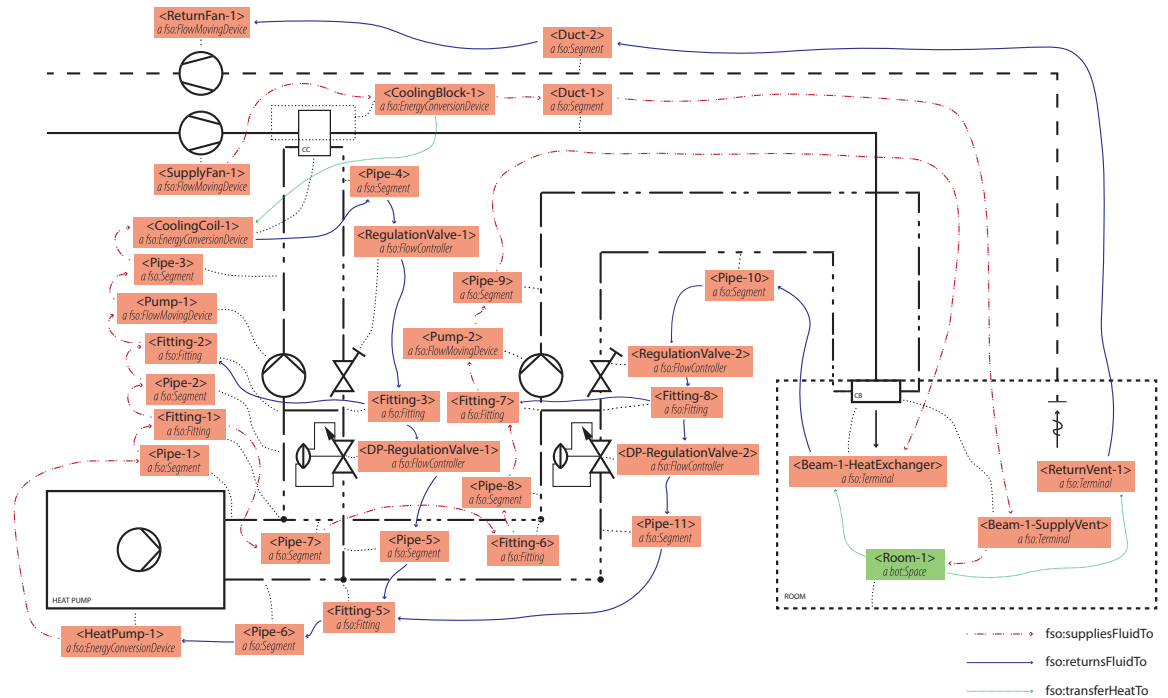


Fig. 8. Illustration of the active chilled beam system example model as a schematic with FSO terms.

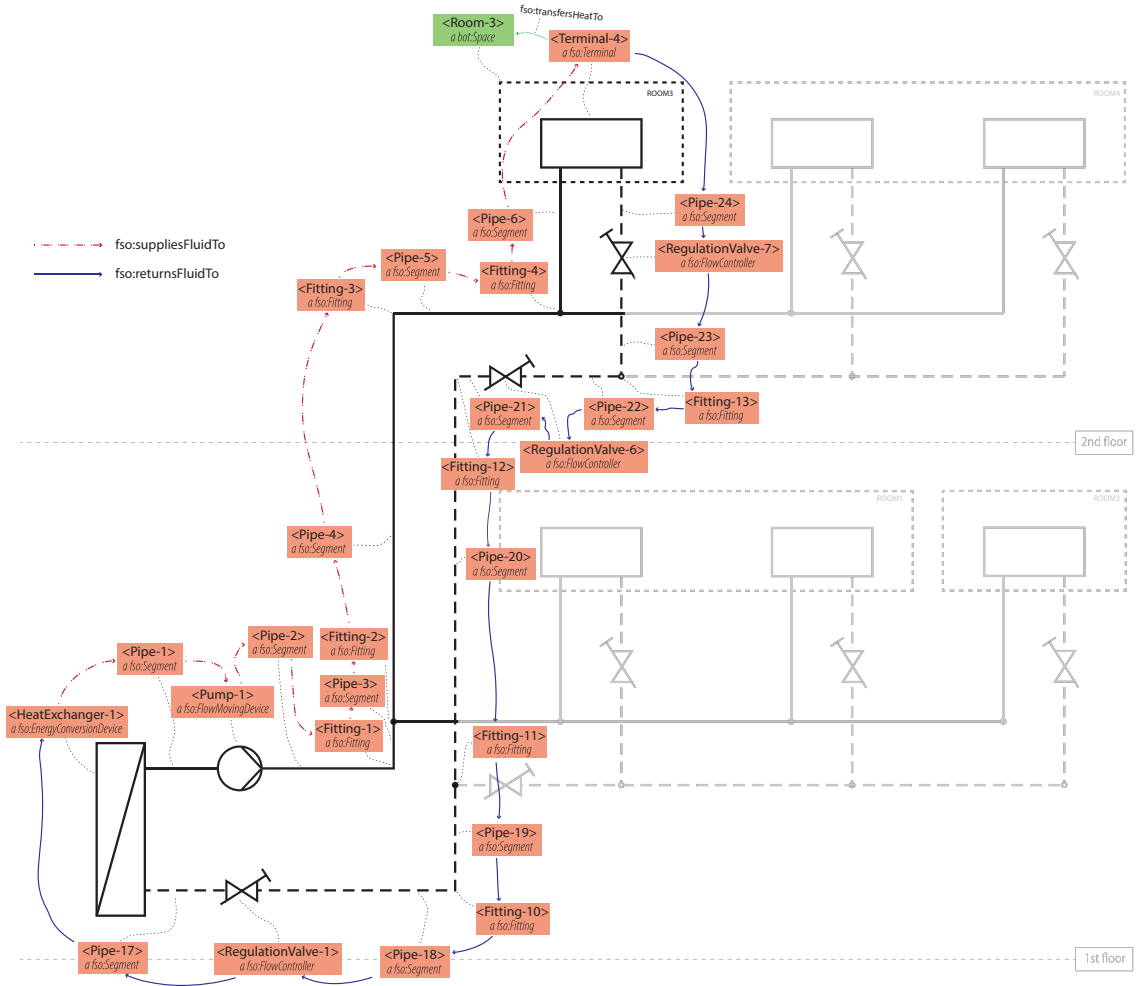


Fig. 9. Illustration of the heating system example model as a schematic with FSO annotations. While only a subset of the schematic is annotated here, a similar approach has been followed for the rest of the model to express the whole schematic in RDF. The complete model is used in the use case demonstrations.

The system composition is not illustrated in Fig. 8, but is shown in Listing 1. Four instances of `fso:System` are modeled: one for the ventilation system, one for the cooling system, one for the cooling coil consisting of the liquid and air sides, and one for the active chilled beam consisting of two terminals.

```
inst:CoolingCoil-1 a fso:System ;
  fso:hasComponent
    inst:CoolingCoil-1-LiquidSide ,
    inst:CoolingCoil-1-AirSide .

inst:CoolingSystem-1 a fso:System ;
  fso:hasSubSystem inst:Beam-1 , inst:CoolingCoil-1 ;
  fso:hasComponent
    inst:HeatPump-1 ,
    inst:Pump-1 ,
    inst:Pump-2 ,
    inst:RegulationValve-1 ,
    inst:RegulationValve-2 ,
    inst:DP-RegulationValve-1 ,
    inst:DP-RegulationValve-2 ,
    inst:Pipe-1 ,
    # ... pipes and fittings ...
    inst:Fitting-8 .

inst:VentilationSystem-1 a fso:System ;
  fso:hasSubSystem inst:Beam-1 , inst:CoolingCoil-1 ;
  fso:hasComponent
    inst:CoolingCoil-1-LiquidSide ,
    inst:SupplyFan-1 ,
    inst:CoolingCoil-1-AirSide ,
    inst:Duct-1 ,
    inst:ReturnVent-1 ,
    inst:Duct-2 ,
    inst:ReturnFan-1 .
```

Listing 1. System composition of the example active chilled beam model expressed in Turtle syntax.

4.2. Heating system

This section describes an example model of a heating system used in the use case demonstrations in Sections 4.3 and 4.4. An illustration of the model is shown in Fig. 9.

The example model depicts a simple heating system in a two-storey residential building. There is a heat exchanger (`fso:EnergyConversionDevice`) supplying heat to six radiators (`fso:Terminal`) in four rooms (`bot:Space`). Rooms 1 and 4 have two radiators each, while rooms 2 and 3 have one radiator each. The fluid is circulated by a pump (`fso:FlowMovingDevice`) through various pipe segments (`fso:Segment`) connected with various fittings (`fso:Fitting`) such as tees and elbows. For balancing, the system has multiple manual balancing valves (`fso:FlowController`).

The fluid flow is described by connecting the components with `fso:suppliesFluidTo` on the supply side, i.e. before the radiators, and `fso:returnsFluidTo` on the return side, i.e. after the radiators. Additionally, the rooms are modeled to contain the radiators with `bot:containsElement`, while simultaneously the radiators are modeled to transfer heat to the rooms with `fso:transfersHeatTo`.

The composition of the system is not shown in Fig. 9. This is implemented as shown in Listing 2. There is a top-level system `inst:HeatingSystem-A` with the supply system `inst:A-S` and return system `inst:A-R`. The subsystems have the concrete components. The radiators and the heat exchanger are components of both the supply and the return systems.

```
# Composition of the heating system.
inst:HeatingSystem-A a fso:System ;
  fso:hasSupplySystem inst:A-S ;
  fso:hasReturnSystem inst:A-R .

# Composition of the heating supply system.
inst:A-S a fso:SupplySystem ;
  fso:hasComponent ... # supply components

# Composition of the heating return system.
inst:A-R a fso:ReturnSystem ;
  fso:hasComponent ... # return components
```

Listing 2. Composition of the heating system example model expressed in Turtle syntax.

4.3. Design phase

In the design of flow systems in buildings, there are many subsystems to keep track of for the individual engineer. As mentioned in Section 1, there is a lack of data interoperability. This means that the logic of separate flow systems (ventilation, cooling, heating, etc.) is often interpreted differently by specific disciplines (ventilation designer, cooling designer, heating designer), though they are correlated with each other. Such different interpretations create calculation inconsistencies between different model elements. In the following use cases, examples are carried out, to display the potential use for mechanical designers.

4.3.1. Querying for systems and subsystems

HVAC systems are complex in nature. In large projects, BIM models of HVAC systems consist of possibly hundreds of subsystems. This means that it can be a laborious task to validate that the integrity and the logic between subsystems have been maintained in the modeling process.

Based on the heating system introduced in Section 4.2 a query can be formulated to return any `fso:System` and their supersystems, along with a string representation of the classes. This is shown in Listing 3.

```
PREFIX fso: <https://w3id.org/fso#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

# Find all instances of type fso:System, their
# supersystems, and system types.
SELECT
  ?system
  ?superSystem
  (GROUP_CONCAT(REPLACE(STR(?systemType),
    "https://w3id.org/fso#", "")) as ?types)
WHERE {
  ?system a fso:System .
  OPTIONAL { ?system fso:isSubSystemOf ?superSystem . }
  ?system a ?systemType .
  ?systemType rdfs:subClassOf+ fso:System .
}
GROUP BY ?system ?superSystem
```

Listing 3. Querying for systems and subsystems.

As seen in Table 4, `inst:HeatingSystem-A` has two subsystems `inst:A-S` and `inst:A-R`. This means that for the heating system, there exist two subsystems of the heating system, which is the supply

Table 4
Result of running query in Listing 3 on the heating system example model.

?system	?superSystem	?types
inst:A-S	inst:HeatingSystem-A	System SupplySystem
inst:A-R	inst:HeatingSystem-A	DistributionSystem
inst:HeatingSystem-A		System DistributionSystem
		ReturnSystem
		System

and return system. The result in Table 4 may not seem useful for this use case, since there are only two subsystems. However, large projects could consist of hundreds of subsystems, which means that it could prove useful to generate an overview to ensure modeling integrity.

4.3.2. Querying for components that serve a room

With large BIM models displayed in 3D, it can be hard to generate a visual overview of the integrity of a mechanical system. In some cases, the designer will only want to see specific flow components and their connections. Whilst visualizations are usually available in existing model viewers, the user does not have full control of which methods are used to display the connection of specific components.

Listing 4 matches all components that serve a specific room and belong to a specific system.

```
PREFIX bot: <https://w3id.org/bot#>
PREFIX fso: <https://w3id.org/fso#>
PREFIX inst: <https://example.com/inst#>

# Find all components of a specific system supplying
# fluid to a terminal in a specific room.
SELECT DISTINCT ?component
WHERE {
  BIND(inst:Room-3 as ?room)
  BIND(inst:HeatingSystem-A as ?system)
  ?room bot:containsElement ?terminal .
  ?terminal a fso:Terminal ;
    fso:hasFluidSuppliedBy+
    | fso:returnsFluidTo+ ?component .
  ?system fso:hasComponent ?component .
}
```

Listing 4. Querying for components of a specific system supplying a specific room.

Running the query of Listing 4 returned 27 of the 76 components in the heating system example model shown in Fig. 9. 5 of the components are displayed in Table 5. This use case demonstrates a way to query components by their relationships to systems and indirect connections to zones. This could be used to, for example, support customizable filtering options in a model viewer.

4.3.3. Calculating the mass flow rate of a system

Part of the mechanical designer's responsibility is to size the flow systems. As mentioned in Section 4.3, the methods used to make the calculations often vary with different designers, meaning that there are calculation inconsistencies between different designers. Therefore, the query in Listing 5 demonstrates a way to calculate the flow rate directly in SPARQL. The mass flow rate formula is written as:

$$\dot{m} = \Phi / (c_p \times \Delta T) \quad (1)$$

where \dot{m} is the mass flow rate [kg/s]; Φ is the heating/cooling demand [W]; c_p is the specific heat capacity [J/(kg K)]; ΔT is the temperature difference [K].

The query in Listing 5 returns the supply side mass flow rate demand, converted to kg/h, for each HVAC component based on the downstream heating demand.

Table 5
Result of running query in Listing 4 on the heating system example model.

?components
inst:HeatExchanger-1
inst:Pipe-1
inst:Pump-1
inst:Fitting-3
inst:Pipe-6
...

```
PREFIX bot: <https://w3id.org/bot#>
PREFIX fso: <https://w3id.org/fso#>
PREFIX ex: <https://example.com/ex#>

# Find all instances of type fso:Component that supply
# fluid to a terminal.
# Calculate the heating demand and mass flow rate for
# those components.
SELECT
  ?element
  (SUM(?demand) AS ?downstreamHeatDemand)
  ((?downstreamHeatDemand /
    (?heatCapacity*(?supplyTemp - ?returnTemp))*3600)
   AS ?massflow)
WHERE {
  # Specific heat capacity of water
  BIND(4180 AS ?heatCapacity) .

  ?supplySystem a fso:SupplySystem ;
    ex:temp ?supplyTemp ;
    fso:suppliesFluidTo ?returnSystem .
  ?returnSystem a fso:ReturnSystem ;
    ex:temp ?returnTemp .

  {
    # Assume equal distribution of demand for each
    # terminal in a room.
    SELECT ?room (?q / COUNT(?terminal) as ?demand)
    WHERE {
      ?terminal a fso:Terminal ;
        fso:transfersHeatTo ?room .
      ?room ex:hasHeatingDemand ?q .
    } GROUP BY ?room ?q
  }

  ?supplySystem fso:hasComponent ?element .
  ?element fso:suppliesFluidTo+ ?terminal .
  ?terminal fso:transfersHeatTo ?room .
}
GROUP BY ?supplySystem ?element ?supplyTemp ?returnTemp
?heatCapacity
```

Listing 5. Calculating the mass flow rate of a system.

Listing 5 has a nested SELECT clause. The inner SELECT subquery is evaluated first, to divide the heating demand of each room equally to the terminals located inside the room. In Table 6, the result is shown for 5 out of 27 components.

By running the query in Listing 5, the mass flow rate for each supply-side component was calculated. This could be used to size the heating system piping and components. Whilst this calculation can be done by hand, the use case demonstrates the potential for running calculations in SPARQL queries. If advanced algorithms were needed to run a flow simulation, this approach could be extended to supply input to an external application, as discussed later in Section 5.

4.4. Operations phase

In building operation and maintenance, being able to detect – or possibly even predict – faults and other issues relies on monitoring the various variables describing the state of the systems. These variables, if

Table 6
Result of running query in Listing 5 on the heating system example model.

?element	?downstreamHeatDemand	?massflow
inst:HeatExchanger-1	5900	254.07
inst:Pipe-1	5900	254.07
inst:Pump-1	5900	254.07
inst:Fitting-3	3800	163.64
inst:Pipe-6	800	34.45
...		

at all observable, may be either directly observable and have actual readings available, or they may be indirectly observable and estimated from the available observations with so-called virtual sensors. One obstacle for setting up continuous monitoring of system variables in buildings is the cost of setup, which is in part caused by varying naming schemes of data points [4]. Using a flow model to describe the relationships of components and connecting the available data points to the components could support various monitoring applications. The following examples show simplified queries supporting use cases from building operations and maintenance.

4.4.1. Querying for flow control components in the same loop

Listing 6 shows an example of a query for retrieving all the flow-altering components in the same loop as a chosen radiator. This could be useful for diagnosing anomalous room temperatures, assuming the components have some control values available. It also showcases a useful pattern that can find all the components in the same loop as a given target using the concepts of fluid supply and return with SPARQL property paths.

```
PREFIX fso: <https://w3id.org/fso#>

# Find all instances of fso:FlowMovingDevice and
# fso:FlowController in the same loop as given target
SELECT ?ctrlComp ?kind WHERE {
  BIND(<https://example.com/inst#Terminal-4> as ?target)

  ?ctrlComp
    (fso:suppliesFluidTo*/fso:returnsFluidTo*)
    | (fso:hasFluidReturnedBy*/fso:hasFluidSuppliedBy*)
    ?target .

  VALUES ?kind { fso:FlowMovingDevice fso:FlowController }

  ?ctrlComp a ?kind .
}
```

Listing 6. Querying for flow affecting components in the same loop as a given target.

The result of running the query on the heating system are shown in Table 7. The result shows that there is one pump and three valves in the same loop as the given radiator. Each of these devices could have more information attached to them, such as operating manuals or data labels. This kind of query could be particularly useful if a component, such as a radiator, in the loop is not working as expected. This could be caused by another faulty component in the same loop, such as a stuck or fluctuating valve.

4.4.2. Querying the upstream components

When diagnosing faults in flow components, it is often useful to know which components are found upstream of the component with anomalous behavior. For example, if a room is consistently too cold, it would seem obvious to take a look at the radiator in the room. While a single radiator may be faulty, the fault could also lie upstream of the radiator, possibly causing symptoms in other radiators as well.

An example of how the FSO object properties can be used to query for components upstream of a given component is shown in Listing 7. This information could be useful for an automatic diagnosis tool, or a manual diagnosis user interface.

Table 7

Result of running the query in Listing 6 on the heating system example model.

?ctrlComp	?kind
inst:Pump-1	fso:FlowMovingDevice
inst:RegulationValve-1	fso:FlowController
inst:RegulationValve-7	fso:FlowController
inst:RegulationValve-6	fso:FlowController

```
PREFIX fso: <https://w3id.org/fso#>

# Find all components upstream of a target, calculating
# the distance (in components) to them.
SELECT ?upstreamComp (COUNT(?step) AS ?distance)
WHERE {
  BIND(<https://example.com/inst#Terminal-4> AS ?target)
  ?upstreamComp fso:suppliesFluidTo* ?step .
  ?step fso:suppliesFluidTo+ ?target .
  ?target a fso:Terminal .
}
GROUP BY ?upstreamComp ?target
ORDER BY ?target ?distance
```

Listing 7. Querying components upstream of a specific radiator, with a distance for ordering.

The result of the query in Listing 7 is shown in Table 8. In an application, if the upstream components had, for example, pressure or temperature observations attached to them, those could be visualized and analyzed to discover potential problems.

4.4.3. Calculating the average room temperature error by valve

During the life cycle of a heating system, it is usually necessary to balance the pressure of the liquid circulating in the radiators to ensure occupant comfort and to optimize the heating system's energy consumption. To monitor that the system is working as intended, it can be useful to track the average temperature error of spaces served by a specific balancing valve.

After augmenting the heating system example model with naive representations of room temperature sensor readings and setpoints, querying the average temperature error of rooms grouped by flow controllers in the same loop could be done as shown in Listing 8. While an actual application likely would not store the values as queried in Listing 8, the query could, for example, be modified to retrieve identifiers used as keys for the sensors and setpoints in another storage system more suited for time series data. Additionally, while the example calculates the average in SPARQL, it could be simply retrieving the data point identifiers for an analysis tool to perform more advanced analysis.

```
PREFIX fso: <https://w3id.org/fso#>
PREFIX bot: <https://w3id.org/bot#>
PREFIX ex: <https://example.com/ex#>

# Calculate the average room temperature error for
# each valve.
SELECT ?ctrlCmp (avg(?error) as ?avg)
WHERE {
  {
    # Some rooms have multiple radiators.
    SELECT DISTINCT ?ctrlCmp ?sensor ?setpoint
    WHERE {
      ?room a bot:Space ;
        ex:hasTemperatureSensor ?sensor ;
        ex:hasTemperatureSetpoint ?setpoint .
      ?ctrlCmp a fso:FlowController ;
        (fso:suppliesFluidTo+|fso:hasFluidReturnedBy+) /
        fso:transfersHeatTo ?room .
    }
  }

  ?sensor ex:hasValue ?sensorValue .
  ?setpoint ex:hasValue ?setpointValue .
  BIND(?sensorValue - ?setpointValue as ?error)
}
GROUP BY ?ctrlCmp
ORDER BY DESC(ABS(?avg))
```

Listing 8. Average room temperature error grouped by valve device.

A similar approach could be used to discover the average temperature error for each heat source in a more complex system with multiple heat sources.

Table 8

Results of running query in Listing 7 on the heating system example model.

?upstreamComp	?distance
inst:Pipe-6	1
inst:Fitting-4	2
inst:Pipe-5	3
inst:Fitting-3	4
inst:Pipe-4	5
inst:Fitting-2	6
inst:Pipe-3	7
inst:Fitting-1	8
inst:Pipe-2	9
inst:Pump-1	10
inst:Pipe-1	11
inst:HeatExchanger-1	12

The result of running the query in Listing 8 on the heating system is shown in Table 9. With the example values, some of the valves have an average error of 0 degrees while one has an error of up to 2.0 degrees. This could indicate an imbalance in the heating system.

4.4.4. Projecting to a simplified logical topology

It is often useful to consider the logical topology of the components in a flow system, disregarding the actual flow segments and fittings that make up the physical flow connections. This kind of simplified model could also be the best thing available when modeling a building with no BIM model.

One way to project a detailed model to a more simple one is the CONSTRUCT query form which constructs a new graph. These constructed graphs could be used similarly as materialized views in relational databases, where the data is projected into a certain format for specialized queries, simplifying and improving the efficiency of those queries. For example, dropping the segments and fittings could be useful for fault detection, where the focus is on the active components. An example query that constructs a graph of the fluid supply and return connections between components without flow segments and fittings is shown in Listing 9.

Table 9

Result of running the query in Listing 8 on the heating system example model augmented with room temperatures and setpoints.

?ctrlCmp	?avg
inst:RegulationValve-7	2.0
inst:RegulationValve-3	1.5
inst:RegulationValve-4	1.5
inst:RegulationValve-2	1.25
inst:RegulationValve-1	1.125
inst:RegulationValve-5	1.0
inst:RegulationValve-6	1.0
inst:RegulationValve-8	0.0
inst:RegulationValve-9	0.0

```
# Construct a graph of component fluid supply and return
# connections, excluding segments and fittings.
PREFIX fso: <https://w3id.org/fso#>
CONSTRUCT {
  ?s fso:suppliesFluidTo ?o .
  ?s1 fso:returnsFluidTo ?o1 .
}
WHERE {
  { # Supply connections
    ?s fso:suppliesFluidTo+ ?o .
    ?s a fso:Component .
    FILTER NOT EXISTS { ?s a fso:Segment }
    FILTER NOT EXISTS { ?s a fso:Fitting }
    FILTER NOT EXISTS { ?o a fso:Segment }
    FILTER NOT EXISTS { ?o a fso:Fitting }
    FILTER NOT EXISTS {
      ?s fso:suppliesFluidTo+ ?intermediate .
      ?intermediate fso:suppliesFluidTo+ ?o .
      FILTER NOT EXISTS { ?intermediate a fso:Segment }
      FILTER NOT EXISTS { ?intermediate a fso:Fitting }
    }
  } UNION { # Return connections
    ?s1 fso:returnsFluidTo+ ?o1 .
    ?s1 a fso:Component .
    FILTER NOT EXISTS { ?s1 a fso:Fitting }
    FILTER NOT EXISTS { ?s1 a fso:Segment }
    FILTER NOT EXISTS { ?o1 a fso:Fitting }
    FILTER NOT EXISTS { ?o1 a fso:Segment }
    FILTER NOT EXISTS {
      ?s1 fso:returnsFluidTo+ ?intermediate .
      ?intermediate fso:returnsFluidTo+ ?o1 .
      FILTER NOT EXISTS { ?intermediate a fso:Segment }
      FILTER NOT EXISTS { ?intermediate a fso:Fitting }
    }
  }
}
```

Listing 9. CONSTRUCT query for a simplified graph.

The result of running the query in Listing 9 is shown in Fig. 10 as a visual graph, manually organized for ease of reading. The result shows that the fluid flows from a central heat exchanger through a pump to all the radiators. Additionally, it can be seen that each radiator has a valve on the return side and that the radiators are grouped into sets of three with second valves. Again, further information could be linked to the components, supporting use cases such as fault diagnosis.

5. Discussion and future work

The example models and the example queries described in Section 4 illustrate simplified use cases. They are an attempt at capturing the essentials of more complex, real-world use cases without going into detail. The models and queries show that fluid circuits, such as used in heating, can be isolated and analyzed using the component relationships. They also allude to the potential in expressing and tracking the composition of systems, which is useful as the number of systems increases. For actual extended use cases, more specific information about the model would be required, and thus the vocabulary for describing that information would be required as well. Examples of further information include product details, such as pipe materials and dimensions, as well as functional requirements for spaces, such as ventilation rates. For expressing this information, other ontologies alongside FSO would need to be used or possibly created.

Some of the existing ontologies for the AEC industry that FSO complements were discussed in Section 2, and include the likes of BOT [21] and SAREF [24]. In addition to these, other ontologies relevant to the BIM and linked building data context include Ontology for Property Management (OPM) [2] for managing properties and Ontology for Managing Geometry (OMG) [31] for managing geometry descriptions. It is worth noting that while BIM tools generally view elements, such as

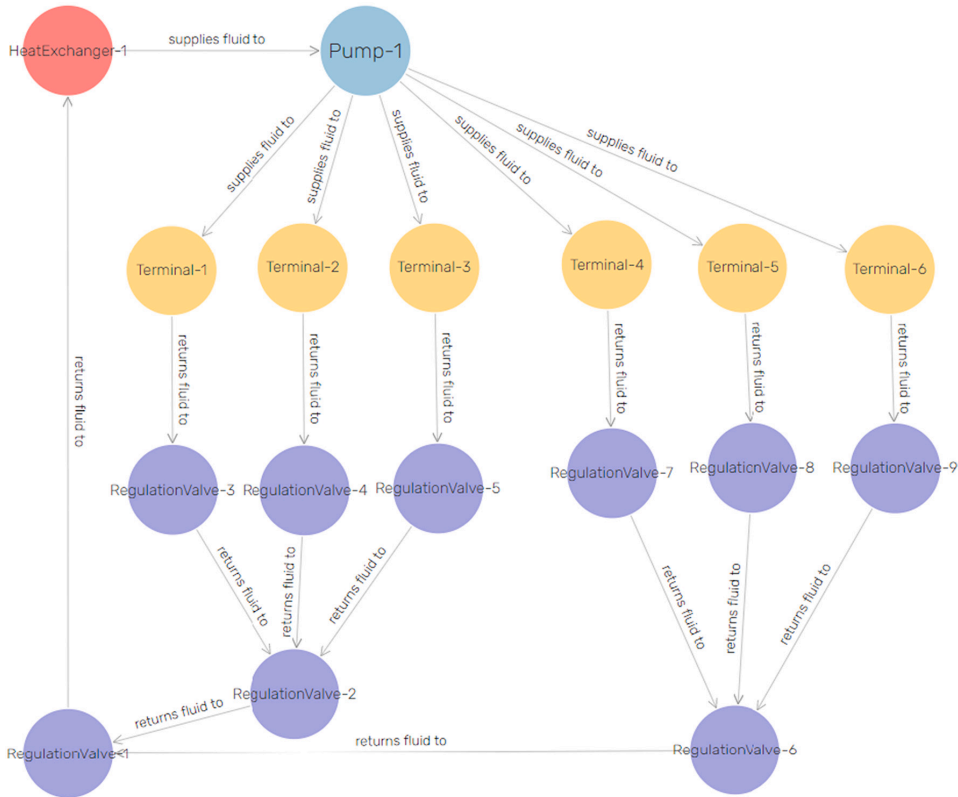


Fig. 10. Example of a simplified logical topology of component connectivity resulting from the CONSTRUCT query in Listing 9. The view was constructed by running the query in GraphDB and manually arranging the nodes for legibility.

flow components, as 3D objects with particular attributes, in the linked data approach the geometry of an object is an attribute among others.

Other common representations for information similar to covered by FSO and used in the HVAC and industrial process domains include Process Flow Diagrams (PFDs). While PFDs are a primarily graphical representation, FSO is a computer-interpretable vocabulary for representing some of the information that would be required to generate a PFD. Additionally, while there is significant overlap between the information contents, FSO needs to be combined with other ontologies to represent all the information representable in PFDs. Similarly, a PFD will generally not include all information representable by FSO.

5.1. Limitations of the study

One limitation of the research presented is the lack of use of a formal ontology development methodology, which is something that has been called for in relation to previous ontology developments [9]. Although no formal, structured method is used, some of the ontology engineering best practices have been implicitly followed. Firstly, the existing ontologies have been researched and their reuse has been considered, resulting in some alignments already mentioned in Section 3.5. Secondly, there are efforts at validating the ontology, although the extent of validation is still limited, and more structured validation should be carried out in future work. Finally, competency questions are used to explicate the scope of the proposed ontology.

Another limitation is that while expert knowledge has been used in developing the scope and terms of the ontology, the number of

participants is limited mainly to the authors. Besides the component classes based on IFC, the authors' own experience from different fields has been the primary source of definitions for the ontology and is something that needs to be opened to further refinement by additional domain experts.

Additionally, while the ontology is theorized to support descriptions outside the HVAC domain, these were not investigated. This means that there are likely unknown limitations when applying to other flow system domains.

Further, while FSO is intended to support various applications, including simulations and monitoring, this paper does not use real data from a particular tool. Once real data is included from a tool such as an HVAC simulation, then the paper needs to go into specific details of that application domain, ruling out other domains. Demonstrating the use of FSO in integrating real-world applications would require further – potentially application-specific – ontology and software tooling development. The need to develop new tooling and combine FSO with other ontologies for practical use cases is discussed in Section 5.2 and illustrated in Fig. 11. The proposed ontology only contains the terminology for describing the topology of a flow system in terms of (sub-)system composition, component classifications, and flow connections. It, by design, does not contain the terminology for describing the properties of components or the instrumentation of HVAC processes. As such, FSO alone is not sufficient for describing the inputs or outputs of simulation tools, nor the instrumentation of building automation systems. The use case descriptions presented in Section 4 use imaginary complementary ontologies and data to showcase the potential for integrations on a

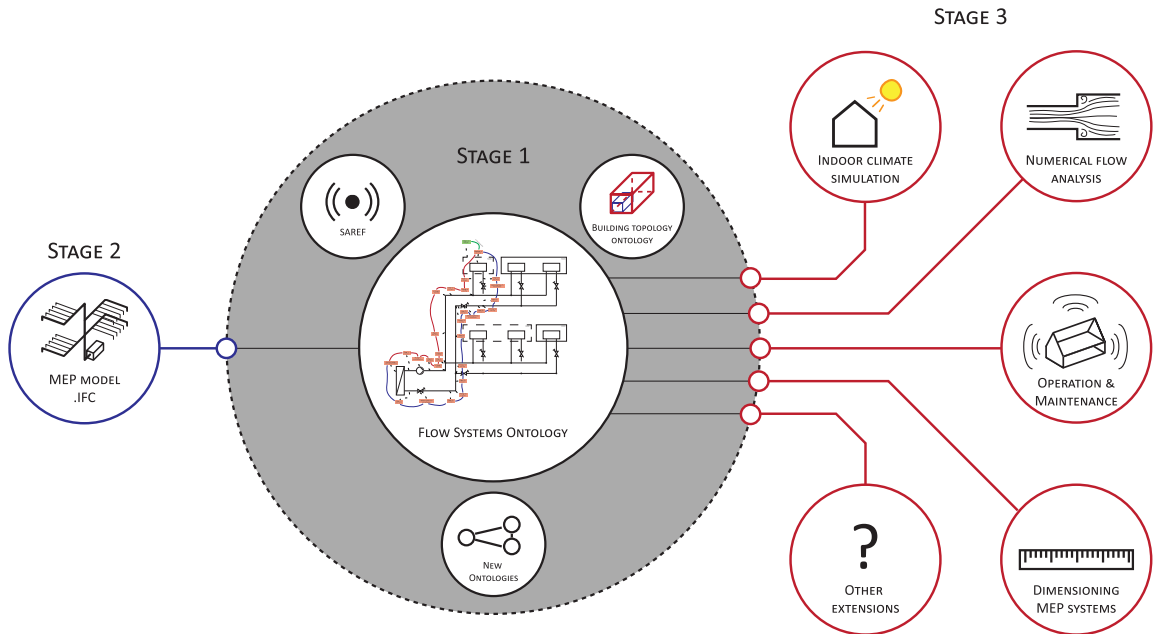


Fig. 11. Illustration of the envisioned workflow for supporting information interoperability, and the planned 3 stages of research focus: 1 (grey), 2 (blue), 3 (red). From left to right, the information sources are transformed into linked data and integrated using FSO and other ontologies, including both existing ontologies and potential future extensions to those. The linked data can then be converted into formats expected by existing tools, and new tools can be developed utilizing the linked data directly. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

conceptual level.

Finally, as briefly mentioned above, it is worth highlighting that the validation has been mostly done using two example models with manually written triples, described in Section 4. To more extensively evaluate the expressiveness, computational characteristics, and potential shortcomings of the ontology, larger models from actual design processes should be used. This would require the development of a tool that is capable of converting a format such as IFC into FSO annotated linked data.

5.2. Roadmap for further development

The development of FSO is envisioned as a part of a three-stage process, pointing to future applications of FSO and further research in supporting information interoperability for flow systems. The stages are illustrated in Fig. 11 and described in more detail next.

Stage 1 is the development of an abstract ontology for describing flow systems, as done in this article. The concrete use of FSO in future applications would be to annotate information either directly, or via more domain-specific ontologies extending FSO, thus enabling reasoning and queries as illustrated in this study. The current version of FSO provides a stepping stone on the way to utilizing semantic web technologies in the context of flow systems such as in HVAC systems, which supports improving data interoperability [1]. The abstract definitions leave room for extending into concrete use cases, allowing alignment via a common upper terminology. The development of the ontology is the first stage in the process as it includes implicit considerations for the use cases downstream and the information available in various sources common in the industry. However, future developments in either direction will inform the refinement of the ontology itself.

Stage 2 consists of the development of tools to convert IFC into linked data representations annotated with FSO. Tools for converting IFC to RDF with IFCOWL exist [29], as well as tools built on top of it for

converting to the ontologies developed by the W3C LBD Community Group [30]. This stage is important for enabling the utilization of BIM models created with current modeling practices. While linked data representations of the information are unlikely to replace IFC files in the industry, tools for consistently and transparently breaking the information out of the IFC document context have value for downstream use of the information. Therefore, the primary avenue of research should be integrating IFC into the pipeline of information flow to future applications that use FSO.

Stage 3 involves the development of more specific ontologies extending FSO, and tools such as parsers bridging the gap from linked data representations to existing and new industry applications. Some of the potential applications that either require or could benefit from the knowledge of flow system relationships include: indoor climate simulations; numerical flow analysis; automated fault detection and diagnosis; and sizing of Mechanical, Electrical and Plumbing (MEP) systems. The development of these use cases can be used to further inform the development of FSO, creating a feedback loop for future iterations. In particular, the authors are working to apply FSO as a part of a new hydraulic simulation process, and to provide a data model for fault detection in HVAC systems.

6. Conclusions

Linked data and semantic web technologies have been gaining interest within the architecture, engineering and construction (AEC) industry, which has brought about the development of multiple ontologies within the domain. This study summarized the prominent ontologies in the AEC field and showed that there is no ontology for the description of the mass and energy flow relationships of systems. To fulfill this gap, this research introduced a new ontology for describing the flow systems in terms of their composition and mass and energy flow relationships. To this extent, the study enumerated the competency questions driving the

development of the ontology, and related the term definitions to their relevant competency questions. The proposed ontology was demonstrated and validated using example models of a heating system and an active chilled beam system. Further, queries in SPARQL Protocol and RDF Query Language (SPARQL) were formulated and evaluated against the heating system, simulating various use cases. The ontology development was justified in the context of the bigger picture of information interoperability, discussing the plans for future extensions and information source integrations.

Declaration of Competing Interest

The authors recognize that there is a potential for conflicts of interest via industry affiliations. Ville Kukkonen is working on a doctoral dissertation in Aalto University while also working in Granlund. Ali Küçükavci is working on a doctoral dissertation in Technical University of Denmark, and was sponsored by Tyréns A/S during the reported study. Mikki Seidenschur is working on a doctoral dissertation in Technical University of Denmark, while also working in Ramboll A/S. Mads Holten Rasmussen works in NIRAS A/S.

Acknowledgements

This work was supported by Tyréns A/S; EU-Interreg ÖKS “Data-driven Energy Management in Public Buildings”; Granlund; the K.V. Lindholm foundation; the Ramboll Foundation; and Innovation Fund Denmark.

References

- [1] P. Pauwels, S. Zhang, Y.C. Lee, Semantic web technologies in AEC industry: a literature overview, *Autom. constr.* 73 (2017) 145–165, <https://doi.org/10.1016/j.autcon.2016.10.003>.
- [2] M.H. Rasmussen, M. Lefrançois, P. Pauwels, C.A. Hviid, J. Karlshøj, Managing interrelated project information in AEC knowledge graphs, *Autom. constr.* 108 (2019) 102956, <https://doi.org/10.1016/j.autcon.2019.102956>.
- [3] W. Kim, S. Katipamula, A review of fault detection and diagnostics methods for building systems, *Sci. Tech. Built Environ.* 24 (1) (2018) 3–21, <https://doi.org/10.1080/23744731.2017.1318008>.
- [4] H.B. Gunay, W. Shen, G. Newsham, Data analytics to improve building performance: a critical review, *Autom. constr.* 97 (2019) 96–109, <https://doi.org/10.1016/j.autcon.2018.10.020>.
- [5] B. Dong, Z. O'Neill, Z. Li, A BIM-enabled information infrastructure for building energy fault detection and diagnostics, *Autom. constr.* 44 (2014) 197–211, <https://doi.org/10.1016/j.autcon.2014.04.007>.
- [6] A. Golabchi, M. Akula, V.R. Kamat, Leveraging BIM for automated fault detection in operational buildings, in: F. Hassani, O. Moselhi, C. Haas (Eds.), *Proceedings of the 30th International Symposium on Automation and Robotics in Construction and Mining (ISARC 2013): Building the Future in Automation and Robotics*, International Association for Automation and Robotics in Construction (IAARC), Montreal, Canada, 2013, pp. 187–197, <https://doi.org/10.22260/ISARC2013/0020>.
- [7] K. Kim, H. Kim, W. Kim, C. Kim, J. Kim, J. Yu, Integration of IFC objects and facility management work information using semantic web, *Autom. constr.* 87 (2018) 173–187, <https://doi.org/10.1016/j.autcon.2017.12.019>.
- [8] S. Tang, D.R. Shelden, C.M. Eastman, P. Pishdad-Bozorgi, X. Gao, BIM assisted building automation system information exchange using BACnet and IFC, *Autom. constr.* 110 (2020) 103049, <https://doi.org/10.1016/j.autcon.2019.103049>.
- [9] G.F. Schneider, G.D. Kontes, H. Qiu, F.J. Silva, M. Bucur, J. Malanik, Z. Schindler, P. Andriopoulos, P. de Agustin-Camacho, A. Romero-Amorrortu, G. Grün, Design of knowledge-based systems for automated deployment of building management services, *Autom. constr.* 119 (2020) 103402, <https://doi.org/10.1016/j.autcon.2020.103402>.
- [10] S. Attia, J.L.M. Hensen, L. Beltrán, A.D. Herde, Selection criteria for building performance simulation tools: contrasting architects' and engineers' needs, *J. Building Perf. Simulation* 5 (3) (2012) 155–169, <https://doi.org/10.1080/19401493.2010.549573>.
- [11] G. Schreiber, Y. Raimond, RDF 1.1 Primer, W3C Working Group Note, W3C, 2014. Available at <https://www.w3.org/TR/rdf11-primer> (Accessed: 2020-11-11).
- [12] SPARQL 1.1 Overview, W3C Recommendation, W3C, 2013. Available at <https://www.w3.org/TR/sparql11-overview/> (Accessed: 2020-11-11).
- [13] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, S. Rudolph, OWL 2 Web Ontology Language Primer (Second Edition), W3C Recommendation, W3C, 2012. Available at <https://www.w3.org/TR/owl2-primer/> (Accessed: 2020-11-11).
- [14] G.E. Modoni, M. Doukas, W. Terkaj, M. Sacco, D. Mourtzis, Enhancing factory data integration through the development of an ontology: from the reference models reuse to the semantic conversion of the legacy models, *Int. J. Comput. Integr. manuf.* 30 (10) (2017) 1043–1059, <https://doi.org/10.1080/0951192X.2016.1268720>.
- [15] E. Maleki, F. Belkadi, N. Boli, B.J. van der Zwaag, K. Alexopoulos, S. Koukas, M. Marin-Perianu, A. Bernard, D. Mourtzis, Ontology-based framework enabling smart product-service systems: application of sensing systems for machine health monitoring, *IEEE Inter. Things J.* 5 (6) (2018) 4496–4505, <https://doi.org/10.1109/JIOT.2018.2831279>.
- [16] T.E. El-Diraby, H. Osman, A domain ontology for construction concepts in urban infrastructure products, *Autom. constr.* 20 (8) (2011) 1120–1132, <https://doi.org/10.1016/j.autcon.2011.04.014>.
- [17] S. Zhang, F. Boukamp, J. Teizer, Ontology-based semantic modeling of construction safety knowledge: towards automated safety planning for job hazard analysis (JHA), *Autom. constr.* 52 (2015) 29–41, <https://doi.org/10.1016/j.autcon.2015.02.005>.
- [18] M. Niknam, S. Karshenas, A shared ontology approach to semantic representation of BIM data, *Autom. constr.* 80 (2017) 22–36, <https://doi.org/10.1016/j.autcon.2017.03.013>.
- [19] C. Wu, P. Wu, J. Wang, R. Jiang, M. Chen, X. Wang, Ontological knowledge base for concrete bridge rehabilitation project management, *Autom. constr.* 121 (2021) 103428, <https://doi.org/10.1016/j.autcon.2020.103428>.
- [20] P. Pauwels, W. Terkaj, EXPRESS to OWL for construction industry: towards a recommendable and usable IFCOWL ontology, *Autom. constr.* 63 (2016) 100–133, <https://doi.org/10.1016/j.autcon.2015.12.003>.
- [21] M.H. Rasmussen, M. Lefrançois, G.F. Schneider, P. Pauwels, BOT: the building topology ontology of the W3C linked building data group, *Semantic Web Preprint (Preprint)* (2020) 1–19, <https://doi.org/10.3233/SW-200385>.
- [22] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Bergés, D. Culler, R.K. Gupta, M.B. Kjærsgaard, M. Srivastava, K. Whitehouse, Brick: metadata schema for portable smart building applications, *Appl. Energy* 226 (2018) 1273–1292, <https://doi.org/10.1016/j.apenergy.2018.02.091>.
- [23] M. Lefrançois, J. Kalaja, T. Ghariani, A. Zimmermann, Smart Energy Aware Systems, 2016. Available at <https://hal.archives-ouvertes.fr/hal-02016334> (Accessed: 2020-11-11).
- [24] ETSI, TS 103 264 V3.1.1 SmartM2M; Smart Applications; Reference Ontology and oneM2M Mapping, Technical Specification, ETSI, 2020. Available at https://www.etsi.org/deliver/etsi_ts/103200/103299/103264/03.01.01_60/ts_103264v030101p.pdf (Accessed: 2020-06-03).
- [25] ETSI, TS 103 410-3 V1.1.2 SmartM2M; Extension to SAREF; Part 3: Building Domain, Technical Specification, ETSI, 2020. Available at https://www.etsi.org/deliver/etsi_ts/103400/103499/10341003/01.01.02_60/ts_10341003v010102p.pdf (Accessed: 2020-06-03).
- [26] ETSI, TS 103 548 V1.1.1 SmartM2M; SAREF consolidation with new reference ontology patterns, based on the experience from the SEAS project, Technical Specification, ETSI, 2019. Available at https://www.etsi.org/deliver/etsi_ts/103500/103599/103548/01.01.01_60/ts_103548v010101p.pdf (Accessed: 2020-06-03).
- [27] K. Hammar, E.O. Wallin, P. Karlberg, D. Hälleberg, The RealEstateCore Ontology, in: C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, F. Gandon (Eds.), *The Semantic Web – ISWC 2019, Lecture Notes in Computer Science*, Springer International Publishing, Auckland New Zealand, 2019, pp. 130–145, https://doi.org/10.1007/978-3-030-30796-7_9.
- [28] V. Kukkonen, A. Küçükavci, M. Seidenschur, M.H. Rasmussen, K.M. Smith, C. A. Hviid, Example Models and Queries for Flow Systems Ontology, 2021, <https://doi.org/10.5281/zenodo.4492645>.
- [29] J. Oraskari, Jyrkioraskari, IFCtoRDF-Desktop: The IFCtoRDF Desktop Application 2.8, Zenodo, 2020, <https://doi.org/10.5281/ZENODO.4005935>.
- [30] J. Oraskari, K. McClinn, P. Pauwels, F. Priyatna, A. Wagner, J. Lehtonen, Jyrkioraskari, IFCtoLBD: IFCtoLBD 2.11, Zenodo, 2020, <https://doi.org/10.5281/ZENODO.4075365>.
- [31] A. Wagner, M. Bonduel, P. Pauwels, R. Uwe, Relating geometry descriptions to its derivatives on the web, *University College Dublin, Chania, Crete*, 2019, pp. 304–313, <https://doi.org/10.35490/ec3.2019.146>.

6.4 Paper IV - Coupling Modelica and a Common Data Environment for simulation of HVAC systems.

Coupling BIM and detailed Modelica simulations of HVAC systems in a Common Data Environment

Esben Visby Fjerbæk^{1,2}, Mikki Seidenschnur^{1,2}, Ali Küçükavcı^{1,3}, Kevin Michael Smith¹, and Christian Anker Hviid¹

¹Department of Civil & Mechanical Engineering, Technical University of Denmark, Kgs. Lyngby, Denmark

²Rambøll, Copenhagen, Denmark

³COWI, Kgs. Lyngby, Denmark

ABSTRACT

In current building design practices, the operation and performance of HVAC systems are rarely validated in detail. This is mainly caused by the manual burden of generating detailed HVAC simulation models. To lower the barrier for detailed HVAC simulations, this paper presents an automated toolchain that generates and simulates models in Modelica language using building information structured in web-based Common Data Environments. This approach differs from previous approaches by its focus on HVAC systems and integration with a Common Data Environment over file-based BIM, which increases interoperability and allows users to run simulation studies in the cloud, without depending on the computation power of their local PCs. The tool successfully generated and simulated a model of the HVAC systems in a small building and thus demonstrated a fully interoperable data exchange between a CDE and a simulation environment while showcasing the potential of analysing HVAC systems with Modelica.

Keywords: HVAC, BIM, Simulation, Modelica, Digital Twin

1 INTRODUCTION

High-performing and low-energy buildings is an ever-growing domain in the architecture, engineering, and construction industry. Accurate estimation of energy consumption is vital in the design of low-energy buildings and for documentation of regulation compliance. With the increase of computation power over the past 30 years, tools for estimating building energy performance have evolved from simple, quasi-steady-state models to detailed dynamic simulations with calculation of multiple heat losses and gains. Choices for building energy performance simulation (BEPS) tools are vast, and within whole building simulations, more than 70 free and commercial tools are listed in the IBPSA-USA Directory of Building Energy Software Tools [1].

Despite the multitude of sophisticated BEPS tools, a significant discrepancy between the estimated and measured energy consumption, known as the energy performance gap, exists [2–4]. Menezes et al. [2] showed magnitudes of more than 2.5 times the expected energy consumption compared to the simulated energy performance. Several researchers have analyzed the causes of the energy performance gap and found that the root causes are a) the inability to predict occupant behavior [5–8] and b) inefficient/faulty operation of heating, ventilation, and air conditioning (HVAC) systems [4, 9–11].

Occupant behavior significantly affects energy consumption through different actions: heat gains, venting, set-point changes, shading and occupancy schedules. Recent developments in occupant behavior modeling have shown considerable improvements in this topic,

and proper implementation of occupant behavior models in BEPS tools should close the occupant-related energy performance gap [8].

Inefficient and faulty HVAC components and unexpected system operation cause a discrepancy since this changes the assumptions of ideal operation, usually made in BEPS software [12]. Additionally, these problems may lead to poor occupant comfort [9, 13] and high return temperatures in heating systems [14].

The contribution of faulty operation to the energy performance gap is difficult to quantify, especially in existing buildings. However, analyses of commissioning projects indicate the impact. In an analysis of more than 500 commissioning projects, Mills [11] found that correction of faulty HVAC components and inefficient operation led to a median savings potential of 13%. Examples of savings potentials include adjusting operation schedules, optimizing control sequences, replacing or fixing faulty components, etc. Thus, correction of HVAC faults can help lower the energy performance gap significantly.

Based on the literature mentioned above, the faults leading to inefficient and unexpected operation of HVAC systems can be categorized in the following groups:

1. **Design errors** such as unbalanced systems and wrong component sizes
2. **Un-implemented design intents** such as operation schedules, setpoints and balancing valves
3. **Construction errors**, including last-minute design changes
4. **Component degradation** such as clogged filters, stuck-open valves and calcium build-up

(1) should be addressed during design, where designers must put more effort into properly designing HVAC systems and controls. Often, the design of HVAC components is based on worst-case conditions and safety margins which may lead to oversized components that perform inefficiently under part load conditions [15–18]. Moreover, traditional simulation programs often use simplified and idealized models for terminal units, such as radiators, and their controls [12]. Without detailed simulations of HVAC systems modeling flows, temperatures, controls and valve openings, it is impossible to quantify the system’s response to varying loads, especially when components are often oversized.

(2) is caused by interoperability problems between and within the phases of construction [4]. Designers must improve documentation of design intents and the construction team should respect design intents whenever possible. If the design changes, the consequences should be assessed in greater detail, and the changes should be documented. Documentation of design intents should be incorporated into the building information modeling (BIM) domain. Today, BIM documentation of HVAC systems and their geometry is available and widely used, but other information, such as physical properties, is rarely present in BIM models.

(3) and (4) should be handled during commissioning and operation. Here, automated fault detection and diagnosis (FDD) has been proposed by many to identify which components are behaving unexpectedly [19]. This can be done in many ways, and Kim and Katipamula [19] carried out a thorough study of different methodologies. Where some methods rely on black- and grey-box models developed on data, *quantitative model-based* approaches rely solely on detailed, physical models of the system. These detailed simulations for components or subsystems are compared with measurements to identify the components whose behavior differs from the expected. While model-based approaches are computationally intensive and require significant modeling efforts, they are superior to black- and grey-box models at modeling both ideal and faulty operation [19].

To reduce the energy performance gap throughout the building’s lifetime, it is clear that both designers and operators must utilize detailed HVAC simulations that are able to model HVAC components in detail under correct and faulty operation. During design, the simulations provide crucial insights into the system’s response to dynamic loads. In the operation phase, they provide a benchmark for expected system performance and a starting point for model-based FDD.

Traditionally, wide-spread BEPS tools such as EnergyPlus, IDA-ICE, IES-VE, BSim, and many others only have limited support for detailed HVAC modeling. Some of these tools have models for terminal units and/or production plants, but usually the models are limited to predefined models of HVAC systems. Moreover, such tools often use idealized control sequences, to improve

numerical efficiency, instead of using actual feedback controls [12]. This makes traditional tools unsuited for the above-mentioned use-cases, and especially as tools for the analysis of novel and innovative HVAC solutions.

To combat the inflexible and idealized nature of traditional BEPS tools, the use of Modelica for BEPS has gained momentum in recent years. The Modelica language is object-oriented, which means it is extensible through addition or combination of new or existing models, allowing modeling of all thinkable system configurations. To build Modelica models, pre-built component models, grouped in third-party libraries, are used to construct comprehensive, full-system models. In the Modelica Association’s library index¹ around 150 free and commercial libraries are listed, covering engineering disciplines such as aviation, automotive, biochemistry, energy, and electronics.

In recent years, the IBPSA project 1 [20] has driven the development of Modelica libraries for buildings. The project resulted in the Modelica IBPSA library, which provides base classes used in four libraries; *AirLib* [21], *BuildingSystems* [22], *IDEAS* [23] and *Buildings* [24]. These libraries include similar models for simulation of thermal zones, HVAC systems, outdoor conditions, etc. The Modelica Buildings Library has recently implemented *Spawn of EnergyPlus* [25], which implements EnergyPlus models in Modelica through the Functional Mock-up Interface [26]. This allows simulation of the envelope heat balance, load profiles and solar gains in EnergyPlus, while all systems are simulated in Modelica. With *Spawn of Energyplus*, the issues related to HVAC modeling in traditional BEPS tools are removed, while preserving EnergyPlus’s efficient thermal simulation capabilities.

No matter which BEPS tool is used, the modeling process for HVAC systems is laborious and time-consuming. In many cases, the manual processes lead to errors, often caused by misjudgments or imprecisions from the modelers [27, 28]. The building design process is iterative, and any design change requires simulation models to be updated. The extra time and uncertain outcome make these procedures costly in commercial applications, meaning that design teams will rarely prioritize detailed HVAC simulations.

To make the BEPS modeling process more efficient, many efforts are made to align BIM and BEPS, both in research and commercial settings [29, 30]. Simulation and BIM professionals drive this development by either integrating simulation engines in BIM tools or supporting BIM import in simulation tools [30, 31]. Hosseini et al. [30] did an extensive review of BIM-integrated BEPS and found a large number of tools in both research and the industry. However, none of the identified tools explicitly implemented HVAC systems in the automatic model generation, which correlates with the lack of HVAC support in traditional BEPS tools.

With the rise of Modelica for BEPS, several tools for

¹Available at <https://modelica.org/libraries.html>

Table 1. Comparison of three tools for BIM-generated Modelica models.

Name	IFC2Modelica	BIM2Modelica	Revit2Modelica
Language	Python	Python	C#
Data source	IFC4	IFC2x3	Revit
HVAC systems	Yes	No	No
Open-source	No	Yes	No
Modelica library	<i>IDEAS</i>	<i>Building Systems</i>	<i>Buildings</i>
Reference	[31]	[32]	[33]

1 BIM-based model generation have also emerged. Gen-
2 eration of Modelica models from BIM data is limited,
3 with only three tools, all developed in research, identified.
4 Table 1 shows a comparison of the three tools, described
5 in the following paragraphs.

6 *IFC2Modelica*, developed at KU Leuven and described
7 by Andriamamonjy et al. [31], parses information from
8 models defined with the Industry Foundation Classes
9 (IFC) format to Modelica models based on the IDEAS
10 library. The tool uses model view definitions to check
11 that the IFC file contains the needed information in the
12 BIM model. Python processes the IFC file through the
13 IfcOpenShell module. The paper describes an example of
14 a ventilation system, and the tool is used for automatic
15 FDD of an air handling unit [34]. While rooms and
16 thermal zones are also considered in the paper, the focus
17 is on HVAC system modeling.

18 Similarly to *IFC2Modelica*, the open-source
19 *BIM2Modelica* [22], developed at UdK Berlin, is a
20 tool using IfcOpenShell to translate IFC files to Modelica
21 models. In addition to multizone models, the tool
22 supports generation of computational fluid dynamics and
23 district heating models. The Modelica models are based
24 on the BuildingSystems library. The paper describes a
25 case study for a multi-zone model, but does not cover
26 HVAC systems.

27 Whereas *IFC2Modelica* and *BIM2Modelica* use IFC as
28 input file types, *Revit2Modelica* [33] uses models from the
29 BIM tool *Revit* to create BEPS models. Through a plugin
30 for Revit, which accesses Revit’s API, the tool parses
31 BIM data to the *Buildings* library. The main intent of
32 the tool is to provide thermal BEPS, and thus HVAC
33 systems have not been considered in the presented cases.

34 Of the three existing applications for generating Mod-
35 elica models from BIM, only *IFC2Modelica* explicitly
36 supports the generation of HVAC models.

37 All of the BIM to BEPS tools mentioned above, both for
38 traditional and Modelica models, are based on file-based
39 BIM, where, e.g., IFC files are imported into the tool.
40 Several critics argue that the use of file-based BIM models
41 where data only flows one-way limits interoperability and
42 argue for a shift from file-based to web-based BIM [35, 36].
43 The web-based approach prevents local version control
44 and information loss every time a BIM model is updated.
45 A web-based BIM setting demands interoperable, open
46 data formats, that allow full integration between different
47 tools and platforms. This corresponds to maturity level

3 on the Bew-Richards BIM maturity model described
in [37]. In BIM maturity level 3 two-way information
exchange is handled through standardized, open data
formats for integration with various tools.

Seidenschur et al. [38] introduced a common data
environment (CDE) to contain HVAC BIM models in
a centralized database, thereby moving HVAC BIM to
BIM level 3. The CDE system architecture is structured
around the Flow System Classes (FSC) object model,
which is used to represent the topology and performance
properties of the components and systems. Seidenschur
et al. mentions that the CDE might be developed further
by adding a microservice architecture to the platform.
The microservice architecture allows for future compu-
tational tasks to be implemented as microservices and
executed when needed from the main application.

1.1 Aim

This article presents an automated toolchain that per-
forms detailed simulations of HVAC systems using Mod-
elica and technical building service system information
stored in a CDE. This tool lowers the barrier for detailed
analysis of the dynamics in HVAC systems during design
and operation and thereby facilitates a better inclusion
of HVAC components and their non-idealized behavior
in building performance simulations.

The aims of this article are to:

1. describe the underlying methodology and approach
of the toolchain
2. showcase the versatility and interoperability of the
tool in combination with a common data environ-
ment.
3. show the potentials of Modelica for HVAC analyses
through a demonstration case.
4. propose future work for scalability and usability of
the toolchain and similar applications.

1.2 Outline

Section 2 describes the translation from FSC to Modelica,
including a description of the FSC object model and
the microservice architecture in section 2.1. Section 3.1
presents a case study, where the toolchain is used to
analyze the consequences of changing the heating curve
in a fictional building. In section 4 we discuss the usability
of the application, and how it can be scaled to actual
workflows and present a roadmap for future work.

2 TOOLCHAIN PRESENTATION

In this paper, the CDE designed by Seidenschur et al.
[38] is used to create an application that uses the data
structure presented in section 2.1 for automated genera-
tion of HVAC simulation models in Modelica language.
This section describes the application and its integration
with the CDE. The system architecture of the CDE pro-
posed by Seidenschur et al. [38] will not be discussed in
this article.

1 The toolchain is developed as a microservice, which
2 means that it is a stand-alone service, that communicates
3 with the CDE through an API, which allows the CDE
4 to request simulation results through the HTTP request-
5 response protocol. When the toolchain receives an HTTP
6 *request* with the FSC object model, the application trans-
7 lates the model into Modelica language, simulates the
8 model with Dymola, and returns an HTTP *response* with
9 the results in a JSON format. Figure 1 shows the main
10 parts of the process.

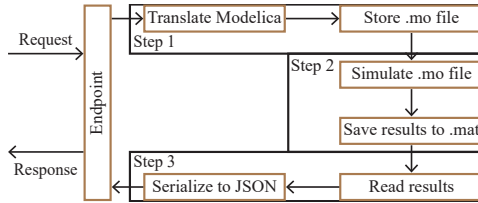


Figure 1. The application's main processes.

11 Section 2.1 describes the FSC object model, used in
12 the CDE. Section 2.2 presents modeling requirements
13 and section 2.3 explains the translation between FSC and
14 Modelica.

15 2.1 FSC object model

16 The CDE uses the FSC object model, developed by Sei-
17 denschnur et al. [38], as a description language for HVAC
18 systems in web-based environments. The FSC object
19 model is a standardized way to describe HVAC compo-
20 nents, their properties and connections. The object
21 model is defined with a class hierarchy, consisting of 27
22 classes to represent a wide range of HVAC components
23 on a generic and detailed level. Such components include;
24 **FlowMovingDevice**, **FlowSegment**, **EnergyConversion-**
25 **Device**, **Fitting**, **FlowTerminal**, **FlowController**. All
26 components share attributes from the generic top-level
27 class **Component**. The shared attributes are listed and
28 explained in table 2.

29 The **ConnectedWith** attribute holds a list of
30 **Connectors**, that define the component's connections
31 to other components. A connector holds information
32 about its dimension, shape, flow, and position. A con-
33 nector can be interpreted as the end of a component,
34 and thereby contain important geometrical information.
35 E.g., the length of a pipe segment can be calculated using
36 the distance between its two connectors. All attributes
37 related to the **Connector** class are listed in table 3.

38 Figure 2 shows a ventilation duct, represented in FSC.
39 As seen, the geometrical information, such as the di-
40 mension, is only represented by the connectors in the
41 **ConnectedWith** attribute. This allows transition pieces,
42 tees, etc., to have different shapes and dimensions in each
43 connector.

44 For the toolchain, we extended the FSC object model
45 with a simple description of control logics. The extension

Table 2. Attributes for the **Component** class.

Attributes	Description
Id	Unique identifier across all plat- forms
Tag	Revit component tag, used as pri- mary identifier
ComponentType	Type of the component - equal to the class name
SystemName	Name of the component's system - free text
SystemType	Type of the component's system; heating, ventilation or cooling
ConnectedWith	List of connectors (see table 3) that describe connections to other components
ContainedInSpaces	List of spaces, that contain the component - usually one space

Table 3. Attributes for the **Connector** class.

Attributes	Description
Tag	Tag of the connected component
Dimension	List with either diameter or height/width
Shape	The shape of the connector/end
DesignFlow	The design flow rate through the con- nector
Coordinate	List with X, Y and Z coordinate for the connection point
ConnectorType	Describes the direction and type of flow

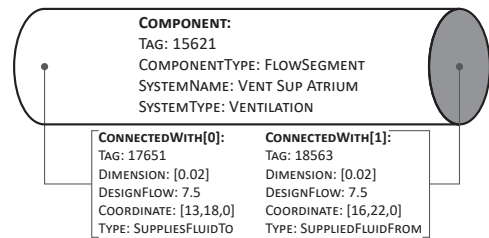


Figure 2. Example of a duct represented by FSC.

1 includes a new attribute, **Control**, for components that
 2 are controlled externally, such as **MotorizedValves** or
 3 **Fans**. The attribute refers to a new **Controller** that
 4 specifies (1) the controller type, (2) the sensor's **Tag**,
 5 (3) the process variable (temperature or CO₂) and (4)
 6 the setpoint (in °C for temperature or PPM for CO₂).
 7 Controllers that control the temperature or CO₂ levels
 8 in rooms simply refer to the room tag instead of a sensor,
 9 since room sensors are not supported in FSC.

10 We added two sensor classes, **TemperatureSensor** and
 11 **PressureSensor**, to the FSC object model to support
 12 measurements within duct and pipe systems. This ex-
 13 tension of FSC allows the representation of simple con-
 14 trol relations, but more complicated cases with varying
 15 setpoints or cascade control, require a more extensive,
 16 stand-alone format.

17 2.2 Modeling requirements

18 This section presents general requirements for successful
 19 HVAC simulations with the toolchain. Specific require-
 20 ments for component information such as performance
 21 data are specified in the component templates (see section
 22 2.3) and are partly covered in section 3.2.

23 The following general requirements apply to HVAC
 24 components:

- 25 1. All component connections must be correctly speci-
 26 fied in the FSC model.
- 27 2. In components that interact with spaces, e.g.,
 28 through heat or air transfer, the connected space
 29 must be specified with the **contained_in_spaces**
 30 attribute.
- 31 3. All components must be modeled explicitly unless
 32 the implicit objects are specified in the component
 33 templates (see section 2.3 and 3.2.1).
- 34 4. Chillers and boilers are not covered in FSC, and
 35 thus unconnected pipe ends represent a production
 36 unit. The same applies to duct ends connected to
 37 the outside.
- 38 5. Component dimensions must be specified in the com-
 39 ponent's connectors.
- 40 6. Where applicable, performance properties for nomi-
 41 nal conditions must be specified. The nominal con-
 42 ditions are not required to be identical to the design
 43 conditions. The needed properties depend on the
 44 specified mapping template, such as the one in sec-
 45 tion 3.2.
- 46 7. Components, such as control valves, that are regu-
 47 lated by a controller, must include information on
 48 the control logic, as described in 2.1.

49 2.3 Parsing the FSC object model to Modelica

50 The key functionality of the toolchain is translating the
 51 FSC object model into Modelica language. This section
 52 describes the translation process and with the default
 53 mapping scheme, presented in section 3.2, and the source

code on GitHub², the reader should understand the steps
 needed to expand or modify the translation for other use
 cases.

To explain the translation process, it is important to
 understand the structure of both FSC and Modelica mod-
 els. Figure 3 represents a system with three components
 in both FSC (figure 3a) and Modelica (figure 3b) to
 showcase similarities and differences. A significant dif-
 ference between FSC and Modelica is that FSC defines
 each connection twice, whereas Modelica needs only one
 definition. This introduces redundant definitions in FSC,
 but since connections are defined inside each component
 they must be represented in both connected components.
 In Modelica, connections are established independently,
 in the **equation** part of the model definition (see listing
 1). In contrast to FSC, Modelica works with so-called
ports for connections, adding additional information on
 the connectivity. In most flow system cases, a component
 has an input port and an output port, typically named
port_a and **port_b**, respectively. In addition, there may
 be ports for heat transfer, control input, etc.

Listing 1 shows a simplified Modelica model for the
 three-component system in figure 3b. The three com-
 ponents are instantiated with a *component string* for
 each component in the top part of the model. After the
equation statement, each connection is instantiated with
 a *connection string*. Thus, components and connections
 are instantiated separately and in separate parts of the
 file.

```

1 model ExamplePumpModel
2   ToolchainLib.PumpConstantPressure
3     pump1250835(
4       Medium = MediumHeating,
5       pumpCurve = {...},
6       ...);
7   FixedResistances.HydraulicDiameter
8     seg1458644(
9       Medium = MediumHeating,
10      m_flow_nominal=0.12,
11      dh=0.022,
12      length=1.34);
13   FixedResistances.HydraulicDiameter
14     seg1458738(
15       Medium = MediumHeating,
16       m_flow_nominal=0.12,
17       dh=0.022,
18       length=0.45);
19 equation
20   connect(seg1458644.port_b,pump1250835.port_a);
21   connect(pump1250835.port_b,seg1458738.port_a);
22 end ExamplePumpModel;
```

Listing 1. Simplified Modelica model of the system
 presented in figure 3b.

²<https://github.com/Virtual-Commissioning/VC-Modelica-Service>

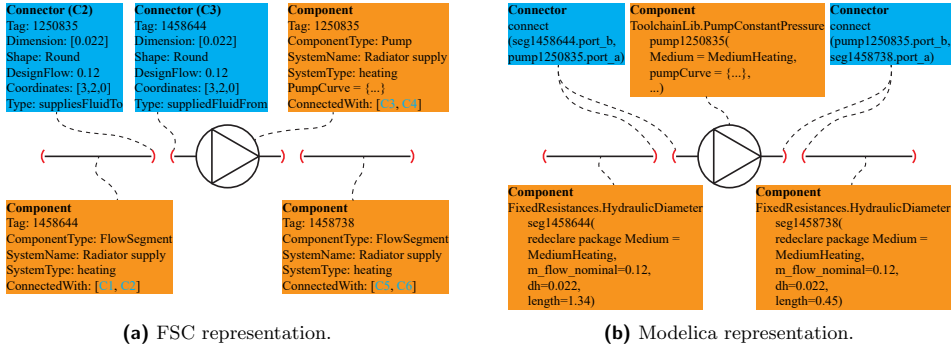


Figure 3. Representation of three components with FSC and Modelica.

1 All components in FSC are mapped to a Modelica
2 component template to handle all necessary information.
3 This template contains a set of attributes and functions
4 to create (1) the *component string*, which is the Modelica
5 representation of the single component and (2) the *con-*
6 *nection string* which is the Modelica representation of a
7 connection. To build the *component string*, the template
8 needs component attributes, such as the pressure curves
9 for pumps or Kv values for valves. Most component at-
10 tributes are either given directly in the FSC component
11 representation or derived from other attributes, which
12 makes it simple to build the *component string*. The
13 *connection string* requires no information on component
14 attributes, and the only component-related information
15 is the names of the components and connection ports.
16 This information is defined in the component templates
17 that return the Modelica name and the port name to the
18 application that serializes them into *connection strings*.
19 The main attributes and functions in all templates are
20 shown in table 4.

21 As stated above, the initial step in the translation
22 process is to map each component in the FSC model to
23 a component template and generates a *component string*.
24 If the component has an external controller, a controller
25 is also instantiated, based on a controller template.

26 After all components are mapped to component tem-
27 plates, the Modelica *connection strings* are created. To
28 avoid duplicate connections, the application considers
29 only input connections for each component. For each
30 input, a connection string is added to the model. In the
31 process, the application accesses information on Model-
32 ica and port names in both the main component and
33 the input component. Since FSC provides no informa-
34 tion about which port a component is connected to, the
35 `get_output_port` and `get_input_port` functions are used
36 to implicitly find the name of the connected ports. If
37 the component is connected to a space, the application
38 invokes the `connect_to_space` function in the compo-
39 nent template. Similarly, any controller connections are
40 instantiated.

41 Lastly, the program combines the *component strings*

Table 4. Main attributes and functions in component templates.

Function/attribute	Description
<code>FSC_object</code>	The original FSC object
<code>get_component_string</code>	Function that instantiates the component in Modelica language
<code>get_input_port</code>	Returns the name of the component's input port. Needs information on the connected component
<code>get_output_port</code>	Returns the name of the component's output port. Needs information on the connected component
<code>connect_to_space</code>	Function that instantiates a connection between the component and the containing room. By default, the function establishes no connection

1 and *connection strings* in a Modelica file and returns it
2 to the application.

3 2.3.1 Translation of spaces

4 In the FSC object model, the main objective of defin-
5 ing spaces is to establish boundary conditions for flow
6 calculations and system sizing [38]. Consequently, the
7 format does not provide sufficient information for energy
8 simulations. However, spaces are translated equivalently
9 to components, and space templates are defined similarly
10 as component templates. In the default template set,
11 presented in 3.2, spaces are mapped to a generic model
12 of a thermal zone of fixed size.

13 Since the FSC object model specifies no space adja-
14 cency, this is not supported in the application.

15 2.3.2 Example: Heating coil template

16 This section describes the attributes and functions in the
17 template for a heating coil and shows how it is used in
18 the process described in the previous section. Listing 2
19 shows the serialized FSC model of the heating coil.

```

1 {
2   "Tag": "1321808",
3   "ComponentType": "HeatingCoil",
4   "SystemType": "heating",
5   "NomPower": 5220.0,
6   "NomSupplyTemperaturePrimary": -15.0,
7   "NomReturnTemperaturePrimary": 28.4,
8   "NomSupplyTemperatureSecondary": 60.0,
9   "NomReturnTemperatureSecondary": 40.0,
10  "NomFlowPrimary": 100.0,
11  "NomFlowSecondary": 0.06,
12  "NomDpPrimary": 6.5,
13  "NomDpSecondary": 1620.0,
14  "ConnectedWith": [
15    {
16      "Tag": "1458980",
17      "DesignFlow": 0.06,
18      "ConnectorType": "suppliesFluidTo"
19    },
20    {
21      "Tag": "1458698",
22      "DesignFlow": 0.06,
23      "ConnectorType": "suppliesFluidFrom"
24    },
25    {
26      "Tag": "1519849",
27      "DesignFlow": 168.0,
28      "ConnectorType": "suppliesFluidTo"
29    },
30    {
31      "Tag": "1324249",
32      "DesignFlow": 168.0,
33      "ConnectorType": "suppliesFluidFrom"
34    }
35  ]
36 }

```

21 Listing 2. Serialized FSC model of a heating coil.

The `get_component_string` function instantiates 22
the model in Modelica language, based on the at- 23
tributes in the FSC model. Listing 3 shows 24
the python function that specifies the corresponding 25
Modelica model (`Buildings.Fluid.HeatExchangers.- 26`
`DryCoilEffectivenessNTU`) and writes nominal perfor- 27
mance data. 28

```

1 def get_component_string():
2     component_string += f'''
3     Buildings.Fluid.HeatExchangers.
4     ↪ DryCoilEffectivenessNTU
5     ↪ {modelica_name}{
6         redeclare package Medium1 = MediumVentilation,
7         redeclare package Medium2 = {medium.name},
8         m1_flow_nominal = {NomFlowPrimary * 10**(-3)},
9         m2_flow_nominal = {NomFlowSecondary * 10**(-3)},
10        dp1_nominal = {NomDpPrimary},
11        dp2_nominal = {NomDpSecondary},
12        Q_flow_nominal = {NomPower},
13        T_a1_nominal =
14        ↪ {NomSupplyTemperaturePrimary+273.15},
15        T_a2_nominal =
16        ↪ {NomSupplyTemperatureSecondary+273.15});
17        '''
18     return component_string

```

Listing 3. `create_component_string` function for the
heating coil template.

The `get_input_port` and `get_output_port` functions 31
return the name of a component's port when connecting 32
to another component. For most components with only 33
two ports, this is trivial; the input port is `port_a` and 34
the output port is `port_b`. However, the heating coil has 35
four fluid ports; two on the primary side and two on the 36
secondary side. Consequently, the `get_input_port` and 37
`get_output_port` functions must be able to return the 38
correct port, based on the connected component. 39

The Modelica heating coil model has four ports: 40
`port_a1` and `port_b1` are input and output on the pri- 41
mary side, and `port_a2` and `port_b2` are input and output 42
on the secondary side. Thus, if the connected compo- 43
nent is in the ventilation system, the functions returns 44
`port_a1` and `port_b1`, otherwise `port_a2` and `port_b2`. 45
This approach assumes that at least one of the streams 46
is a ventilation stream. 47

The `connect_to_room` returns no room connection 48
string since heat transfer between the heat exchanger 49
and the surroundings are not considered. 50

Translating the component itself is straightforward, 51
through the `create_component_string` function in list- 52
ing 3. The component's Modelica name is a short ver- 53
sion of the component type, followed by the `tag`. For 54
this component it is `heatCoil1321808`. The medium 55
depends on the system type. For the heating system, it is 56
`MediumHeating`, which is defined at the top of the model 57
file. Thus, the translation process returns the component 58

1 string in listing 4.

```
1 Buildings.Fluid.HeatExchangers.DryCoilEffectivenessNTU
  ↳ heatCoil1324249(
2   redeclare package Medium1 = MediumVentilation,
3   redeclare package Medium2 = MediumCooling,
4   m1_flow_nominal=0.12,
5   m2_flow_nominal=0.06,
6   dp1_nominal=6.5,
7   dp2_nominal=1620,
8   Q_flow_nominal=5220,
9   T_a1_nominal=258.15,
10  T_a2_nominal=333.15);
```

2 **Listing 4.** Modelica representation of the heating coil in
3 listing 2.

4 When translating the connections, the algorithm must
5 find the connected components and determine their sys-
6 tem type. Here, components 1519849 and 1324249 are
7 in the ventilation system, whereas 1458980 and 1458698
8 are in the heating system. Listing 5 shows the resulting
9 connections.

```
1 connect(heatCoil1324249.port_b1, heatCoil1321808.port_a1)
2 connect(seg1458698.port_b, heatCoil1321808.port_a2)
3 connect(heatCoil1321808.port_b2, seg1458980.port_a)
4 connect(heatCoil1321808.port_b1, red1519849.port_a)
```

10 **Listing 5.** Modelica connections for the heating coil in
listing 2.

11 3 DEMONSTRATION CASE

12 To show the potential of the application we tested it on
13 a purpose-built, fictional building model. In the exam-
14 ple, the toolchain simulates the building’s response to
15 two different heating curves during ideal and faulty user
16 operation. Section 3.1 describes the building and HVAC
17 system. Section 3.2 describes the model generation pro-
18 cedure, the custom-built library and the default mapping
19 template. Section 3.3 explains the background for such
20 an analysis and the investigated scenarios and section 3.4
21 shows the simulation results.

22 3.1 Use case model

23 The use case is a building with four identical rooms of 25
24 m², as shown in figure 4 and 5. Each room has 3.2 m²
25 south-facing windows and is occupied from 7:30 to 16:30.
26 To model varying loads, the occupancy in each room
27 ranges between 1 and 6 people within a 7-day period.

28 A VAV-controlled ventilation system controls the CO₂
29 levels in the rooms with a setpoint of 1000 ppm. The min-
30 imum supply air temperature is 20 °C, and the maximum
31 is 24 °C. The cooling system limits the ventilation supply
32 temperature but does not control the room temperature.
33 For simplicity, the air handling unit has been reduced

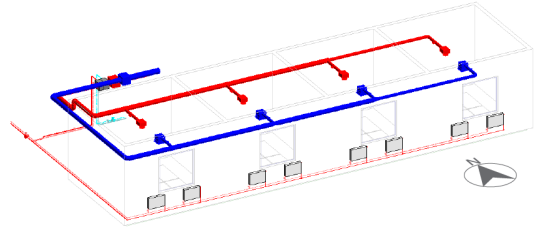


Figure 4. 3D representation of the example building.

to supply/return fans and heating/cooling coils (see fig- 34
ure 5). Motorized control valves control the hydronic 35
flow through the coils to maintain the desired ventilation 36
supply temperature. 37

The heating system consists of 8 radiators for space 38
heating and a heating coil to heat the supply ventilation 39
air. The heating system is connected to a generic heat 40
source, where a weather compensation curve defines the 41
supply temperature. 42

The cooling system, which feeds the cooling coil in 43
the ventilation system, is connected to a source with a 44
constant temperature of 5 °C. The pumps in the heating 45
and cooling systems provide a constant head. 46

51 3.2 Model generation

We modeled the use case in Autodesk Revit and added all 48
necessary attributes to the project’s component families. 49
Through the FSC converter (see [38]), we parsed the Revit 50
model to the database. All information came directly 51
from the Revit model, but performance data can also be 52
added after parsing to the database if needed. 53

To generate the simulation model, we developed a set 54
of component templates, shown in table 5, along with a 55
custom Modelica library, described in section 3.2.1. The 56
template set and Modelica library make up the default 57
translation capabilities, distributed with the application. 58
Since the use case was designed to include a large variety 59
of components, the template set is sufficient for many 60
applications. However, extending the default set with 61
other components is simple and requires few programming 62
skills. 63

Several assumptions were made in the default template 64
set. These assumptions, listed below, are built into the 65
component templates or the custom Modelica library. 66

- Occupancy schedules were defined directly in the cus- 67
tom Modelica room model. The schedule defines the 68
load profile in percentage of the maximum number 69
of occupants. 70
- To avoid fan operation during unoccupied hours, 71
the Modelica controller template for fans includes a 72
predefined operation schedule from 7:30 to 16:30. 73
- FSC does not support the definition of setpoints 74
and weather compensation curves in the heating and 75
cooling plants. Thus, plant setpoints and weather 76

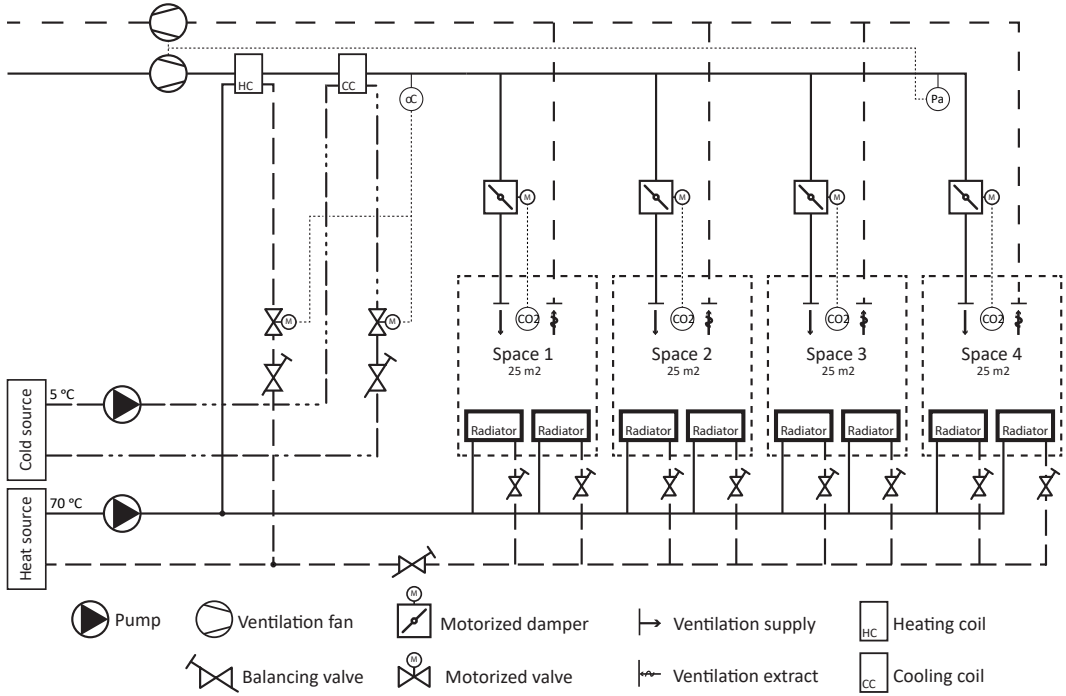


Figure 5. Schematic representation of the HVAC system in the use case. HC: Heating coil. CC: Cooling coil.

compensation curves were defined in the component templates.

- P- and I constants for controllers were hardcoded in the templates. E.g., for fans, $k = 0.01$ and $t_i = 1000$.
- Radiator setpoints were specified in the templates, and in the use case, they were manually changed for each scenario.

Table 5 shows the mapping between FSC component types and Modelica model templates with corresponding port names, as defined in the default template set.

As seen in table 5 fittings, such as bends and reductions, are modeled as pressure drops with a fixed resistance factor, even though dedicated Modelica models exist. With this approach, the resistance factors are calculated in the model template instead of at each time step in a complicated Modelica model. This reduces the simulation time but consequently reduces the precision.

All spaces are mapped to a standard room of 25 m², defined by the *RoomDetailedProfile*. This means that all spaces, no matter their size or shape, are mapped to the same room model in Modelica.

3.2.1 Custom Modelica library

For this use case, we built a Modelica library³ with custom models to support components that do not translate to a single model. Alternatively, the corresponding model

³Distributed on GitHub with the toolchain.

templates could instantiate and connect multiple Modelica models, but with the library approach, it is simpler to combine Modelica models through a graphical user interface. The models, described in table 6, are all extensions of existing models in *Buildings* and *Modelica Standard Library*. The corresponding FSC component types can be seen in table 5, which shows the link between FSC types and Modelica models.

3.3 Scenarios

Decreasing the supply temperature in local and centralized heating systems is a key component in decarbonizing space heating of buildings [39]. Low-temperature district heating increases the efficiency of large-scale heat pumps, allows recycling of low-temperature heat sources and reduces heat losses in the distribution network. However, in existing systems, low-temperature district heating is vulnerable to inefficient operation, since the low supply temperature leaves a small margin for the heat output [40, 41]. Thus, low-temperature district heating requires tools for investigating heating curves, pump pressures, system temperatures, and operational faults. The toolchain has all the necessary functionalities for such analyses.

This case study illustrates how the toolchain can compare two different weather compensation curves, shown in figure 6, and their responses to faulty operation. Under faulty operation, the setpoint for one radiator in each room is increased by 1.5 °C. On a standard, numeric-

Table 5. Mapping between FSC and Modelica classes.

FSC component type	Modelica model	Input ports	Output ports	Other ports
FlowSegment	FixedResistances.HydraulicDiameter*	port_a	port_b	
Pump	PumpConstantSpeed**	port_a	port_b	
Radiator	Radiator**	port_a	port_b	heaPor
HeatingCoil	HeatExchangers.DryCoilEffectivenessNTU*	port_a1	port_b1	
		port_a2	port_b2	
Bend	FixedResistances.PressureDrop*,***	port_a	port_b	
Tee	FixedResistances.Junction*	port_1	port_2	port_3
BalancingValve	Actuators.Valves.TwoWayLinear*	port_a	port_b	
MotorizedValve	Actuators.Valves.TwoWayEqualPercentage*	port_a	port_b	y
Reduction	FixedResistances.PressureDrop*,***	port_a	port_b	
AirTerminal	FixedResistances.PressureDrop*	port_a	port_b	
MotorizedDamper	Actuators.Dampers.Exponential*	port_a	port_b	y
BalancingDamper	Actuators.Dampers.Exponential*	port_a	port_b	
Fan	Movers.SpeedControlled_y*	port_a	port_b	y
PressureSensor	PressureSensor**	port_a	port_b	port_external statPres
TemperatureSensor	Sensors.TemperatureTwoPort*	port_a	port_b	
Space	RoomDetailedProfile**	airPorIn	airPorOut	heaPorAir heaPorRad
Cooling plant****	GenericPlant**	port_a	port_b	
Heating plant****	GenericPlantWC**	port_a	port_b	

*From *Buildings.Fluid* library **From custom library ***Pressure loss factor calculated from geometry ****Mapped implicitly - not represented in FSC

Table 6. Description of the custom library.

Model	Description
PumpConstPressure	A pump operating with a constant pressure setting
Radiator	A radiator and a thermostatic radiator valve
GenericPlant	A generic plant with a constant supply temperature (used for both heating and cooling)
GenericPlantWC	A generic plant with a supply temperature regulated by the external temperature
RoomDetailedProfile	A space energy model with a detailed occupancy profile
PressureSensor	Pressure sensor for measuring the static pressure in ducts
PIDControl	A PID controller with a constant setpoint
PIDControl_IO	A PID controller with a constant setpoint and an on/off schedule

scale thermostatic radiator valve, this corresponds to an increase of approximately 0.5 [42]. This is a commonly encountered issue, which results in high return temperatures because the high-setpoint radiator operates above its design output. The faulty scenario can also be interpreted as a scenario where two adjacent rooms have different setpoints and one room heats the other, which is also a commonly seen issue. In such a case, the effect is similar but smaller.

Figure 6 presents the investigated weather compensation curves and table 7 summarizes the four scenarios. Scenario *high_i* and *low_i* assume ideal operation of the radiators, and scenario *high_f* and *low_f* introduce faulty operation. In scenario *low_i* and *low_f*, the lower supply temperature is compensated with an increased pump pressure to ensure sufficient radiator output.

To investigate the effect of the heating curves under maximum heating loads, we disabled internal heat gains from occupants. Since the ventilation system is controlled for the CO₂-level, this means that the ventilation system did not operate in the simulations, and therefore the heating and cooling coils were out of operation for the entire period.

The simulation was carried out for seven days in the end of January, where the coldest period of the weather data occurs.

Table 7. Weather compensation curve scenarios.

	$high_i$	$high_f$	low_i	low_f
Heating curve*	High	High	Low	Low
Pump pressure [kPa]	11.5	11.5	30.0	30.0
Rad. 1 setpoint	22.0	22.0	22.0	22.0
Rad. 2 setpoint	22.0	23.5	22.0	23.5

*Refer to figure 6

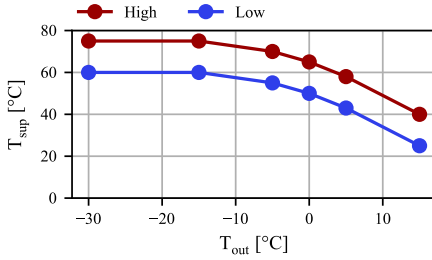


Figure 6. The investigated weather compensation curves.

3.4 Simulation results

Figure 7 and 8 show the results for all scenarios, and table 8 shows summarized results. From figure 7, it is seen that both scenarios are able to maintain the desired room temperature throughout the simulation period. Hence, the low heating curve is sufficient when the pump pressure is increased accordingly.

Under ideal operation (scenario $high_i$ and low_i), the low heating curve results in a 5 °C increase in the return temperature, which suggests that the low heating curve may have downsides under ideal operation, even though it is sufficient for maintaining the room temperature. However, under faulty operation (scenario $high_f$ and low_f), the low heating curve results in a 5 °C decrease in return temperature. This happens because radiator 1 has a higher share of the total heat output, as seen in figure 8 and table 8. Figure 8 shows the outputs of the radiators and their valve openings under faulty operation. Here, it's visible how the low heating curve balances the output of the two radiators.

The low heating curve provides a balanced solution that is more robust to faulty operation. In the ideal scenario it may not be optimal, but in cases with high user autonomy, it may be beneficial for robustness. Additionally it shows that low-temperature district heating in existing systems is not only possible but also has benefits. The low heating curve requires additional pump pressure, which has a cost of approximately 1.5 kWh in the simulation period. This may be neglected in a cost-perspective, since reduced return temperatures can provide monetary savings in many district heating networks.

Table 8. Key figures for the four scenarios.

		$high_i$	$high_f$	low_i	low_f
Pump power	[kWh]	0.37	0.41	1.67	1.85
Avg. supply temp.*	[°C]	72.4	72.4	57.3	57.3
Avg. return temp.*	[°C]	37.4	54.2	42.5	49.1
Accumulated flow	[m ³]	10.9	20.9	27.6	45.9
Heat consumption	[kWh]	436	441	432	440
Rad. 2 share	[%]	50	88	50	66

*Weighted according to heat consumption

4 DISCUSSION

The toolchain combines BIM maturity level 3, the state-of-the-art and open-source modeling capabilities of Modelica, and the simulation engine of Dymola, to create a fully interoperable framework for detailed simulations of HVAC systems. This lowers the barrier for detailed modeling and simulation of HVAC systems while creating a strong bond between BEPS and BIM to ensure that design progressions are represented in the BEPS results. As shown in section 3, where the tool was tested on an example building, it can successfully translate an FSC representation of an HVAC system to a Modelica model and simulate it through Dymola. Through minor adjustments to the Modelica model, we compared two heating curves under ideal and faulty operation to see the potential for lowering the heating curve and the impact on faulty operation. These specific results could be obtained without the toolchain through manual model generation, but in larger projects, the modeling task may be too time-demanding.

The tool translated the test case through the default template set that covers 16 component types (see table 5) commonly found in HVAC systems. The mapping can be changed or extended to support other Modelica libraries or new component types by modifying or adding templates in the program code.

Compared to previous approaches to combining BIM and Modelica, described in section 1, the presented toolchain distinguishes itself since it (1) is part of a web-based microservice architecture, (2) focuses primarily on HVAC systems and (3) showcases the powers of Modelica for analysis of HVAC systems. In the previous approaches, only *IFC2Modelica* supports HVAC systems, while all of the approaches work in a file-based manner and thus do not support BIM level 3.

The tool can support all phases of a building design, but the presented showcase and the default mapping template suit the detailed design phase. The tool is built for the CDE presented by Seidenschur et al. [38], who aim to add a virtual step to the traditional commissioning process, before the physical commissioning, called virtual commissioning. The toolchain provides crucial capabilities in this regard, but before it is fully usable for virtual commissioning, relevant use cases and test scenarios must

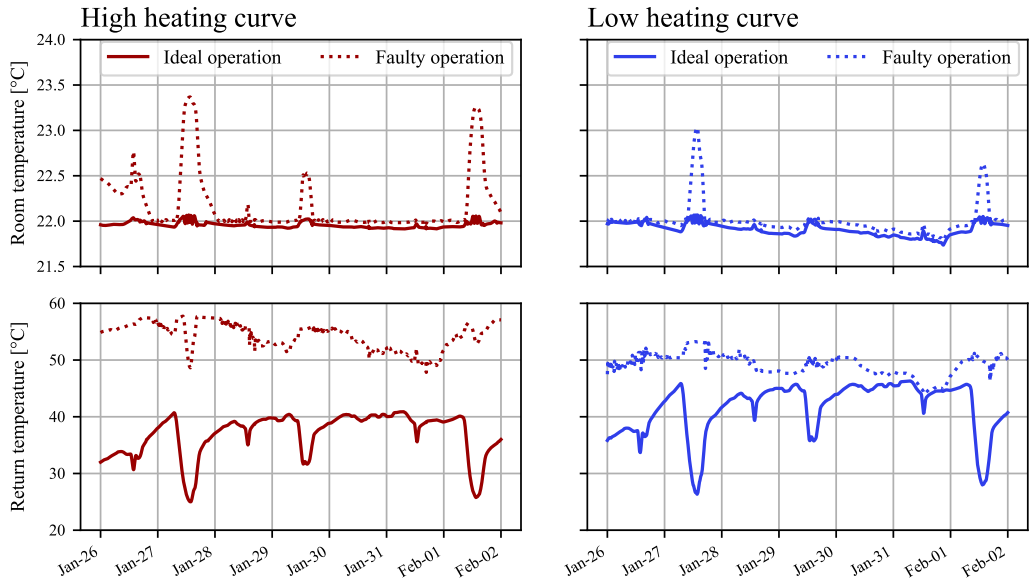


Figure 7. Simulated room and system return temperature.

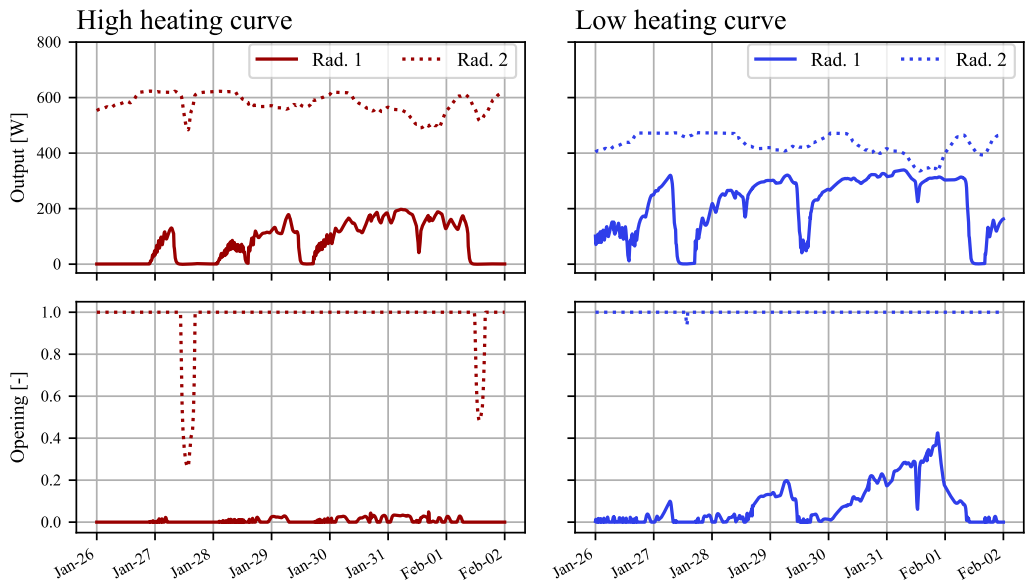


Figure 8. Simulated radiator output and valve opening under faulty operation.

1 be identified. Even though the tool was developed for the
2 CDE, it is a stand-alone application that is deployed on
3 a separate server. Therefore, the toolchain can be reused
4 in other platforms using the FSC object model.

5 The advantages of the microservice architecture are
6 manifold and while the advantages to interoperability and
7 data management are presented thoroughly in [38], the
8 server-based approach has obvious benefits when running
9 detailed simulations. With large models, the computing
10 requirements for simulations can become significant, and
11 by hosting the application on a stand-alone server, the
12 burden is shifted from the user's local PC to a more
13 powerful server.

14 4.1 Limitations and future work

15 The lack of HVAC controls in FSC is a significant barrier
16 to the general scalability of the application. Without
17 information on the controls, the simulation models are
18 incomplete since they are a vital part of detailed HVAC
19 simulations. This barrier applies to the entire BIM do-
20 main since no traditional BIM formats and tools support
21 control relations and sequences.

22 To overcome the barrier of controls, we extended FSC
23 with support for simple PI controllers for valves, dampers,
24 fans, and pumps, along with in-stream sensors for tem-
25 perature and pressure. In this implementation, relevant
26 components are connected to a controller with a fixed
27 setpoint and a connection to a sensor, as described in
28 section 2.1. However, HVAC controls are often more
29 complicated than simple PI controllers, connected to one
30 actuator and one sensor, with a fixed setpoint. Control
31 sequences are built of multiple, time-varying setpoints,
32 state changes, cascade controls, etc. In such cases, the
33 presented extension of FSC does not suffice. Therefore,
34 future applications should include a schema supporting
35 detailed definitions of controls. Such a schema could
36 be adapted from the Brick schema [43], which supports
37 description of points, commands, etc. With such a devel-
38 opment, naturally, a new mapping between that schema
39 and Modelica should be carried out.

40 As mentioned in section 2.3.1 and 3.2, all spaces are
41 mapped to a generic model of a thermal zone with a fixed
42 size, geometry and user profile. Since this application
43 focuses on HVAC systems, spaces were not considered
44 in detail. As long as FSC and the CDE do not support
45 the needed information for thermal zones, this will be
46 the case. An object model supporting thermal zones is
47 under development in the work presented by Seidenschur
48 et al. [44], where the thermal zones are translated into an
49 EnergyPlus model. Future work should aim to implement
50 this object model or other, similar models and translate
51 those to Modelica or use the EnergyPlus files directly with
52 Spawn of EnergyPlus, which is a part of the *Buildings*
53 library [25].

54 All pipe and duct segments, bends, and fittings are
55 mapped from FSC to Modelica and modelled individually
56 as fixed resistances, although the results for each segment
57 are without interest. This means that the solver must

calculate each segment individually in each time step, 58
which adds unnecessary calculation steps and can lead 59
to non-linear equations that complicate the solution and 60
increase computation time significantly. From a user 61
point of view, this makes the Modelica models immense, 62
if viewed through a graphical interface. Mathematically, 63
these fixed resistances can be summarized into a single 64
component, or better, added to other components without 65
any loss of precision. This requires a restructuring of 66
the application code but should be considered in future 67
work to reduce the complexity of the generated Modelica 68
models. 69

When using the toolchain for simulations, the require- 70
ments for the information level and completeness in the 71
BIM models are increased significantly. E.g., Kv val- 72
ues for valves and fan characteristics must be available, 73
and all components must be connected correctly. In some 74
cases, this shifts the modeling burden from the simulation 75
to the BIM process. However, as more tools utilize the 76
information in BIM models, one can expect an increased 77
motivation for creating and maintaining this information. 78
The information needed for simulations can also bene- 79
fit many other processes in the building lifecycle. E.g., 80
maintenance can more easily identify product specifica- 81
tions when replacing products and see previous settings 82
on balancing valves. In a fully interoperable framework, 83
the toolchain could be reversed to populate BIM models 84
with information from the simulation models. Thus, this 85
tool, along with many others, may provide an incentive 86
to improve the overall quality of BIM. 87

58 5 CONCLUSION

This article presented an automated toolchain, which is 89
a server-based application that interfaces the common 90
data environment (CDE) presented in [38]. The toolchain 91
uses fully interoperable formats and connections to gen- 92
erate and simulate models for HVAC systems, derived 93
from BIM models in the CDE. The simulation models are 94
written in the Modelica language and simulated with the 95
Dymola simulation environment. The tool was success- 96
fully tested on a purpose-built BIM model of a building, 97
where an analysis of the weather compensation curve's 98
impact on the systems' robustness showed the powers of 99
detailed HVAC analyses in Modelica. 100

The work showed that BIM-generated Modelica mod- 101
els is successful if the BIM model is defined correctly. 102
However, it also highlights a large gap in BIM, regarding 103
support for control sequences. Neither FSC or traditional 104
BIM schemas, support description of controls, except for 105
high-level properties, such as room temperature setpoints. 106
Support for control sequences would not only benefit 107
the domain of automated model generation but also the 108
design process in general. 109

110 ACKNOWLEDGEMENTS

This work was supported by the Innovation Fund Den- 111
mark under grant number 2040-00027B and 8090-00046B; 112

1 the Ramboll Foundation under grant number 2022-068;
 2 and Elforsk under grant number 352-042.

3 REFERENCES

- 4 [1] US DOE. Building Energy Software
5 Tools Directory, 2013. URL [https://www.
6 buildingenergysoftwaretools.com/](https://www.buildingenergysoftwaretools.com/).
- 7 [2] Anna Carolina Menezes, Andrew Cripps, Dino
8 Bouchlaghem, and Richard Buswell. Predicted vs.
9 actual energy performance of non-domestic buildings:
10 Using post-occupancy evaluation data to reduce the
11 performance gap. *Applied Energy*, 97:355–364, 2012.
12 doi: 10.1016/j.apenergy.2011.11.075.
- 13 [3] Pieter De Wilde. The gap between predicted and
14 measured energy performance of buildings: A frame-
15 work for investigation. *Automation in Construction*,
16 41:40–49, 2014. doi: 10.1016/j.autcon.2014.02.009.
- 17 [4] Guy R. Newsham, Sandra Mancini, and Benjamin J.
18 Birt. Do LEED-certified buildings save energy? Yes,
19 but... *Energy and Buildings*, 41(8):897–905, 2009.
20 doi: 10.1016/j.enbuild.2009.03.014.
- 21 [5] C. Carpino, E. Loukou, P. Heiselberg, and N. Arcuri.
22 Energy performance gap of a nearly Zero Energy
23 Building (nZEB) in Denmark: the influence of oc-
24 cupancy modelling. *Building Research and Informa-
25 tion*, 48(8):899–921, 2020. doi: 10.1080/09613218.
26 2019.1707639.
- 27 [6] Jesper Kragh, Jørgen Rose, Henrik N. Knudsen, and
28 Ole Michael Jensen. Possible explanations for the
29 gap between calculated and measured energy con-
30 sumption of new houses. *Energy Procedia*, 132:69–74,
31 2017. doi: 10.1016/j.egypro.2017.09.638.
- 32 [7] Emeka E. Osaji, Subashini Suresh, and Ezekiel
33 Chinyio. The impacts of contributory factors in
34 the gap between predicted and actual office build-
35 ing energy use. In *Smart Innovation, Systems and
36 Technologies*, volume 22, pages 757–778. Springer
37 Berlin Heidelberg, 2013. ISBN 9783642366444. doi:
38 10.1007/978-3-642-36645-1_68.
- 39 [8] Salvatore Carlucci, Marilena De Simone, Steven K.
40 Firth, Mikkel B. Kjærgaard, Romana Markovic,
41 Mohammad Saiedur Rahaman, Masab Khalid
42 Annaqeeb, Silvia Biandrate, Anooshmita Das,
43 Jakub Wladyslaw Dziedzic, Gianmarco Fajilla, Mat-
44 teo Favero, Martina Ferrando, Jakob Hahn, Mengjie
45 Han, Yuzhen Peng, Flora Salim, Arno Schlüter,
46 and Christoph van Treeck. Modeling occupant be-
47 havior in buildings. *Building and Environment*,
48 174(December 2019):106768, 2020. doi: 10.1016/
49 j.buildenv.2020.106768.
- 50 [9] Rongpeng Zhang and Tianzhen Hong. Modeling of
51 HVAC operational faults in building performance
52 simulation. *Applied Energy*, 202:178–188, sep 2017.
53 doi: 10.1016/j.apenergy.2017.05.153.
- 54 [10] Fu Xiao and Shengwei Wang. Progress and method-
55 ologies of lifecycle commissioning of HVAC systems
56 to enhance building sustainability. *Renewable and
Sustainable Energy Reviews*, 13(5):1144–1149, jun 57
2009. doi: 10.1016/j.rser.2008.03.006. 58
- 59 [11] Evan Mills. Building commissioning: A golden
60 opportunity for reducing energy costs and green-
61 house gas emissions in the United States. *Energy
62 Efficiency*, 4(2):145–173, may 2011. doi: 10.1007/
63 s12053-011-9116-8.
- 64 [12] Michael Wetter, Marco Bonvini, and Thierry S.
65 Nouidui. Equation-based languages - A new
66 paradigm for building energy modeling, simulation
67 and optimization. *Energy and Buildings*, 117:290–
68 300, apr 2016. doi: 10.1016/j.enbuild.2015.10.017.
- 69 [13] Cheong Peng Au-Yong, Azlan Shah Ali, and Faizah
70 Ahmad. Improving occupants’ satisfaction with ef-
71 fective maintenance management of HVAC system
72 in office buildings. *Automation in Construction*, 43:
73 31–37, 2014. doi: 10.1016/j.autcon.2014.03.013.
- 74 [14] Dorte Skaarup Østergaard, Kevin Michael Smith,
75 Michele Tunzi, and Svend Svendsen. Low-
76 temperature operation of heating systems to enable
77 4th generation district heating: A review. *Energy*,
78 248:123529, jun 2022. doi: 10.1016/j.energy.2022.
79 123529.
- 80 [15] Daniel H. Nall. Rightsizing HVAC equipment. *ASHRAE Journal*, 57(1):48–51, 2015. 81
- 82 [16] Paul Mathew, Steve Greenberg, David Frenze,
83 Michael Morehead, Dale Sartor, and William Starr.
84 Using measured equipment load profiles to ”right-
85 size” HVAC systems and reduce energy use in labo-
86 ratory buildings. *HPAC Engineering*, 6-29-05:1–14,
87 2005. 88
- 89 [17] Ery Djunaedy, Kevin van den Wymelenberg, Brad
90 Acker, and Harshana Thimmana. Oversizing of
91 HVAC system: Signatures and penalties. *En-
92 ergy and Buildings*, 43(2-3):468–475, feb 2011. doi:
93 10.1016/j.enbuild.2010.10.011. 94
- 95 [18] Steve Doty. Part-load HVAC Efficiency. *Energy
96 Engineering*, 107(3):6–28, mar 2010. doi: 10.1080/
97 01998591009709874. 98
- 99 [19] Woohyun Kim and Srinivas Katipamula. A review of
100 fault detection and diagnostics methods for building
101 systems. *Science and Technology for the Built Envi-
102 ronment*, 24(1):3–21, 2018. doi: 10.1080/23744731.
103 2017.1318008. 104
- 105 [20] IBPSA. IBPSA Project 1, 2016. URL [https://
106 ibpsa.github.io/project1/index.html](https://ibpsa.github.io/project1/index.html). 107
- 108 [21] D Müller, M Lauster, A Constantin, M Fuchs, and
109 P Remmen. Aixlib - an Open-Source Modelica Li-
110 brary Within the IEA-EBC Annex 60 Framework. In
111 *Proceedings of the CESBP Central European Sym-
112 posium on Building Physics and BauSIM 2016*, pages
113 3–9, 2016. 114
- 115 [22] Christoph Nytsch-Geusen, Jörg Huber, Manuel Lju-
116 bijankic, and Jörg Rädler. Modelica BuildingSystems
117 - eine Modellbibliothek zur Simulation komplexer en-
118 ergietechnischer Gebäudesysteme. *Bauphysik*, 35(1):
119 21–29, 2013. doi: 10.1002/bapi.201310045. 120

- 1 [23] F. Jorissen, G. Reynders, R. Baetens, D. Picard,
2 D. Saelens, and L. Helsen. Implementation and
3 verification of the IDEAS building energy simulation
4 library. *Journal of Building Performance Simulation*,
5 11(6):669–688, nov 2018. doi: 10.1080/19401493.
6 2018.1428361.
- 7 [24] Michael Wetter, Wangda Zuo, Thierry S. Nouidui,
8 and Xiufeng Pang. Modelica Buildings library. *Jour-
9 nal of Building Performance Simulation*, 7(4):253–
10 270, jul 2014. doi: 10.1080/19401493.2013.765506.
- 11 [25] Lawrence Berkeley National Laboratory - Simulation
12 Research Group. Spawn of EnergyPlus, 2021. URL
13 <https://lbnl-srg.github.io/soep/>.
- 14 [26] Torsten Blockwitz, Martin Otter, Johan Akeson,
15 Martin Arnold, Christoph Clauss, Hilding Elmqvist,
16 Markus Friedrich, Andreas Junghanns, Jakob Mauss,
17 Dietmar Neumerkel, Hans Olsson, and Antoine Viel.
18 Functional Mockup Interface 2.0: The Standard for
19 Tool independent Exchange of Simulation Models.
20 *Proceedings of the 9th International MODELICA
21 Conference, September 3-5, 2012, Munich, Germany*,
22 76:173–184, 2012. doi: 10.3384/ecp12076173.
- 23 [27] Gilles Guyon. Role of the model user in results
24 obtained from simulation software program. *Inter-
25 national Building Simulation Conference*, pages 377–
26 384, 1997.
- 27 [28] Salah Imam, David A. Coley, and Ian Walker.
28 The building performance gap: Are modellers liter-
29 ate? *Building Services Engineering Research
30 and Technology*, 38(3):351–375, 2017. doi: 10.1177/
31 0143624416684641.
- 32 [29] Yujie Lu, Zhilei Wu, Ruidong Chang, and Yongkui
33 Li. Building Information Modeling (BIM) for green
34 buildings: A critical review and future directions.
35 *Automation in Construction*, 83(February):134–148,
36 2017. doi: 10.1016/j.autcon.2017.08.024.
- 37 [30] Seyed Mohsen Hosseini, Reza Shirmohammadi, Al-
38 ibakhsh Kasaeian, and Fathollah Pourfayaz. Dy-
39 namic thermal simulation based on building infor-
40 mation modeling: A review. *International Journal
41 of Energy Research*, 45(10):14221–14244, 2021. doi:
42 10.1002/er.6740.
- 43 [31] Ando Andriamamonjy, Dirk Saelens, and Ralf Klein.
44 An automated IFC-based workflow for building en-
45 ergy performance simulation with Modelica. *Auto-
46 mation in Construction*, 91(March):166–181, 2018.
47 doi: 10.1016/j.autcon.2018.03.019.
- 48 [32] Christoph Nytsch-Geusen, Alexander Inderfurth,
49 Werne Kaul, Katharina Mucha, Jörg Rädler, Matthias
50 Thorade, and Carles Ribas Tugores. Template based
51 code generation of Modelica building energy simu-
52 lation models. In *Proceedings of the 12th Interna-
53 tional Modelica Conference, Prague, Czech Repub-
54 lic, May 15-17, 2017*, volume 132, pages 199–207.
55 Linköping University Electronic Press, jul 2017. doi:
56 10.3384/ecp17132199.
- 57 [33] Jong Bum Kim, Woonseong Jeong, Mark J. Clayton,
Jeff S. Haberl, and Wei Yan. Developing a physical
BIM library for building thermal energy simulation.
Automation in Construction, 50(C):16–28, 2015. doi:
10.1016/j.autcon.2014.10.011.
- [34] Ando Andriamamonjy, Dirk Saelens, and Ralf Klein.
An auto-deployed model-based fault detection and
diagnosis approach for Air Handling Units using
BIM and Modelica. *Automation in Construction*, 96
(August):508–526, dec 2018. doi: 10.1016/j.autcon.
2018.09.016.
- [35] Walter Terkaj, Georg Ferdinand Schneider, and
Pieter Pauwels. Reusing domain ontologies in linked
building data: The case of building automation and
control. *CEUR Workshop Proceedings*, 2050, 2017.
- [36] Mads Holten Rasmussen, Maxime Lefrançois,
Georg Ferdinand Schneider, and Pieter Pauwels.
BOT: The building topology ontology of the W3C
linked building data group. *Semantic Web*, 12(1):
143–161, nov 2020. doi: 10.3233/SW-200385.
- [37] M Bew and M Richards. BIM Maturity Model. In
Construct IT Autumn 2008 Members’ Meeting, 2008.
- [38] Mikki Seidenschur, Ali Küçükavci, Esben Visby
Fjerbæk, Kevin Michael Smith, Pieter Pauwels, and
Christian Anker Hviid. A common data environment
for HVAC design and engineering. *Automation in
Construction*, 142(March):104500, oct 2022. doi: 10.
1016/j.autcon.2022.104500.
- [39] Helge Averfalk, Theofanis Benakopoulos, Isabelle
Best, Frank Dammel, Christian Engel, Roman Geyer,
Oddgeir Gudmundsson, Kristina Lygnerud, Natasa
Nord, Johannes Oltmanns, Karl Ponweiser, Diet-
rich Schmidt, Harald Schrammel, Dorte Skaarup
Østergaard, Svend Svendsen, Michele Tunzi, and
Sven Werner. Low-Temperature District Heating
Implementation Guidebook. Final Report. Techni-
cal report, IEA DHC, 2021. URL <http://publica.fraunhofer.de/dokumente/N-640204.html>.
- [40] Benakopoulos, Salenbien, Vanhoudt, and Svendsen.
Improved Control of Radiator Heating Systems with
Thermostatic Radiator Valves without Pre-Setting
Function. *Energies*, 12(17):3215, aug 2019. doi:
10.3390/en12173215.
- [41] Theofanis Benakopoulos, Michele Tunzi, Robbe
Salenbien, and Svend Svendsen. Strategy for low-
temperature operation of radiator systems using data
from existing digital heat cost allocators. *Energy*, 231:
120928, sep 2021. doi: 10.1016/j.energy.2021.120928.
- [42] Danfoss. Data Sheet - Thermostatic Sensors Type
RA 2000, 2014.
- [43] Bharathan Balaji, Arka Bhattacharya, Gabriel
Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong,
Aslak Johansen, Jason Koh, Joern Ploennigs, Yu-
vraj Agarwal, Mario Bergés, David Culler, Rajesh K.
Gupta, Mikkel Baun Kjærgaard, Mani Srivastava,
and Kamin Whitehouse. Brick: Metadata schema
for portable smart building applications. *Applied
Energy*, 226(September 2017):1273–1292, 2018. doi:

1 10.1016/j.apenergy.2018.02.091.

2 ^[44] Mikki Seidenschmur, Ali Küçükavci, Esben Visby
3 Fjerbæk, Kevin Michael Smith, and Christian Anker
4 Hviid. A Common Data Environment with an Ener-
5 gyPlus microservice for Post-occupancy evaluation
6 of the Energy Performance Gap. 2023. [Manuscript
7 submitted for publication].

6.5 Paper V - Introducing a Semantic Web
Ontology and Rule-Set to Support Capacity-
and Size-Related Property Descriptions and
Validation of Heating, Ventilation and Air
Conditioning Components in The Design
Phase of Buildings

Introducing a Semantic Web Ontology and Rule-Set to Support Capacity- and Size-Related Property Descriptions and Validation of Heating, Ventilation and Air Conditioning Components in The Design Phase of Buildings

Ali Küçükavcı^{a,*}, Mikki Seidenschmur^{a,b}, Pieter Pauwels^d, Mads Holten Rasmussen^c, Christian Anker Hviid^a

^aDepartment of Civil Engineering, Technical University of Denmark, Copenhagen, Denmark

^bRamboll, Copenhagen, Denmark

^cNiras, Allerød, Denmark

^dDepartment of the Built Environment, Eindhoven University of Technology, Eindhoven, Netherlands

Abstract

Several OWL ontologies have been developed for the AEC domain, yet they often overlook the domain of building systems, e.g. HVAC components. The Flow Systems Ontology was recently proposed to address this need, but it does not include HVAC components' size and capacity-related properties. Despite their strengths in representing domain-specific knowledge, ontologies cannot solve poor data quality in BIM models. A five-fold contribution is made in this research paper to define and improve the data quality of HVAC knowledge: (1) extending Flow Systems Ontology, (2) proposing the Flow Properties Ontology, (3) proposing the HVAC rule model, (4) introducing a method for compliance checking on HVAC models (5) and designing HVAC component capacity using semantic web technologies. The demonstration case shows that we can represent the data model in a distributed way, validate it using 36 SHACL shapes and use SPARQL to determine the pressure and flow rate of fans and pumps.

Keywords: Building Information Modelling, Heating, Ventilation and Air Conditioning (HVAC), SHACL, Semantic Web technologies, Linked Data, Compliance checking, SPARQL

1. Introduction

1.1. A Document-centric AEC Industry

Architecture, Engineering and Construction (AEC) projects have become more technically complex and involve many stakeholders that must exchange information to complete a project successfully [1]. Since the Building Information Modeling (BIM) methodology was introduced in the early to mid-2000s [2], the AEC industry has experienced improvements in coordination and communication between project stakeholders and digital tools. The BIM methodology aims to achieve a more collaborative workflow and addresses the need for a Digital Information Hub [3]. It provides a method for managing structured, accessible, and reliable building data to represent the physical and functional characteristics of a 3D building model. Current BIM applications have improved the workflows across the building life cycle and typically include 3D modelling. For that reason, its use is focused on phases of the building life cycle where 3D modelling is a requirement [4]. Today, BIM methodology is mainly

based on a document-centric approach in the AEC industry, leading to poor data management across the building life cycle, disciplines, and digital tools [5]. Data is often outdated and not in sync with the real building model, for which no live access is available.

The Industry Foundation Classes (IFC) is currently the standard format of building information and has been applied to exchange the needed information among stakeholders, mainly in a file-based or document-centric approach. Extending the IFC schema with new domain-specific knowledge becomes difficult due to its monolithic structure and complexity [6]. In addition, the schema does not describe cross-domain information such as occupancy data, meteorological data, data from building automation and control systems (BACS), etc., nor information that links the different domain information to each other [4].

1.2. Linked Data & Semantic Web

The World Wide Web Consortium (W3C), with its participants consisting of academic and industrial partners, has developed open data standards for software developers to support the shift from a "Web of Documents" to a "Web of Data" [7]. They have developed the Semantic Web Technologies consisting of Resource Description Framework (RDF), RDF Schema (RDFS), Web Ontology Language (OWL), SPARQL Protocol and RDF Query

*Corresponding author

Email addresses: alikuc@byg.dtu.dk (Ali Küçükavcı), msei@ramboll.dk (Mikki Seidenschmur), p.pauwels@tue.nl (Pieter Pauwels), mhra@niras.dk (Mads Holten Rasmussen), cah@byg.dtu.dk (Christian Anker Hviid)

Language (SPARQL), and Shapes Constraint Language (SHACL). It is a framework that enables sharing, accessing, conforming, and linking data over the web in a machine-interpretable format [8, 9].

Contrary to the IFC schema, which has well-known limitations such as limited-expression range, difficulty partitioning information, and describing the same information in multiple ways, the W3C suggests more modular, polythetic, and simple data formats, also called ontologies, that can be interlinked and easily extended over time [6, 10, 11]. Figure 1 shows the concept of interconnected ontologies, and it can be seen that the domain-specific ontologies can be separated as smaller graphs and linked with other ontologies. An ontology does not need to cover an entire domain, such as HVAC systems. It can also cover minor subdomains for HVAC, such as representing different component types and their properties alone or the connectivity of HVAC components and their relations to systems and subsystems. Developing smaller ontologies that target one building domain will yield a practical and flexible way of modelling knowledge when combined [4, 12].

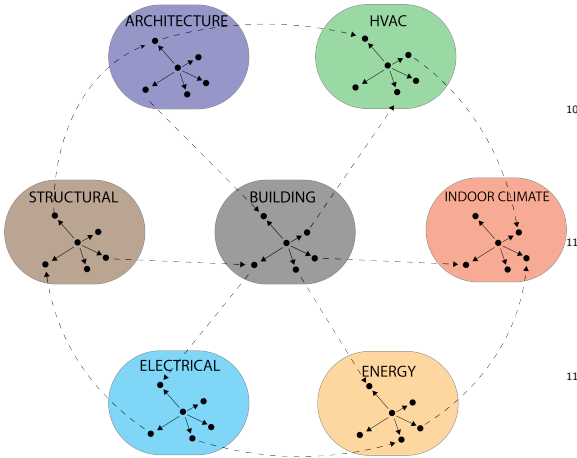


Figure 1: Interlinked domain-specific ontologies.

1.3. Interlinking Domain-specific Knowledge

In this context, the W3C LBD Community Group (W3C LBD CG) has defined and shared a set of ontologies like Building Topology Ontology (BOT) [13], Flow Systems Ontology (FSO) [14], TUBES System Ontology (TUBES) [15], Property Set Definition Ontology (PROPS) [16], and Product Ontology (PRODUCT) [17] etc. for the AEC industry. While FSO describes the energy and mass flow relationships between systems and their components and their compositions [14, 18], it lacks system components' capacity- and size-related properties. A key research question here is whether such properties need to be added directly to the FSO ontology, or can be kept separate, e.g. in its own module or ontology. In our research, we intend to

investigate whether the best approach is to create an ontology, called the Flow Properties Ontology (FPO), that includes only those properties and aligns it with other existing ontologies in the Linked Building Data (LBD) context, in particular with the FSO ontology that focuses on HVAC domain.

1.4. Conforming Domain-Specific Knowledge

Despite their strengths in representing domain-specific knowledge, ontologies cannot solve the problem that many BIM models are poorly modelled and lack building elements or metadata. Currently, poor data quality in building models contributes to faulty design decisions and downfalls in the information stream. Due to the increasing level of information, it is challenging to create sufficient BIM models [10, 19–21]. Architects and owners can spend hundreds of hours manually assessing conformity [22]. Due to the time-consuming process and the need for high-performing BIM models, many research publications have addressed conformance checking. The most prominent publications on conformance checking of BIM models cover various frameworks, tools, rule languages, rule models, and rule engines [23–33]. As their data models rely on IFC or their rule models lack semantic expressivity, they all have limitations and cause poor query performance [34, 35]. Soman et al. [36], Stolk and McGlinn [9], and Oraskari et al. [37] describe a promising approach to surpass the limitations of IFC and improve conformance checking. They use a semantic web approach with a data model written in OWL and a rule model written in SHACL to verify constraint violations. Soman et al. [36] applied the method to the construction field, while Stolk and McGlinn [9] applied the method to geospatial field, and Oraskari et al. [37] to the energy simulation field. However, these publications do not describe how to validate an HVAC model with SHACL, nor do they apply the framework to a real-world large building project. In addition, we intend to develop a rule model written in SHACL for validating HVAC-related constraints.

1.5. Contribution

Considering the above, several innovations are needed. In fact, our research includes five contributions. Firstly, our research aims to extend FSO to support an alignment with the proposed FPO ontology. Secondly, we propose the FPO ontology itself to represent HVAC components' capacity and size-related properties. Thirdly, we propose a set of rules to validate HVAC-related constraints. Fourthly, our work produces a demonstration environment for a real-world building project, showcasing how to conform a HVAC model using semantic web technologies. Lastly, the demonstration environment will showcase how FPO and the HVAC rule model can support the description and validation of hydraulics in HVAC components and the capacity of HVAC components.

1.6. Outline

Table 1 shows the namespaces and prefixes used in this article. The remainder of this article is structured as follows. Section 2 describes previous work on knowledge representation and rule checking related to buildings and systems. The presented work is limited to OWL-based data models and SHACL-based rule models. The development of FPO and extension of FSO are explained in Section 3. Section 4 outlines our framework and rules for validating HVAC-related constraints. We utilize a real-world building model in Section 5 to illustrate how FPO can represent capacity- and size-related properties and be used to design an HVAC device. Additionally, the real-world building model will be validated against our rule model in Section 5 where a process of four steps and a web application is introduced and applied to generate validation and capacity design results and display the results within a web interface. The validation results pinpoint the components or properties in the data model that are violating our rule model, while the capacity results show the flow rate and pressure of each flow-moving device that is represented in the data model. The validation and capacity design results are discussed in Section 6, and conclusions are presented in Section 7.

Table 1: Used prefixes and namespaces.

Prefix	Namespaces
fpo	https://w3id.org/fpo#
fso	https://w3id.org/fso#
fsosh	https://w3id.org/fsosh#
bot	https://w3id.org/bot#
s4bldg	https://saref.etsi.org/saref4bldg#
s4syst	https://saref.etsi.org/saref4syst#
brick	https://brickschema.org/schema/1.1/Brick#
seas	https://w3id.org/seas#
rdfs	http://www.w3.org/2000/01/rdf-schema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
ex	https://example.com/ex#
inst	https://example.com/inst#
owl	https://www.w3.org/2002/07/owl#

2. Background

2.1. Scope of the HVAC domain

The HVAC engineer is responsible for designing a building's HVAC system. The purpose of an HVAC system is to provide building occupants with acceptable thermal comfort and indoor air quality. HVAC engineers must go through a series of steps to design an HVAC system, such as defining the distribution strategy for HVAC, defining the control strategy, calculating HVAC demand by zones, and determining the capacity and size of HVAC systems and their components. To determine whether an

HVAC system is designed sufficiently, its cooling, ventilation and heating effects are compared with the building's cooling, ventilation, and heating demands. The HVAC system is considered sufficient when the capacity exceeds the building's demand. The HVAC engineer must design each HVAC component's capacity individually since an HVAC system's capacity equals the sum of its components. The HVAC component's size is then determined based on its capacity. The HVAC engineer can choose a product from a manufacturer once the capacity and size have been defined. By the time all HVAC components have been designed, the HVAC engineer has completed the HVAC design process.

Since our research project seeks to represent and validate an HVAC system's and HVAC component's capacity and size-related properties in a semantic web context, Section 2.2 provides an overview of what research has been achieved in this field and what is missing.

2.2. System representation in a Semantic Web context

A number of ontologies have been proposed to handle data within the AEC industry since the early 2000s. The first significant contribution towards moving BIM data into the Semantic Web is the ifcOWL ontology. IfcOWL is an OWL representation of the IFC schema [38, 39], and it is available at the buildingSMART website¹ as just another serialisation of the IFC schema, next to eXtensible Markup Language Schema Definition (XSD) and EXPRESS [40]. It is recognized that IFC is not the easiest method to model a building or infrastructure due to the complex relationships between building elements (mostly n-ary relationships) and the fact that it is an extremely extensive schema that is difficult to extend. Hence, this has hampered its direct use among AEC stakeholders [8, 41]. Moreover, it covers a wide range of domains, making it monolithic, rigid, and hard to extend [42]. The direct translation from the IFC schema to an OWL ontology does not change these inherent features of IFC, and so also the OWL ontology has the same limitations (complexity, limited extensibility, size). To resolve the issues, the W3C LBD CG developed a more modular and lightweight principle named LBD. This LBD approach takes a small, simple, and extensible building ontology at its core, known as the Building Topology Ontology (BOT) [13]. A BOT graph can be expanded with more specific details by interlinking with other ontologies like FSO, DOT, Brick, SAREF, etc.

BOT describes the relationship between building zones and elements [43]. A zone can be a building, a floor, a space, or a group of spaces. The building can be connected to the floor level by asserting that an entity of bot:Building is related to an entity of bot:Storey with bot:hasStorey. The same method can be applied between the storey and the space. Zones are related in BOT in a similar way

¹<https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications>

to the Babushka concept. In Babushka, smaller dolls are nested in larger dolls, whereas in BOT smaller zones are nested in larger zones. BOT can be used to describe the connections between zones in a building, but it cannot describe building systems.

SEAS describes the relationships between physical systems [44]. There are three main modules in the ontology, namely, The System Ontology, The Features Of Interest Ontology, and The Evaluation Ontology. The Features Of Interest Ontology allows to describe features of interest and their properties. A car, as an example, can be considered a feature of interest with a property called speed. Properties are either evaluated directly or through a qualified evaluation in the Evaluation Ontology. In a direct evaluation, a value is assigned to the property. A qualified evaluation needs to outline three categories: type, the context of validity, and provenance data. The System Ontology describes the systems and the relationships between them. There are three levels of connectivity: between systems, connections, or connection points. The SEAS ontology focuses primarily on electrical systems but can also be used to represent higher-level building services systems [44]. Yet, it does not describe any building service components or their relationships to building service systems.

Building service components are included in the Brick ontology [45] and the Smart Applications REference (SAREF) ontology [46] at different conceptual levels and scopes. The Brick ontology describes data points and their relationships to physical, logical, and virtual assets in buildings. It consists of a core ontology to describe fundamental concepts and their relationships and a domain-specific taxonomy. The ontology focuses on data points and their relations to location, equipment, and resource [45]. Relating a data point to a location expresses in which area of the building the data point is located. It can be located in a room, on a floor, in a duct, etc. Relating a data point to a specific equipment expresses how the data point controls the system or component. For example, take a room temperature sensor positioned in a room. The room temperature sensor regulates how much air an air handling unit (AHU) must supply to the room. Lastly, the resource is the medium being measured and regulated by the data point and equipment. For example, the medium of an AHU is the air that is being supplied to a room.

The SAREF Smart Appliances Reference ontology is a reference ontology for smart appliances (devices) [46]. It aims to bring meaningful interactions between Internet of Things (IoT) devices in various domains. There are currently 13 extensions to the core ontology. SAREF4SYST is based on the concepts of `seas:SystemOntology` to describe higher-level building service systems. SAREF4BLDG is based on the IFC taxonomy and describes building service devices. Even if it is similar to IFC and BOT, these structures are not fully the same [47]. Together, SAREF4SYST and SAREF4BLDG can represent building systems and their connectivity with IoT devices. Like Brick, the

SAREF ontology represents medium-level building system devices such as a fan or pump. Furthermore, SAREF4BLDG represents capacity-related building service devices to some extent. Those parameters are based on the IFC taxonomy. However, both Brick and SAREF ontologies are primarily focused on the operational phase of the building life cycle. As a result, they do not represent any passive building service devices such as pipes, ducts, tees, elbows, etc., nor their properties.

An OWL ontology that is similar to the SAREF4BLDG ontology, but does not include any building topology to avoid semantically overlapping ontologies, is the Mechanical, Electrical and Plumbing (MEP) ontology². This ontology is structured as a very simple hierarchical taxonomy for devices and is directly created based on the DistributionElement subtree in the IFC schema. It needs to be combined with the BOT ontology to be of use and works well to classify distribution elements such as air terminals, etc.

FSO focuses on the design and operational phase of the building life cycle [14]. It describes the mass flow and energy relationships between systems and components and the composition of such systems [14]. FSO gives the ability to connect both passive and active components to systems and subsystems. For example, a heating system can include a supply and return system as subsystems. A segment or fitting can be related to a supply or return system. A component can also be connected to a supply and return system, such as a heat exchanger. A segment can supply or return fluid to another component based on what system it belongs to. Unlike Brick and SAREF ontologies, FSO only represents higher-level components such as flow-moving device or flow-controlling devices (also included in the MEP ontology). The taxonomy of building service devices for all four ontologies is based on the IFC taxonomy. However, FSO does not represent both active and passive components' size- and capacity-related properties. Without that representation, HVAC engineers cannot design an HVAC system nor an HVAC component during the design phase using FSO.

FPO and an extended version of FSO are introduced in Section 3 to fill this research gap and describe the size- and capacity-related properties of both active and passive components within the design phase. Ontologies are mainly used to represent domain-specific knowledge. To check whether a BIM model lack building elements or metadata, we need a rule language. Section 2.3 describes the process of compliance checking, which rule languages exists and what research have achieved in this area in a Semantic Web context.

2.3. Compliance checking in a Semantic Web context

Compliance checking, code-checking, rule-based checking, and constraint checking are all terms that describe

²<https://pi.pauwel.be/voc/distributionelement>

a passive process that notifies whether a rule has been violated [48]. The process does not modify the building but validates the building design against different types of requirements such as client requirements, functional requirements, aesthetic requirements, building performance requirements, building code and regulations, complete discipline assessment and complete BIM data [22, 49]. Currently, companies primarily apply compliance checking to assess the quality and perform collision control on BIM models by utilizing the commercial tool Solibri Model Checker (SMC). Solibri Model Checker uses predefined rules for geometrical clashes, property completeness, and relationships between building elements. Using SMC does not allow the use of predefined rules in other applications or the creation of customized or complex rules [22]. In order to perform compliance checking on BIM models without being restricted to specific types of constraints or applications in general, Eastman et al. [50] provide a four-step manual approach.

1. Rule interpretation: Human-readable rules are converted into a machine-interpretable format that contains the information needed to be checked in the correct format, also known as the rule model.
2. Building model preparation: Building information is converted into a machine-readable format, also known as the data model.
3. Rule execution: The data model is validated against the rule model.
4. Rule check reporting: A validation report describing whether the data model has passed or violated any constraints.

By following these steps, custom rules can be written without being limited to a particular application. However, the process is passive and only informs the user or system whether any constraints have been met or violated. For actively correcting the violation in the data model, Solihin et al. [49] introduce a fifth step:

5. Automatic correction: If any constraints are violated, the user or system is not only notified, but new data is created to correct the violation. Users can be notified to implement the new data as an option or the new data can be implemented automatically. As some violations can be solved by multiple solutions, the system should be able to notify the user of all the possible solutions, allowing them to choose the appropriate one.

Moreover, Solihin et al. [49] suggest categorizing the defined rules based on their complexity into four categories:

Class 1: entities and attributes are queried and checked against a single value.

Class 2: additional values are calculated (e.g. distance) and checked.

Class 3: additional geometry is created, in order to calculate spatial relationships.

Class 4: problem solutions are calculated, and new data is created.

Defining each rule in the rule interpretation phase requires a rule language. In the following subsection, we describe several prominent rule languages developed by the W3C.

2.3.1. Rule languages

In 2004, the W3C introduced the Semantic Web Rule Language (SWRL) as a member submission³. SWRL is a combination of the OWL Description Language (DL) and OWL Lite sublanguages of OWL with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. OWL knowledge bases are integrated with Horn-like rules in the rule language. The rules are expressed in terms of OWL concepts, such as classes, properties and individuals. Because OWL ontologies are limited in their ability to express complex logical reasoning, SWRL allows users to create custom rules and apply them to OWL ontologies [51, 52].

Similar to SWRL, the Rule Interchange Format (RIF) introduced in 2005 by W3C allows rules to be expressed in XML syntax. In order to enhance interoperability between rule languages, RIF was designed to be the standard exchange format for rules on the Semantic Web. As of today, RIF consists of 12 parts, including RIF-core, which is the core of all RIF dialects [52, 53].

Notation3 (N3), is an assertion and logic language that supports expressing RDF-based rules. It was introduced in 2011 by W3C as a team submission to extend RDF by adding formulae, variables, logical implication, and functional predicates, as well as to provide an alternative syntax to the XML syntax that SWRL and RIF use. By using shortcuts and syntactic sugar, it is able to simplify statements in the form of triples [54].

The SPARQL Inferencing Notation (SPIN) was introduced by W3C in 2011 as a member submission and has become a de facto industry standard for describing SPARQL rules and constraints. The key feature of SPIN, compared to SWRL, RIF, and N3, is the ability to specify constraints using SPARQL queries. In this way, property values can be calculated based on other properties, or a set of rules can be isolated for execution under certain conditions. It is also possible to use SPIN to check the validity of constraints based on the assumption of a closed world [55].

SHACL is the successor to SPIN and was published as a W3C Recommendation in 2017 [56, 57]. A higher

³<https://www.w3.org/2021/Process-20211102/>

status has been granted to SHACL by W3C in comparison to SWRL, RIF, N3 and SPIN. As a result, SHACL has become the web standard today for validating RDF graphs. SHACL is heavily inspired by SPIN, but it offers far more flexibility in defining target constraints. SPIN is limited to classes, while SHACL can be applied to classes or sets of nodes by various target mechanisms, including customized targets. Furthermore, SHACL advanced features allow validation of more complex constraint types, such as sub-graph pattern validation, conditional validation, etc.. SHACL contains two major components:

Data graph: A data model containing domain-specific knowledge.

Shape graph: A rule model, consisting of user-defined constraints. User-defined shapes can be node shapes or property shapes. Node shapes specify constraints on target nodes, while property shapes specify constraints on target properties and their values.

By separating the data model and rule model, SHACL follows the Business Rule Management Systems (BRMS) principle of decomposing knowledge into logic and data, enabling them to be independently manipulated [36]. In addition, SHACL outputs an RDF graph with validation results, which describes whether a data model passed or failed a given rule-set.

The following section highlights the research gap based on an overview of recent research on applying SHACL to perform conformance checking within the AEC industry.

2.3.2. The research gap in case studies

Stolk and McGlinn [9] demonstrated how ifcOWL can be validated using SHACL. The authors showed how ifc:LengthValueIfcQuantityLength can be restricted to only have values of type ifc:IfcLengthMeasure and how cardinality constraints can be used to restrict IfcDoorPanel properties.

Hagedorn and König [56] developed an approach for compliance checking linked building models. The proposed method implements the four steps mentioned by Eastman using semantic web technologies. Using the IFC2RDF converter, the authors converted an IFC schema into ifcOWL. Their rule model involved a set of rules to validate the path between an identifier of a link and the original identifier. In order to validate their data model against the rule model and receive a validation report, they used the W3C SHACL Test Suite.

To define and check complex and dynamic scheduling constraints in construction, Soman et al. [36] developed a linked-data based constraint-checking approach utilizing semantic web technologies. The approach was implemented through a web application that validated construction scheduling violations using different types of constraints. The pySHACL library was used to define and validate SHACL shapes and the RDFlib library was used

to design and store a RDF graph. They used IfcOWL and LinkOnt to capture the model information of a real-building model.

Oraskari et al. [58] defined rules within the energy simulation field for validating windows of specific sizes, checksums of properties, and alignments of BOT classes and properties. They validated two data models against each other in order to align BOT classes and properties with ifcOWL. The IFC schema of a conceptual building model was converted to ifcOWL and BOT using the IFCtoLBD and IFC2BOT converters. The rule modelling, validation and reporting was performed using the TopBraid SHACL Application Programming Interface (API).

None of the mentioned authors developed a SHACL-based rule model nor performed a conformance check against an OWL-based HVAC model. Soman et al. [36] is the only author that uses a real building model, but a model of low complexity and size. For that reason, a constraint-checking approach to define and validate HVAC-related constraints on a large real-building model using semantic web technologies is introduced in Section 4 to fill this research gap.

3. Flow Properties Ontology

FPO is developed as an extension to FSO [14] to represent FSO component's capacity and size-related properties. The development of FPO is closely related to FSO, but the authors in [14] sought to keep FSO as lightweight as possible, to describe a myriad of different flow systems. As a result, we developed FPO as an extension to FSO. It contains 50 classes, 50 object properties and 6 data properties and has a Description Logic expressivity of $\mathcal{ALRF}(\mathcal{D})$ [59]. A practical guide [60] was used to design and structure the classes, object properties and data properties in FPO. Classes, for instance, should always begin with capital letters, also known as upper camel case, and should not contain spaces. In contrast, object properties and data properties should always be written in lower camel case and with verb senses.

It is necessary to know the HVAC component type to describe its properties. A property of one HVAC component may differ from another, and the data type or unit of one property may vary from another property. A pump has different properties than a fan, and the flow rate can be expressed in liters per second or cubic meters per hour which is different from a ventilation fan. An elbow can differ in properties from a tee by having an angle even if both are fittings. Moreover, while a tee has three flow ports and elbow has two flow ports. Conceptually, Figure 2 illustrates how a component can have a property, and the property a value. As there are two steps between the component (Type / Object) and the value, this property modelling approach is a Level 2 (L2) property modelling approach, as defined by Bonduel and Pauwels [61]. Other property modelling approaches are L1 (direct object

and data properties), and L2 (more metadata for tracking property states over time).



Figure 2: Relationship between components, properties, and property values.

It is possible to represent buildings, spaces, and their relationships with systems and components using FSO and BOT. Adding FPO, the representation can identify whether a particular system or component is able to heat, cool, or ventilate a specific building or space.

The following subsections provide a more detailed description of FPO. To determine the scope of the ontology, Section 3.1 lists a set of competency questions. In Section 3.2.2, FSO is extended with medium-level components to represent component interfaces and their connections with other components. Section 3.3 reviews FPO classes and their properties. Finally, reasoning examples will be enabled in Section 3.4, followed by alignments to FSO, SAREF4BLDG, MEP, and Brick in Section 3.5. Both the extension of FSO, the development of FPO and the alignments are made available on GitHub⁴.

3.1. Competency questions

Competency questions are listed in Table 2 to determine FPO’s scope and purpose formally. The scope of the ontology is verified in Section 5 with SPARQL queries.

Table 2: Competency questions

Reference	Competency question
CQ1	What is the heating, cooling or ventilation capacity of a system?
CQ2	What is the heating, cooling or ventilation capacity of an HVAC component?
CQ3	What is the size of a given HVAC component?

3.2. Flow System Ontology Extended

3.2.1. Connection between components

FSO represents the energy and mass flow relationships between systems and their components and their composition. However, the current version of FSO does not express the opening or passage that directs the flow of energy or mass. The existing version of FSO expresses a segment. This simplistic representation is insufficient to determine an HVAC component’s size or capacity during the building

design phase. An actual component contains a fluid, which is in motion. This is known as flow. Ports are added for the fluid to flow in and out of each component. The existing FSO taxonomy is therefore extended with **fso:Port** and **fso:Flow**. As a result, a hierarchical relationship can be described among systems, components, ports, and flows.

The concept of relating a **fso:Port** and a **fso:Flow** for multiple components is shown in Figure 3. An **fso:Segment** can be linked to an **fso:Port** with **fso:hasPort**, and an **fso:Port** can be linked to a flow with **fso:hasFlow**. With **fso:hasPort** and **fso:hasFlow** available, an **fso:Fitting** can be related to its ports and flow. The direct relationship between the ports of both components is expressed using **fso:suppliesFluidTo**. In some cases, it is sufficient to just represent the ports and not to explicitly indicate the flow. In that case, the **fso:Flow** instances can simply be left out.

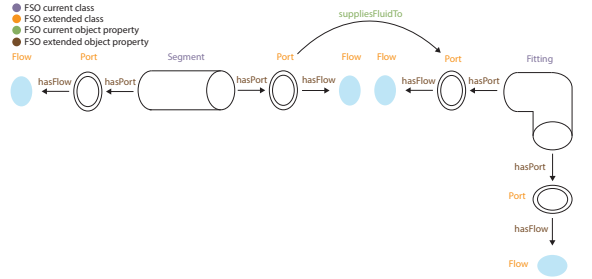


Figure 3: A segment partitioned with ports and flow connects to an fitting through its ports

In addition, the opening can also be expressed as a **fso:ConnectionPoint** instead of a **fso:Port**. A single connection point can be used to represent connections between components instead of multiple ports. The **fso:ConnectionPoint** is an interface between two components that transports fluid. Figure 4 illustrates how multiple components can be related using **fso:ConnectionPoint**. The **fso:Segment** relates to a **fso:ConnectionPoint** with **fso:ConnectsTo**, while the **fso:Fitting** relates to a **fso:ConnectionPoint** with **fso:ConnectsFrom**. A connection point’s relationship to a component also determines the intended direction of the flow, which is crucial information when performing hydraulic calculations. The fluid is transported from the **fso:Segment** to the **fso:Fitting** in Figure 4. Both **fso:Port** and **fso:ConnectionPoint** are subclasses of **bot:Interface**.

A relationship can be described among systems, and components as shown in Figure 5. The components share the same **fso:ConnectionPoint**. Flows and Ports are not available in this example, but could be modelled as well, after the example in Figure 3.

The proposed extension to FSO makes it capable of representing components and interfaces in multiple ways, which adds some flexibility. The definition of the mentioned classes and relationships in this section is defined

⁴<https://github.com/Semantic-Web-Tool/Orchestrator-Service/tree/main/public/Ontologies>

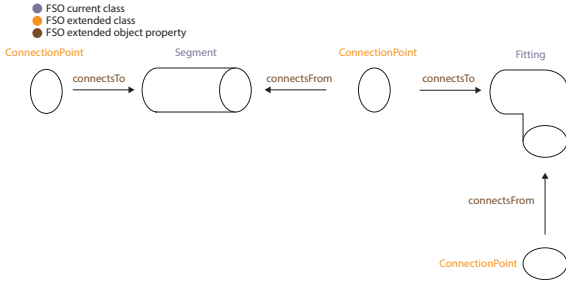


Figure 4: A segment connects to a fitting through connection points.

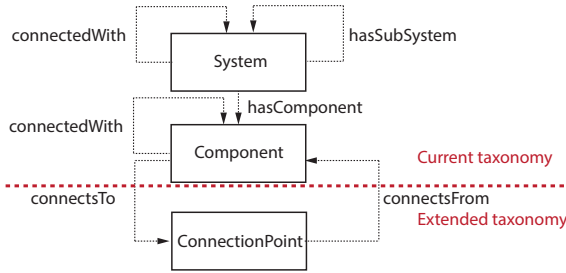


Figure 5: Current and extended taxonomy of FSO with connection points.

as follows:

- **fso:Port** is defined as “An opening or passage that directs flow of a mass or energy”.
- **fso:Flow** is defined as a “A fluid flowing into or out of a port to another port”.
- **fso:ConnectionPoint** is defined as “A point of interaction between components”.
- **fso:hasPort** is defined as “The relation from a component to a port.”
- **fso:hasFlow** is defined as “The relation from a port to a flow.”
- **fso:connectsTo** is defined as “The relation from a connection point to a component.”
- **fso:connectsFrom** is defined as “The relation from a connection point to a component.”

3.2.2. Extended component abstraction level

Currently, FSO represents eight high-level component types. For several reasons, we must subdivide the eight high-level component types into 19 medium-level components. For instance, the hydraulic sizing of a pump or a fan are different. The sizing of a pump includes the pressure drop from both supply system components and return system components, but sizing of a fan only includes pressure

drop of either supply or return side. We have to define the types explicitly when performing hydraulic calculations.

Often components lack the required properties to perform a hydraulic calculation. For example, if an elbow does not have a specified angle, we will not be able to differentiate between an elbow or transition since they both are represented as a **fso:Fitting** and have two ports. To accommodate the difference in properties, the eight high-level FSO components have been nested into 19 medium-level components as shown in Figure 6.

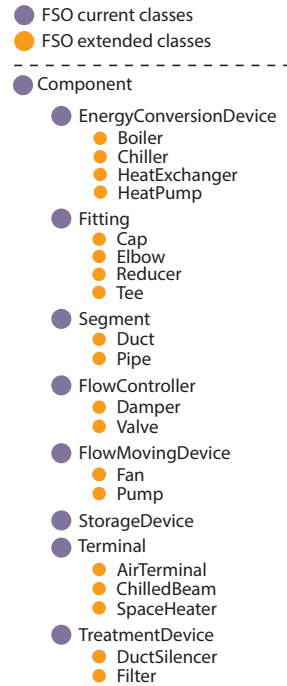


Figure 6: A class hierarchy of current and extended FSO components.

3.3. Property relationships

FPO provides 6 data properties: value, unit, abbreviation, design condition and curve. They can be used to relate an entity literal to an entity class. Combined, the 50 classes, 50 object properties and 6 data properties represent the size and capacity of the FSO components. Figure 7 demonstrates how properties are added to components, ports, or flows. An **fso:Segment** can be related to the property **fpo:Length** with **fpo:hasProperty**. With **fpo:hasValue** and **fpo:hasUnit**, **fpo:Length** can be connected to the value '15' and the unit *meter*. In this example, **fso:Segment** and **fpo:Length** are both classes, while **fpo:hasProperty** is an object property and **fpo:hasUnit** and **fpo:hasValue** are data properties. This method is applied to both **fso:Port** and **fso:Flow**. With this approach, we entirely follow the L2 property modelling ap-

proach that is documented by Bonduel and Pauwels [61] and in principle follows a one-to-many pattern.

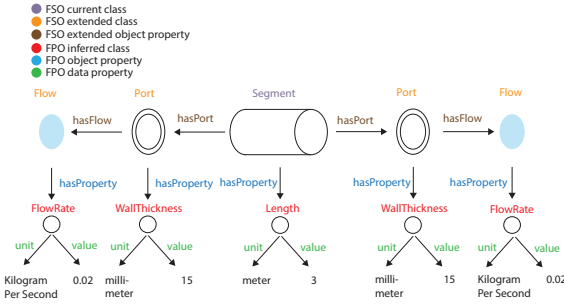


Figure 7: Describing the relationship between an `fso:Segment` and its properties with FPO classes, object and data properties.

3.4. Reasoning

Semantic Web technologies enable deductive reasoning as well as explicit assertions. A few examples of how FPO and the extended FSO allow for reasoning are presented in this section. Every object property in FPO is assigned a domain and a range. For example, the attribute `fpo:hasLength` has the domain `fso:Component` and range `fpo:Length`. This means that, whenever we have a subject of type `fso:Component` and a predicate of type `fpo:hasLength`, then the object must be of type `fpo:Length`. This also means that a reasoning engine will automatically infer the class `fpo:Length` when the object property `fpo:hasLength` is provided in the input instance data. This can similarly be done for all the other properties shown in Figure 7.

An `fso:Segment` is shown in Figure 3 supplying fluid to an `fso:Fitting` with the property `fso:suppliesFluidTo`. However, with the extended FSO, it is possible to infer that if a segment port supplies fluid to another port of a fitting, then the segment must also feed fluid to the fitting (transitive object property). Figure 8 illustrates the inferred knowledge. This can similarly be done for an `fso:connectionPoint` (example shown in Figure 4). If a connection point is connected to a segment and connected from a fitting, it can be inferred that the segment feeds fluid to the fitting.

3.5. Alignments

Figure 9 shows the relation between BOT, FSO and FPO. The figure also illustrates how this network of ontologies can be used to show the relationship between a heating system, its components, properties, and the buildings it serves. It simplifies the relationship between the HVAC components and their properties for illustration purposes.

The taxonomy of components in FPO, FSO, MEP, SAREF4BLDG, and Brick is based mainly on the IFC taxonomy and can therefore be aligned. Of course, they

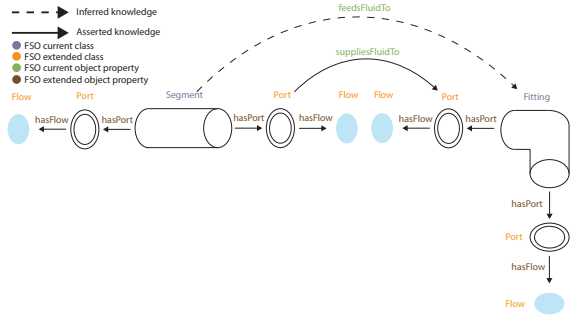


Figure 8: Deducing that the segment feeds fluid to the fitting as a port of the segment supplies fluid to a port of the fitting.

can never be fully aligned because of their difference in semantic meaning and definitions. Mappings between these and other ontologies always remain limited, faulty, and very much open to interpretation and use; by the very nature of mapping ontologies [62]. The mentioned ontologies do not represent all the same components, nor are they conceptually equivalent. Both SAREF4BLDG and Brick represent some component properties but are not intended to describe the capacity or size of each component as FPO does. Even the definition for Zone, which is available in SAREF4BLDG and BOT, for example, has very different meanings in both ontologies and should not be translated or mapped to one another [13, 46].

Classes, object properties, and data properties are nevertheless compared between the ontologies in this section. It is nevertheless recommended to not rely fully on these ontology mappings and instead rely much more on instance linking, as recommended by Schneider [63], Rasmussen [43] and Terkaj [64]. An instance can hereby be annotated as a Brick class, BOT class, and FPO class using the advantage of multi-typing in RDF graphs [14, 65, 66].

For the ontology mapping in the below section, we follow standard approaches and aim to organize FPO classes as either sub-classes or equivalents to classes in another ontology. This notion also applies to object and data properties. It can either be a sub-property or equivalent to another ontology. This is the case when aligning FPO and SAREF4BLDG as shown in Table 3. We are able to align 14 object properties between FPO and SAREF4BLDG. For example, `fpo:hasKv` is a sub-property of `s4blbg:flowCoefficient`, while `fpo:hasVolume` is an equivalent property to `s4blbg:volume`. Moreover, `fpo:hasDesignAirflowRate` is equivalent to `s4blbg:airFlowRateMin`, as their definitions are equivalent.

Just like SAREF4BLDG, Brick components can be equally aligned with FPO components. We can align 2 of the 50 FPO object properties with Brick. For example, `fpo:hasVolume` is equivalent to `brick:volume` as shown in Appendix A. Care needs to be taken, as it is very easy to introduce false assumptions in the data using these mappings.

The shape `fso:Pipe` applies 7 constraints to an `fso:Pipe` and has a complexity level of 1 and are described as follows:

Constraint 1: An `fso:Pipe` must have exactly two flow ports.

Constraint 2: A pipe must feed fluid to exactly one component.

Constraint 3: A pipe must be fed with fluid by exactly one component.

Constraint 4: A pipe must be connected to exactly one system.

Constraint 5: Exactly one property of material type must be present in a pipe.

Constraint 6: Exactly one property of length must be present for a pipe.

Constraint 7: Exactly one property of roughness type must be present for a pipe.

In Listing 1, only the first constraint is expressed in SHACL. The remaining 6 SHACL constraints are made available on GitHub⁶. In the first constraint, the cardinality constraints `sh:minCount` and `sh:maxCount` are applied to check that the `fso:Pipe` has two ports. A minimum and maximum cardinality of 2 will satisfy this constraint. In addition, we use the value type constraint `sh:datatype` with the value `xsd:anyURI` to ensure the triple includes an URI. If the cardinality constraint or value type constraint is not satisfied, the message “A pipe must have exactly two flow ports” will be thrown.

Listing 1: A SHACL shape to constrain the number of `fso:Ports` with `fso:hasPort` for each `fso:Pipe`.

```
1  fsosh:Pipe
2  a sh:NodeShape;
3  sh:nodeKind sh:IRI ;
4  sh:targetClass fso:Pipe ;
5  sh:property[
6    sh:path fso:hasPort ;
7    sh:datatype xsd:anyURI;
8    sh:minCount 2;
9    sh:maxCount 2;
10   sh:message "A pipe must have exactly two flow
11   ports"
12 ]; #... the shape continues
```

4.2. Verifying the demand versus capacity by derived information

HVAC systems and their components must be designed to provide sufficient heating, cooling, and/or ventilation to

buildings. For example, an HVAC terminal is designed correctly if its capacity to heat, cool, and ventilate a space exceeds the space’s demand. With the following constraint, we demonstrate how the capacity of a supply air terminal can be compared with the supply airflow demand of a space:

Constraint 1: The supply air terminal capacity should be higher than the space’s required supply airflow demand.

The rule is expressed in a single SHACL shape, as shown in Listing 2 and the constraint belongs to the shape `fsosh:AirTerminalCapacityCheck`. A SPARQL-based constraint is used to implicitly find the comparison between capacity and demand since it is not explicitly defined. Because this rule requires derived information, it reaches complexity level 2. A nested SPARQL select query is shown in Listing 2. There can be more than one supply air terminal in a space. To sum the capacity of all air terminals grouped by space, we apply an inner select query. In the outer select query, we find the supply airflow demand for each space and filter them according to the constraint. This rule will be violated when the supply air terminal capacity exceeds the supply airflow demand of the space.

Listing 2: The listing shows a SHACL shape to constrain the capacity of an supply air terminal versus the supply airflow demand of an space.

```
fsosh:AirTerminalCapacityCheck
a sh:NodeShape;
sh:nodeKind sh:IRI ;
sh:targetClass bot:Space ;
sh:sparql [
  a sh:SPARQLConstrain ;
  sh:message "The supply air terminal capacity shall
  not be lower the required supply air flow demand of
  the space" ;
  sh:prefixes (fpo: fso: ex: inst: bot:);
  sh:select """PREFIX bot:<https://w3id.org/bot#>
  PREFIX ex: <https://example.com/ex#> PREFIX fso:
  <http://w3id.org/fso#> PREFIX fpo:
  <http://w3id.org/fpo#>
  SELECT ?this {
    ?this ex:designSupplyAirflowDemand ?flowDemand .
    ?flowDemand fpo:hasValue ?flowDemandValue .
    BIND (ROUND(?flowDemandValue) AS ?demand) .
    {
      SELECT ?this (ROUND(SUM(?flowCapValue)) AS
      ?capacity) WHERE {
        ?this a bot:Space .
        ?airTerminal a fso:AirTerminal .
        ?airTerminal fpo:hasAirTerminalType
        ?airTerminalType .
        ?airTerminalType fpo:hasValue "inlet" .
        ?airTerminal fso:feedsFluidTo ?this .
        ?airTerminal fso:hasPort ?port .
        ?port fpo:hasFlowDirection ?flowDirection .
        ?flowDirection fpo:hasValue "Out" .
        ?port fpo:hasFlowRate ?flowCapacity .
        ?flowCapacity fpo:hasValue ?flowCapValue .
      } GROUP BY ?this
    }
  }
```

⁶<https://github.com/Semantic-HVAC-Tool/Rule-Service/tree/main/Public/Shapes/fsosh.ttl>


```

29 BIND (((?capacity/?demand)-1)*10 as ?oversizing) .
30 FILTER (?demand > ?capacity || ?oversizing > 10 )
31 } "" ; ] .

```

```

bind ((?pressureDropValue / ?lengthvalue) AS
↪ ?value) .
FILTER (?value > 100)} "" ; ] .

```

850

4.3. A rule of thumb to verify pressure drop in pipes

The pressure drop in pipes affects the economy of building projects, the material's lifetime and the energy consumption of HVAC systems. A high pressure loss will result in a lower cost price, a shorter lifetime, and higher energy consumption. As a result, most HVAC engineers apply a guideline to their design, e.g. a maximum pipe pressure loss of 100 Pa/m. This guideline or rule cannot be conveyed through explicit information. Calculations and derived information are also required. The complexity level of the shape `fsosh:PipePressureDrop` reaches 3 because an engine is used to calculate the pressure drop and velocity of each distribution component. The engine is discussed in detail in Section 5.1. The only constraint in this rule is targeting an `fso:Pipe` and is described as follows:

Constraint 1: The pressure drop of a `fso:Pipe` shall not exceed 100 Pa/m.

Listing 3 shows the rule expression in SHACL. The pressure drop in pipes is not explicitly defined in Pa/m in FSO or FPO. We can, however, implicitly find the information using a SPARQL constraint. Our SPARQL-based constraint contains a SPARQL select query. The select query returns all instances of `fso:Pipe` that exceeds 100 Pa/m in pressure drop. By dividing the length of the pipe by the pressure drop at the outlet port, we can determine the pressure drop in Pa/m for each `fso:Pipe` instance.

Listing 3: A SHACL shape to constrain the maximum pressure drop of each `fso:Pipe`.

```

1 fsosh:PipePressureDrop
2   a sh:NodeShape;
3   sh:nodeKind sh:IRI ;
4   sh:targetClass fpo:Pipe ;
5   sh:sparql [
6     a sh:SPARQLConstraint ;
7     sh:message "The pressure drop of a fso:Pipe shall
↪ not exceed 100 Pa/m";
8     sh:prefixes (fpo: fso: inst:);
9     sh:select ""PREFIX fso: <http://w3id.org/fso#>
10    PREFIX fpo: <http://w3id.org/fpo#>
11    PREFIX inst: <https://example.com/inst#>
12    SELECT ?this ?value
13    WHERE {
14      ?this a fso:Pipe .
15      ?this fpo:hasLength ?length .
16      ?length fpo:hasValue ?lengthvalue .
17      ?this fso:hasPort ?port .
18      ?port fpo:hasFlowDirection ?flowDirection .
19      ?flowDirection fpo:hasValue "Out" .
20      ?port fpo:hasPressureDrop ?pressureDrop .
21      ?pressureDrop fpo:hasValue ?pressureDropValue .

```

4.4. Redesigning the size of pipes automatically

During the HVAC design process, HVAC components are often oversized or undersized due to limited time. Rather than just creating a rule that notifies whether HVAC components are right-sized passively, we will generate new data actively and add it to the model. By increasing the diameter of the pipe, we can decrease the pressure drop. That is precisely what Listing 4 is doing. Listing 4 is an inference rule expressed in SHACL. Using a SPARQL construct query, the pipe diameter is increased based on the material type and standard manufacturer size. The dimensions are limited to the material type PEX⁷ and range from 0.012 to 0.050 meters. For every `fso:Pipe` that violates the previous rule, `fsosh:PipePressureDrop`, the active rule generates a new diameter. For instance, a pipe diameter of 0.012 meters will automatically be increased to 0.015 meters and added to the data model. Since this rule can generate new information, it reaches a complexity level of 4.

Listing 4: A SHACL shape to increase the size of a `fso:Pipe` automatically

```

fsosh:PipePexSizing
  a sh:NodeShape ;
  sh:targetClass fso:Pipe ;
  sh:rule [
    a sh:SPARQLRule ;
    sh:prefixes (fpo: fso: ex: );
    sh:construct ""
    CONSTRUCT {?diameter fpo:hasValue ?newSize.}
    WHERE {
      ?this a fso:Pipe .
      ?this fpo:hasMaterialType ?type .
      ?type fpo:hasValue "PEX 6 bar varme" .
      ?this fso:hasPort ?port .
      ?port fpo:hasOuterDiameter ?diameter .
      ?diameter fpo:hasValue ?diameterValue .
      BIND (
        IF(?diameterValue = 0.012, 0.015,
          IF(?diameterValue = 0.015, 0.018,
            IF(?diameterValue = 0.018, 0.020,
              IF(?diameterValue = 0.020, 0.022,
                IF(?diameterValue = 0.022, 0.028,
                  IF(?diameterValue = 0.028, 0.032,
                    IF(?diameterValue = 0.032, 0.040,
                      IF(?diameterValue = 0.040, 0.050,
                        ?diameterValue)))))))))
        AS ?newSize)} "" ;
      condition: fsosh: PipePressureDrop
    ] .

```

⁷<https://www.bobvila.com/articles/pex-pipe>

5. Demonstration Environment

This section aims to demonstrate how capacity and size-related properties within the HVAC domain can be represented and validated for a real-world BIM model. The use case process is illustrated in Figure 10.

The first step of the process is to create a data graph and shape graph. As the shape graph is already produced in Section 4, it does not require further processing and can be used as-is⁸. In contrast, converting a BIM model will create the data graph. This step is identical to the building model preparation phase of Eastman et al. [50]. The data graph contains BOT, FSO, and FPO vocabularies so that it matches with the rules in our shape graph and can proceed to the rule execution phase of Eastman et al. [50]. Using these three vocabularies, we can describe the building, its services, its interactions, and properties. For example, we can express how the HVAC system or an HVAC component relates to the building or a specific room.

In the second step, a rule execution process will be performed to check the shape graph against the data graph. The data graph will be manually corrected if any constraints are violated during rule execution. Depending on the violation type, manual correction can be achieved at three levels; BIM model, parser or data graph. In cases where we do not want to modify the BIM model, we can use SPARQL on the data graph or add the information through the parser.

When the rule execution conforms, we can proceed to step 3. This step involves hydraulic calculations for ducts, pipes, and fittings to determine each distribution component's pressure drop and fluid velocity. A second conformance check will be conducted to check the shape graph against the data graph and the hydraulic results. Whenever a constraint is violated, an HVAC rule at level 4 in complexity from the shape graph will be used to correct the violation.

When the rule execution conforms, we will have all the information necessary to size the flow-moving device. Step 4 will therefore involve calculating the capacity of each flow-moving device, represented in the data graph. After the flow-moving devices' capacities has been calculated, the result is given, and the process ends.

5.1. A Semantic HVAC tool

We developed the Semantic HVAC tool to perform the process shown in Figure 10. The web tool has a microservice-oriented system architecture and contains four layers, which is illustrated in Figure 11. The source code of the Semantic HVAC Tool and the material used to perform the process shown in Figure 10 is made available on GitHub⁹. The following sections first describe the data

flow in detail and then demonstrate the Semantic HVAC tool in a use case.

5.1.1. Presentation layer

The presentation layer handles the user interface logic and displays data on the page. The Graphical User Interface (GUI) relies on React components to improve page rendering [67]. Using the GUI, users can perform conformance checking, perform hydraulic calculations, calculate the capacity of flow-moving devices, and view the results. The user has to initiate the conformance checking and calculations in the right order as shown in Figure 10. It is therefore necessary for the user to initiate the conformance check first. The user must correct all violations manually if any exist. If any violation exists, the GUI will not allow the user to perform the hydraulic calculation. Using this method, we ensure that the data model contains all the information we need to calculate the hydraulics. The same applies to the capacity calculation of flow-moving devices. If any violations occur after the second conformance check, the GUI will not allow the user to initiate the flow-moving device calculation.

The GUI displays the conformance check results in two different tables. Based on the type of HVAC component, the HVAC system, and size and capacity properties, the first table shows the number of violations. The first table is interactive. By clicking on a specific HVAC component type in the first table, the GUI will display the second table. The second table lists the violations for that specific HVAC component in more detail, including the instance ID, constraint type, and violation description. Additionally, the GUI shows the results of the flow-moving device calculation in a table. The table displays the type, ID, flow rate, and pressure of each flow-moving device.

5.1.2. Communication layer

The orchestrator handles the communication between the service components in the Semantic HVAC Tool via HTTP requests.

There are two ways to communicate between services: decentralized and centralized. Decentralized communication allows microservice components to communicate directly with each other. In central communication, microservices will communicate through an orchestrator service. As illustrated in Figure 11, we have implemented a central orchestrator to handle the communication between the presentation layer, the business layer, and the database layer. The orchestrator is developed as an ExpressJS server [68] in NodeJS [69]. When the user initiates the conformance checking, the following communication will happen:

1. the client requests conformance checking results from the orchestrator.
2. the orchestrator requests conformance checking results from the rule service.

⁸<https://github.com/Semantic-HVAC-Tool/Rule-Service/tree/main/Public/Shapes/fsosh.ttl>

⁹<https://github.com/Semantic-HVAC-Tool>

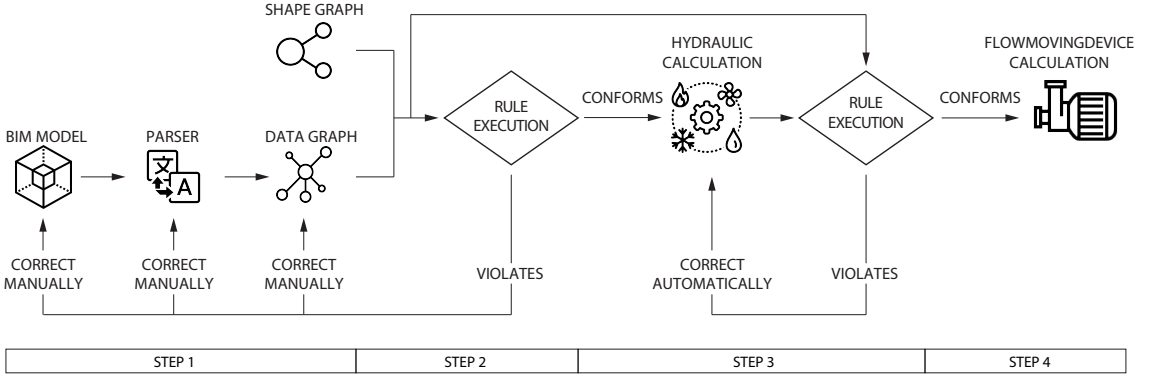


Figure 10: The process of performing conformance checking and design calculations for an HVAC model.

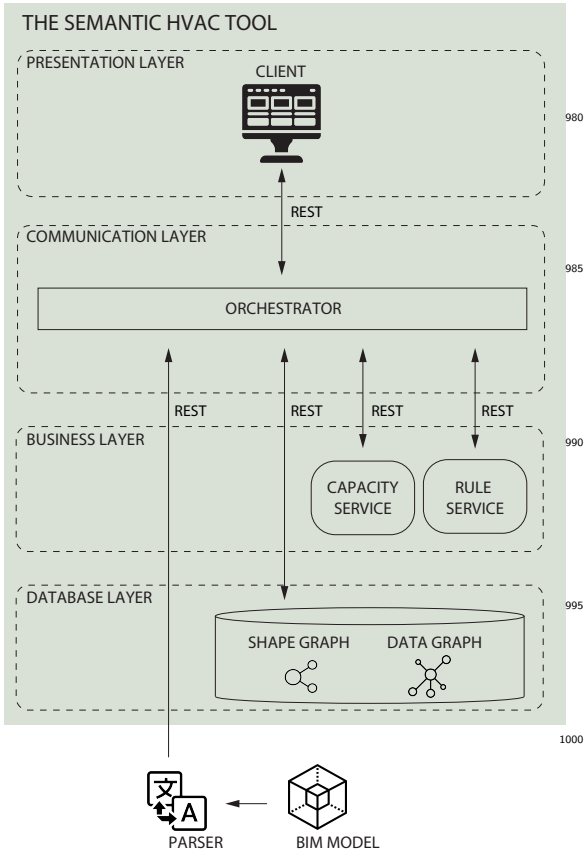


Figure 11: The system architecture of the Semantic HVAC Tool.

- the rule service sends a rule model expressed in turtle format to the orchestrator.
- the orchestrator sends the rule model to the database.
- as the database already stores the data graph, it performs the rule execution and sends the conformance checking results expressed in JSON-LD to the orchestrator.
- the orchestrator sends the conformance checking results to the client.
- the client displays the conformance checking results in two tables.

Similar to the conformance checking, the orchestrator handles communication between the different services when performing hydraulic- and flow-moving device calculations.

5.1.3. Business layer

The business logic is spread over multiple microservices in the web application. We have divided our logic into two microservices: the capacity service and the rule service, as shown in Figure 11. Rule logic is handled by the rule service, while the capacity service handles HVAC design logic. The rule service consists of two functions. When requested, the first function provides a shape graph in turtle format, while the second function performs an automatic conformance check and produces a validation report in JSON-LD format.

The capacity service has one function. When requested, it performs a hydraulic calculation and delivers the pressure drop result for each distribution component, which is of type `fso:Pipe`, `fso:Duct`, `fso:Elbow`, `fso:Transition` and `fso:Tee`. The output of the function is expressed in

JSON-LD format. Both microservices are developed separately in FastAPI. To perform hydraulic calculations, we use the fluids library [70].

5.1.4. Database layer

The database layer consists of a Jena Fuseki server [71]¹⁰ that stores RDF data. The microservices in the business layer share the same database to access information from different domains easily. Jena Fuseki has SPARQL, SHACL, and Update endpoints. The SPARQL endpoint retrieves data, while the Update endpoint inserts, deletes¹⁰⁶⁵ or updates data.

For example, when the user initiates the flow-moving device calculation, the client requests a list of flow-moving devices from the orchestrator. The orchestrator then requests three SPARQL queries¹⁰. The first SPARQL query¹⁰⁷⁰ is illustrated in Appendix B and is able to sum the pressure drops of the critical branch to determine the necessary pressure of each `fso:Pump` represented in the data graph. The second SPARQL query performs the same calculation for every `fso:Fan`, while the third query calculates the total flow rate of each flow-moving device. Once the orchestrator hits the SPARQL endpoint in the Jena Fuseki Server with the SPARQL queries, it retrieves the results and sends them to the client to be displayed in the flow-moving device table.

5.1.5. Parsing the BIM model

The parser¹¹ and the BIM model¹² are not part of the Semantic HVAC Tool. The parser is developed as a .NET Framework (C-Sharp) plugin in Revit [72], using the Revit API, while the BIM model is developed as a BIM model in Revit. The parser has two functions; the first function serializes Revit BIM objects into a data graph expressed in turtle syntax and, while the second sends the data graph to the orchestrator via an HTTP request. The orchestrator then redirects the data graph to the database for storage.

5.2. Results

To showcase the tool in use, we used a BIM model of a real-world building located in Sorø, Denmark. The building is a primary school constructed in 2017 and named Frederiksberg Skole. Frederiksberg Skole has a gross floor area of 6970 m² and is divided into a northern building¹⁰⁸⁰ and southern building. Each building has three floor levels, as shown in Figure 12. The original BIM model has been modified by Seeberg and Tangeraas [73] to include only the northern building and its heating and ventilation system. It has 86 rooms, each heated with radiators¹⁰⁸⁵ and ventilated with supply and extract air terminals. Both systems are located in the basement of the northern

building. The results of parsing Frederiksberg Skole as a data model, performing two conformance checks, calculating the hydraulics and designing flow-moving devices with the Semantic HVAC tool are presented in this section.

5.2.1. Parsing the data model

The process of serializing Frederiksberg Skole from Revit to the Semantic HVAC Tool took 17.1 seconds to complete. Moreover it took the Semantic HVAC Tool 8.3 seconds to store the data model of 369054 triples in the database. The triples are also made available on GitHub¹³. Since FSO represent HVAC components, we can extract the sum of components by type. Table 4 shows that the data model consists of 6137 HVAC components, 36 HVAC systems and 65851 HVAC size- and capacity-related properties. In total, the data model consists of 84887 instances.

Table 4: The table shows the amount of HVAC components, systems and size- and capacity-related properties in the data model

Type	Amount
fso:EnergyConversionDevice	1
fso:Segment	2766
fso:Fitting	2912
fso:FlowMovingDevice	3
fso:FlowController	85
fso:Terminal	370
fso:System	36
fso:Port	12827
fso:Flow	36
fpo:Property	65851
total	84887

5.2.2. Conformance checking Frederiksberg Skole

The process of validating the data model against the rule model took 3.1 seconds to complete. Table 5 shows the results of the first conformance check. For example, Table 5 shows that instances of type `fso:System` in the data model have violated the constraints 32 times. The HVAC rule model is also violated by instances of type `fso:Duct`, `fso:SpaceHeater`, `fso:Port`, and `fpo:Property`. The total amount of violations is 372. Since the data graph contains 84887 instances this means that approximately 0.5% of the components are violating the HVAC rule model. We can also observe, that the majority of violations are caused by instances of type `fso:Port`, which accounts for approx. 73% of the total violating instances.

We can access Table 6 in the client by clicking on `fso:System` in the first conformance checking table. Table 6 lists the violation details for `fso:System`. The GUI displays all 32 violations, but Table 6 is limited to the first two violations, indicating that instance `inst:5eb8aa6a...`

¹⁰<https://github.com/Semantic-HVAC-Tool/Orchestrator-Service/tree/main/public/Queries>

¹¹<https://github.com/Semantic-HVAC-Tool/Parser>

¹²<https://github.com/Semantic-HVAC-Tool/Other/blob/main/BIM-Model.rvt>

¹³<https://github.com/Semantic-HVAC-Tool/Other/blob/main/Data-Model.ttl>

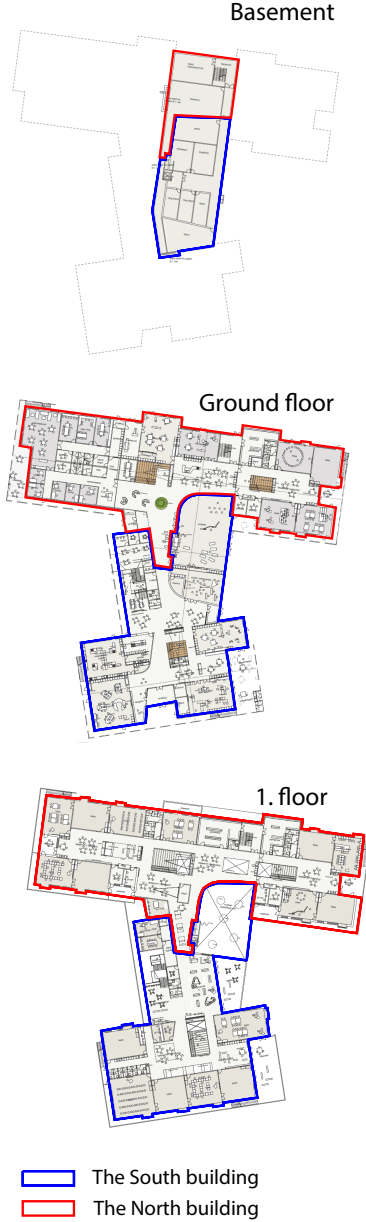


Figure 12: The illustration shows the floor plans of Frederiksberg Skole in Sorø, Denmark. The south building is marked with red, while the north building is marked with blue [73]

Table 5: Results of the first conformance check, showing the number of violations, based on HVAC component type, HVAC system and size- and capacity-related properties

Type	Amount
fso:HeatExchanger	0
fso:Pipe	2
fso:Duct	2
fso:Elbow	0
fso:Transition	0
fso:Tee	0
fso:Fan	0
fso:Pump	0
fso:AirTerminal	0
fso:SpaceHeater	3
fso:Damper	0
fso:Valve	0
fso:System	32
fso:Port	251
fso:Flow	0
fpo:Property	82
Total	372

violates the SHACL constraint type `sh:MinCountConstraintComponent` and throws the message “A return system must contain at least one component”.

Table 6: Results of the first conformance check, showing the first two results of `fso:System` violations in details

ID	Constraint type	Description
inst:5eb8aa6a-0ed0-4fea-b226-dd7fa9ae035e-0019ec8a	sh:MinCountConstraintComponent	A return system must have at least one component
inst:98e9914f-25c6-4c43-a0fb-912eba89c13d-0019dbff	sh:MinCountConstraintComponent	A supply system must have at least one component

All 32 violations were corrected in the data graph by performing the SPARQL update query shown in Appendix C directly in the Jena Fuseki Server. The query deletes all `fso:SupplySystem` and `fso:ReturnSystem` instances that lack the predicate `fso:hasComponent`.

The remaining violations were corrected manually in the BIM model, parser, and data graph, which results in an empty validation table. A blank validation table at this stage indicates that the data graph conforms, and we have completed step 2 of the process illustrated in Figure 10.

5.2.3. Hydraulic calculation and second conformance check

Performing the hydraulic calculation on Frederiksberg Skole took 5.4 seconds. The violation results of the second conformance check are shown in Table 7. It can be seen that instances of `fso:Pipe` are violating the HVAC rule

model 14 times, and the total number of violations in step 3 of the process illustrated in Figure 10 is 14.

Table 7: Results of performing the second conformance check, showing the amount of violations, when the hydraulic results are added to the data graph

Type	Amount
fso:HeatExchanger	0
fso:Pipe	14
fso:Duct	0
fso:Elbow	0
fso:Transition	0
fso:Tee	0
fso:Fan	0
fso:Pump	0
fso:AirTerminal	0
fso:SpaceHeater	0
fso:Damper	0
fso:Valve	0
fso:System	0
fso:Port	0
fso:Flow	0
fpo:Property	0
Total	14

Clicking on **fso:Pipe** in the first conformance checking table in the client will display Table 8. The table displays the violation details within the category of **fso:Pipe**. While the GUI of the Semantic HVAC Tool displays the violation details of all 14 violations, Table 8 is limited to the first two violations. The first result indicates that the instance **inst:745522df...** is violating the SHACL constraint type **sh:SPARQLConstraintComponent**. The message it throws indicates that the pressure drop of the **fpo:Pipe** instance is exceeding 100 Pa/m.

Table 8: Results of the second conformance check, displaying the first two results of fso:Pipe violations in detail after running the hydraulic calculation

ID	Constraint type	Description
inst:745522df-9a78-4732-8b22-f56765e86201-002bec43	sh:SPARQL-Constraint-Component	The pressure drop of a pipe should not exceed 100 Pa/m
inst:745522df-9a78-4732-8b22-f56765e86201-002bec25	sh:SPARQL-Constraint-Component	The pressure drop of a pipe should not exceed 100 Pa/m

In the GUI the user can implement the correction of all 14 violations automatically. If the corrections are implemented, the violations will be removed from Table 7, and the total number of violations will be decreased to 0.

The violations at this stage were corrected automatically in this way, which resulted in an empty validation table. A blank validation table at this stage indicates that the data graph conforms, and we have completed step 3 of the process illustrated in Figure 10.

5.2.4. Flow-moving device capacity calculation and second validation

Since we have performed the rule execution and hydraulic calculation, we are now ready to calculate the capacity of each flow-moving device represented in the data graph. The results of the flow-moving device calculation are shown in Table 9. It took 87 seconds to calculate the total amount of flow rate and pressure for each flow-moving device using three SPARQL queries and to display the results in the flow-moving device table. Two fans and one pump are shown in Table 9 as flow-moving devices. Table 9 provides the component ID, flow rate, and pressure for each **fso:Fan** and **fso:Pump**. For example, it shows that the instance **inst:0fc738e3...** of type **fso:Pump** has a total flow rate of 0.84 L/s and a total pressure of 16867 pascal. The fan pressure includes the ductwork, air terminal, and AHU pressure drop. Using this information, correctly sized fans and pumps can be selected from manufacturers product catalogues.

Table 9: Flow-moving device results showing the type of each flow-moving device, its component ID, flow rate and pressure.

Type	Component ID	Flow rate [L/s]	Pressure [Pa]
fso:Fan	inst:36aec977-8efa-403c-b1e6-3b29521aac43-002f6bf5	7943	724
fso:Fan	inst:f4ad7dcb-2875-4fe5-be51-f41510b75979-002f583e	8124	822
fso:Pump	inst:0fc738e3-3eb1-4344-b913-b3883e4083b0-0033212a	0.84	16867

6. Discussion

This section describes the achievements, limitations, and future work.

6.1. Achievements

This paper shows how an ontology can be extended, constructed and aligned from scratch to represent the capacity and size-related properties of HVAC systems and their components. We also demonstrated how separate and lightweight ontologies such as BOT, FSO and FPO can be interconnected to represent the building, its services and their relationships in a modular way. Moreover, we

developed a set of constraints to increase the data quality of BIM models within the HVAC domain. We developed the Semantic HVAC tool and applied it to a real-world building to demonstrate the feasibility of expressing and conforming an HVAC model. We have created a reliable data model to perform hydraulic calculations and designing the capacity of flow-moving devices. Considering the time spent on conformance checking, (re-)sizing and quality control in the industry, this study implements technical solutions and demonstrates a path towards better data quality in BIM models, time savings due to computerization and increased transparency.

6.2. Limitations

6.2.1. Logical complexity

Schwabe et al. [33], Oraskari et al. [58], and Hagedorn and König [56] applied a reasoner to perform an automatic rule check. In the same way, we used a SHACL inference rule to automatically increase the diameter of a pipe when the pressure in the pipe exceeded 100 Pa/m. Although the SHACL component could generate the new data, it could not delete the old data. SHACL inferencing rules can only infer new knowledge. We implemented a separate SPARQL query in the Semantic HVAC Tool to delete the existing data after the SHACL inferencing rule was performed. In any web tool, spreading logic this way will increase its logical complexity.

6.2.2. Query efficiency

The rule execution is performing well since it took only 3.1 seconds to validate The HVAC rule model consisting of 36 shapes and 122 constraints against Frederiksberg Skole with 369054 triples. In contrast, it took 87 seconds to calculate the total pressure and flow rate of each flow-moving device, represented in the data graph using three SPARQL queries. Two of the SPARQL queries have a Filter Not Exists statement, which is responsible for the slow query performance. Using the Filter Not Exists statement, we iterate through all HVAC components in the graph and return only those with ports that belong to the same HVAC system. Iterating through all HVAC components and their ports slows down the query efficiency. This could be improved by replacing the Filter Not Exists statement.

6.2.3. Abstraction level of HVAC components

FSO is limited to eight high-level HVAC components and 19 medium-level HVAC components. In practice, it is possible to subdivide FSO further. For example, a pump can be subdivided into centrifugal pumps, positive displacement pumps, etc. There are also several levels of centrifugal pumps. To retain FSO as a lightweight ontology we did not nest further.

6.2.4. Geometry-based constraints

The data graph and shape graph we developed in our research do not represent HVAC component geometry and

its geometry-related properties nor validate geometry-based constraints, such as separation distances between HVAC components and components from other domains or service distances, such as structural components. The delivery of BIM models with incorrect separation and service distances between HVAC components from the design phase to the construction phase is a common problem affecting a building project's economy and schedule and should therefore be a focal point in further development.

6.3. Future work

The proposal for future work in this paper can be divided into three steps.

A literature review of geometry-related ontologies should be conducted first. If a sufficient geometry-related ontology doesn't exist, an existing one should be extended, or a new one should be developed to describe the geometry and the relation between geometries.

Secondly, to represent separation and service distances for HVAC components, the geometry-related ontology should be interconnected with BOT, FSO, and FPO.

Lastly, a set of geometry-based constraints should be added to the HVAC rule model and validated against the data graph.

7. Conclusions

This paper presents a demonstration environment to represent and validate the composition of HVAC components, their systems, and their capacity and size-related properties using semantic web technologies. This paper aimed to:

1. Extend FSO to support an alignment with the proposed FPO ontology.
2. Propose the FPO ontology to represent HVAC components' capacity and size-related properties.
3. Propose a rule model for the HVAC domain.
4. Produce a demonstration environment to show the conformance of an HVAC model.
5. Use the demonstration environment to show how FPO and the HVAC rule model can support the description and validation of hydraulics in HVAC components and the capacity of HVAC components.

We extended FSO with three classes and four properties related to the connectivity between ports and fluids. This made it possible to describe the relationship between HVAC components, their flow ports and the fluid being transported in three ways and aligned with FPO. We also extended FSO to represent 19-medium level component types. We developed FPO to represent the size- and capacity-related properties of HVAC components. FPO

has a Description Logic expressivity of $\mathcal{ALRF}(\mathcal{D})$ and contains 50 classes, 50 object properties and 6 data properties¹²²⁰

Moreover, we developed an HVAC rule model that restricts the composition of HVAC components, their systems, and their size- and capacity-related properties. The rule model consists of 36 shapes and 122 constraints.¹³²⁵

A four-step process and the Semantic HVAC Tool were developed to demonstrate how a real-world building model can be represented, validated, and used to compute hydraulic calculations and design the capacity of a flow-moving device. Frederiksberg Skole consists of 369054 triples and was used as the real-world building model. We managed to perform conformance checking twice. The first rule execution resulted in 372 constraint violations, and the second¹³³⁰ resulted in 14 constraint violations. These rule violations were fixed both manually and automatically. Finally, using the conformed model, we performed hydraulic calculations and used the results to design the capacity of two fans and a pump, which were represented in the real-world building model.¹³³⁵

8. Acknowledgements

This work was supported by EU-Interreg ÖKS “Data-driven Energy Management in Public Buildings”; the Innovation Fund Denmark (grant 9065-00266A); the Ram-boll Foundation; and COWI A/S. We thank Sorø municipality for providing the BIM model for Frederiksberg Skole.¹³⁵⁵

References

- [1] M. Niknam, S. Karshenas, A shared ontology approach to semantic representation of BIM data, *Automation in Construction* (2017). doi:10.1016/j.autcon.2017.03.013.¹³⁶⁰
- [2] R. Sacks, C. Eastman, G. Lee, P. Teicholz, *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers*, 2018.
- [3] J. M. Werbrouck, M. Senthilvel, J. Beetz, *Querying Heterogeneous Linked Building Data with Context-expanded GraphQL Queries*, Tech. rep. URL <https://www.w3.org/TR/sparql11-query/>¹³⁶⁵
- [4] A.-H. Hamdan, M. Bonduel, R. J. Scherer, An ontological model for the representation of damage to constructions, Tech. rep. URL <http://www.w3.org/1999/02/22-rdf-syntax-ns#>¹³⁷⁰
- [5] I. Esnaola-Gonzalez, F. J. Diez, Integrating Building and IoT data in Demand Response solutions, Tech. rep. URL <http://project-respond.eu>
- [6] M. H. Rasmussen, M. Lefrançois, P. Pauwels, C. A. Hviid, J. Karlshøj, Managing interrelated project information in AEC Knowledge Graphs, *Automation in Construction* (2019). doi: 10.1016/j.autcon.2019.102956.¹³⁷⁵
- [7] A. Hogan, *The Web of Data*, 2020. doi:10.1007/978-3-030-51580-5.¹³⁸⁰
- [8] J. Flore, T. Djuedja, Integration of environmental data in BIM tool & Linked Building Data, Tech. rep. URL <http://www.enit.fr>¹³⁸⁵
- [9] S. Stolk, K. McGlinn, Validation of ifcowl datasets using shacl, Vol. 2636, 2020.
- [10] P. Pauwels, D. V. Deursen, R. Verstraeten, J. D. Roo, R. D. Meyer, R. V. D. Walle, J. V. Campenhout, A semantic rule checking environment for building performance checking, *Automation in Construction* 20 (2011). doi:10.1016/j.autcon.2010.11.017.¹³⁹⁰
- [11] J. Lee, Y. Jeong, User-centric knowledge representations based on ontology for AEC design collaboration, *Computer-Aided Design* 44 (2012) 735–748. doi:10.1016/j.cad.2012.03.011. URL www.elsevier.com/locate/cad
- [12] J. F. Tchouanguem Djuedja, F. H. Abanda, B. Kamsu-Foguem, P. Pauwels, C. Magniont, M. H. Karay, An integrated Linked Building Data system: AEC industry case, *Advances in Engineering Software* 152 (feb 2021). doi:10.1016/j.advengsoft.2020.102930.
- [13] M. H. Rasmussen, M. Lefrançois, G. F. Schneider, P. Pauwels, Bot: The building topology ontology of the w3c linked building data group, *Semantic Web* 12 (1) (2020) 143–161. doi:10.3233/SW-200385.
- [14] V. Kukkonen, A. Kücükavci, M. Seidenschmur, M. H. Rasmussen, K. M. Smith, C. A. Hviid, An ontology to support flow system descriptions from design to operation of buildings, *Automation in Construction* 134 (December 2021) (2022) 104067. doi:10.1016/j.autcon.2021.104067. URL <https://doi.org/10.1016/j.autcon.2021.104067>
- [15] N. Pauen, D. Schlütter, J. Siwiecki, J. Frisch, C. van Treeck, Integrated representation of building service systems: topology extraction and tubes ontology, *Bauphysik* 42 (6) (2020) 299–305. doi:10.1002/bapi.202000027.
- [16] M. Bonduel, Towards a props ontology (2018), URL: <https://github.com/w3c-lbdcg/lbd/blob/gh-pages/presentations/props/presentation/LBDCall20180312>.
- [17] A. Wagner, W. Sprenger, C. Maurer, T. E. Kuhn, U. Rüppel, Building product ontology: Core ontology for linked building product data, *Automation in Construction* 133 (2022) 103927. doi:10.1016/j.autcon.2021.103927.
- [18] N. Pauen, D. Schlütter, J. Siwiecki, J. Frisch, C. van Treeck, Integrated representation of building service systems: topology extraction and TUBES ontology, *Bauphysik* 42 (6) (2020) 299–305. doi:10.1002/bapi.202000027.
- [19] E. van den Bersselaar, J. Heinen, M. Chaudron, P. Pauwels, Automatic validation of technical requirements for a bim model using semantic web technologies, 2022, 1st 4TU/14USA research day on Digitalization in the Built Environment ; Conference date: 01-04-2022.
- [20] A. S. Ismail, K. N. Ali, N. A. Iahad, A review on bim-based automated code compliance checking system, in: 2017 International Conference on Research and Innovation in Information Systems (ICRIIS), 2017, pp. 1–6. doi:10.1109/ICRIIS.2017.8002486.
- [21] R. Ren, J. Zhang, Model information checking to support interoperable bim usage in structural analysis, *ASCE International Conference on Computing in Civil Engineering* 2019doi: 10.1061/9780784482421.046. URL <https://par.nsf.gov/biblio/10104661>
- [22] A. T. Kovacs, A. Micsik, Bim quality control based on requirement linked data, *International Journal of Architectural Computing* 19 (3) (2021) 431–448. doi:10.1177/14780771211012175.
- [23] W. Solihin, N. Shaikh, X. Rong, L. K. Poh, Beyond interoperability of building model: a case for code compliance, *Carnegie Mellon University (CMU)*, 2004. URL <https://www.researchgate.net/publication/280598933BEYOND>
- [24] E. Hjelseth, N. Nisbet, Capturing normative constraints by use of the semantic mark-up rase methodology, *Proceedings of CIB* (2011).
- [25] T. H. Beach, T. Kasim, H. Li, N. Nisbet, Y. Rezgui, Towards automated compliance checking in the construction industry, Vol. 8055 LNCS, 2013. doi:10.1007/978-3-642-40285-2_32.
- [26] J. K. Lee, C. M. Eastman, Y. C. Lee, Implementation of a bim domain-specific language for the building environment rule and analysis, *Journal of Intelligent and Robotic Systems: Theory and Applications* 79 (2015). doi:10.1007/s10846-014-0117-7.
- [27] J. Dimyadi, W. Solihin, W. Solihin, C. Eastman, A knowledge representation approach in bim rule requirement analysis using the conceptual graph, *Journal of Information Technology in*

- Construction 21 (2016).
- [28] J. Dimyadi, P. Pauwels, R. Amor, Modelling and accessing regulatory knowledge for computer-assisted compliance audit, *Journal of Information Technology in Construction* 21 (2016).
- [29] J. Dimyadi, C. Clifton, M. Spearpoint, R. Amor, Computerizing regulatory knowledge for building engineering design, *Journal of Computing in Civil Engineering* 30 (2016). doi:10.1061/(asce)cp.1943-5487.0000572.
- [30] T. Chipman, T. Liebich, M. Weise, mvdxml specification 1.1, specification of a standardized format to define and exchange model view definitions with exchange requirements and validation rules. by model support group (msg) of buildingsmart, BuildingSMART 1 (2016).
- [31] S. Park, Y. C. Lee, J. K. Lee, Definition of a domain-specific language for korean building act sentences as an explicit computable form, Vol. 21, 2016.
- [32] G. Governatori, M. Hashmi, H. P. Lam, S. Villata, M. Palmirani, Semantic business process regulatory compliance checking using legalruleml, Vol. 10024 LNAI, 2016. doi:10.1007/978-3-319-49004-5_48.
- [33] K. Schwabe, J. Teizer, M. König, Applying rule-based model-checking to construction site layout planning tasks, *Automation in Construction* 97 (2019). doi:10.1016/j.autcon.2018.10.012.
- [34] G. Lee, J. Jeong, J. Won, C. Cho, S. joon You, S. Ham, H. Kang, Query performance of the ifc model server using an object-relational database approach and a traditional relational database approach, *Journal of Computing in Civil Engineering* 28 (2014). doi:10.1061/(asce)cp.1943-5487.0000256.
- [35] W. Solihin, J. Dimyadi, Y.-C. Lee, C. Eastman, R. Amor, The critical role of accessible data for bim-based automated rule checking systems, 2017. doi:10.24928/jc3-2017/0161.
- [36] R. K. Soman, M. Molina-Solana, J. K. Whyte, Linked-data based constraint-checking (ldcc) to support look-ahead planning in construction, *Automation in Construction* 120 (2020) 103369. doi:10.1016/j.autcon.2020.103369.
- [37] J. Oraskari, M. Senthilvel, J. Beetz, M. Senthilvel, J. Beetz, SHACL is for LBD what mvdxML is for IFC, *Proceedings of the 38th International Conference of CIB W78 (October) (2021)* 693–702.
- URL <https://www.cibw78-ldac-2021.lu/>
- [38] J. Beetz, J. Van Leeuwen, B. De Vries, IfcOWL: A case of transforming EXPRESS schemas into ontologies, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM 23 (1) (2009)*. doi:10.1017/S0890060409000122.
- [39] W. Terkaj, A. Šojić, Ontology-based representation of IFC EXPRESS rules: An enhancement of the ifcOWL ontology, *Automation in Construction* 57 (2015). doi:10.1016/j.autcon.2015.04.010.
- [40] D. A. Koonce, R. P. Judd, A visual modelling languages for express schema, *International Journal of Computer Integrated Manufacturing* 14 (5) (2001) 457–472. doi:10.1080/09511920010022495.
- [41] K. Afari, C. M. Eastman, D. Castro-Lacouture, Javascript object notation (json) data serialization for ifc schema in web-based bim data exchange, *Automation in Construction* 77 (2017) 24–51. doi:https://doi.org/10.1016/j.autcon.2017.01.011.
- URL <https://www.sciencedirect.com/science/article/pii/S0926580517300316>
- [42] P. Pauwels, S. Zhang, Y.-C. Lee, Semantic web technologies in aec industry: A literature overview, *Automation in Construction* 73 (2017) 145–165. doi:https://doi.org/10.1016/j.autcon.2016.10.003.
- URL <https://www.sciencedirect.com/science/article/pii/S0926580516302928>
- [43] M. H. Rasmussen, P. Pauwels, C. A. Hviid, J. Karlshøj, Proposing a Central AEC Ontology That Allows for Domain Specific Extensions, 2017. doi:10.24928/jc3-2017/0153.
- [44] M. Lefrançois, J. Kalaoja, T. Ghariani, A. Zimmermann, T. Seas, D2 . 2 SEAS Knowledge Model, *Tech. Rep. December* (2014).
- [45] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Bergés, D. Culler, R. K. Gupta, M. B. Kjærgaard, M. Srivastava, K. Whitehouse, Brick: Metadata schema for portable smart building applications, *Applied Energy* 226 (September 2017) (2018) 1273–1292. doi:10.1016/j.apenergy.2018.02.091.
- URL <https://doi.org/10.1016/j.apenergy.2018.02.091>
- [46] L. Daniele, F. den Hartog, J. Roes, Created in Close Interaction with the Industry: The Smart Appliances REference (SAREF) Ontology, in: *Lecture Notes in Business Information Processing*, Vol. 225, 2015. doi:10.1007/978-3-319-21545-7_9.
- [47] P. Pauwels, A. Costin, M. H. Rasmussen, Knowledge graphs and linked data for the built environment, *Structural Integrity* 20 (2022) 157–183. doi:10.1007/978-3-030-82430-3_7.
- [48] J. P. Martins, A. Monteiro, Lica: A bim based automated code-checking application for water distribution systems, *Automation in Construction* 29 (2013). doi:10.1016/j.autcon.2012.08.008.
- [49] W. Solihin, C. Eastman, Classification of rules for automated bim rule checking development, *Automation in Construction* 53 (2015). doi:10.1016/j.autcon.2015.03.003.
- [50] C. Eastman, J. min Lee, Y. suk Jeong, J. kook Lee, Automatic rule-based checking of building designs (2009). doi:10.1016/j.autcon.2009.07.002.
- [51] W3C, Swrl: A semantic web rule language combining owl and ruleml (5 2004).
- URL <https://www.w3.org/Submission/SWRL/>
- [52] S. Mehla, S. Jain, Rule languages for the semantic web, Vol. 755, 2019. doi:10.1007/978-981-13-1951-8_73.
- [53] W3C, Rif overview (second edition) (2 2013).
- URL <https://www.w3.org/TR/rif-overview/>
- [54] W3C, Notation3 (n3): A readable rdf syntax (3 2011).
- URL <https://www.w3.org/TeamSubmission/n3/>
- [55] W3C, Spin - overview and motivation (2 2011).
- URL <https://www.w3.org/Submission/spin-overview/>
- [56] P. Hagedorn, M. König, Rule-based semantic validation for standardized linked building models (2021). doi:10.1007/978-3-030-51295-8_53.
- [57] W3C, Shapes constraint language (shacl) (7 2017).
- URL <https://www.w3.org/TR/shacl/>
- [58] J. Oraskari, J. Beetz, M. Senthilvel, Shacl is for lbd what mvdxml is for ifc (10 2021).
- URL <https://github.com/w3c-lbd-cg/opm>
- [59] Description Logic Expressivity.
- URL <http://protegeproject.github.io/protege/views/ontology-metrics/>
- [60] M. Debellis, A practical guide to building owl ontologies using protégé 5.5 and plugins (04 2021).
- [61] P. Pauwels, Buildings and Semantics : Data Models and Web Technologies for the Built Environment, Buildings and Semantics, Taylor Francis Group, 2022. doi:10.1201/9781003204381.
- [62] J. Euzenat, P. Shvaiko, *Ontology matching*, 2nd Edition, 2013.
- [63] G. F. Schneider, Towards aligning domain ontologies with the building topology ontology, *Proceedings of the 5th Linked Data in Architecture and Construction Workshop (LDAC 2017)* (2017).
- [64] W. Terkaj, G. F. Schneider, P. Pauwels, Reusing domain ontologies in linked building data: The case of building automation and control, Vol. 2050, 2017.
- [65] D. Mavropapadidis, K. Katsigarakis, P. Pauwels, E. Petrova, I. Korolija, D. Rovas, A linked-data paradigm for the integration of static and dynamic building data in digital twins, 2021. doi:10.1145/3486611.3491125.
- [66] L. Bindra, K. Eng, O. Ardukanian, E. Stroulia, Flexible, decentralised access control for smart buildings with smart contracts, *Cyber-Physical Systems* (2021). doi:10.1080/23335777.2021.1922502.
- [67] React, a JavaScript library for building user interfaces.
- URL <https://github.com/reactjs/reactjs.org>

- [68] Fast, unopinionated, minimalist web framework for node.
URL <https://github.com/expressjs/express>
- [69] Node.js is an open-source, cross-platform, JavaScript runtime environment.
URL <https://github.com/nodejs/node>
- [70] FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.
URL <https://github.com/tiangolo/fastapi>
- [71] Apache Jena Fuseki is a SPARQL server.
URL <https://github.com/apache/jena/blob/main/jena-fuseki2/apache-jena-fuseki/fuseki-server>
- [72] Revit: BIM software for designers, builders, and doers.
URL <https://www.autodesk.eu/products/revit>
- [73] F. Seeberg, J. Tangeraas, Integration of Thermal Building Simulation Tools and Cloud-Based Building Information Models (2022).

Appendix A. Mapping between Flow Properties: Ontology (FPO) and Brick

Table A.10: Alignments between FPO and Brick.

owl:Class	rdfs:subClassOf
owl:ObjectProperty	rdfs:subPropertyOf
	owl:equivalentClass
fpo:hasDesignCoolingPower	brick:coolingCapacity
fpo:hasVolume	brick:volume

Appendix B. Querying fso:Pump pressure

```

SELECT ?pump (MAX(?sumOfSupplyPressureDrop +
↪ ?sumOfReturnPressureDrop +
↪ ?terminalPressureDropValue) AS ?pressure)
WHERE {
  {
    SELECT ?pump ?terminal ( SUM(?supplyValue) AS
↪ ?sumOfSupplyPressureDrop)
    WHERE {
      ?pump a fso:Pump .
      VALUES ?terminalType {fso:HeatExchanger
↪ fso:SpaceHeater}
      ?terminal a ?terminalType .
      ?supplySystem fso:hasComponent ?pump .
      ?supplyComponent fso:feedsFluidTo+ ?terminal .
      ?supplySystem fso:hasComponent ?supplyComponent .
      ?supplySystem a fso:SupplySystem .
      ?supplyComponent fso:hasPort ?supplyPort .
      ?supplyPort fpo:hasFlowDirection ?flowDirection .
      ?flowDirection fpo:hasValue "Out" .
      ?supplyPort fpo:hasPressureDrop ?pressureDrop .
      ?pressureDrop fpo:hasValue ?supplyValue .
      FILTER NOT EXISTS {
        ?supplyPort fso:suppliesFluidTo ?connectedPort .
        ?connectedComponent fso:hasPort ?connectedPort .
        ?connectedComponent fso:feedsFluidTo+ ?terminal .
        ?connectedComponent a fso:Tee .
      }} GROUP BY ?pump ?terminal
    }
  }
  SELECT ?pump ?terminal ?terminalPressureDropValue
↪ ?sumOfReturnPressureDrop
  WHERE {
    ?terminal fso:hasPort ?port .
    ?port fso:returnsFluidTo ?anotherPort .
    ?port fpo:hasPressureDrop ?pressureDrop .
    ?pressureDrop fpo:hasValue
↪ ?terminalPressureDropValue .
    {
      SELECT ?pump ?terminal ( SUM(?returnValue) AS
↪ ?sumOfReturnPressureDrop)
      WHERE {{
        ?pump a fso:Pump .
        VALUES ?terminalType {fso:HeatExchanger
↪ fso:SpaceHeater}
        ?terminal a ?terminalType .
        ?supplySystem fso:hasComponent ?pump .
        ?terminal fso:feedsFluidTo+ ?returnComponent
↪ .
        ?returnSystem fso:hasComponent
↪ ?returnComponent .
        ?returnSystem a fso:ReturnSystem .
        ?returnComponent fso:hasPort ?returnPort .
        ?returnPort fpo:hasFlowDirection
↪ ?flowDirection .
        ?flowDirection fpo:hasValue "Out" .
        ?returnPort fpo:hasPressureDrop ?pressureDrop
↪ .
        ?pressureDrop fpo:hasValue ?returnValue .
      }} GROUP BY ?pump ?terminal
    }}} GROUP BY ?pump
  }

```

Listing 5: A SPARQL query to calculate the pressure of each fso:Pump

Appendix C. Deleting systems, which doesn't have any components

```
1 DELETE {
2   ?system a ?systemType .
3   ?system ?systemPred ?systemObj .
4   ?system fso:hasFlow ?flow .
5   ?flow ?flowPred ?flowObj .
6   ?flow fpo:hasTemperature ?temperature .
7   ?temperature ?tempPred ?tempObj
8 }
9 WHERE {
10  VALUES ?systemType {fso:ReturnSystem fso:SupplySystem}
11  ↪ ?system a ?systemType .
12  ?system ?systemPred ?systemObj .
13  ?system fso:hasFlow ?flow .
14  ?flow ?flowPred ?flowObj .
15  ?flow fpo:hasTemperature ?temperature .
16  ?temperature ?tempPred ?tempObj
17  FILTER NOT EXISTS {?system fso:hasComponent ?component}
18  ↪ .
19 }
```

Listing 6: A SPARQL update query to remove all `fpo:SupplySystem` and `fpo:ReturnSystem`, which is missing the predicate `fso:hasComponent` from the data model

6.6 Paper VI - Efficient management and compliance check of HVAC information in the building design phase using semantic web technologies

Efficient management and compliance check of HVAC information in the building design phase using semantic web technologies

Ali Küçükavcı^{a,*}, Mikki Seidenschmur^{a,b}, Pieter Pauwels^d, Mads Holten Rasmussen^c, Christian Anker Hviid^a

^aDepartment of Civil Engineering, Technical University of Denmark, Copenhagen, Denmark

^bRamboll, Copenhagen, Denmark

^cNiras, Allerød, Denmark

^dDepartment of the Built Environment, Eindhoven University of Technology, Eindhoven, Netherlands

Abstract

Several OWL ontologies have been developed for the AEC industry to manage domain-specific information, yet they often overlook the domain of building services and HVAC components. The Flow Systems Ontology was recently proposed to address this need, but it does not include HVAC components' size and capacity-related properties. Also, despite their strengths in representing domain-specific knowledge, ontologies cannot efficiently identify poor data quality in BIM models. A four-fold contribution is made in this research paper to define and improve the data quality of HVAC information by: (1) extending the existing Flow Systems Ontology, (2) proposing the new Flow Properties Ontology, (3) proposing an HVAC rule set for compliance checking. Moreover, we use semantic web technologies to demonstrate the benefits of efficient HVAC data management when sizing components. The demonstration case shows that we can represent the data model in a distributed way, validate it using 36 SHACL shapes and use SPARQL to determine the pressure and flow rate of fans and pumps.

Keywords: Building Information Modelling, Heating, Ventilation and Air Conditioning (HVAC), SHACL, Semantic Web technologies, Linked Data, Compliance checking, SPARQL

1. Introduction

1.1. A Document-centric AEC Industry

Architecture, Engineering and Construction (AEC) projects have become more technically complex and involve many stakeholders that must exchange information to complete a project successfully [1]. Since the Building Information Modeling (BIM) methodology was introduced in the early to mid-2000s [2], the AEC industry has experienced improvements in coordination and communication between project stakeholders and digital tools. The BIM methodology aims to achieve a more collaborative workflow and addresses the need for a Digital Information Hub [3]. It provides a method for managing structured, accessible, and reliable building data to represent the physical and functional characteristics of a 3D building model. Current BIM applications have improved the workflows across the building life cycle and typically include 3D modelling. For that reason, its use is focused on phases of the building life cycle where 3D modelling is a requirement [4]. Today, BIM methodology is mainly

based on a document-centric approach in the AEC industry, leading to poor data management across the building life cycle, disciplines, and digital tools [5]. Data is often outdated and not in sync with the real building model, for which no live access is available.

The Industry Foundation Classes (IFC) is currently the standard format of building information and has been applied to exchange the needed information among stakeholders, mainly in a file-based or document-centric approach. Extending the IFC schema with new domain-specific knowledge becomes difficult due to its monolithic structure and complexity [6]. In addition, the schema does not describe cross-domain information such as occupancy data, meteorological data, data from building automation and control systems (BACS), etc., nor information that links the different domain information to each other [4].

1.2. Linked Data & Semantic Web

The World Wide Web Consortium (W3C), with its participants consisting of academic and industrial partners, has developed open data standards for software developers to support the shift from a “Web of Documents” to a “Web of Data” [7]. They have developed the Semantic Web Technologies consisting of Resource Description Framework (RDF), RDF Schema (RDFS), Web Ontology Language (OWL), SPARQL Protocol and RDF Query Language (SPARQL), and Shapes Constraint Language

*Corresponding author

Email addresses: alikuc@byg.dtu.dk (Ali Küçükavcı), msei@ramboll.dk (Mikki Seidenschmur), p.pauwels@tue.nl (Pieter Pauwels), mhra@niras.dk (Mads Holten Rasmussen), cah@byg.dtu.dk (Christian Anker Hviid)

(SHACL). It is a framework that enables sharing, accessing, conforming, and linking data over the web in a machine-interpretable format [8, 9].

Contrary to the IFC schema, which has well-known limitations such as limited-expression range, difficulty partitioning information, and describing the same information in multiple ways, the W3C suggests more modular, polythetic, and simple data formats, also called ontologies, that can be interlinked and easily extended over time [6, 10, 11]. Figure 1 shows the concept of interconnected ontologies, and it can be seen that the domain-specific ontologies can be separated as smaller graphs and linked with other ontologies. An ontology does not need to cover an entire domain, such as HVAC systems. It can also cover minor subdomains for HVAC, such as representing different component types and their properties alone or the connectivity of HVAC components and their relations to systems and subsystems. Developing smaller ontologies that target one building domain will yield a practical and flexible way of modelling knowledge when combined [4, 12].

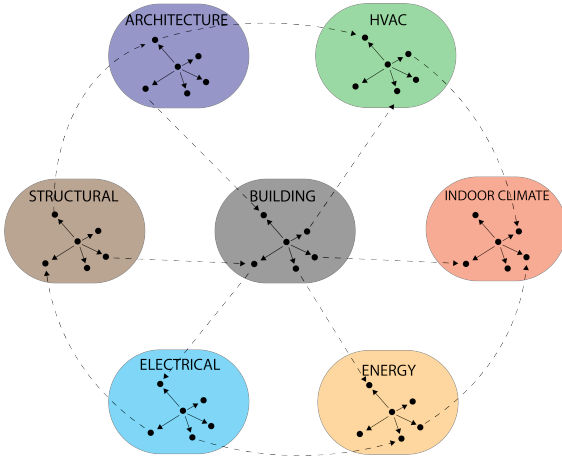


Figure 1: Interlinked domain-specific ontologies.

1.3. Interlinking Domain-specific Knowledge

In this context, the W3C LBD Community Group (W3C LBD CG) has defined and shared a set of ontologies like Building Topology Ontology (BOT) [13], Flow Systems Ontology (FSO) [14], TUBES System Ontology (TUBES) [15], Property Set Definition Ontology (PROPS) [16], and Product Ontology (PRODUCT) [17] etc. for the AEC industry. While FSO describes the energy and mass flow relationships between systems and their components and their compositions [14, 18], it lacks system components' capacity- and size-related properties. A key research question here is whether such properties need to be added directly to the FSO ontology, or can be kept separate, e.g. in its own module or ontology. In our research, we intend to

investigate whether the best approach is to create an ontology, called the Flow Properties Ontology (FPO), that includes only those properties and aligns it with other existing ontologies in the Linked Building Data (LBD) context, in particular with the FSO ontology that focuses on HVAC domain.

1.4. Conforming Domain-Specific Knowledge

Despite their strengths in representing domain-specific knowledge, ontologies cannot solve the problem that many BIM models are poorly modelled and lack building elements or metadata. Currently, poor data quality in building models contributes to faulty design decisions and downfalls in the information stream. Due to the increasing level of information, it is challenging to create sufficient BIM models [10, 19–21]. Architects and owners can spend hundreds of hours manually assessing conformity [22]. Due to the time-consuming process and the need for high-performing BIM models, many research publications have addressed conformance checking. The most prominent publications on conformance checking of BIM models cover various frameworks, tools, rule languages, rule models, and rule engines [23–33]. As their data models rely on IFC or their rule models lack semantic expressivity, they all have limitations and cause poor query performance [34, 35]. Soman et al. [36], Stolk and McGlinn [9], and Oraskari et al. [37] describe a promising approach to surpass the limitations of IFC and improve conformance checking. They use a semantic web approach with a data model written in OWL and a rule model written in SHACL to verify constraint violations. Soman et al. [36] applied the method to the construction field, while Stolk and McGlinn [9] applied the method to geospatial field, and Oraskari et al. [37] to the energy simulation field. However, these publications do not describe how to validate an HVAC model with SHACL, nor do they apply the framework to a real-world large building project. In addition, we intend to develop a rule model written in SHACL for validating HVAC-related constraints.

1.5. Contribution

Considering the above, several innovations are needed. In fact, our research includes five contributions. Firstly, our research aims to extend FSO to support an alignment with the proposed FPO ontology. Secondly, we propose the FPO ontology itself to represent HVAC components' capacity and size-related properties. Thirdly, we propose a set of rules to validate HVAC-related constraints. Fourthly, our work produces a demonstration environment for a real-world building project, showcasing how to conform a HVAC model using semantic web technologies. Lastly, the demonstration environment will showcase how FPO and the HVAC rule model can support the description and validation of hydraulics in HVAC components and the capacity of HVAC components.

1.6. Outline

Table 1 shows the namespaces and prefixes used in this article. The remainder of this article is structured as follows. Section 2 describes previous work on knowledge representation and rule checking related to buildings and systems. The presented work is limited to OWL-based data models and SHACL-based rule models. The development of FPO and extension of FSO are explained in Section 3. Section 4 outlines our framework and rules for validating HVAC-related constraints. We utilize a real-world building model in Section 5 to illustrate how FPO can represent capacity- and size-related properties and be used to design an HVAC device. Additionally, the real-world building model will be validated against our rule model in Section 5 where a process of four steps and a web application is introduced and applied to generate validation and capacity design results and display the results within a web interface. The validation results pinpoint the components or properties in the data model that are violating our rule model, while the capacity results show the flow rate and pressure of each flow-moving device that is represented in the data model. The validation and capacity design results are discussed in Section 6, and conclusions are presented in Section 7.

Table 1: Used prefixes and namespaces.

Prefix	Namespaces
fpo	https://w3id.org/fpo#
fso	https://w3id.org/fso#
fsosh	https://w3id.org/fsosh#
bot	https://w3id.org/bot#
s4bldg	https://saref.etsi.org/saref4bldg#
s4syst	https://saref.etsi.org/saref4syst#
brick	https://brickschema.org/schema/1.1/Brick#
seas	https://w3id.org/seas#
rdfs	http://www.w3.org/2000/01/rdf-schema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
ex	https://example.com/ex#
inst	https://example.com/inst#
owl	https://www.w3.org/2002/07/owl#

2. Background

2.1. Scope of the HVAC domain

The HVAC engineer is responsible for designing a building's HVAC system. The purpose of an HVAC system is to provide building occupants with acceptable thermal comfort and indoor air quality. HVAC engineers must go through a series of steps to design an HVAC system, such as defining the distribution strategy for HVAC, defining the control strategy, calculating HVAC demand by zones, and determining the capacity and size of HVAC systems and their components. To determine whether an

HVAC system is designed sufficiently, its cooling, ventilation and heating effects are compared with the building's cooling, ventilation, and heating demands. The HVAC system is considered sufficient when the capacity exceeds the building's demand. The HVAC engineer must design each HVAC component's capacity individually since an HVAC system's capacity equals the sum of its components. The HVAC component's size is then determined based on its capacity. The HVAC engineer can choose a product from a manufacturer once the capacity and size have been defined. By the time all HVAC components have been designed, the HVAC engineer has completed the HVAC design process.

Since our research project seeks to represent and validate an HVAC system's and HVAC component's capacity and size-related properties in a semantic web context, Section 2.2 provides an overview of what research has been achieved in this field and what is missing.

2.2. System representation in a Semantic Web context

A number of ontologies have been proposed to handle data within the AEC industry since the early 2000s. The first significant contribution towards moving BIM data into the Semantic Web is the ifcOWL ontology. IfcOWL is an OWL representation of the IFC schema [38, 39], and it is available at the buildingSMART website¹ as just another serialisation of the IFC schema, next to eXtensible Markup Language Schema Definition (XSD) and EXPRESS [40]. It is recognized that IFC is not the easiest method to model a building or infrastructure due to the complex relationships between building elements (mostly n-ary relationships) and the fact that it is an extremely extensive schema that is difficult to extend. Hence, this has hampered its direct use among AEC stakeholders [8, 41]. Moreover, it covers a wide range of domains, making it monolithic, rigid, and hard to extend [42]. The direct translation from the IFC schema to an OWL ontology does not change these inherent features of IFC, and so also the OWL ontology has the same limitations (complexity, limited extensibility, size). To resolve the issues, the W3C LBD CG developed a more modular and lightweight principle named LBD. This LBD approach takes a small, simple, and extensible building ontology at its core, known as the Building Topology Ontology (BOT) [13]. A BOT graph can be expanded with more specific details by interlinking with other ontologies like FSO, DOT, Brick, SAREF, etc.

BOT describes the relationship between building zones and elements [43]. A zone can be a building, a floor, a space, or a group of spaces. The building can be connected to the floor level by asserting that an entity of bot:Building is related to an entity of bot:Storey with bot:hasStorey. The same method can be applied between the storey and the space. Zones are related in BOT in a similar way

¹<https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications>

to the Babushka concept. In Babushka, smaller dolls are nested in larger dolls, whereas in BOT smaller zones are nested in larger zones. BOT can be used to describe the connections between zones in a building, but it cannot describe building systems.

SEAS describes the relationships between physical systems [44]. There are three main modules in the ontology, namely, The System Ontology, The Features Of Interest Ontology, and The Evaluation Ontology. The Features Of Interest Ontology allows to describe features of interest and their properties. A car, as an example, can be considered a feature of interest with a property called speed. Properties are either evaluated directly or through a qualified evaluation in the Evaluation Ontology. In a direct evaluation, a value is assigned to the property. A qualified evaluation needs to outline three categories: type, the context of validity, and provenance data. The System Ontology describes the systems and the relationships between them. There are three levels of connectivity: between systems, connections, or connection points. The SEAS ontology focuses primarily on electrical systems but can also be used to represent higher-level building services systems [44]. Yet, it does not describe any building service components or their relationships to building service systems.

Building service components are included in the Brick ontology [45] and the Smart Applications REference (SAREF) ontology [46] at different conceptual levels and scopes. The Brick ontology describes data points and their relationships to physical, logical, and virtual assets in buildings. It consists of a core ontology to describe fundamental concepts and their relationships and a domain-specific taxonomy. The ontology focuses on data points and their relations to location, equipment, and resource [45]. Relating a data point to a location expresses in which area of the building the data point is located. It can be located in a room, on a floor, in a duct, etc. Relating a data point to a specific equipment expresses how the data point controls the system or component. For example, take a room temperature sensor positioned in a room. The room temperature sensor regulates how much air an air handling unit (AHU) must supply to the room. Lastly, the resource is the medium being measured and regulated by the data point and equipment. For example, the medium of an AHU is the air that is being supplied to a room.

The SAREF Smart Appliances Reference ontology is a reference ontology for smart appliances (devices) [46]. It aims to bring meaningful interactions between Internet of Things (IoT) devices in various domains. There are currently 13 extensions to the core ontology. SAREF4SYST is based on the concepts of `seas:SystemOntology` to describe higher-level building service systems. SAREF4BLDG is based on the IFC taxonomy and describes building service devices. Even if it is similar to IFC and BOT, these structures are not fully the same [47]. Together, SAREF4SYST and SAREF4BLDG can represent building systems and their connectivity with IoT devices. Like Brick, the

SAREF ontology represents medium-level building system devices such as a fan or pump. Furthermore, SAREF4BLDG represents capacity-related building service devices to some extent. Those parameters are based on the IFC taxonomy. However, both Brick and SAREF ontologies are primarily focused on the operational phase of the building life cycle. As a result, they do not represent any passive building service devices such as pipes, ducts, tees, elbows, etc., nor their properties.

An OWL ontology that is similar to the SAREF4BLDG ontology, but does not include any building topology to avoid semantically overlapping ontologies, is the Mechanical, Electrical and Plumbing (MEP) ontology². This ontology is structured as a very simple hierarchical taxonomy for devices and is directly created based on the DistributionElement subtree in the IFC schema. It needs to be combined with the BOT ontology to be of use and works well to classify distribution elements such as air terminals, etc.

FSO focuses on the design and operational phase of the building life cycle [14]. It describes the mass flow and energy relationships between systems and components and the composition of such systems [14]. FSO gives the ability to connect both passive and active components to systems and subsystems. For example, a heating system can include a supply and return system as subsystems. A segment or fitting can be related to a supply or return system. A component can also be connected to a supply and return system, such as a heat exchanger. A segment can supply or return fluid to another component based on what system it belongs to. Unlike Brick and SAREF ontologies, FSO only represents higher-level components such as flow-moving device or flow-controlling devices (also included in the MEP ontology). The taxonomy of building service devices for all four ontologies is based on the IFC taxonomy. However, FSO does not represent both active and passive components' size- and capacity-related properties. Without that representation, HVAC engineers cannot design an HVAC system nor an HVAC component during the design phase using FSO.

FPO and an extended version of FSO are introduced in Section 3 to fill this research gap and describe the size- and capacity-related properties of both active and passive components within the design phase. Ontologies are mainly used to represent domain-specific knowledge. To check whether a BIM model lack building elements or metadata, we need a rule language. Section 2.3 describes the process of compliance checking, which rule languages exists and what research have achieved in this area in a Semantic Web context.

2.3. Compliance checking in a Semantic Web context

Compliance checking, code-checking, rule-based checking, and constraint checking are all terms that describe

²<https://pi.pauwel.be/voc/distributionelement>

a passive process that notifies whether a rule has been violated [48]. The process does not modify the building but validates the building design against different types of requirements such as client requirements, functional requirements, aesthetic requirements, building performance requirements, building code and regulations, complete discipline assessment and complete BIM data [22, 49]. Currently, companies primarily apply compliance checking to assess the quality and perform collision control on BIM models by utilizing the commercial tool Solibri Model Checker (SMC). Solibri Model Checker uses predefined rules for geometrical clashes, property completeness, and relationships between building elements. Using SMC does not allow the use of predefined rules in other applications or the creation of customized or complex rules [22]. In order to perform compliance checking on BIM models without being restricted to specific types of constraints or applications in general, Eastman et al. [50] provide a four-step manual approach.

1. Rule interpretation: Human-readable rules are converted into a machine-interpretable format that contains the information needed to be checked in the correct format, also known as the rule model.
2. Building model preparation: Building information is converted into a machine-readable format, also known as the data model.
3. Rule execution: The data model is validated against the rule model.
4. Rule check reporting: A validation report describing whether the data model has passed or violated any constraints.

By following these steps, custom rules can be written without being limited to a particular application. However, the process is passive and only informs the user or system whether any constraints have been met or violated. For actively correcting the violation in the data model, Solihin et al. [49] introduce a fifth step:

5. Automatic correction: If any constraints are violated, the user or system is not only notified, but new data is created to correct the violation. Users can be notified to implement the new data as an option or the new data can be implemented automatically. As some violations can be solved by multiple solutions, the system should be able to notify the user of all the possible solutions, allowing them to choose the appropriate one.

Moreover, Solihin et al. [49] suggest categorizing the defined rules based on their complexity into four categories:

Class 1: entities and attributes are queried and checked against a single value.

Class 2: additional values are calculated (e.g. distance) and checked.

Class 3: additional geometry is created, in order to calculate spatial relationships.

Class 4: problem solutions are calculated, and new data is created.

Defining each rule in the rule interpretation phase requires a rule language. In the following subsection, we describe several prominent rule languages developed by the W3C.

2.3.1. Rule languages

In 2004, the W3C introduced the Semantic Web Rule Language (SWRL) as a member submission³. SWRL is a combination of the OWL Description Language (DL) and OWL Lite sublanguages of OWL with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. OWL knowledge bases are integrated with Horn-like rules in the rule language. The rules are expressed in terms of OWL concepts, such as classes, properties and individuals. Because OWL ontologies are limited in their ability to express complex logical reasoning, SWRL allows users to create custom rules and apply them to OWL ontologies [51, 52].

Similar to SWRL, the Rule Interchange Format (RIF) introduced in 2005 by W3C allows rules to be expressed in XML syntax. In order to enhance interoperability between rule languages, RIF was designed to be the standard exchange format for rules on the Semantic Web. As of today, RIF consists of 12 parts, including RIF-core, which is the core of all RIF dialects [52, 53].

Notation3 (N3), is an assertion and logic language that supports expressing RDF-based rules. It was introduced in 2011 by W3C as a team submission to extend RDF by adding formulae, variables, logical implication, and functional predicates, as well as to provide an alternative syntax to the XML syntax that SWRL and RIF use. By using shortcuts and syntactic sugar, it is able to simplify statements in the form of triples [54].

The SPARQL Inferencing Notation (SPIN) was introduced by W3C in 2011 as a member submission and has become a de facto industry standard for describing SPARQL rules and constraints. The key feature of SPIN, compared to SWRL, RIF, and N3, is the ability to specify constraints using SPARQL queries. In this way, property values can be calculated based on other properties, or a set of rules can be isolated for execution under certain conditions. It is also possible to use SPIN to check the validity of constraints based on the assumption of a closed world [55].

SHACL is the successor to SPIN and was published as a W3C Recommendation in 2017 [56, 57]. A higher

³<https://www.w3.org/2021/Process-20211102/>

status has been granted to SHACL by W3C in comparison to SWRL, RIF, N3 and SPIN. As a result, SHACL has become the web standard today for validating RDF graphs. SHACL is heavily inspired by SPIN, but it offers far more flexibility in defining target constraints. SPIN is limited to classes, while SHACL can be applied to classes or sets of nodes by various target mechanisms, including customized targets. Furthermore, SHACL advanced features allow validation of more complex constraint types, such as sub-graph pattern validation, conditional validation, etc.. SHACL contains two major components:

Data graph: A data model containing domain-specific knowledge.

Shape graph: A rule model, consisting of user-defined constraints. User-defined shapes can be node shapes or property shapes. Node shapes specify constraints on target nodes, while property shapes specify constraints on target properties and their values.

By separating the data model and rule model, SHACL follows the Business Rule Management Systems (BRMS) principle of decomposing knowledge into logic and data, enabling them to be independently manipulated [36]. In addition, SHACL outputs an RDF graph with validation results, which describes whether a data model passed or failed a given rule-set.

The following section highlights the research gap based on an overview of recent research on applying SHACL to perform conformance checking within the AEC industry.

2.3.2. The research gap in case studies

Stolk and McGlinn [9] demonstrated how ifcOWL can be validated using SHACL. The authors showed how ifc:LengthValueIfcQuantityLength can be restricted to only have values of type ifc:IfcLengthMeasure and how cardinality constraints can be used to restrict IfcDoorPanel properties.

Hagedorn and König [56] developed an approach for compliance checking linked building models. The proposed method implements the four steps mentioned by Eastman using semantic web technologies. Using the IFC2RDF converter, the authors converted an IFC schema into ifcOWL. Their rule model involved a set of rules to validate the path between an identifier of a link and the original identifier. In order to validate their data model against the rule model and receive a validation report, they used the W3C SHACL Test Suite.

To define and check complex and dynamic scheduling constraints in construction, Soman et al. [36] developed a linked-data based constraint-checking approach utilizing semantic web technologies. The approach was implemented through a web application that validated construction scheduling violations using different types of constraints. The pySHACL library was used to define and validate SHACL shapes and the RDFlib library was used

to design and store a RDF graph. They used IfcOWL and LinkOnt to capture the model information of a real-building model.

Oraskari et al. [58] defined rules within the energy simulation field for validating windows of specific sizes, checksums of properties, and alignments of BOT classes and properties. They validated two data models against each other in order to align BOT classes and properties with ifcOWL. The IFC schema of a conceptual building model was converted to ifcOWL and BOT using the IFCtoLBD and IFC2BOT converters. The rule modelling, validation and reporting was performed using the TopBraid SHACL Application Programming Interface (API).

None of the mentioned authors developed a SHACL-based rule model nor performed a conformance check against an OWL-based HVAC model. Soman et al. [36] is the only author that uses a real building model, but a model of low complexity and size. For that reason, a constraint-checking approach to define and validate HVAC-related constraints on a large real-building model using semantic web technologies is introduced in Section 4 to fill this research gap.

3. Flow Properties Ontology

FPO is developed as an extension to FSO [14] to represent FSO component's capacity and size-related properties. The development of FPO is closely related to FSO, but the authors in [14] sought to keep FSO as lightweight as possible, to describe a myriad of different flow systems. As a result, we developed FPO as an extension to FSO. It contains 50 classes, 50 object properties and 6 data properties and has a Description Logic expressivity of $\mathcal{ALRF}(\mathcal{D})$ [59]. A practical guide [60] was used to design and structure the classes, object properties and data properties in FPO. Classes, for instance, should always begin with capital letters, also known as upper camel case, and should not contain spaces. In contrast, object properties and data properties should always be written in lower camel case and with verb senses.

It is necessary to know the HVAC component type to describe its properties. A property of one HVAC component may differ from another, and the data type or unit of one property may vary from another property. A pump has different properties than a fan, and the flow rate can be expressed in liters per second or cubic meters per hour which is different from a ventilation fan. An elbow can differ in properties from a tee by having an angle even if both are fittings. Moreover, while a tee has three flow ports and elbow has two flow ports. Conceptually, Figure 2 illustrates how a component can have a property, and the property a value. As there are two steps between the component (Type / Object) and the value, this property modelling approach is a Level 2 (L2) property modelling approach, as defined by Bonduel and Pauwels [61]. Other property modelling approaches are L1 (direct object

and data properties), and L2 (more metadata for tracking property states over time).



Figure 2: Relationship between components, properties, and property values.

It is possible to represent buildings, spaces, and their relationships with systems and components using FSO and BOT. Adding FPO, the representation can identify whether a particular system or component is able to heat, cool, or ventilate a specific building or space.

The following subsections provide a more detailed description of FPO. To determine the scope of the ontology, Section 3.1 lists a set of competency questions. In Section 3.2.2, FSO is extended with medium-level components to represent component interfaces and their connections with other components. Section 3.3 reviews FPO classes and their properties. Finally, reasoning examples will be enabled in Section 3.4, followed by alignments to FSO, SAREF4BLDG, MEP, and Brick in Section 3.5. Both the extension of FSO, the development of FPO and the alignments are made available on GitHub⁴.

3.1. Competency questions

Competency questions are listed in Table 2 to determine FPO’s scope and purpose formally. The scope of the ontology is verified in Section 5 with SPARQL queries.

Table 2: Competency questions

Reference	Competency question
CQ1	What is the heating, cooling or ventilation capacity of a system?
CQ2	What is the heating, cooling or ventilation capacity of an HVAC component?
CQ3	What is the size of a given HVAC component?

3.2. Flow System Ontology Extended

3.2.1. Connection between components

FSO represents the energy and mass flow relationships between systems and their components and their composition. However, the current version of FSO does not express the opening or passage that directs the flow of energy or mass. The existing version of FSO expresses a segment. This simplistic representation is insufficient to determine an HVAC component’s size or capacity during the building

design phase. An actual component contains a fluid, which is in motion. This is known as flow. Ports are added for the fluid to flow in and out of each component. The existing FSO taxonomy is therefore extended with **fso:Port** and **fso:Flow**. As a result, a hierarchical relationship can be described among systems, components, ports, and flows.

The concept of relating a **fso:Port** and a **fso:Flow** for multiple components is shown in Figure 3. An **fso:Segment** can be linked to an **fso:Port** with **fso:hasPort**, and an **fso:Port** can be linked to a flow with **fso:hasFlow**. With **fso:hasPort** and **fso:hasFlow** available, an **fso:Fitting** can be related to its ports and flow. The direct relationship between the ports of both components is expressed using **fso:suppliesFluidTo**. In some cases, it is sufficient to just represent the ports and not to explicitly indicate the flow. In that case, the **fso:Flow** instances can simply be left out.

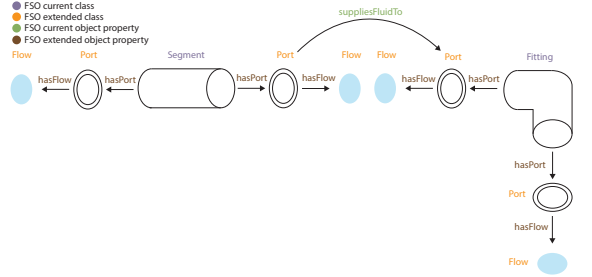


Figure 3: A segment partitioned with ports and flow connects to an fitting through its ports

In addition, the opening can also be expressed as a **fso:ConnectionPoint** instead of a **fso:Port**. A single connection point can be used to represent connections between components instead of multiple ports. The **fso:ConnectionPoint** is an interface between two components that transports fluid. Figure 4 illustrates how multiple components can be related using **fso:ConnectionPoint**. The **fso:Segment** relates to a **fso:ConnectionPoint** with **fso:ConnectsTo**, while the **fso:Fitting** relates to a **fso:ConnectionPoint** with **fso:ConnectsFrom**. A connection point’s relationship to a component also determines the intended direction of the flow, which is crucial information when performing hydraulic calculations. The fluid is transported from the **fso:Segment** to the **fso:Fitting** in Figure 4. Both **fso:Port** and **fso:ConnectionPoint** are subclasses of **bot:Interface**.

A relationship can be described among systems, and components as shown in Figure 5. The components share the same **fso:ConnectionPoint**. Flows and Ports are not available in this example, but could be modelled as well, after the example in Figure 3.

The proposed extension to FSO makes it capable of representing components and interfaces in multiple ways, which adds some flexibility. The definition of the mentioned classes and relationships in this section is defined

⁴<https://github.com/Semantic-Web-Tool/Orchestrator-Service/tree/main/public/Ontologies>

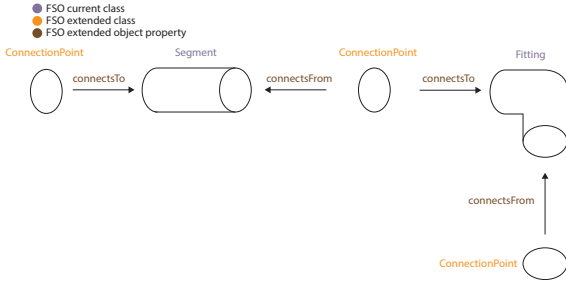


Figure 4: A segment connects to a fitting through connection points.

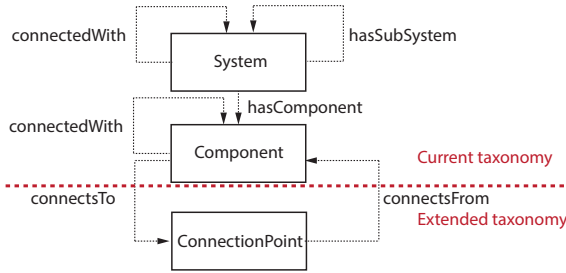


Figure 5: Current and extended taxonomy of FSO with connection points.

as follows:

- **fso:Port** is defined as “An opening or passage that directs flow of a mass or energy”.
- **fso:Flow** is defined as a “A fluid flowing into or out of a port to another port”.
- **fso:ConnectionPoint** is defined as “A point of interaction between components”.
- **fso:hasPort** is defined as “The relation from a component to a port.”
- **fso:hasFlow** is defined as “The relation from a port to a flow.”
- **fso:connectsTo** is defined as “The relation from a connection point to a component.”
- **fso:connectsFrom** is defined as “The relation from a connection point to a component.”

3.2.2. Extended component abstraction level

Currently, FSO represents eight high-level component types. For several reasons, we must subdivide the eight high-level component types into 19 medium-level components. For instance, the hydraulic sizing of a pump or a fan are different. The sizing of a pump includes the pressure drop from both supply system components and return system components, but sizing of a fan only includes pressure

drop of either supply or return side. We have to define the types explicitly when performing hydraulic calculations.

Often components lack the required properties to perform a hydraulic calculation. For example, if an elbow does not have a specified angle, we will not be able to differentiate between an elbow or transition since they both are represented as a **fso:Fitting** and have two ports. To accommodate the difference in properties, the eight high-level FSO components have been nested into 19 medium-level components as shown in Figure 6.

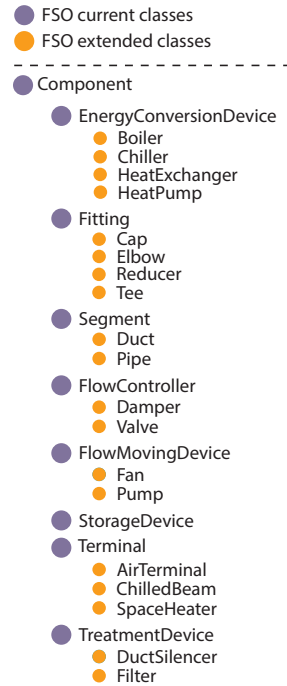


Figure 6: A class hierarchy of current and extended FSO components.

3.3. Property relationships

FPO provides 6 data properties: value, unit, abbreviation, design condition and curve. They can be used to relate an entity literal to an entity class. Combined, the 50 classes, 50 object properties and 6 data properties represent the size and capacity of the FSO components. Figure 7 demonstrates how properties are added to components, ports, or flows. An **fso:Segment** can be related to the property **fpo:Length** with **fpo:hasProperty**. With **fpo:hasValue** and **fpo:hasUnit**, **fpo:Length** can be connected to the value '15' and the unit *meter*. In this example, **fso:Segment** and **fpo:Length** are both classes, while **fpo:hasProperty** is an object property and **fpo:hasUnit** and **fpo:hasValue** are data properties. This method is applied to both **fso:Port** and **fso:Flow**. With this approach, we entirely follow the L2 property modelling ap-

proach that is documented by Bonduel and Pauwels [61] and in principle follows a one-to-many pattern.

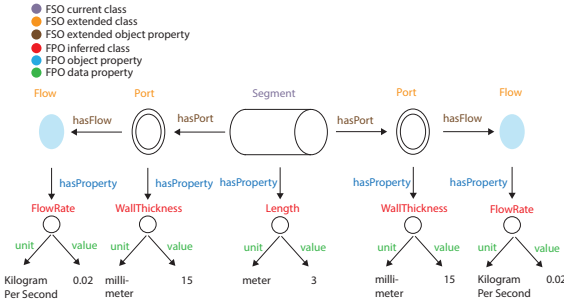


Figure 7: Describing the relationship between an `fso:Segment` and its properties with FPO classes, object and data properties.

3.4. Reasoning

Semantic Web technologies enable deductive reasoning as well as explicit assertions. A few examples of how FPO and the extended FSO allow for reasoning are presented in this section. Every object property in FPO is assigned a domain and a range. For example, the attribute `fpo:hasLength` has the domain `fso:Component` and range `fpo:Length`. This means that, whenever we have a subject of type `fso:Component` and a predicate of type `fpo:hasLength`, then the object must be of type `fpo:Length`. This also means that a reasoning engine will automatically infer the class `fpo:Length` when the object property `fpo:hasLength` is provided in the input instance data. This can similarly be done for all the other properties shown in Figure 7.

An `fso:Segment` is shown in Figure 3 supplying fluid to an `fso:Fitting` with the property `fso:suppliesFluidTo`. However, with the extended FSO, it is possible to infer that if a segment port supplies fluid to another port of a fitting, then the segment must also feed fluid to the fitting (transitive object property). Figure 8 illustrates the inferred knowledge. This can similarly be done for an `fso:connectionPoint` (example shown in Figure 4). If a connection point is connected to a segment and connected from a fitting, it can be inferred that the segment feeds fluid to the fitting.

3.5. Alignments

Figure 9 shows the relation between BOT, FSO and FPO. The figure also illustrates how this network of ontologies can be used to show the relationship between a heating system, its components, properties, and the buildings it serves. It simplifies the relationship between the HVAC components and their properties for illustration purposes.

The taxonomy of components in FPO, FSO, MEP, SAREF4BLDG, and Brick is based mainly on the IFC taxonomy and can therefore be aligned. Of course, they

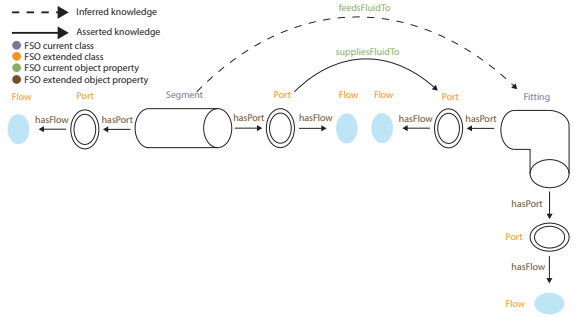


Figure 8: Deducing that the segment feeds fluid to the fitting as a port of the segment supplies fluid to a port of the fitting.

can never be fully aligned because of their difference in semantic meaning and definitions. Mappings between these and other ontologies always remain limited, faulty, and very much open to interpretation and use; by the very nature of mapping ontologies [62]. The mentioned ontologies do not represent all the same components, nor are they conceptually equivalent. Both SAREF4BLDG and Brick represent some component properties but are not intended to describe the capacity or size of each component as FPO does. Even the definition for Zone, which is available in SAREF4BLDG and BOT, for example, has very different meanings in both ontologies and should not be translated or mapped to one another [13, 46].

Classes, object properties, and data properties are nevertheless compared between the ontologies in this section. It is nevertheless recommended to not rely fully on these ontology mappings and instead rely much more on instance linking, as recommended by Schneider [63], Rasmussen [43] and Terkaj [64]. An instance can hereby be annotated as a Brick class, BOT class, and FPO class using the advantage of multi-typing in RDF graphs [14, 65, 66].

For the ontology mapping in the below section, we follow standard approaches and aim to organize FPO classes as either sub-classes or equivalents to classes in another ontology. This notion also applies to object and data properties. It can either be a sub-property or equivalent to another ontology. This is the case when aligning FPO and SAREF4BLDG as shown in Table 3. We are able to align 14 object properties between FPO and SAREF4BLDG. For example, `fpo:hasKv` is a sub-property of `s4blbg:flowCoefficient`, while `fpo:hasVolume` is an equivalent property to `s4blbg:volume`. Moreover, `fpo:hasDesignAirflowRate` is equivalent to `s4blbg:airFlowRateMin`, as their definitions are equivalent.

Just like SAREF4BLDG, Brick components can be equally aligned with FPO components. We can align 2 of the 50 FPO object properties with Brick. For example, `fpo:hasVolume` is equivalent to `brick:volume` as shown in Appendix A. Care needs to be taken, as it is very easy to introduce false assumptions in the data using these mappings.

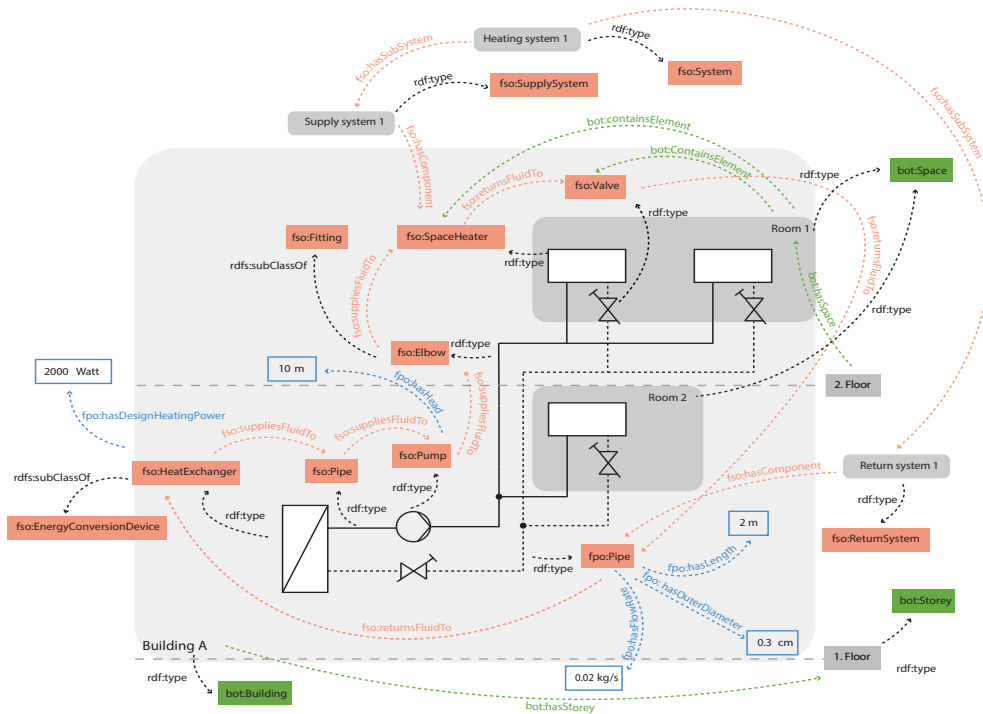


Figure 9: Combining multiple ontologies to represent building, spaces, systems, HVAC components, their properties and their relationships

740 4. HVAC rule model

The HVAC rule model was developed to check the composition of HVAC components, their systems, and their capacity and size-related properties. The HVAC rule model consists of 36 shapes and 122 constraints and is made available on GitHub⁵. A shape of constraints can, for example, determine whether a pipe is a part of a system, has two flow ports and is connected to other components. It can also check whether the port of a pipe has the capacity-related property flow rate or the pipe has the size-related property diameter. In a validation process, the HVAC rule model will ensure that the necessary BIM information is available to calculate the size and capacity of HVAC systems and their components. The calculation is also known as the hydraulic calculation.

A rule can differ in complexity and range from 1-4, as defined by Solihin et al. [49]. In this section, we showcase a SHACL-based rule for each complexity level.

4.1. Verifying pipes explicitly

In hydraulic calculations, it is essential to know the location of each pipe segment in relation to upstream and downstream components, as well as roughness and length.

Table 3: Alignments between FPO and s4bldg.

owl:Class	rdfs:subClassOf
owl:ObjectProperty	rdfs:subPropertyOf
	owl:equivalentClass
fpo:hasDesignAirflowRate	s4bldg:airFlowRateMin ⁷⁴⁵
fpo:hasCrossSectionalArea	s4bldg:faceArea
fpo:hasKv	s4bldg:flowCoefficient ⁷⁵⁰
fpo:hasHeight	s4bldg:height
fpo:hasOuterDiameter	s4bldg:inletConnectionSize
fpo:hasDesignVolume	s4bldg:waterStorageCapacity
fpo:hasPressure	s4bldg:openPressureDrop
fpo:hasOuterDiameter	s4bldg:outletConnectionSize ⁷⁵⁵
fpo:hasDesignHeatingPower	s4bldg:outputCapacity
fpo:hasOuterDiameter	s4bldg:outerDiameter
fpo:hasRoughness	s4bldg:roughness
fpo:hasThermalConductivity	s4bldg:thermalConductivity
fpo:hasVolume	s4bldg:volume
fpo:hasLength	s4bldg:length ⁷⁶⁰

⁵<https://github.com/Semantic-HVAC-Tool/Rule-Service/tree/main/Public/Shapes/fsosh.ttl>

The shape `fso:Pipe` applies 7 constraints to an `fso:Pipe` and has a complexity level of 1 and are described as follows:

Constraint 1: An `fso:Pipe` must have exactly two flow ports.

Constraint 2: A pipe must feed fluid to exactly one component.

Constraint 3: A pipe must be fed with fluid by exactly one component.

Constraint 4: A pipe must be connected to exactly one system.

Constraint 5: Exactly one property of material type must be present in a pipe.

Constraint 6: Exactly one property of length must be present for a pipe.

Constraint 7: Exactly one property of roughness type must be present for a pipe.

In Listing 1, only the first constraint is expressed in SHACL. The remaining 6 SHACL constraints are made available on GitHub⁶. In the first constraint, the cardinality constraints `sh:minCount` and `sh:maxCount` are applied to check that the `fso:Pipe` has two ports. A minimum and maximum cardinality of 2 will satisfy this constraint. In addition, we use the value type constraint `sh:datatype` with the value `xsd:anyURI` to ensure the triple includes an URI. If the cardinality constraint or value type constraint is not satisfied, the message “A pipe must have exactly two flow ports” will be thrown.

Listing 1: A SHACL shape to constrain the number of `fso:Ports` with `fso:hasPort` for each `fso:Pipe`.

```
1  fsosh:Pipe
2  a sh:NodeShape;
3  sh:nodeKind sh:IRI ;
4  sh:targetClass fso:Pipe ;
5  sh:property[
6    sh:path fso:hasPort ;
7    sh:datatype xsd:anyURI;
8    sh:minCount 2;
9    sh:maxCount 2;
10   sh:message "A pipe must have exactly two flow
11   ports"
12 ]; #... the shape continues
```

4.2. Verifying the demand versus capacity by derived information

HVAC systems and their components must be designed to provide sufficient heating, cooling, and/or ventilation to

buildings. For example, an HVAC terminal is designed correctly if its capacity to heat, cool, and ventilate a space exceeds the space’s demand. With the following constraint, we demonstrate how the capacity of a supply air terminal can be compared with the supply airflow demand of a space:

Constraint 1: The supply air terminal capacity should be higher than the space’s required supply airflow demand.

The rule is expressed in a single SHACL shape, as shown in Listing 2 and the constraint belongs to the shape `fsosh:AirTerminalCapacityCheck`. A SPARQL-based constraint is used to implicitly find the comparison between capacity and demand since it is not explicitly defined. Because this rule requires derived information, it reaches complexity level 2. A nested SPARQL select query is shown in Listing 2. There can be more than one supply air terminal in a space. To sum the capacity of all air terminals grouped by space, we apply an inner select query. In the outer select query, we find the supply airflow demand for each space and filter them according to the constraint. This rule will be violated when the supply air terminal capacity exceeds the supply airflow demand of the space.

Listing 2: The listing shows a SHACL shape to constrain the capacity of an supply air terminal versus the supply airflow demand of an space.

```
fsosh:AirTerminalCapacityCheck
a sh:NodeShape;
sh:nodeKind sh:IRI ;
sh:targetClass bot:Space ;
sh:sparql [
  a sh:SPARQLConstrain ;
  sh:message "The supply air terminal capacity shall
  not be lower the required supply air flow demand of
  the space" ;
  sh:prefixes (fpo: fso: ex: inst: bot:);
  sh:select """PREFIX bot:<https://w3id.org/bot#>
  PREFIX ex: <https://example.com/ex#> PREFIX fso:
  <http://w3id.org/fso#> PREFIX fpo:
  <http://w3id.org/fpo#>
  SELECT ?this {
    ?this ex:designSupplyAirflowDemand ?flowDemand .
    ?flowDemand fpo:hasValue ?flowDemandValue .
    BIND (ROUND(?flowDemandValue) AS ?demand) .
    {
      SELECT ?this (ROUND(SUM(?flowCapValue)) AS
      ?capacity) WHERE {
        ?this a bot:Space .
        ?airTerminal a fso:AirTerminal .
        ?airTerminal fpo:hasAirTerminalType
        ?airTerminalType .
        ?airTerminalType fpo:hasValue "inlet" .
        ?airTerminal fso:feedsFluidTo ?this .
        ?airTerminal fso:hasPort ?port .
        ?port fpo:hasFlowDirection ?flowDirection .
        ?flowDirection fpo:hasValue "Out" .
        ?port fpo:hasFlowRate ?flowCapacity .
        ?flowCapacity fpo:hasValue ?flowCapValue .
      } GROUP BY ?this
    }
  }
```

⁶<https://github.com/Semantic-HVAC-Tool/Rule-Service/tree/main/Public/Shapes/fsosh.ttl>

```

29 BIND (((?capacity/?demand)-1)*10 as ?oversizing) .
30 FILTER (?demand > ?capacity || ?oversizing > 10 )
31 } "" ; ] .

```

```

bind ((?pressureDropValue / ?lengthvalue) AS
↪ ?value) .
FILTER (?value > 100)} "" ; ] .

```

850

4.3. A rule of thumb to verify pressure drop in pipes

The pressure drop in pipes affects the economy of building projects, the material's lifetime and the energy consumption of HVAC systems. A high pressure loss will result in a lower cost price, a shorter lifetime, and higher energy consumption. As a result, most HVAC engineers apply a guideline to their design, e.g. a maximum pipe pressure loss of 100 Pa/m. This guideline or rule cannot be conveyed through explicit information. Calculations and derived information are also required. The complexity level of the shape `fsosh:PipePressureDrop` reaches 3 because an engine is used to calculate the pressure drop and velocity of each distribution component. The engine is discussed in detail in Section 5.1. The only constraint in this rule is targeting an `fso:Pipe` and is described as follows:

Constraint 1: The pressure drop of a `fso:Pipe` shall not exceed 100 Pa/m.

Listing 3 shows the rule expression in SHACL. The pressure drop in pipes is not explicitly defined in Pa/m in FSO or FPO. We can, however, implicitly find the information using a SPARQL constraint. Our SPARQL-based constraint contains a SPARQL select query. The select query returns all instances of `fso:Pipe` that exceeds 100 Pa/m in pressure drop. By dividing the length of the pipe by the pressure drop at the outlet port, we can determine the pressure drop in Pa/m for each `fso:Pipe` instance.

Listing 3: A SHACL shape to constrain the maximum pressure drop of each `fso:Pipe`.

```

1 fsosh:PipePressureDrop
2   a sh:NodeShape;
3   sh:nodeKind sh:IRI ;
4   sh:targetClass fpo:Pipe ;
5   sh:sparql [
6     a sh:SPARQLConstraint ;
7     sh:message "The pressure drop of a fso:Pipe shall
↪ not exceed 100 Pa/m";
8     sh:prefixes (fpo: fso: inst:);
9     sh:select ""PREFIX fso: <http://w3id.org/fso#>
10    PREFIX fpo: <http://w3id.org/fpo#>
11    PREFIX inst: <https://example.com/inst#>
12    SELECT ?this ?value
13    WHERE {
14      ?this a fso:Pipe .
15      ?this fpo:hasLength ?length .
16      ?length fpo:hasValue ?lengthvalue .
17      ?this fso:hasPort ?port .
18      ?port fpo:hasFlowDirection ?flowDirection .
19      ?flowDirection fpo:hasValue "Out" .
20      ?port fpo:hasPressureDrop ?pressureDrop .
21      ?pressureDrop fpo:hasValue ?pressureDropValue .

```

4.4. Redesigning the size of pipes automatically

During the HVAC design process, HVAC components are often oversized or undersized due to limited time. Rather than just creating a rule that notifies whether HVAC components are right-sized passively, we will generate new data actively and add it to the model. By increasing the diameter of the pipe, we can decrease the pressure drop. That is precisely what Listing 4 is doing. Listing 4 is an inference rule expressed in SHACL. Using a SPARQL construct query, the pipe diameter is increased based on the material type and standard manufacturer size. The dimensions are limited to the material type PEX⁷ and range from 0.012 to 0.050 meters. For every `fso:Pipe` that violates the previous rule, `fsosh:PipePressureDrop`, the active rule generates a new diameter. For instance, a pipe diameter of 0.012 meters will automatically be increased to 0.015 meters and added to the data model. Since this rule can generate new information, it reaches a complexity level of 4.

Listing 4: A SHACL shape to increase the size of a `fso:Pipe` automatically

```

fsosh:PipePexSizing
  a sh:NodeShape ;
  sh:targetClass fso:Pipe ;
  sh:rule [
    a sh:SPARQLRule ;
    sh:prefixes (fpo: fso: ex: );
    sh:construct ""
    CONSTRUCT {?diameter fpo:hasValue ?newSize.}
    WHERE {
      ?this a fso:Pipe .
      ?this fpo:hasMaterialType ?type .
      ?type fpo:hasValue "PEX 6 bar varme" .
      ?this fso:hasPort ?port .
      ?port fpo:hasOuterDiameter ?diameter .
      ?diameter fpo:hasValue ?diameterValue .
      BIND (
        IF(?diameterValue = 0.012, 0.015,
          IF(?diameterValue = 0.015, 0.018,
            IF(?diameterValue = 0.018, 0.020,
              IF(?diameterValue = 0.020, 0.022,
                IF(?diameterValue = 0.022, 0.028,
                  IF(?diameterValue = 0.028, 0.032,
                    IF(?diameterValue = 0.032, 0.040,
                      IF(?diameterValue = 0.040, 0.050,
                        ?diameterValue)))))))))
        AS ?newSize)} "" ;
      condition: fsosh: PipePressureDrop
    ] .

```

⁷<https://www.bobvila.com/articles/pex-pipe>

5. Demonstration Environment

This section aims to demonstrate how capacity and size-related properties within the HVAC domain can be represented and validated for a real-world BIM model. The use case process is illustrated in Figure 10.

The first step of the process is to create a data graph and shape graph. As the shape graph is already produced in Section 4, it does not require further processing and can be used as-is⁸. In contrast, converting a BIM model will create the data graph. This step is identical to the building model preparation phase of Eastman et al. [50]. The data graph contains BOT, FSO, and FPO vocabularies so that it matches with the rules in our shape graph and can proceed to the rule execution phase of Eastman et al. [50]. Using these three vocabularies, we can describe the building, its services, its interactions, and properties. For example, we can express how the HVAC system or an HVAC component relates to the building or a specific room.

In the second step, a rule execution process will be performed to check the shape graph against the data graph. The data graph will be manually corrected if any constraints are violated during rule execution. Depending on the violation type, manual correction can be achieved at three levels; BIM model, parser or data graph. In cases where we do not want to modify the BIM model, we can use SPARQL on the data graph or add the information through the parser.

When the rule execution conforms, we can proceed to step 3. This step involves hydraulic calculations for ducts, pipes, and fittings to determine each distribution component's pressure drop and fluid velocity. A second conformance check will be conducted to check the shape graph against the data graph and the hydraulic results. Whenever a constraint is violated, an HVAC rule at level 4 in complexity from the shape graph will be used to correct the violation.

When the rule execution conforms, we will have all the information necessary to size the flow-moving device. Step 4 will therefore involve calculating the capacity of each flow-moving device, represented in the data graph. After the flow-moving devices' capacities has been calculated, the result is given, and the process ends.

5.1. A Semantic HVAC tool

We developed the Semantic HVAC tool to perform the process shown in Figure 10. The web tool has a microservice-oriented system architecture and contains four layers, which is illustrated in Figure 11. The source code of the Semantic HVAC Tool and the material used to perform the process shown in Figure 10 is made available on GitHub⁹. The following sections first describe the data

flow in detail and then demonstrate the Semantic HVAC tool in a use case.

5.1.1. Presentation layer

The presentation layer handles the user interface logic and displays data on the page. The Graphical User Interface (GUI) relies on React components to improve page rendering [67]. Using the GUI, users can perform conformance checking, perform hydraulic calculations, calculate the capacity of flow-moving devices, and view the results. The user has to initiate the conformance checking and calculations in the right order as shown in Figure 10. It is therefore necessary for the user to initiate the conformance check first. The user must correct all violations manually if any exist. If any violation exists, the GUI will not allow the user to perform the hydraulic calculation. Using this method, we ensure that the data model contains all the information we need to calculate the hydraulics. The same applies to the capacity calculation of flow-moving devices. If any violations occur after the second conformance check, the GUI will not allow the user to initiate the flow-moving device calculation.

The GUI displays the conformance check results in two different tables. Based on the type of HVAC component, the HVAC system, and size and capacity properties, the first table shows the number of violations. The first table is interactive. By clicking on a specific HVAC component type in the first table, the GUI will display the second table. The second table lists the violations for that specific HVAC component in more detail, including the instance ID, constraint type, and violation description. Additionally, the GUI shows the results of the flow-moving device calculation in a table. The table displays the type, ID, flow rate, and pressure of each flow-moving device.

5.1.2. Communication layer

The orchestrator handles the communication between the service components in the Semantic HVAC Tool via HTTP requests.

There are two ways to communicate between services: decentralized and centralized. Decentralized communication allows microservice components to communicate directly with each other. In central communication, microservices will communicate through an orchestrator service. As illustrated in Figure 11, we have implemented a central orchestrator to handle the communication between the presentation layer, the business layer, and the database layer. The orchestrator is developed as an ExpressJS server [68] in NodeJS [69]. When the user initiates the conformance checking, the following communication will happen:

1. the client requests conformance checking results from the orchestrator.
2. the orchestrator requests conformance checking results from the rule service.

⁸<https://github.com/Semantic-HVAC-Tool/Rule-Service/tree/main/Public/Shapes/fsosh.ttl>

⁹<https://github.com/Semantic-HVAC-Tool>

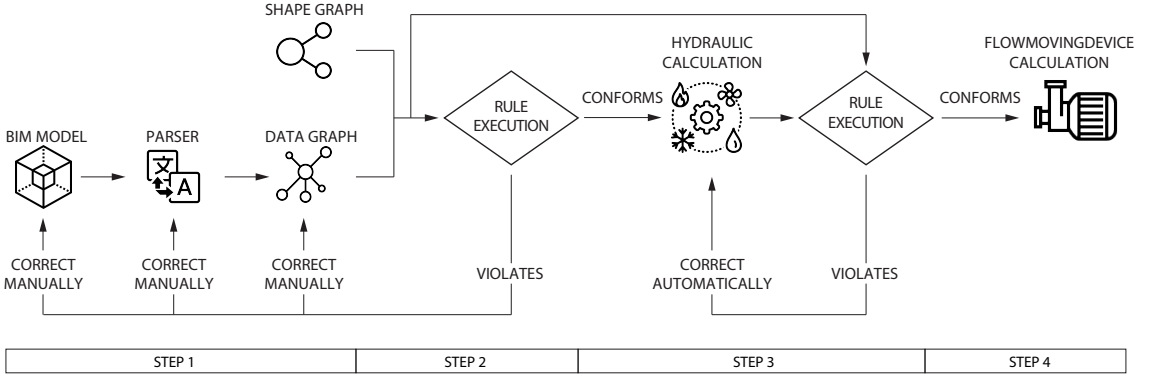


Figure 10: The process of performing conformance checking and design calculations for an HVAC model.

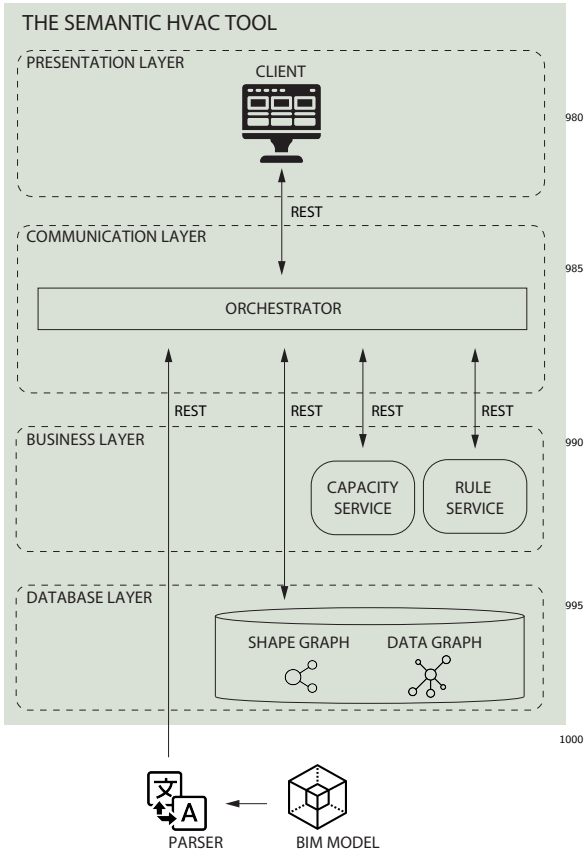


Figure 11: The system architecture of the Semantic HVAC Tool.

3. the rule service sends a rule model expressed in turtle format to the orchestrator.
4. the orchestrator sends the rule model to the database.
5. as the database already stores the data graph, it performs the rule execution and sends the conformance checking results expressed in JSON-LD to the orchestrator.
6. the orchestrator sends the conformance checking results to the client.
7. the client displays the conformance checking results in two tables.

Similar to the conformance checking, the orchestrator handles communication between the different services when performing hydraulic- and flow-moving device calculations.

5.1.3. Business layer

The business logic is spread over multiple microservices in the web application. We have divided our logic into two microservices: the capacity service and the rule service, as shown in Figure 11. Rule logic is handled by the rule service, while the capacity service handles HVAC design logic. The rule service consists of two functions. When requested, the first function provides a shape graph in turtle format, while the second function performs an automatic conformance check and produces a validation report in JSON-LD format.

The capacity service has one function. When requested, it performs a hydraulic calculation and delivers the pressure drop result for each distribution component, which is of type `fso:Pipe`, `fso:Duct`, `fso:Elbow`, `fso:Transition` and `fso:Tee`. The output of the function is expressed in

JSON-LD format. Both microservices are developed separately in FastAPI. To perform hydraulic calculations, we use the fluids library [70].

5.1.4. Database layer

The database layer consists of a Jena Fuseki server [71] that stores RDF data. The microservices in the business layer share the same database to access information from different domains easily. Jena Fuseki has SPARQL, SHACL, and Update endpoints. The SPARQL endpoint retrieves data, while the Update endpoint inserts, deletes or updates data.

For example, when the user initiates the flow-moving device calculation, the client requests a list of flow-moving devices from the orchestrator. The orchestrator then requests three SPARQL queries¹⁰. The first SPARQL query is illustrated in Appendix B and is able to sum the pressure drops of the critical branch to determine the necessary pressure of each `fso:Pump` represented in the data graph. The second SPARQL query performs the same calculation for every `fso:Fan`, while the third query calculates the total flow rate of each flow-moving device. Once the orchestrator hits the SPARQL endpoint in the Jena Fuseki Server with the SPARQL queries, it retrieves the results and sends them to the client to be displayed in the flow-moving device table.

5.1.5. Parsing the BIM model

The parser¹¹ and the BIM model¹² are not part of the Semantic HVAC Tool. The parser is developed as a .NET Framework (C-Sharp) plugin in Revit [72], using the Revit API, while the BIM model is developed as a BIM model in Revit. The parser has two functions; the first function serializes Revit BIM objects into a data graph expressed in turtle syntax and, while the second sends the data graph to the orchestrator via an HTTP request. The orchestrator then redirects the data graph to the database for storage.

5.2. Results

To showcase the tool in use, we used a BIM model of a real-world building located in Sorø, Denmark. The building is a primary school constructed in 2017 and named Frederiksberg Skole. Frederiksberg Skole has a gross floor area of 6970 m² and is divided into a northern building and southern building. Each building has three floor levels, as shown in Figure 12. The original BIM model has been modified by Seeberg and Tangeraas [73] to include only the northern building and its heating and ventilation system. It has 86 rooms, each heated with radiators and ventilated with supply and extract air terminals. Both systems are located in the basement of the northern

building. The results of parsing Frederiksberg Skole as a data model, performing two conformance checks, calculating the hydraulics and designing flow-moving devices with the Semantic HVAC tool are presented in this section.

5.2.1. Parsing the data model

The process of serializing Frederiksberg Skole from Revit to the Semantic HVAC Tool took 17.1 seconds to complete. Moreover it took the Semantic HVAC Tool 8.3 seconds to store the data model of 369054 triples in the database. The triples are also made available on GitHub¹³. Since FSO represent HVAC components, we can extract the sum of components by type. Table 4 shows that the data model consists of 6137 HVAC components, 36 HVAC systems and 65851 HVAC size- and capacity-related properties. In total, the data model consists of 84887 instances.

Table 4: The table shows the amount of HVAC components, systems and size- and capacity-related properties in the data model

Type	Amount
fso:EnergyConversionDevice	1
fso:Segment	2766
fso:Fitting	2912
fso:FlowMovingDevice	3
fso:FlowController	85
fso:Terminal	370
fso:System	36
fso:Port	12827
fso:Flow	36
fpo:Property	65851
total	84887

5.2.2. Conformance checking Frederiksberg Skole

The process of validating the data model against the rule model took 3.1 seconds to complete. Table 5 shows the results of the first conformance check. For example, Table 5 shows that instances of type `fso:System` in the data model have violated the constraints 32 times. The HVAC rule model is also violated by instances of type `fso:Duct`, `fso:SpaceHeater`, `fso:Port`, and `fpo:Property`. The total amount of violations is 372. Since the data graph contains 84887 instances this means that approximately 0.5% of the components are violating the HVAC rule model. We can also observe, that the majority of violations are caused by instances of type `fso:Port`, which accounts for approx. 73% of the total violating instances.

We can access Table 6 in the client by clicking on `fso:System` in the first conformance checking table. Table 6 lists the violation details for `fso:System`. The GUI displays all 32 violations, but Table 6 is limited to the first two violations, indicating that instance `inst:5eb8aa6a...`

¹⁰<https://github.com/Semantic-HVAC-Tool/Orchestrator-Service/tree/main/public/Queries>

¹¹<https://github.com/Semantic-HVAC-Tool/Parser>

¹²<https://github.com/Semantic-HVAC-Tool/Other/blob/main/BIM-Model.rvt>

¹³<https://github.com/Semantic-HVAC-Tool/Other/blob/main/Data-Model.ttl>



Figure 12: The illustration shows the floor plans of Frederiksberg Skole in Sorø, Denmark. The south building is marked with red, while the north building is marked with blue [73]

Table 5: Results of the first conformance check, showing the number of violations, based on HVAC component type, HVAC system and size- and capacity-related properties

Type	Amount
fso:HeatExchanger	0
fso:Pipe	2
fso:Duct	2
fso:Elbow	0
fso:Transition	0
fso:Tee	0
fso:Fan	0
fso:Pump	0
fso:AirTerminal	0
fso:SpaceHeater	3
fso:Damper	0
fso:Valve	0
fso:System	32
fso:Port	251
fso:Flow	0
fpo:Property	82
Total	372

violates the SHACL constraint type `sh:MinCountConstraintComponent` and throws the message “A return system must contain at least one component”.

Table 6: Results of the first conformance check, showing the first two results of `fso:System` violations in details

ID	Constraint type	Description
inst:5eb8aa6a-0ed0-4fea-b226-dd7fa9ae035e-0019ec8a	sh:MinCountConstraintComponent	A return system must have at least one component
inst:98e9914f-25c6-4c43-a0fb-912eba89c13d-0019dbff	sh:MinCountConstraintComponent	A supply system must have at least one component

All 32 violations were corrected in the data graph by performing the SPARQL update query shown in Appendix C directly in the Jena Fuseki Server. The query deletes all `fso:SupplySystem` and `fso:ReturnSystem` instances that lack the predicate `fso:hasComponent`.

The remaining violations were corrected manually in the BIM model, parser, and data graph, which results in an empty validation table. A blank validation table at this stage indicates that the data graph conforms, and we have completed step 2 of the process illustrated in Figure 10.

5.2.3. Hydraulic calculation and second conformance check

Performing the hydraulic calculation on Frederiksberg Skole took 5.4 seconds. The violation results of the second conformance check are shown in Table 7. It can be seen that instances of `fso:Pipe` are violating the HVAC rule

model 14 times, and the total number of violations in step 3 of the process illustrated in Figure 10 is 14.

Table 7: Results of performing the second conformance check, showing the amount of violations, when the hydraulic results are added to the data graph

Type	Amount
fso:HeatExchanger	0
fso:Pipe	14
fso:Duct	0
fso:Elbow	0
fso:Transition	0
fso:Tee	0
fso:Fan	0
fso:Pump	0
fso:AirTerminal	0
fso:SpaceHeater	0
fso:Damper	0
fso:Valve	0
fso:System	0
fso:Port	0
fso:Flow	0
fpo:Property	0
Total	14

Clicking on **fso:Pipe** in the first conformance checking table in the client will display Table 8. The table displays the violation details within the category of **fso:Pipe**. While the GUI of the Semantic HVAC Tool displays the violation details of all 14 violations, Table 8 is limited to the first two violations. The first result indicates that the instance **inst:745522df...** is violating the SHACL constraint type **sh:SPARQLConstraintComponent**. The message it throws indicates that the pressure drop of the **fpo:Pipe** instance is exceeding 100 Pa/m.

Table 8: Results of the second conformance check, displaying the first two results of fso:Pipe violations in detail after running the hydraulic calculation

ID	Constraint type	Description
inst:745522df-9a78-4732-8b22-f56765e86201-002bec43	sh:SPARQL-Constraint-Component	The pressure drop of a pipe should not exceed 100 Pa/m
inst:745522df-9a78-4732-8b22-f56765e86201-002bec25	sh:SPARQL-Constraint-Component	The pressure drop of a pipe should not exceed 100 Pa/m

In the GUI the user can implement the correction of all 14 violations automatically. If the corrections are implemented, the violations will be removed from Table 7, and the total number of violations will be decreased to 0.

The violations at this stage were corrected automatically in this way, which resulted in an empty validation table. A blank validation table at this stage indicates that the data graph conforms, and we have completed step 3 of the process illustrated in Figure 10.

5.2.4. Flow-moving device capacity calculation and second validation

Since we have performed the rule execution and hydraulic calculation, we are now ready to calculate the capacity of each flow-moving device represented in the data graph. The results of the flow-moving device calculation are shown in Table 9. It took 87 seconds to calculate the total amount of flow rate and pressure for each flow-moving device using three SPARQL queries and to display the results in the flow-moving device table. Two fans and one pump are shown in Table 9 as flow-moving devices. Table 9 provides the component ID, flow rate, and pressure for each **fso:Fan** and **fso:Pump**. For example, it shows that the instance **inst:0fc738e3...** of type **fso:Pump** has a total flow rate of 0.84 L/s and a total pressure of 16867 pascal. The fan pressure includes the ductwork, air terminal, and AHU pressure drop. Using this information, correctly sized fans and pumps can be selected from manufacturers product catalogues.

Table 9: Flow-moving device results showing the type of each flow-moving device, its component ID, flow rate and pressure.

Type	Component ID	Flow rate [L/s]	Pressure [Pa]
fso:Fan	inst:36aec977-8efa-403c-b1e6-3b29521aac43-002f6bf5	7943	724
fso:Fan	inst:f4ad7dcb-2875-4fe5-be51-f41510b75979-002f583e	8124	822
fso:Pump	inst:0fc738e3-3eb1-4344-b913-b3883e4083b0-0033212a	0.84	16867

6. Discussion

This section describes the achievements, limitations, and future work.

6.1. Achievements

This paper shows how an ontology can be extended, constructed and aligned from scratch to represent the capacity and size-related properties of HVAC systems and their components. We also demonstrated how separate and lightweight ontologies such as BOT, FSO and FPO can be interconnected to represent the building, its services and their relationships in a modular way. Moreover, we

developed a set of constraints to increase the data quality of BIM models within the HVAC domain. We developed the Semantic HVAC tool and applied it to a real-world building to demonstrate the feasibility of expressing and conforming an HVAC model. We have created a reliable data model to perform hydraulic calculations and designing the capacity of flow-moving devices. Considering the time spent on conformance checking, (re-)sizing and quality control in the industry, this study implements technical solutions and demonstrates a path towards better data quality in BIM models, time savings due to computerization and increased transparency.

6.2. Limitations

6.2.1. Logical complexity

Schwabe et al. [33], Oraskari et al. [58], and Hagedorn and König [56] applied a reasoner to perform an automatic rule check. In the same way, we used a SHACL inference rule to automatically increase the diameter of a pipe when the pressure in the pipe exceeded 100 Pa/m. Although the SHACL component could generate the new data, it could not delete the old data. SHACL inferencing rules can only infer new knowledge. We implemented a separate SPARQL query in the Semantic HVAC Tool to delete the existing data after the SHACL inferencing rule was performed. In any web tool, spreading logic this way will increase its logical complexity.

6.2.2. Query efficiency

The rule execution is performing well since it took only 3.1 seconds to validate The HVAC rule model consisting of 36 shapes and 122 constraints against Frederiksberg Skole with 369054 triples. In contrast, it took 87 seconds to calculate the total pressure and flow rate of each flow-moving device, represented in the data graph using three SPARQL queries. Two of the SPARQL queries have a Filter Not Exists statement, which is responsible for the slow query performance. Using the Filter Not Exists statement, we iterate through all HVAC components in the graph and return only those with ports that belong to the same HVAC system. Iterating through all HVAC components and their ports slows down the query efficiency. This could be improved by replacing the Filter Not Exists statement.

6.2.3. Abstraction level of HVAC components

FSO is limited to eight high-level HVAC components and 19 medium-level HVAC components. In practice, it is possible to subdivide FSO further. For example, a pump can be subdivided into centrifugal pumps, positive displacement pumps, etc. There are also several levels of centrifugal pumps. To retain FSO as a lightweight ontology we did not nest further.

6.2.4. Geometry-based constraints

The data graph and shape graph we developed in our research do not represent HVAC component geometry and

its geometry-related properties nor validate geometry-based constraints, such as separation distances between HVAC components and components from other domains or service distances, such as structural components. The delivery of BIM models with incorrect separation and service distances between HVAC components from the design phase to the construction phase is a common problem affecting a building project's economy and schedule and should therefore be a focal point in further development.

6.3. Future work

The proposal for future work in this paper can be divided into three steps.

A literature review of geometry-related ontologies should be conducted first. If a sufficient geometry-related ontology doesn't exist, an existing one should be extended, or a new one should be developed to describe the geometry and the relation between geometries.

Secondly, to represent separation and service distances for HVAC components, the geometry-related ontology should be interconnected with BOT, FSO, and FPO.

Lastly, a set of geometry-based constraints should be added to the HVAC rule model and validated against the data graph.

7. Conclusions

This paper presents a demonstration environment to represent and validate the composition of HVAC components, their systems, and their capacity and size-related properties using semantic web technologies. This paper aimed to:

1. Extend FSO to support an alignment with the proposed FPO ontology.
2. Propose the FPO ontology to represent HVAC components' capacity and size-related properties.
3. Propose a rule model for the HVAC domain.
4. Produce a demonstration environment to show the conformance of an HVAC model.
5. Use the demonstration environment to show how FPO and the HVAC rule model can support the description and validation of hydraulics in HVAC components and the capacity of HVAC components.

We extended FSO with three classes and four properties related to the connectivity between ports and fluids. This made it possible to describe the relationship between HVAC components, their flow ports and the fluid being transported in three ways and aligned with FPO. We also extended FSO to represent 19-medium level component types. We developed FPO to represent the size- and capacity-related properties of HVAC components. FPO

has a Description Logic expressivity of $\mathcal{ALRF}(\mathcal{D})$ and contains 50 classes, 50 object properties and 6 data properties¹²²⁰

Moreover, we developed an HVAC rule model that restricts the composition of HVAC components, their systems, and their size- and capacity-related properties. The rule model consists of 36 shapes and 122 constraints.¹³²⁵

A four-step process and the Semantic HVAC Tool were developed to demonstrate how a real-world building model can be represented, validated, and used to compute hydraulic calculations and design the capacity of a flow-moving device. Frederiksberg Skole consists of 369054 triples and was used as the real-world building model. We managed to perform conformance checking twice. The first rule execution resulted in 372 constraint violations, and the second¹³³⁰ resulted in 14 constraint violations. These rule violations were fixed both manually and automatically. Finally, using the conformed model, we performed hydraulic calculations and used the results to design the capacity of two fans and a pump, which were represented in the real-world building model.¹³³⁵

8. Acknowledgements

This work was supported by EU-Interreg ÖKS “Data-driven Energy Management in Public Buildings”; the Innovation Fund Denmark (grant 9065-00266A); the Ram-boll Foundation; and COWI A/S. We thank Sorø municipality for providing the BIM model for Frederiksberg Skole.¹³⁵⁵

References

- [1] M. Niknam, S. Karshenas, A shared ontology approach to semantic representation of BIM data, *Automation in Construction* (2017). doi:10.1016/j.autcon.2017.03.013.¹³⁶⁰
- [2] R. Sacks, C. Eastman, G. Lee, P. Teicholz, *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers*, 2018.
- [3] J. M. Werbrouck, M. Senthilvel, J. Beetz, *Querying Heterogeneous Linked Building Data with Context-expanded GraphQL Queries*, Tech. rep. URL <https://www.w3.org/TR/sparql11-query/>¹³⁶⁵
- [4] A.-H. Hamdan, M. Bonduel, R. J. Scherer, An ontological model for the representation of damage to constructions, Tech. rep. URL <http://www.w3.org/1999/02/22-rdf-syntax-ns#>¹³⁷⁰
- [5] I. Esnaola-Gonzalez, F. J. Diez, Integrating Building and IoT data in Demand Response solutions, Tech. rep. URL <http://project-respond.eu>
- [6] M. H. Rasmussen, M. Lefrançois, P. Pauwels, C. A. Hviid, J. Karlshøj, Managing interrelated project information in AEC Knowledge Graphs, *Automation in Construction* (2019). doi: 10.1016/j.autcon.2019.102956.¹³⁷⁵
- [7] A. Hogan, *The Web of Data*, 2020. doi:10.1007/978-3-030-51580-5.¹³⁸⁰
- [8] J. Flore, T. Djuedja, Integration of environmental data in BIM tool & Linked Building Data, Tech. rep. URL <http://www.enit.fr>¹³⁸⁵
- [9] S. Stolk, K. McGlinn, Validation of ifcowl datasets using shacl, Vol. 2636, 2020.
- [10] P. Pauwels, D. V. Deursen, R. Verstraeten, J. D. Roo, R. D. Meyer, R. V. D. Walle, J. V. Campenhout, A semantic rule checking environment for building performance checking, *Automation in Construction* 20 (2011). doi:10.1016/j.autcon.2010.11.017.¹³⁹⁰
- [11] J. Lee, Y. Jeong, User-centric knowledge representations based on ontology for AEC design collaboration, *Computer-Aided Design* 44 (2012) 735–748. doi:10.1016/j.cad.2012.03.011. URL www.elsevier.com/locate/cad
- [12] J. F. Tchouanguem Djuedja, F. H. Abanda, B. Kamsu-Foguem, P. Pauwels, C. Magniont, M. H. Karay, An integrated Linked Building Data system: AEC industry case, *Advances in Engineering Software* 152 (feb 2021). doi:10.1016/j.advengsoft.2020.102930.
- [13] M. H. Rasmussen, M. Lefrançois, G. F. Schneider, P. Pauwels, Bot: The building topology ontology of the w3c linked building data group, *Semantic Web* 12 (1) (2020) 143–161. doi:10.3233/SW-200385.
- [14] V. Kukkonen, A. Küçükavci, M. Seidenschmur, M. H. Rasmussen, K. M. Smith, C. A. Hviid, An ontology to support flow system descriptions from design to operation of buildings, *Automation in Construction* 134 (December 2021) (2022) 104067. doi:10.1016/j.autcon.2021.104067. URL <https://doi.org/10.1016/j.autcon.2021.104067>
- [15] N. Pauen, D. Schlütter, J. Siwiecki, J. Frisch, C. van Treeck, Integrated representation of building service systems: topology extraction and tubes ontology, *Bauphysik* 42 (6) (2020) 299–305. doi:10.1002/bapi.202000027.
- [16] M. Bonduel, Towards a props ontology (2018), URL: <https://github.com/w3c-lbdcg/lbd/blob/gh-pages/presentations/props/presentation/LBDCall20180312>.
- [17] A. Wagner, W. Sprenger, C. Maurer, T. E. Kuhn, U. Rüppel, Building product ontology: Core ontology for linked building product data, *Automation in Construction* 133 (2022) 103927. doi:10.1016/j.autcon.2021.103927.
- [18] N. Pauen, D. Schlütter, J. Siwiecki, J. Frisch, C. van Treeck, Integrated representation of building service systems: topology extraction and TUBES ontology, *Bauphysik* 42 (6) (2020) 299–305. doi:10.1002/bapi.202000027.
- [19] E. van den Bersselaar, J. Heinen, M. Chaudron, P. Pauwels, Automatic validation of technical requirements for a bim model using semantic web technologies, 2022, 1st 4TU/14USA research day on Digitalization in the Built Environment ; Conference date: 01-04-2022.
- [20] A. S. Ismail, K. N. Ali, N. A. Iahad, A review on bim-based automated code compliance checking system, in: 2017 International Conference on Research and Innovation in Information Systems (ICRIIS), 2017, pp. 1–6. doi:10.1109/ICRIIS.2017.8002486.
- [21] R. Ren, J. Zhang, Model information checking to support interoperable bim usage in structural analysis, *ASCE International Conference on Computing in Civil Engineering* 2019doi: 10.1061/9780784482421.046. URL <https://par.nsf.gov/biblio/10104661>
- [22] A. T. Kovacs, A. Micsik, Bim quality control based on requirement linked data, *International Journal of Architectural Computing* 19 (3) (2021) 431–448. doi:10.1177/14780771211012175.
- [23] W. Solihin, N. Shaikh, X. Rong, L. K. Poh, Beyond interoperability of building model: a case for code compliance, *Carnegie Mellon University (CMU)*, 2004. URL <https://www.researchgate.net/publication/280598933BEYOND>
- [24] E. Hjelseth, N. Nisbet, Capturing normative constraints by use of the semantic mark-up rase methodology, *Proceedings of CIB* (2011).
- [25] T. H. Beach, T. Kasim, H. Li, N. Nisbet, Y. Rezgui, Towards automated compliance checking in the construction industry, Vol. 8055 LNCS, 2013. doi:10.1007/978-3-642-40285-2_32.
- [26] J. K. Lee, C. M. Eastman, Y. C. Lee, Implementation of a bim domain-specific language for the building environment rule and analysis, *Journal of Intelligent and Robotic Systems: Theory and Applications* 79 (2015). doi:10.1007/s10846-014-0117-7.
- [27] J. Dimyadi, W. Solihin, W. Solihin, C. Eastman, A knowledge representation approach in bim rule requirement analysis using the conceptual graph, *Journal of Information Technology in*

- Construction 21 (2016).
- [28] J. Dimyadi, P. Pauwels, R. Amor, Modelling and accessing regulatory knowledge for computer-assisted compliance audit, *Journal of Information Technology in Construction* 21 (2016).
- [29] J. Dimyadi, C. Clifton, M. Spearpoint, R. Amor, Computerizing regulatory knowledge for building engineering design, *Journal of Computing in Civil Engineering* 30 (2016). doi:10.1061/(asce)cp.1943-5487.0000572.
- [30] T. Chipman, T. Liebich, M. Weise, mvdxml specification 1.1, specification of a standardized format to define and exchange model view definitions with exchange requirements and validation rules. by model support group (msg) of buildingsmart, BuildingSMART 1 (2016).
- [31] S. Park, Y. C. Lee, J. K. Lee, Definition of a domain-specific language for korean building act sentences as an explicit computable form, Vol. 21, 2016.
- [32] G. Governatori, M. Hashmi, H. P. Lam, S. Villata, M. Palmirani, Semantic business process regulatory compliance checking using legalruleml, Vol. 10024 LNAI, 2016. doi:10.1007/978-3-319-49004-5_48.
- [33] K. Schwabe, J. Teizer, M. König, Applying rule-based model-checking to construction site layout planning tasks, *Automation in Construction* 97 (2019). doi:10.1016/j.autcon.2018.10.012.
- [34] G. Lee, J. Jeong, J. Won, C. Cho, S. joon You, S. Ham, H. Kang, Query performance of the ifc model server using an object-relational database approach and a traditional relational database approach, *Journal of Computing in Civil Engineering* 28 (2014). doi:10.1061/(asce)cp.1943-5487.0000256.
- [35] W. Solihin, J. Dimyadi, Y.-C. Lee, C. Eastman, R. Amor, The critical role of accessible data for bim-based automated rule checking systems, 2017. doi:10.24928/jc3-2017/0161.
- [36] R. K. Soman, M. Molina-Solana, J. K. Whyte, Linked-data based constraint-checking (ldcc) to support look-ahead planning in construction, *Automation in Construction* 120 (2020) 103369. doi:10.1016/j.autcon.2020.103369.
- [37] J. Oraskari, M. Senthilvel, J. Beetz, M. Senthilvel, J. Beetz, SHACL is for LBD what mvdxML is for IFC, *Proceedings of the 38th International Conference of CIB W78 (October)* (2021) 693–702.
- URL <https://www.cibw78-ldac-2021.lu/>
- [38] J. Beetz, J. Van Leeuwen, B. De Vries, IfcOWL: A case of transforming EXPRESS schemas into ontologies, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM 23* (1) (2009). doi:10.1017/S0890060409000122.
- [39] W. Terkaj, A. Šojić, Ontology-based representation of IFC EXPRESS rules: An enhancement of the ifcOWL ontology, *Automation in Construction* 57 (2015). doi:10.1016/j.autcon.2015.04.010.
- [40] D. A. Koonce, R. P. Judd, A visual modelling language for express schema, *International Journal of Computer Integrated Manufacturing* 14 (5) (2001) 457–472. doi:10.1080/09511920010022495.
- [41] K. Afari, C. M. Eastman, D. Castro-Lacouture, Javascript object notation (json) data serialization for ifc schema in web-based bim data exchange, *Automation in Construction* 77 (2017) 24–51. doi:https://doi.org/10.1016/j.autcon.2017.01.011.
- URL <https://www.sciencedirect.com/science/article/pii/S0926580517300316>
- [42] P. Pauwels, S. Zhang, Y.-C. Lee, Semantic web technologies in aec industry: A literature overview, *Automation in Construction* 73 (2017) 145–165. doi:https://doi.org/10.1016/j.autcon.2016.10.003.
- URL <https://www.sciencedirect.com/science/article/pii/S0926580516302928>
- [43] M. H. Rasmussen, P. Pauwels, C. A. Hviid, J. Karlshøj, Proposing a Central AEC Ontology That Allows for Domain Specific Extensions, 2017. doi:10.24928/jc3-2017/0153.
- [44] M. Lefrançois, J. Kalaoja, T. Ghariani, A. Zimmermann, T. Seas, D2 . 2 SEAS Knowledge Model, *Tech. Rep.* December (2014).
- [45] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Bergés, D. Culler, R. K. Gupta, M. B. Kjærgaard, M. Srivastava, K. Whitehouse, Brick: Metadata schema for portable smart building applications, *Applied Energy* 226 (September 2017) (2018) 1273–1292. doi:10.1016/j.apenergy.2018.02.091.
- URL <https://doi.org/10.1016/j.apenergy.2018.02.091>
- [46] L. Daniele, F. den Hartog, J. Roes, Created in Close Interaction with the Industry: The Smart Appliances REference (SAREF) Ontology, in: *Lecture Notes in Business Information Processing*, Vol. 225, 2015. doi:10.1007/978-3-319-21545-7_9.
- [47] P. Pauwels, A. Costin, M. H. Rasmussen, Knowledge graphs and linked data for the built environment, *Structural Integrity* 20 (2022) 157–183. doi:10.1007/978-3-030-82430-3_7.
- [48] J. P. Martins, A. Monteiro, Lica: A bim based automated code-checking application for water distribution systems, *Automation in Construction* 29 (2013). doi:10.1016/j.autcon.2012.08.008.
- [49] W. Solihin, C. Eastman, Classification of rules for automated bim rule checking development, *Automation in Construction* 53 (2015). doi:10.1016/j.autcon.2015.03.003.
- [50] C. Eastman, J. min Lee, Y. suk Jeong, J. kook Lee, Automatic rule-based checking of building designs (2009). doi:10.1016/j.autcon.2009.07.002.
- [51] W3C, Swrl: A semantic web rule language combining owl and ruleml (5 2004).
- URL <https://www.w3.org/Submission/SWRL/>
- [52] S. Mehla, S. Jain, Rule languages for the semantic web, Vol. 755, 2019. doi:10.1007/978-981-13-1951-8_73.
- [53] W3C, Rif overview (second edition) (2 2013).
- URL <https://www.w3.org/TR/rif-overview/>
- [54] W3C, Notation3 (n3): A readable rdf syntax (3 2011).
- URL <https://www.w3.org/TeamSubmission/n3/>
- [55] W3C, Spin - overview and motivation (2 2011).
- URL <https://www.w3.org/Submission/spin-overview/>
- [56] P. Hagedorn, M. König, Rule-based semantic validation for standardized linked building models (2021). doi:10.1007/978-3-030-51295-8_53.
- [57] W3C, Shapes constraint language (shacl) (7 2017).
- URL <https://www.w3.org/TR/shacl/>
- [58] J. Oraskari, J. Beetz, M. Senthilvel, Shacl is for lbd what mvdxml is for ifc (10 2021).
- URL <https://github.com/w3c-lbd-cg/opm>
- [59] Description Logic Expressivity.
- URL <http://protegeproject.github.io/protege/views/ontology-metrics/>
- [60] M. Debellis, A practical guide to building owl ontologies using protégé 5.5 and plugins (04 2021).
- [61] P. Pauwels, Buildings and Semantics : Data Models and Web Technologies for the Built Environment, Buildings and Semantics, Taylor Francis Group, 2022. doi:10.1201/9781003204381.
- [62] J. Euzenat, P. Shvaiko, *Ontology matching*, 2nd Edition, 2013.
- [63] G. F. Schneider, Towards aligning domain ontologies with the building topology ontology, *Proceedings of the 5th Linked Data in Architecture and Construction Workshop (LDAC 2017)* (2017).
- [64] W. Terkaj, G. F. Schneider, P. Pauwels, Reusing domain ontologies in linked building data: The case of building automation and control, Vol. 2050, 2017.
- [65] D. Mavropapadidis, K. Katsigarakis, P. Pauwels, E. Petrova, I. Korolija, D. Rovas, A linked-data paradigm for the integration of static and dynamic building data in digital twins, 2021. doi:10.1145/3486611.3491125.
- [66] L. Bindra, K. Eng, O. Ardukanian, E. Stroulia, Flexible, decentralised access control for smart buildings with smart contracts, *Cyber-Physical Systems* (2021). doi:10.1080/23335777.2021.1922502.
- [67] React, a JavaScript library for building user interfaces.
- URL <https://github.com/reactjs/reactjs.org>

- [68] Fast, unopinionated, minimalist web framework for node.
URL <https://github.com/expressjs/express>
- [69] Node.js is an open-source, cross-platform, JavaScript runtime environment.
URL <https://github.com/nodejs/node>
- [70] FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.
URL <https://github.com/tiangolo/fastapi>
- [71] Apache Jena Fuseki is a SPARQL server.
URL <https://github.com/apache/jena/blob/main/jena-fuseki2/apache-jena-fuseki/fuseki-server>
- [72] Revit: BIM software for designers, builders, and doers.
URL <https://www.autodesk.eu/products/revit>
- [73] F. Seeberg, J. Tangeraas, Integration of Thermal Building Simulation Tools and Cloud-Based Building Information Models (2022).

Appendix A. Mapping between Flow Properties: Ontology (FPO) and Brick

Table A.10: Alignments between FPO and Brick.

owl:Class	rdfs:subClassOf
owl:ObjectProperty	rdfs:subPropertyOf
	owl:equivalentClass
fpo:hasDesignCoolingPower	brick:coolingCapacity
fpo:hasVolume	brick:volume

Appendix B. Querying fso:Pump pressure

```

SELECT ?pump (MAX(?sumOfSupplyPressureDrop +
↪ ?sumOfReturnPressureDrop +
↪ ?terminalPressureDropValue) AS ?pressure)
WHERE {
  {
    SELECT ?pump ?terminal ( SUM(?supplyValue) AS
↪ ?sumOfSupplyPressureDrop)
    WHERE {
      ?pump a fso:Pump .
      VALUES ?terminalType {fso:HeatExchanger
↪ fso:SpaceHeater}
      ?terminal a ?terminalType .
      ?supplySystem fso:hasComponent ?pump .
      ?supplyComponent fso:feedsFluidTo+ ?terminal .
      ?supplySystem fso:hasComponent ?supplyComponent .
      ?supplySystem a fso:SupplySystem .
      ?supplyComponent fso:hasPort ?supplyPort .
      ?supplyPort fpo:hasFlowDirection ?flowDirection .
      ?flowDirection fpo:hasValue "Out" .
      ?supplyPort fpo:hasPressureDrop ?pressureDrop .
      ?pressureDrop fpo:hasValue ?supplyValue .
      FILTER NOT EXISTS {
        ?supplyPort fso:suppliesFluidTo ?connectedPort .
        ?connectedComponent fso:hasPort ?connectedPort .
        ?connectedComponent fso:feedsFluidTo+ ?terminal .
        ?connectedComponent a fso:Tee .
      }} GROUP BY ?pump ?terminal
    }
  }
  SELECT ?pump ?terminal ?terminalPressureDropValue
↪ ?sumOfReturnPressureDrop
  WHERE {
    ?terminal fso:hasPort ?port .
    ?port fso:returnsFluidTo ?anotherPort .
    ?port fpo:hasPressureDrop ?pressureDrop .
    ?pressureDrop fpo:hasValue
↪ ?terminalPressureDropValue .
    {
      SELECT ?pump ?terminal ( SUM(?returnValue) AS
↪ ?sumOfReturnPressureDrop)
      WHERE {{
        ?pump a fso:Pump .
        VALUES ?terminalType {fso:HeatExchanger
↪ fso:SpaceHeater}
        ?terminal a ?terminalType .
        ?supplySystem fso:hasComponent ?pump .
        ?terminal fso:feedsFluidTo+ ?returnComponent
↪ .
        ?returnSystem fso:hasComponent
↪ ?returnComponent .
        ?returnSystem a fso:ReturnSystem .
        ?returnComponent fso:hasPort ?returnPort .
        ?returnPort fpo:hasFlowDirection
↪ ?flowDirection .
        ?flowDirection fpo:hasValue "Out" .
        ?returnPort fpo:hasPressureDrop ?pressureDrop
↪ .
        ?pressureDrop fpo:hasValue ?returnValue .
      }} GROUP BY ?pump ?terminal
    }}} GROUP BY ?pump
  }

```

Listing 5: A SPARQL query to calculate the pressure of each fso:Pump

Appendix C. Deleting systems, which doesn't have any components

```
1 DELETE {
2   ?system a ?systemType .
3   ?system ?systemPred ?systemObj .
4   ?system fso:hasFlow ?flow .
5   ?flow ?flowPred ?flowObj .
6   ?flow fpo:hasTemperature ?temperature .
7   ?temperature ?tempPred ?tempObj
8 }
9 WHERE {
10  VALUES ?systemType {fso:ReturnSystem fso:SupplySystem}
11  ↪ ?system a ?systemType .
12  ?system ?systemPred ?systemObj .
13  ?system fso:hasFlow ?flow .
14  ?flow ?flowPred ?flowObj .
15  ?flow fpo:hasTemperature ?temperature .
16  ?temperature ?tempPred ?tempObj
17  FILTER NOT EXISTS {?system fso:hasComponent ?component}
18  ↪ .
19 }
```

Listing 6: A SPARQL update query to remove all `fpo:SupplySystem` and `fpo:ReturnSystem`, which is missing the predicate `fso:hasComponent` from the data model

6.7 Paper VII - A Web-based Common Data Environment for Continuous Commissioning of buildings

A Web-based Common Data Environment for Continuous Commissioning of buildings

Mikki Seidenschnur^{1,2}, Ali Küçükavci², Kevin Michael Smith², Christian Anker Hviid²

¹Ramboll, Copenhagen, Denmark

²Technical University of Denmark, Kongens Lyngby, Denmark

Abstract

With ever increasing demands for high performance buildings in the Architecture, Engineering, Construction, and Operation (AECO) industry, a transition towards Common Data Environments (CDEs) is happening. Such CDEs has been created initially to raise the Building Information Modeling (BIM) maturity to level 3. CDEs have been discussed to be the next step of full digitalization of the AECO industry. It is believed, that by creating a CDE as a Single Source of Truth (SSOT) for the BIM model, the AECO industry can minimize the Energy Performance Gap (EPG) from predicted energy performance to measured energy performance. However, most existing CDEs do not involve the Heating, Ventilation, and Cooling (HVAC) model and are not capable of performing advanced hydraulic simulations based on the BIM model. In this article, we discuss the technological changes that need to be made for CDEs to be widely employed within the HVAC industry. We propose a system architecture based on literature, making it possible to perform continuous commissioning of buildings on a digital platform after the building has been built. By introducing a CDE, BIM models will be stored in a centralized database so that all project stakeholders have access to an SSOT.

Introduction

There is a gap in the predicted energy consumption of buildings to the measured energy consumption, and this is called the EPG by De Wilde (2014). Several reasons have been stated for this gap, one of them being that there is a lack of continuous commissioning (Jradi et al., 2018). It should be the original designer of the HVAC system that should perform it. Therefore, we suggest a framework for a CDE to enable continuous commissioning and therefore minimize the EPG from predicted to measured energy performance. Previous work describes parts of the CDE by the authors of this report. A CDE is defined as a web-based collaboration space for all the stakeholders of a construction project (Preidel et al., 2016; Kirby, 2022; BIM Wiki, 2021). The main idea is that all project stakeholders should be able to access the project from anywhere, using a computer, phone, etc. CDEs have been implemented into parts of the AECO industry, but mainly for the design and construction, on large-scale commercial projects. A CDE provides an SSOT of the entire project so that all stakeholders and designing, constructing, and operating

the building on the same data foundation. However, few efforts have been made to create a CDE for commissioning of building services. One effort was carried out by Jradi et al. (2018), but while presenting a novel idea on how to utilize a CDE for operation, they did not present a system architecture for others to build upon the idea. This article aims to present the foundation of a CDE and which developments is needed to achieve a CDE for the continuous commissioning of building services. The system architecture has been developed in relation to several unpublished (in review) papers by the authors of this report (Seidenschnur et al., 2023; Fjerbæk et al., 2023; Seidenschnur et al., 2023) and published articles (Kukkonen et al., 2022; Fjerbæk et al., 2022).

Objectives

The objectives of this article is to:

1. To identify what a CDE is
2. To present a system architecture that allows for a CDE capable of making continuous commissioning
3. To identify the IT technological developments needed for CDEs to be used for most building projects

Results & Discussion

Figure 6 shows the system architecture of the CDE. The central BIM-database acts as a SSOT. The database is connected with a backend routing module to the microservices that allows for detailed HVAC simulation, energy and indoor climate simulation, and performance monitoring of the building.

BIM Model requirements

The first part of transferring the data from the Revit model is to ensure that the Level of Detail (LOD) of the Revit model is sufficient to have coherent information about the building and its systems. This subsection lists the requirements for what an HVAC model should include to be transferred as a model to a web-based database. According to United-BIM (2021) there are 6 levels of detail. (1) LOD100 is a purely conceptual model; it contains the spaces in the model with the actual area requirements. (2) LOD200 is the approximate geometry, that is represented with generic models with approximate specifications, quantities, size, shape, location, and orientation. (3) LOD300 is usually used as construction documentation. It contains accurate sizes, quantity, location, and orientation. (4) LOD350 is different because it, on top of LOD300,

also supports: interfaces and connections with other building components. (5) LOD400 is a model that is ready to be driven directly into fabrication - it can be handed directly to manufacturers that can supply components based on it. (6) LOD500 is an as-built model. Everything is the same, from the MEP model to the actual building. This means that to be capable of containing the needed information to use the CDE, the BIM model should have an LOD with LOD350 or higher, as the connectivity of the components is necessary. Figure 1 shows an example of an HVAC system modeled to standard LOD350. The figure shows that both the heating, ventilation, and cooling system is modeled with all the components that will be in the system. Furthermore, they are modeled within spaces of a building to provide the connection from system to space.

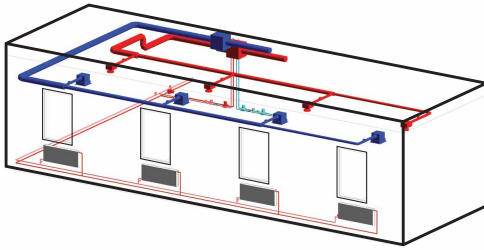


Figure 1: Example of an HVAC system modeled to LOD350, with all components modeled in Revit (unpublished work: Seidenschur et al. (2023))

However, with current modeling standards in companies, such a transformation is not simple to make. While most companies agree to Information and Communications Technologies (ICT) contracts, stating the LOD for certain phases of the construction process, most of them do not follow the contract meticulously, meaning that often connectivity issues occur. Systems are not classified correctly (supply and return system poorly defined in the model), the flow direction of the system is wrong, etc. Figure 2 shows the level of detail needed for modeling if the desire for the 3D model is to be able to make hydraulic simulations in a Modelica environment.

The model shown in 2 does not constitute the current modeling standards for most companies until very late in the design phase. Therefore, to perform hydraulic simulations in Modelica, companies need to invest time in the 3D model to represent the actual flow system.

Class hierarchy for HVAC systems and spaces

Figure 6 shows both a BIM energy model and a BIM HVAC model. Those constitutes two related object models. To parse the two BIM models, the Revit C# Application Programming Interface (API) was used to serialize a JavaScript Object Notation (JSON) object model to represent the energy model and HVAC model. The JSON object model was initialized using a class hierarchy for HVAC and spaces (energy model), as

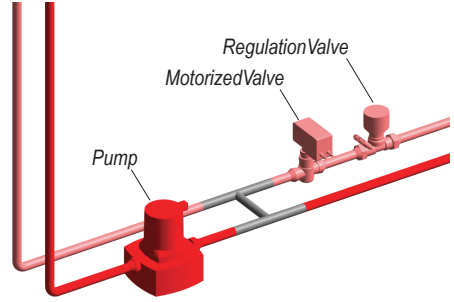


Figure 2: Mixing loop from heating system to ventilation system heating coil, from 3D model shown in 1

seen in Figures 3 and 4. Figure 3 that the container HVACSystem contains a dictionary over the SubSystems. A SubSystem contains a string that describes its type and a list of Components in that SubSystem. A Component acts as a super-class for all possible components in an HVAC system. It contains an Id, Tag, ComponentType, SystemName, SystemType, a list of all components that the component is connected with, and which spaces the component is located in. Below is an example of the C# code that generates the Component class. It has been simplified for readability

```
public class Component
{
    string Id {get;set;}
    string Tag {get;set;}
    string ComponentType {get;set;}
    string SystemName {get;set;}
    string SystemType {get;set;}
    List<Connectors.Connector>
        ConnectedWith {get;set;} =
        new List<Connectors.Connector>();
    List<string>
        ContainedInSpaces {get;set;} =
        new List<string>();

    Component(string id,
               string tag,
               string systemName,
               string systemType)
    {
        Id = id;
        Tag = tag;
        SystemName = systemName;
        SystemType = systemType;
    }
    <Methods not shown for simplicity>
}
```

The Component is a super-class to FlowSegment, FlowTerminal, FlowController, Fitting,

EnergyConversionDevice, and FlowMovingDevice. Below is an example of how the FlowTerminal inherits the properties and methods from the super-class, Component.

```
public class FlowTerminal : Component
{
    FlowTerminal(string id,
                string tag,
                string systemName,
                string systemType)
        : base(id,
              tag,
              systemName,
              systemType)
    {
    }
}
```

The same principle applies to all the components that inherit their properties and methods, as shown in Figure 3. However, some classes add extra information to the specific class, such as a pump. The pump contains information that none of the other components contain, such as a power curve and pressure curve.

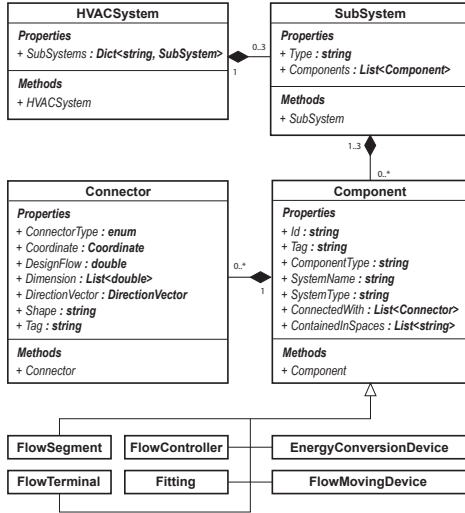


Figure 3: Base classes for a class hierarchy to describe an HVAC system (unpublished work: Seiden-schnur et al. (2023))

Figure 4 shows that the container Spaces contains a list (SpacesInModel) of all spaces in the model. The class Space represents the simple features needed to define a space. It contains an Id, Tag, ThermalZone, and SpaceGeometry. The Id and Tag identify the space with a unique id within Revit. The Thermal-Zone specifies the properties needed to augment the

thermal zone with properties related to indoor climate. The SpaceGeometry describes the geometry of the spaces.

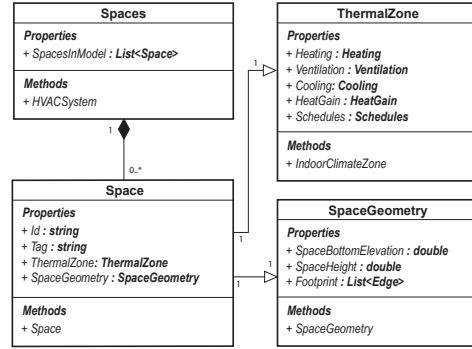


Figure 4: Base classes for a class hierarchy to describe an energy model (unpublished work: Seiden-schnur et al. (2023))

Object model generation

In order to generate the FSC object model and serialize it to JSON, a C# script was written, using the Revit API. The C# script was created with an object-oriented approach. First, the Revit API is used to loop through all of the components within the model. While the components are looped, they are mapped down to the correct component, for instance, a pump. After each component of the HVAC model has been looped through and mapped to a FSC component based on the FSC hierarchy, the HVACSystem is mapped from a list containing all subsystems and components and serialized into a JSON object. The UI of Revit now creates a prompt window, asking under which projectId, UserId, and Uniform Resource Locator (URL) address the JSON should be sent to. It then sends a response to the endpoint with the above URL to post the data within the MongoDB database.

Database infrastructure

The database used for this CDE is called MongoDB. MongoDB is a database following the paradigm of Not only Structured Query Language (NoSQL). NoSQL databases store data non-tabular compared to Structured Query Language (SQL) (relational) databases. The advantage of a NoSQL database is that the schema is more flexible than for an SQL database. Furthermore, a NoSQL database is capable of more easily scaling horizontally rather than vertically. A vertical scale-up means adding extra processing power to the single server but will eventually be limited by the processing power of that server. Horizontal scaling, however, brings in more nodes to distribute the workload of the server - a task that is difficult to perform with relational databases. Finally, MongoDB is compatible with JSON objects, which essentially means that it can take in the JSON that is generated by the C# script from the previous section.

User Interface

Figure 7 shows an example of a user page in CDE. The spaces have been imported from Revit to the central database in the CDE. The top of the figure shows the different types of rooms that the user has generated in the CDE. The bottom shows the rooms that were actually imported from Revit to the database. Each room contains the room number, the name of the room, the function of the room, which level it is located on, what volume the room encloses, the floor area, the type of ventilation, the flow supply, the flow return, and what type of room it is, compared to the mechanical template types, shown in the top of the figure. This makes it possible for the user to select a mechanical template for each room. The engine will automatically calculate the flow supply and return for the given room.

Microservices

With a connection from Revit through an object model to the centralized web-based database in the CDE, the final step is to implement microservices to utilize the object model in the database. Figure 6 shows that the microservices is connected through a routing module within the application. The microservices are created as endpoints in the web framework Flask, which is based on Python, making it easy to develop new microservices. The microservice architecture allows for technologies like Kubernetes to be used for managing containerized workloads and services, making it easy to scale the application to the desired level (horizontal rather than vertical).

The article from Fjerbæk et al. (2022) showed how to transfer an HVAC model from a BIM database into the Modelica-based Dymola environment. In the article, they successfully simulated a small heating system in Modelica and obtained results for the return temperatures of each loop. The setup of this toolchain allowed for easy-to-initiate Modelica simulations of a heating system. This process would, under normal circumstances, be very time-consuming since it is a manual process and the simulations have high complexity. Figure 5 shows the microservice. The microservice receives a post request to perform a Modelica simulation. Step 1 - The FSC object model is translated into a Modelica file (.mo) and stored. Step 2 - The Modelica file (.mo) is simulated, and the results are stored as a Matlab file. Step 3 - the results are read and serialized to a JSON file and then parsed back to the database. Then the results are stored within the original FSC object model for future use.

Conclusion

CDEs will revolutionize the way that the AECO industry works with buildings through the design, construction, and operation phases. If it can be applied to perform continuous commissioning, it has the potential to minimize the EPG from predicted to the measured energy performance of buildings. However, for CDEs to be applicable, the AECO industry must increase the level of BIM maturity from level 2 to level 3. This means that model-

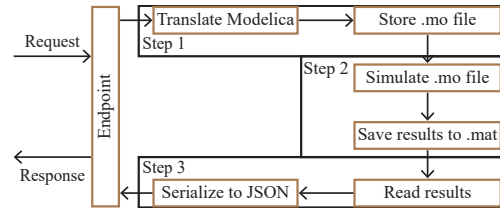


Figure 5: The figure shows the system architecture of the Modelica microservice. (unpublished work: Fjerbæk et al. (2023))

ing standards need to improve in the industry. The CDE centralizes the BIM model into a database with the applications around it instead of keeping external applications in separated silos. In this article, we have shown examples of the technological advances needed to create a CDE based on microservice architecture for continuous commissioning. In previous work by the authors of this article, a microservice for Modelica was created and used to generate automated detailed simulations of building services together with indoor climate parameters. The next step in the CDE shown in this article is to deploy it within a company structure and use it to ultimately reduce the EPG from predicted to the measured energy performance of buildings. For future work, efforts should be made to prove how a CDE for continuous commissioning can reduce the EPG. Finally, the system architecture created for this article has only been tested in a local testbed, which means that future work will focus on the deployment of the application. Finally, this application has been developed by the authors of this article, who have professional careers as HVAC engineers, meaning that most of the concepts have been developed without any user testing. Future work will involve users in further developing features in the project, improving user-friendliness, etc.

Acknowledgment

Funding: This work was funded by the Ramboll Foundation and the Innovation Fund Denmark.

References

- BIM Wiki (2021). Common data environment CDE.
- De Wilde, P. (2014). The gap between predicted and measured energy performance of buildings : A framework for investigation. *Automation in Construction* 41, 40–49.
- Fjerbæk, E. V., M. Seidenschnur, A. Küçükavci, K. Michael, and C. Anker (2022). From BIM databases to Modelica - Automated simulations of heating systems . In *REHVA 14th HVAC World Congress*.
- Fjerbæk, E. V., M. Seidenschnur, A. Küçükavci, K. M. Smith, and C. A. Hviid (2023). FSC2Modelica: Coupling Modelica simulations and a Common Data Environment for BIM.

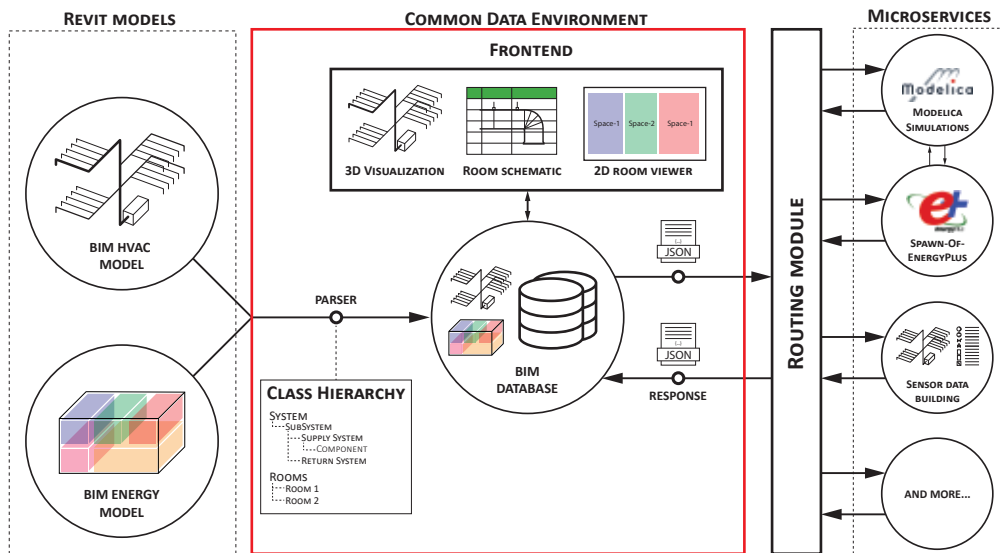


Figure 6: System architecture for the CDE. Data is transferred from the Revit models (left in figure) to the database within the CDE. A frontend is generated (top of figure) to provide insights into the performance of the building. The CDE can now route the information to microservices (right in figure). Among the suggested microservices are Spawn-Of-EnergyPlus with Modelica, a building sensor monitoring module, and potentially many more.

Source Information

Archit information:
DigitalProjektering_Arch
Exported: 2022-02-16T15:46:56 by JEJA

Rooms

ID	Room Type Name:	Ventilation Type:	Flow :
1	Toilet	<input checked="" type="checkbox"/> CAV	<input checked="" type="checkbox"/> Unit Value fix m ³ 36
2	Kontor	<input checked="" type="checkbox"/> VAV	<input checked="" type="checkbox"/> Unit Value m ³ pr. m ² 5
3	Stor Kontor	<input checked="" type="checkbox"/> VAV	<input checked="" type="checkbox"/> Unit Value m ³ pr. m ² 6
4	Kantine	<input checked="" type="checkbox"/> VAV	<input checked="" type="checkbox"/> Unit Value m ³ pr. m ² 8
5	Kokken	<input checked="" type="checkbox"/> VAV	<input checked="" type="checkbox"/> Unit Value m ³ pr. m ² 25
6	Gang	<input checked="" type="checkbox"/> CAV	<input checked="" type="checkbox"/> Unit Value m ³ pr. m ² 5

[ADD TEMPLATE](#)

Rooms

<input type="checkbox"/> Number ↑	Name	ark Function	Level	Volumen [m ³]	Area [m ²]	Vent Type	Flow Supply [m ³]	Flow Return [m ³]	Mech Room Type
101	Vest. 101	Hallway	01 - Entry Level	136.7	38.7	None	683	683	None
102	Lobby 102	Hallway	01 - Entry Level	1190.2	323.6	None	0	0	None

Figure 7: Mock-up of UI for the CDE

- Jradi, M., K. Arendt, F. C. Sangogboye, C. G. Mattera, E. Markoska, M. B. Kjærgaard, C. T. Veje, and B. N. Jørgensen (2018). ObepME: An online building energy performance monitoring and evaluation tool to reduce energy performance gaps. *Energy and Buildings* 166, 196–209.
- Kirby, M. (2022). What Is a Common Data Environment and How Is It Used In Construction?
- Kukkonen, V., A. Küçükavci, M. Seidenschnur, M. H. Rasmussen, K. M. Smith, and C. A. Hviid (2022). An ontology to support flow system descriptions from design to operation of buildings. *Automation in Construction* 134(November 2020), 104067.
- Preidel, C., A. Borrmann, C. Oberender, and M. Tretheway (2016). Seamless integration of common data environment access into BIM authoring applications: The BIM integration framework. *eWork and eBusiness in Architecture, Engineering and Construction*, 119 – 128.
- Seidenschnur, M., A. Küçükavci, E. V. Fjerbæk, K. M. Smith, P. Pauwels, and C. A. Hviid (2023). A Common Data Environment for HVAC Design and Engineering.
- Seidenschnur, M., A. Küçükavci, F. Seeberg, J. M. Tangeraas, E. V. Fjerbæk, P. Pauwels, K. M. Smith, and C. A. Hviid (2023). A web-based approach to close the Energy Performance Gap, using Spawn of EnergyPlus & Modelica in a Common Data Environment.
- United-BIM (2021). A Practical Approach to Level of Detail (LOD).

6.8 Paper VIII - From BIM databases to Modelica - Automated simulations of heating systems

From BIM databases to Modelica - Automated simulations of heating systems

Esben Visby Fjerbæk^a, Mikki Seidenschnur^{a,b}, Ali Küçükavcı^{a,c}, Kevin Michael Smith^a, Christian Anker Hviid^a

^a Department of Civil Engineering, Technical University of Denmark, Kgs. Lyngby, Denmark, evifj@byg.dtu.dk, kevs@byg.dtu.dk, cah@byg.dtu.dk.

^b Rambøll, Copenhagen, Denmark, msei@ramboll.dk

^c Cowi, Kgs. Lyngby, Denmark, alkc@cowi.dk

Abstract. Detailed simulations of HVAC systems play a crucial role in creating 1:1 digital twins of buildings and their systems. In particular, detailed models of hydronic systems are essential for fault detection of building services and control optimization. However, modeling HVAC systems is labour intensive due to the components and connections that one must create based on drawings or models. Creating the HVAC simulation models from BIM data eases the modeling burden, simplifying the creation of digital twins. Straight-forward HVAC simulations can aid the design process. Instead of prescriptive design based on the worst-case conditions, simulations enable performance-based design with partial-loads and dynamic behaviour. This paper presents a preliminary tool using BIM data to create and simulate models of heating systems. The tool uses a central BIM data platform with a dedicated data format – defining components and their relations in a database. Python scripts apply model templates to create heating system models in the Modelica language. The tool simulates the models in Dymola, while Python scripts read and parse the results to the database for visualization and analysis. The tool efficiently simulated a small heating system and obtained results for the return temperatures of several loops.

Keywords. BIM, Modelica, HVAC, simulations

DOI: <https://doi.org/10.34641/clima.2022.365>

1. Introduction

There is a large discrepancy between the estimated energy consumptions of buildings and the one measured, often leading to underestimated energy consumption [1, 2]. The issue, known as the performance gap, has many causes. One significant cause is the precision of building energy performance simulations (BEPS) used to estimate energy consumption. BEPS tools rarely consider HVAC systems in detail [3] or simply assume ideal performance of components and controls [4]. This means that commonly occurring phenomena such as oscillation or system imbalance, that create disturbances, are not identified by the BEPS models.

Modeling HVAC systems in detail ensures that all non-ideal performance is considered in the BEPS simulations, which increases simulation precision. In the design phase, this can help to evolve the design process from a steady-state practice, where the design of HVAC systems is based on a worst-case full-capacity situation, to a dynamic design paradigm, where requirements for part-load conditions and dynamic behavior define the design. Additionally, it will be possible to check that the detail of the design

is sufficient for actual operation under all conditions. Today, this is ensured through guidelines provided by component manufacturers and empirical knowledge of practitioners.

The combination of traditional BEPS models and detailed models of all HVAC systems can act as a digital twin if connected to live weather data. During operation, the digital twin estimates the expected/optimal operation of the building systems alongside the actual building. The expected operation can be used for fault detection and to test different control strategies continuously throughout the building's lifetime.

Modeling HVAC systems can, however, be a laborious task since the systems include many different elements that must be created manually. Often, the simulation models derive from diagrams of the systems, that the simulation engineer manually interprets and translates to the simulation model format. This error-prone process results in two separate models where changes to one does not affect the other.

Generating the HVAC simulation models from BIM

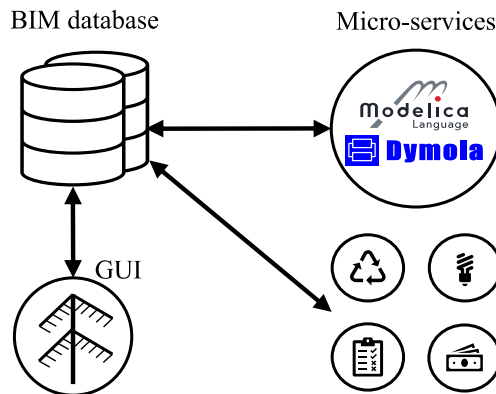


Fig. 1 - Several micro-services in addition to the Modelica service interface the BIM database platform.

data eases the burden of modeling. Integrating BIM and BEPS ensures that the BIM and simulation models share similar information. Both the BIM and BEPS industry work towards linking BIM and BEPS models. Tools such as IESVE [5] and IDA ICE [6] have functionalities to import geometry and construction parameters, whereas the BIM tool Revit contains some BEPS functionalities. Several tools for using BIM as a basis for models in the open-source Modelica language [7] exist [8–10], but all of these, including the traditional BEPS tools, have a primary focus on the envelope and thermal zone model. *IFC2Modelica* [10] includes an example for ventilation systems.

Common for all BIM to BEPS methodologies is the fact that they depend on file-based BIM information. Several critics argue that the use of file-based BIM models limits interoperability [11, 12]. A solution is to transfer from file-based to web-based collaboration, where information is exchanged through open data formats and stored in centralized databases [11, 12]. This corresponds to the BIM level 3 in the Bew-Richards BIM maturity model described in [13]. In the BIM maturity model, the information exchange on levels 0, 1, and 2 has different degrees of standardized data structures. In level 3, the information exchange is handled through standardized, open data formats for integration with various tools.

2. Cloud BIM platform

The toolchain, presented in the following sections, is implemented in a cloud platform that stores BIM models in a database to allow cross-platform access to the models. The platform is built with a micro-service structure, which means that several *micro-services* for design and evaluation of HVAC systems can utilize the data. Amongst these is the Modelica micro-service, which creates models in Modelica language and simulates them with Dymola. As seen in Fig. 1, the data flows back and forth between the

micro-service and the database so that results are read and analyzed in the platform for analysis and visualization in the graphical user interface (GUI).

In the database, components and their relations are defined with the Flow Systems Ontology (FSO) [14, 15]. This ontology uses class hierarchies to define the type of component its relation to other components. E.g., a pipe supplying water to a radiator would have the class *Segment* and have the property *ConnectedWith* equal to the radiator's unique tag. Selected classes relevant to this project are listed in table Tab. 1, whereas the full list of classes and connections can be found online in [15].

Tab. 1 - Selected component classes

FSO class	Examples
Radiator	Radiators for heating
Segment	Pipe or duct segments
FlowController	Valves and dampers
FlowMovingDevice	Pumps and fans
HeatExchanger	Heating coils and heat recovery units

3. Toolchain

The toolchain automatically generates and simulates Modelica models of heating systems from BIM data. In fig. Fig. 2 the main processes of the toolchain are shown along with the flow of data. On the left side, the tool is connected through an API (see Fig. 3), that establishes an integration between the micro-service and the database. As seen in Fig. 3 the JSON format is used to parse data between the database and tool. As seen in Fig. 2 this is maintained throughout the tool, except for the last two steps, where the simulation environment needs a Modelica file and writes the results to a .mat file.

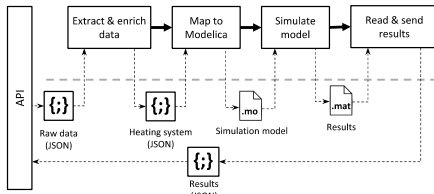


Fig. 2 - System architecture.

All functions are written with Python, since it is a straight-forward tool which is well suited for translation between data formats and since it is used in the BIM platform. Python does not carry out the simulations itself, but simply interfaces the simulation environment Dymola.

When the toolchain is activated, in the BIM platform, the platform sends a Post request, including BIM data for the desired system(s) to the service's API as seen in Fig. 3 where all data exchange between the database, the API and the toolchain is seen.

In the following sections, each step in the tool is described.

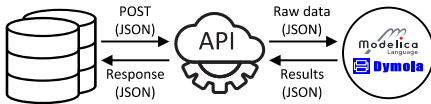


Fig. 3 - Interaction between the database and the toolchain through an API.

3.1. System extraction and data enrichment

When activating the tool, BIM data for the heating system is sent from the database, through the API to the tool. As a precaution and for future scenarios with several systems the tool extracts all components in the heating system from the data. To support the following mapping process, minor changes are made to the data by adding certain parameters based on the component classes. E.g., the length of pipe segments is calculated from the component's start and end coordinates. The enriched/manipulated data is then sent to the mapping process for model generation.

3.2. Mapping

In the mapping step, the Modelica models are generated. This is where the original data format and classes are translated to Modelica language and classes. This step is divided into two separate processes; in the first, the program loops through all components and maps them to a corresponding Modelica class and instantiates it in the model code. In the second, all connections between the components are translated to Modelica connectors. To handle the lack of information on control, this is also where default control connections are established.

In the mapping process, seven FSO classes are mapped to 10 different Modelica classes. Some FSO classes have been mapped to multiple Modelica classes, depending on the value of certain attributes. The full mapping and the translated attributes are seen in Tab. 2. The parameters are all required in the BIM data; if not, the program will fail.

All Modelica classes, except the bend model, originate from the Buildings library [16] which includes models for most components in HVAC systems in addition to detailed models of thermal zones. To simplify the mapping process, a purpose-built library with models that combine component models from *Buildings* was created. The combined models simplify the mapping process, since several Modelica models would otherwise have to be instantiated for each database component. Examples are the radiator model, which combines a radiator model and a thermostat, acting as a proportional controller and the MotorValve class, which combines a motorized valve with a PI controller and a setpoint.

Tab. 2 - Mapping between classes in the database and their corresponding Modelica classes.

Component	Modelica
Segment	model: Pipe ^a parameters: nominal flow, insulation thickness, insulation lambda, diameter, length
FlowMoving-Device ^d	model: PumpConstantSpeed ^b parameters: speed, performance curve
FlowMoving-Device ^d	model: PumpConstantPressure ^b parameters: head, performance curve
Radiator	model: Radiator ^b parameters: nominal heat flux, nominal supply temp, nominal return temp, nominal room temp, nominal pressure loss
HeatExchanger	model: DryCoilCounterFlow ^a parameters: nominal air flow, nominal water flow, dp nominal air, dp nominal water, UA nominal
Bend	model: CurvedBend ^c parameters: angle, diameter, bend radius
Tee	model: Junction ^a
FlowController ^d	model: Valves.TwoWayLinear ^a parameters: nominal flow, Kv-value
FlowController ^d	model: CheckValve ^a parameters: nominal flow, Kvs-value
FlowController ^d	model: MotorValve ^b parameters: nominal flow, Kvs-value

^a In Buildings.Fluid library

- ^b In purpose-built library
- ^c In Modelica.Fluid library
- ^d Mapping depends on component attributes

In the connection process, the connections between components are translated from the database format to Modelica language. In the database, the connection between components are described in *connectors*, in the *ConnectedWith* attribute, as seen in Fig. 4 that shows an example of two connected pipes. All components have at least 2 connectors. Each connector defines the expected direction of flow (in or out) and the connected component's tag, among other properties not relevant to this project.

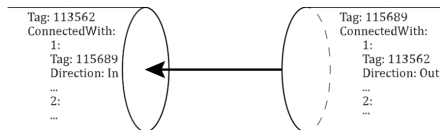


Fig. 4 - Example of connector definition.

The toolchain loops through all components and for every ingoing connector, a corresponding Modelica connector will be established. Only ingoing connections are considered to avoid duplicate connectors. In Modelica, components are connected through *ports*. The name of the ports vary, depending on the component class, and hence, they are stored in the components during data enrichment.

Since the BIM data does not support definition of control logics, default controls are assumed for the components that need control. E.g., all components mapped to the MotorValve class (see Tab. 2) are assumed to be controlling flow in a heating coil. Thus, these all take the measured ventilation supply temperature as an input for the processed variable for the PI controller.

For each component, a component model template is instantiated and added to a text string, containing the model information. After looping through all components, both for class mapping and connection establishment, the text string contains the entire model. The model is saved in a temporary model file for simulation.

3.3. Simulation and results reading

When simulating Modelica models, the models must first be compiled to a machine-readable executable and after that simulated. In the toolchain, this is done through *BuildingsPy* [17], which interfaces the commercial Modelica simulation environment Dymola. BuildingsPy takes the file path to the simulation file and simulation parameters, such as duration and solver, as parameters and parses these to Dymola. After simulation, the results are read through BuildingsPy, and the results are parsed to a JSON format and returned to the database. Since the simulations return many results for each component, the wanted results for each component class are defined in a specific file, and only these results are

sent back to the database.

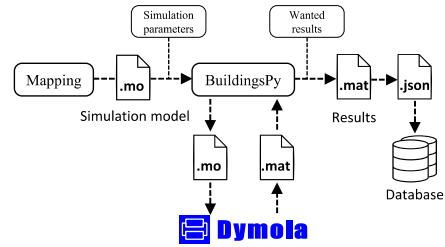


Fig. 5 - Simulation procedure.

4. Testing

To ensure that the tool is usable, it was tested on a small heating system model. The test did not focus on assessing the system's performance but merely to check whether the tool works, and the obtained results make sense and are of interest.

4.1. Test case description

The test case system, depicted in Fig. 6, consists of a heating coil and a radiator, each in separate loops. The main pump supplies flow to both loops, and a secondary pump is connected to the heating coil. To simulate the dynamic behavior, both the heating coil and the radiator are connected to a generic room with the parameters given in Tab. 3. For simplicity, only heat loss through the walls and window was considered.

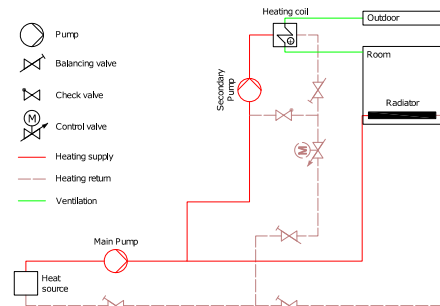


Fig. 6 - Heating system with one radiator and one heating coil used for testing the tool.

The radiator is controlled by a thermostatic radiator valve (TRV), connected to the room temperature. The TRV is not depicted in figure Fig. 6, since it is considered a part of the radiator. To control the heating output of the heating coil, a PI controller adjusts the control valve position to change the heating supply temperature. This is based on the ventilation supply temperature to the room, which has a constant setpoint. For simplicity, both pumps are operated with constant speed, although under normal circumstances, such pumps would either be controlled for constant or proportional pressure. The heat source is not considered, and it is assumed that

it supplies water at a fixed temperature of 70 °C.

Tab. 3 - Room parameters

Parameter	Value
Floor area	30 m ²
Wall area	60 m ²
Window area (south)	4 m ²
Total envelope area	66 m ²
Wall U-value	0.27 W/(m ² K)
Window U-value	1.31 W/(m ² K)
Window g-value	0.73 [-]

4.2. Simulation setup

The simulation was done for the first five days of weather data for Chicago, USA. In this period, the temperature ranges between -15 °C and 0 °C. Hence, it is possible to see the dynamic effects in varying external temperatures, including maximum capacity conditions.

To initialize the solution, in addition to the Modelica initialization, one preceding day was simulated before the first day of the year.

4.3. Simulation results

By simulating the system through the toolchain, the overall temperature curves in Fig. 7 were achieved. Fig. 7 shows the temperature of the room and the ventilation supply air, compared to the external temperature. It is seen that the room temperature is stable, but that there is an offset from the setpoint. This offset is caused by the TRV, which in Modelica is

modeled as a proportional controller, which will normally introduce an offset. Hence, this behavior is expected. The supply temperature is stable with no offset since it is controlled by a PI-controller.

In Fig. 8, the return temperature for both loops and the full system is seen. The return temperatures are stable around 30 °C, with a slight tendency to increase with lower external temperatures, as expected.

5. Discussion and gained experiences

The development process highlighted several points of attention during the mapping process. Most importantly, all needed data for the considered components must be available. If the parameters in Tab. 2 are not available for all components, the simulations will fail. This puts high demands on the level of information in BIM, but with tools for system dimensioning and component databases, the amount of information is not unrealistic. The possibility to perform detailed simulations of, e.g., return temperatures may even motivate designers to populate BIM models with more information on hydronic components.

In the presented work, controls were handled by applying a set of assumptions suitable for the specific test case. To work expand the work to larger systems, unambiguous control relations must be established in the database. This can be handled in two ways. Either the existing data format is extended with

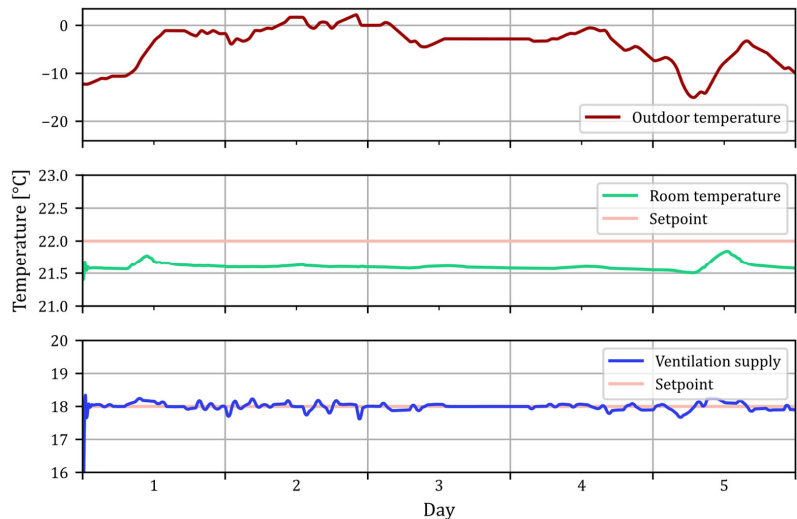


Fig. 7 - Results for the outdoor (t_{ext}), room (t_{room}) and ventilation supply (t_{vent}) temperature, including setpoints (SP).

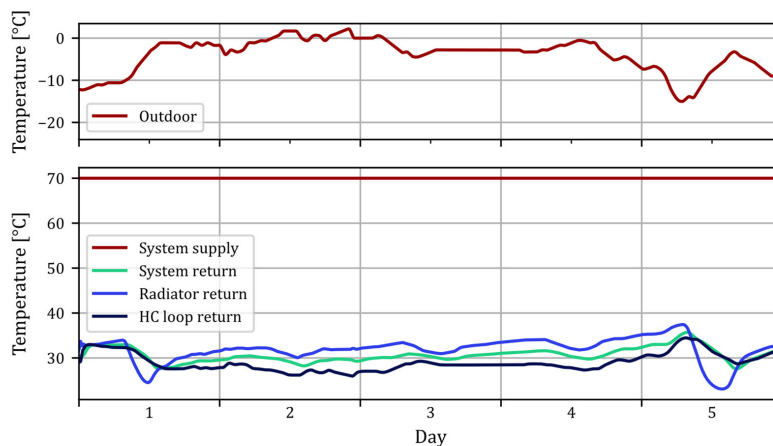


Fig. 8 – Return temperatures for heating coil loop (HC), radiator loop and for the entire system.

component parameters for controls, such as process variables, setpoints, etc., or a new data format for controls must be used. Work towards digitizing control information is in growth with several projects under development [11, 18]. Utilizing these existing frameworks to represent control logic in a database seems like a viable solution, but for simpler systems, the simple approach of added attributes may prove sufficient.

The connection between end units and rooms is a vital piece of information to correctly simulate the systems. Creating a link between end units and rooms may be a simple process in the BIM domain, but it requires additional mapping modules to include the rooms in the simulations.

6. Conclusion and future work

In this paper, it was proved that it is possible to create a tool for the simulation of heating systems based on BIM models. The simulations provided detailed and vital information on the performance of the individual components in the testing case. This showed the value of such simulations that are usually too time-consuming to be made. While the presented results may be trivial for a system as small as the test case, the same analysis for larger system will uncover results that are difficult for normal practitioners to quantify.

Several important attention points for a larger implementation were identified, the biggest being the lack of representation of controls in BIM models. These points resulted in several assumptions built into the tool, especially regarding control strategies. These assumptions mean that the tool is less flexible to different system configurations. By extending the data format to include the needed information on controls, the tool can easily be modified to simulate larger systems with both heating, ventilation, and cooling systems. When this work is done, the full

models can be used in fault detection, detailed analysis of the dynamic effects of coupling the systems, etc.

7. Acknowledgement

This work was funded by the national IFD grant J.nr. 8090-00046B for the project "HEAT 4.0 - Digitally supported Smart District Heating", Elforsk grant 352-042, IFD grant J.nr. 9065-00266A for the project "Virtual Commissioning in Building Services" and "Databaseret energistyring i offentlige bygninger", EU Interreg-ØKS 2020-2022.

8. References

- [1] de Wilde P. The gap between predicted and measured energy performance of buildings: A framework for investigation. *Automation in Construction*. 2014;41:40–9.
- [2] Menezes AC, Cripps A, Bouchlaghem D, Buswell R. Predicted vs. actual energy performance of non-domestic buildings: Using post-occupancy evaluation data to reduce the performance gap. *Applied Energy* [Internet]. 2012;97:355–64. Available from: <http://dx.doi.org/10.1016/j.apenergy.2011.11.075>
- [3] Virta M. Initial Commissioning. In: *HVAC Commissioning Guidebook* [Internet]. REHVA; 2021. p. 13–42. Available from: <https://app.knovel.com/hotlink/khtml/id:k t011ZBWH4/hvac-commissioning-process/initial-commissioning>
- [4] Wetter M. Modelica-based modelling and simulation to support research and development in building energy and control systems. *Journal of Building Performance Simulation* [Internet]. 2009 Jun 19;2(2):143–

61. Available from: <https://www.tandfonline.com/doi/full/10.1080/19401490902818259>
- [5] Integrated Environmental Solutions | IES. Iesve 2021 [Internet]. Available from: <https://www.iesve.com/ve2021>
- [6] EQUA Simulation Technology Group. IDA Indoor Climate and Energy [Internet]. 2014. Available from: <http://www.equa-solutions.co.uk/de/software/idaice>
- [7] Mattsson SE, Elmqvist H. Modelica - An International Effort to Design the Next Generation Modeling Language. IFAC Proceedings Volumes [Internet]. 1997 Apr;30(4):151–5. Available from: <https://linkinghub.elsevier.com/retrieve/pii/S1474667017436287>
- [8] Kim JB, Jeong W, Clayton MJ, Haberl JS, Yan W. Developing a physical BIM library for building thermal energy simulation. Automation in Construction. 2015;50(C):16–28.
- [9] Nytsch-Geusen C, Inderfurth A, Kaul W, Mucha K, Rädler J, Thorade M, et al. Template based code generation of Modelica building energy simulation models. In: Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15–17, 2017. Linköping University Electronic Press; 2017. p. 199–207.
- [10] Andriamamonjy A, Saelens D, Klein R. An automated IFC-based workflow for building energy performance simulation with Modelica. Automation in Construction. 2018;91(March):166–81.
- [11] Terkaj W, Schneider GF, Pauwels P. Reusing domain ontologies in linked building data: The case of building automation and control. CEUR Workshop Proceedings. 2017;2050.
- [12] Janowicz K, Rasmussen MH, Lefrançois M, Schneider GF, Pauwels P. BOT: The building topology ontology of the W3C linked building data group. Janowicz K, editor. Semantic Web [Internet]. 2020 Nov 19;12(1):143–61. Available from: <https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-200385>
- [13] Bew M, Richards M. BIM Maturity Model. Paper presented at the Construct IT Autumn 2008 Members' Meeting. 2008;
- [14] Kukkonen V, Küçükavci A, Seidenschur M, Rasmussen MH, Smith KM, Hviid CA. Proposing a Semantic Web Ontology to Support Flow System Descriptions from Design to Operation of Buildings. Automation in Construction. 2021;134.
- [15] Flow Systems Ontology Web Page [Internet]. 2021. Available from: <https://alikuçukavci.github.io/FSO/>
- [16] Wetter M, Zuo W, Noudui TS, Pang X. Modelica Buildings library. Journal of Building Performance Simulation [Internet]. 2014 Jul 4;7(4):253–70. Available from: <http://www.tandfonline.com/doi/abs/10.1080/19401493.2013.765506>
- [17] Lawrence Berkeley National Laboratory - Simulation Research Group. BuildingsPy [Internet]. 2021 [cited 2021 May 4]. Available from: <https://simulationresearch.lbl.gov/modelica/buildingspy/>
- [18] Wetter M, Ehrlich P, Gautier A, Grahovac M, Haves P, Hu J, et al. OpenBuildingControl: Digitizing the control delivery from building energy modeling to specification, implementation and formal verification. Energy [Internet]. 2022 Jan 1 [cited 2021 Sep 14];238:121501. Available from: <https://doi.org/10.1016/j.energy.2021.121501>

Data Statement

The datasets generated during and/or analysed during the current study are not publicly available because of ongoing development but are/will be available upon request.

6.9 Paper IX - Taking advantage of semantic web ontologies and shape constraints for Heating, Cooling and Ventilation Systems

Taking advantage of semantic web ontologies and shape constraints for Heating, Cooling and Ventilation Systems

Ali Küçükavcı¹, Mikki Seidenschnur^{1,2}, Kristoffer Negendahl¹, Christian Anker Hviid¹

¹Department of Civil and Mechanical Engineering, Technical University of Denmark, Denmark

²Ramboll, Copenhagen, Denmark

Abstract

In recent years semantic web ontologies have improved data interoperability within architecture, engineering, construction, and operation of buildings. One of the persisting issues inhibiting quality assurance is a lack of robust model validation of BIM models used for HVAC flow system simulation and analysis. This article provides a novel approach for automating the BIM validation process using SHACL shapes and FSO/FPO ontologies. Using this approach will ensure that the BIM model contains the required HVAC information for simulating hydraulic systems. The paper presents multiple shapes developed to identify and validate typical HVAC design details in buildings.

Introduction

Knowledge graphs and linked data has proven useful for the representation of Building Information Modeling (BIM) models in the Architecture, Engineering, Construction, and Operation (AECO) industry in the recent years (Rasmussen et al., 2021; Balaji et al., 2016; Pauwels and Terkaj, 2015). Graph representations of BIM models has potential uses in many different aspects of design and engineering. However, one issue in particular ties to the interoperability with the two most common formats for common data exchange, Industry Foundation Classes (IFC) and green building Extensible Markup Language (gbXML). These data formats are not designed to carry over all the data relevant for energy and indoor climate simulations, or Heating, Ventilation, and Cooling (HVAC) simulation data (Redmond et al., 2012). A recent attempt to challenge this issue is Porsani et al. (2021) who shows the difficulty in transforming BIM models to Building Energy Model (BEM) models, where the aspect of transformation between proprietary formats such as Revit (.rvt) to IFC and gbXML inherently generates errors later in the simulation process.

Kukkonen et al. (2022) proposed a semantic web ontology called Flow Systems Ontology (FSO) to describe the composition of flow systems and their energy and mass flow. Kukkonen et al. (2022) provided a comprehensive roadmap for further developments showing that use cases should be developed to further the development of the specific ontology of FSO. Furthermore, it was proposed that an ontology should be developed to include the properties of flow systems (component sizes, flow, material, i.e.). Though FSO proposes a common language to de-

scribe flow systems, the data needs to be parsed from existing BIM tools, which means that model validation is paramount to insure data integrity. FPO is a semantic web ontology developed to describe the capacity- and size-related properties of HVAC components (Küçükavcı et al., 2022).

Several efforts have sought to create validation tools for the IFC schema based on EXPRESS (Ghannad et al., 2019; Lee et al., 2016, 2021; Bolpagni et al., 2015; Lee et al., 2015). However, IFC is a large super-schema which is interpreted differently by BIM software vendors like Autodesk and Graphisoft, meaning that a building made in Archicad will not be parsed the same as a building made in Revit. Efforts proposes the use of Shapes Constraint Language (SHACL) shapes when using semantic web ontologies (Stolk and McGlinn, 2020; Soman et al., 2020; Soman, 2019; Hamdan and Scherer, 2020). SHACL shapes allow for validation of BIM models represented in a linked data format.

In this article we explore the HVAC system validation process of proprietary BIM models using FSO, FPO and SHACL to ensure that the required information for performing flow simulations are represented.

Methods

A Revit model of a typical office building was created and an airflow calculation was carried out for the HVAC system. The Revit model was transformed from a proprietary file format (.rvt) into a web-based Resource Description Framework (RDF)-triplestore containing a BIM model. Finally, SHACL shapes were used to validate that the model contained the necessary information to find the most critical pressure point in the open- or closed-circuit HVAC system.

The following sections describe the methods used to;

1. Parse data from a BIM to an RDF-triplestore
2. Validate and enrich the model using SHACL shapes and SPARQL
3. Use typical HVAC system design queries as use case examples

The process starts with a Revit model that contains a part of an HVAC system and the attached spaces. For this article, a parser was created, using C# and the Revit Application Programming Interface (API). First, the script maps the components and spaces, then it builds a .ttl (turtle format) string based on the FSO and FPO ontologies. Once

the parser has looped through all components and spaces, the .ttl string is parsed as a .txt file and sent through a client to the RDF-triplestore. Then, the triplestore stores the data, and then SHACL shapes are used to validate the data integrity (is all parameters filled, etc.). If the data validation fails, it will send an "actions needed" message to the original BIM model and tell it which components need to be fixed to continue. If it succeeds, it will pass to the final stage, which is the pressure and flow calculation of the critical branch. Finally, it displays the results in a table format for the user. The following sections will go through the process shown in Figure 1.

Parsing HVAC data from a BIM model into a Graph model

In the use case a HVAC model is linked with an architectural Revit model. The architectural model describe an office building with four rooms each heated by a radiator and ventilated by one return air terminal and one supply air terminal. The HVAC model involves three systems: heating, cooling, and ventilation seen in Figure 2. The heating system consists of a pump that feeds eight radiators and a heat exchanger. A pump supplies a heat exchanger via the cooling system. The ventilation system consists of a supply fan and exhaust fan connected by air ducts to four air terminals. A detailed description of the systems are illustrated in Figure 2.

The Revit to RDF parser converts the Revit data to RDF to be read into an RDF-based data model (triplestore). The RDF parser plugin written in C# accomplishes this. Revit's API is used to extract data from both the HVAC model and the architect model from the database, as shown in Figure 1. This data is then converted to RDF format expressed in turtle syntax and appended to a StringBuilder in C#.

Listing 1: Code-snippet from the parser showing how a pipe in Revit is converted to RDF using FPO.

```
1 //Get all pipes
2 FilteredElementCollector pipeCollector = new
  ↳ FilteredElementCollector(doc);
3 ICollection<Element> pipes =
  ↳ pipeCollector.OfClass(typeof(Pipe)).ToElements();
4 List<Pipe> pipeList = new List<Pipe>();
5 foreach (Pipe component in pipeCollector)
6 {
7     Pipe w = component as Pipe;
8     //Type
9     string componentID =
10    ↳ component.UniqueId.ToString();
11    sb.Append($"inst:{componentID} a fpo:Pipe ." +
12    ↳ "\n");
13 }
```

Listing 1 shows a small example of the code from the parser that can be applied to convert Revit data into RDF. The filteredElementCollector class and some other classes from Revit's API are used to retrieve pipes in the model. We extract each pipe's guid number and add it to our StringBuilder object sb.

Validation of model and data integrity

The graph model is validated by parsing the RDF data into a Fuseki database. The server is a SPARQL server that stores knowledge graphs in RDF form. In this case SPARQL is used to enable Create, Read, Update, Delete (CRUD) operations via an endpoint. Queries are used to both enrich or simply request data such as head and flow rates for a specific moving device. Data validation ensure that the necessary data exists and that it has the correct data type, content, and relation to other data in the graph. W3C recommends using SHACL shapes for validating RDF-based data used to describe and constrain RDF graphs. Each shape contains a description of the target it validates. Six SHACL shapes are developed to assure necessary data is present for a query to be performed subsequently to determine head and flow rate:

1. Each supply component must supply fluid to another component of the same system
2. Each return component must return fluid to another component of the same system
3. A component can only supply fluid to one component, except for a tee fitting and heat exchanger
4. A heat exchanger and tee must supply fluid to two components
5. Each supply component must have a parameter of pressure drop. The parameter must have a value and unit and the value must be above 0.0 and have a datatype of double.
6. Each supply component must have a parameter of length. The parameter must have a value and unit and the value must be above 0.0 and have a datatype of double.

In order to validate the data within the triplestore, the database must receive a .ttl file with all six shapes. The SHACL validation engine on the Fuseki server validates our shapes against the RDF graph and returns a validation report. It will then be possible to determine whether our data complies with the rules (shapes) in the validation report. The BIM model is updated if the data set does not comply with the rules. If the dataset matches the graph, we can continue to the next step and query the database. We used Postman, an HTTP client, to send a .ttl file to the Fuseki server and receive a validation report.

Querying head and flow rate

A query is a request for data from a database. Data from the triplestore can be queried using SPARQL. This article shows four queries. To find the head and flow rate of a specific flow moving device, we need to determine whether the flow moving device is part of an open-circuit or closed-circuit and what type of flow medium it is transporting as it impacts how we are going to query the head. In a closed-circuit system, the total head is equal to the dynamic head. When the system is an open-circuit and transports air, the total head is equal to the dynamic head. Lastly, if the system is an open-circuit and transports water, the total head is equal to the dynamic head plus the

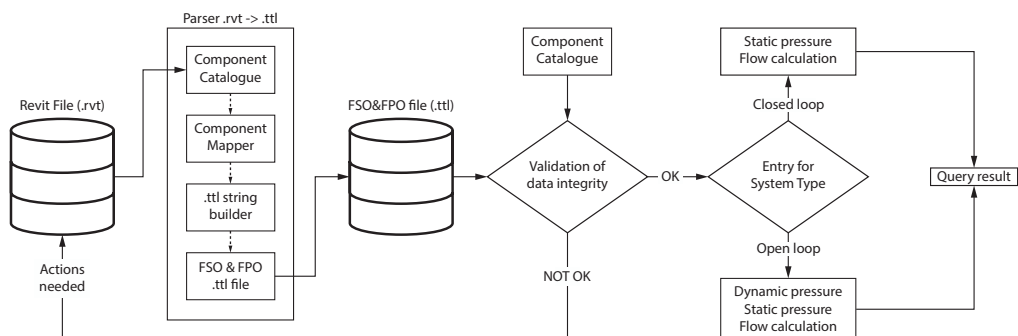


Figure 1: Process diagram for the Revit to FSO and FPO parser. The process consists of a BIM model, a parser, a Fuseki server to store, query and validate RDF models, a set of SHACL shapes and a Simple Protocol and RDF Query Language (SPARQL) select query to find the resulting head and flow rate of a given pump.

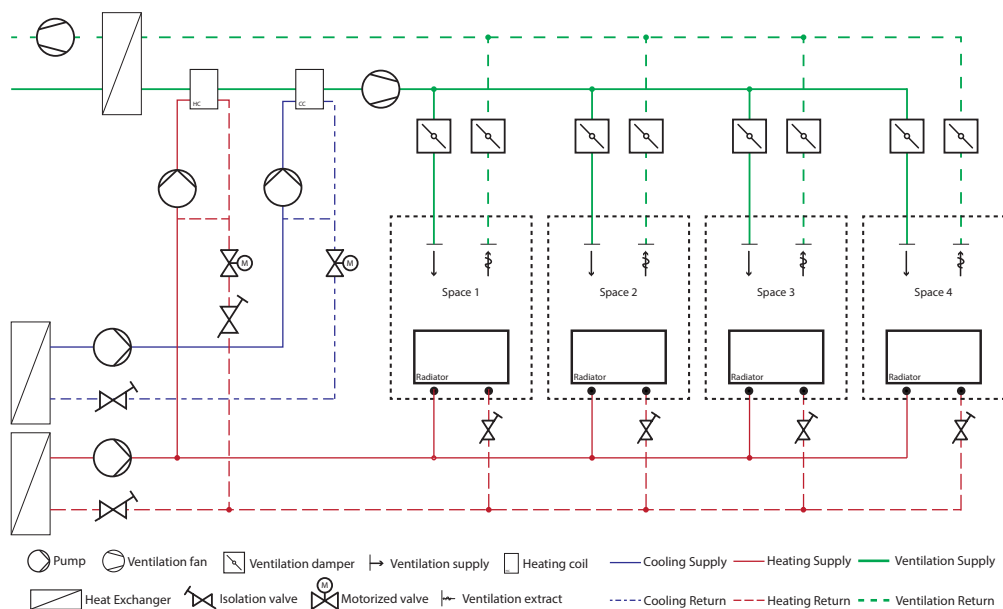


Figure 2: System diagram showing the heating system, cooling system and ventilation system supplying the 4 rooms (spaces) and their components. This schematic was modeled in Revit for the use case of this article.

static head. The first query identifies the type of circuit. A query will be performed according to the circuit type and flow medium type to obtain the head and flow rate of the given flow moving device.

Results

A total of six SHACL shapes and four SPARQL queries have been developed; however, due to article space limitations, only two SHACL shapes and two queries have been described in detail. The descriptions and source code for all SHACL shapes, SPARQL queries and converted triples, has been made available online ¹

To perform a hydraulic calculation to determine the capacity of a flow moving device in an HVAC system two parameters are needed; the total head and flow rate. Based on these, an HVAC designer can select a product. For closed-circuits' water-based systems, the head is the sum of the pressure drop generated by the critical branch. A SPARQL insert query can be used to perform all necessary calculations automatically. This is also possible when the original BIM model lacks critical data containing circuit type information. By using the insert query shown in Listing 2, each system in the triplestore is enriched with circuit type `ex:ClosedCircuit` or `ex:OpenCircuit` based on the medium and consumer components it uses, as well as the supply- and return temperatures.

Listing 2: Sparql update query to determine whether a system is a open-circuit or closed-circuit, and to add that information to that system, expressed in Turtle syntax.

```
1 INSERT {?system a ?circuit . ?system a ?systemType
2   ↳ .}
3 WHERE {
4   ?system fso:hasComponent ?component .
5   ?component fso:feedsFluidTo+ ?componentA .
6   ?componentA a ?componentAType .
7   ?system fso:hasFlow ?flow .
8   ?flow fpo:temperature ?temperature .
9   ?temperature fpo:value ?temperatureValue .
10  BIND (IF((?temperatureValue >= 25 &&
11    ↳ ?temperatureValue <= 70 && (?componentAType =
12    ↳ fpo:SpaceHeater || ?componentAType =
13    ↳ fpo:HeatExchanger)), ex:HeatingSystem, IF (
14    ↳ (?temperatureValue >= 5 &&
15    ↳ ?temperatureValue <= 15 &&
16    ↳ (?componentAType =
17    ↳ fpo:ChilledBeam || ?componentAType =
18    ↳ fpo:HeatExchanger)), ex:CoolingSystem, IF (
19    ↳ ((?temperatureValue >= 16 &&
20    ↳ ?temperatureValue <=
21    ↳ 24 && (?componentAType = fpo:AirTerminal ||
22    ↳ ?componentAType = fpo:HeatExchanger))),
23    ↳ ex:VentilationSystem, "" ))) AS
24    ↳ ?systemType )
25 FILTER (isIRI(?systemType))
26 BIND (IF((?temperatureValue >= 25 &&
27   ↳ ?temperatureValue <= 70 && (?componentAType =
28   ↳ fpo:SpaceHeater || ?componentAType =
```

```
15 fpo:HeatExchanger)), ex:ClosedCircuit,
16   ↳ IF((?temperatureValue >= 5 &&
17   ↳ ?temperatureValue <= 15 &&
18   ↳ (?componentAType =
19   ↳ fpo:ChilledBeam || ?componentAType =
20   ↳ fpo:HeatExchanger)), ex:ClosedCircuit,
21   ↳ IF ((?temperatureValue >= 16 &&
22   ↳ ?temperatureValue <=
23   ↳ 24 && (?componentAType = fpo:AirTerminal ||
24   ↳ ?componentAType = fpo:HeatExchanger))),
25   ↳ ex:OpenCircuit, "" ))) AS ?circuit )
26 }
```

Every HVAC component in the BIM model must be associated with a parameter `fpo:pressureDrop`. The parameter `fpo:pressureDrop` must also have a value associated with it, and the unit must be consistent across all components. Otherwise, the sum will be incorrect. The SHACL shape shown in Listing 3, validates exactly `fpo:pressureDrop` for all HVAC components in our BIM model. Listing 3 shows how we select our target using a SPARQL select query. The listing includes HVAC components on both a closed circuit's supply and return sides. Since the pump itself for a closed-circuit does not have a pressure drop, we omit this by writing `FILTER NOT EXISTS this is a fpo:Pump`. The rules are assigned to the target with the `sh: property`. For example, the maximum and minimum of one `fpo:pressureDrop` property is required for the target.

Listing 3: Shacl shape of each component must have a parameter pressure drop, value and unit. Expressed in Turtle syntax.

```
1 ex:Shape-1 a sh:NodeShape ;
2 sh:nodeKind sh:IRI ;
3 sh:target [
4   a sh:SPARQLTarget ;
5   sh:prefixes (fpo: fso: ex:);
6   sh:select """PREFIX fso: <https://w3id.org/fso#>
7   ↳ PREFIX fpo: <https://w3id.org/fpo#> prefix
8   ↳ ex:<http://example.org/> SELECT ?this WHERE
9   ↳ {?system a ex:ClosedCircuit .?system
10  ↳ fso:hasComponent ?this .filter not exists
11  ↳ {values ?type {fpo:Pump fpo:Fan} ?this a
12  ↳ type} .} """ ;
13 ];
14 sh:property [
15   sh:path fpo:pressureDrop ;
16   sh:minCount 1;
17   sh:maxCount 1;
18 ];
19 sh:property [
20   sh:path (fpo:pressureDrop fpo:value) ;
21   sh:minCount 1;
22   sh:maxCount 1;
23   sh:minInclusive 0.001;
24   sh:dataType xsd:double ;
25 ];
26 sh:property [
27   sh:path (fpo:pressureDrop fpo:unit) ;
28   sh:minCount 1;
29   sh:maxCount 1;
30   sh:dataType xsd:string ;
31   sh:hasValue "Pascal"^^xsd:string ;
32 ].
```

¹<https://github.com/alikucukavci/IBPSA-SPARQL-QUERIES-AND-SHACL-SHAPES>

In the first iteration of Listing 3, eight components violated the rules as they were missing the parameters `fpo:pressureDrop`, `fpo:value` and `fpo:unit`. We fixed these components in the HVAC BIM model, and the process from BIM to validation was repeated. After the second iteration, the validation report was conformant since all components met the conditions in Listing 3.

To verify the existence of a parameter, the composition of HVAC systems and components can also be validated. The tee and heat exchanger, for example, feed fluid to at least two other components. In Listing 2, the composition of the tees and heat exchangers for the RDF model is validated, and the validation report was conformant already in the first iteration.

Listing 4: Shacl shape of a heatexchanger- and tee component must supply fluid to two components expressed in Turtle syntax.

```
1 ex:Shape-2 a sh:NodeShape ;
2 sh:nodeKind sh:IRI ;
3 sh:target [
4   a sh:SPARQLTarget ;
5   sh:prefixes (fpo: fso:);
6   sh:select """
7     PREFIX fso: <https://w3id.org/fso#>
8     PREFIX fpo: <https://w3id.org/fpo#>
9     SELECT ?this WHERE {
10      ?system fso:hasComponent ?this .
11      FILTER EXISTS {
12        VALUE ?type {
13          fpo:Tee fpo:HeatExchanger
14        } ?this a ?type .}} """ ;
15 ];
16 sh:property [
17   sh:path fso:feedsFluidTo ;
18   sh:minCount 2;
19 ] .
```

Using a SPARQL query, listing 4 calculates the head and flow rate of a given pump. This pump is part of a close circuit, so we ignore the static head and only calculate the dynamic head. We must first calculate the total pressure loss of the critical path before we can calculate the dynamic head. The parameter pressure drop is summarized for each path from the given pump to a terminal to determine the critical path. The next step is to filter the path with the largest pressure loss. Due to this being a closed system, the terminals are either `fpo:SpaceHeater` or `fpo:HeatExchanger`. The total pressure loss for the critical path is converted from Pascal to meters. Finally, the flow rate is summarized for the terminals that the given flow moving device supplies in the same system. For the given flow moving device `inst:98172f87-b31e-4363-a01f-2f3f2d13a48f-00131613`, the SPARQL query returns the id number of the critical consumer component, a head of 1.71 meters of head and a flow rate of 0.034 L/s.

Listing 5: Sparql SELECT query to retrieve the dynamic head and flow rate of a given pump for a heating system, expressed in Turtle syntax.

```
1 PREFIX fpo: <https://w3id.org/fpo#>
2 PREFIX fso: <https://w3id.org/fso#>
3 PREFIX inst: <https://example.com/inst#>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT ?consumer
7   ↳ ((?totalPressure/0.00010199773339984) AS
8   ↳ ?dynamicHead) ?totalFlow
9 WHERE{
10   {SELECT (SUM(?flowRateValue) AS ?totalFlow)
11     WHERE{
12       ?system fso:hasComponent inst:98172f87-b31e-
13       ↳ 4363-a01f-2f3f2d13a48f-00131613
14       ↳ .
15       VALUES ?type {fpo:SpaceHeater
16       ↳ fpo:HeatExchanger} ?consumer a ?type .
17       ?consumer fpo:flowRate ?flowRate .
18       ?flowRate fpo:value ?flowRateValue .
19     }
20   }
21   {
22     SELECT ?consumer (?totalComponentPressureDrop +
23     ↳ ?consumerPresserDropValue AS ?totalPressure)
24     WHERE {
25       {
26         SELECT ?consumer (SUM(?returnPressureValue)
27         ↳ + (SUM(?totalSupplyPressureDrop)/COUNT(
28         ↳ ?totalSupplyPressureDrop)) AS
29         ↳ ?totalComponentPressureDrop)
30         WHERE {
31           {
32             SELECT ?consumer
33             ↳ (SUM(?supplyPressureValue) AS
34             ↳ ?totalSupplyPressureDrop)
35             {
36               ?supplySystem fso:hasComponent
37               ↳ inst:98172f87-b31e-4363-a01f-
38               ↳ 2f3f2d13a48f-00131613
39               ↳ .
40               ?supplySystem a fso:SupplySystem .
41               ?supplySystem fso:hasComponent
42               ↳ ?supplySystemComponent .
43               ?supplySystemComponent
44               ↳ fso:feedsFluidTo+ ?consumer .
45               values ?type {fpo:SpaceHeater
46               ↳ fpo:HeatExchanger} ?consumer a
47               ↳ ?type .
48               ?supplySystemComponent
49               ↳ fpo:pressureDrop
50               ↳ ?supplyPressureDrop .
51               ?supplyPressureDrop fpo:value
52               ↳ ?supplyPressureValue .
53             }
54             GROUP BY ?consumer
55           }
56         }
57         ?returnSystem fso:hasComponent ?consumer
58         ↳ .
59         ?returnSystem a fso:ReturnSystem .
60         ?returnSystem fso:hasComponent
61         ↳ ?returnSystemComponent .
62         ?consumer fso:feedsFluidTo+
63         ↳ ?returnSystemComponent .
64         ?returnSystemComponent fpo:pressureDrop
65         ↳ ?returnPressureDrop .
66         ?returnPressureDrop fpo:value
67         ↳ ?returnPressureValue .
68       }
69     }
70   }
71 }
```

```
43     } GROUP BY ?consumer
44   }
45   ?consumer fpo:pressureDrop
46   ↪ ?consumerPressureDrop .
47   ?consumerPressureDrop fpo:value
48   ↪ ?consumerPresserDropValue .
49 }
50 ORDER BY DESC((?totalPressure)) LIMIT 1
51 }
```

Discussion

This article has provided a novel approach to generate FSO triples based on a Revit BIM model. Furthermore, it allows for a systematic and standardized way to perform model validation of an HVAC system with the use of SHACL shapes. Such model validation enables better data interoperability in the future, and therefore eases the designer's burden. The SHACL shapes can be used in future work to ensure that models contain the right BIM information to allow for simulation of the hydraulic systems performed in Modelica. The SHACL shapes created in this article, though advanced, needs to be validated further by industry and academia. Listing 5 is limited to query the head and flow rate of one flow moving device at a time and only for closed-circuits. A more generic query that can calculate head and flow rate for every flow moving device on a construction project, regardless of system and circuit type, will be helpful for the HVAC designer. We created Listing 5 as a starting point, but for future work, it should be modified to be more generic. Furthermore, we created the SHACL shapes specifically for the validation of HVAC systems, but their use is not limited to that. There is a potential to use SHACL shapes for validation of many different sub-disciplines within the AECO industry. The article provides a proof-of-concept for a validation method readily available within the world of semantic web ontologies. To further verify the validity of SHACL shapes as a model validation method, further research should be carried out on the topic. Furthermore, this paper suggested a way to perform dimensioning of pumps, using an RDF-triplestore. In the roadmap for future works, the researchers imagine the work of this paper to pave the way for pump manufacturers to create an RDF-triplestore with all of their product data available. Doing this will close the gap from the manufacturer to the HVAC designers by allowing the HVAC designer to automatically query for product data from the manufacturer, based on the static and dynamic pressure calculations carried out in the RDF-triplestore.

Conclusion

It is possible to transform a typical BIM model with limited HVAC data into an RDF-triplestore using FSO and FPO. Beside being able to make useful HVAC queries on original data we demonstrate that HVAC data can be enriched via SPARQL insert queries. We demonstrate that the HVAC data can be validated for consistency and coherence through the use of generic SHACL shapes. Finally,

we conclude that SPARQL select queries can be used to compute critical pressure and flow rate for a given flow moving device thus exploiting the inherent advantages of an open source graph database using FSO, FPO, SHACL, and SPARQL as key enablers.

Acknowledgements

Funding: This work was funded by; the Ramboll Foundation; the Innovation Fund Denmark; EU-Interreg ÖKS "Data-driven Energy Management in Public Buildings".

References

- Balaji, B., A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M. B. Kjærgaard, M. Srivastava, and K. Whitehouse (2016). Brick: Towards a Unified Metadata Schema For Buildings. In *BuildSys '16: Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, pp. 41–50.
- Bolpagni, M., A. Luigi, C. Ciribini, and S. M. Ventura (2015). Informative content validation is the key to success in a BIM-based project validation is the key.
- Ghannad, P., Y.-c. Lee, J. Dimyadi, and W. Solihin (2019). Automated BIM data validation integrating open-standard schema with visual programming language. *Advanced Engineering Informatics* 40(January), 14–28.
- Hamdan, A. H. and R. J. Scherer (2020). Integration of BIM-related bridge information in an ontological knowledgebase. *CEUR Workshop Proceedings* 2636, 77–90.
- Küçükavci, A., M. Seidenschur, and H. C. A. Pauwels, Pieter (2022). Proposing a Semantic Web Ontology to Support Capacity- and Size-Related Property Descriptions of Heating, Ventilation and Air Conditioning Components in The Design Phase of Buildings.
- Kukkonen, V., A. Küçükavci, M. Seidenschur, M. H. Rasmussen, K. M. Smith, and C. A. Hviid (2022). An ontology to support flow system descriptions from design to operation of buildings. *Automation in Construction* 134(November 2020), 104067.
- Lee, Y.-c., C. M. Eastman, and J.-k. Lee (2015). Validations for ensuring the interoperability of data exchange of a building information model. *Automation in Construction* 58, 176–195.
- Lee, Y.-c., C. M. Eastman, and W. Solihin (2021). Rules and validation processes for interoperable BIM data exchange. *Journal of Computational Design and Engineering* 8(August 2020), 97–114.
- Lee, Y.-c., C. M. Eastman, W. Solihin, and R. See (2016). Modularized rule-based validation of a BIM model pertaining to model views. *Automation in Construction* 63, 1–11.

- Pauwels, P. and W. Terkaj (2015). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction* 63, 100–133.
- Porsani, G. B., K. D. V. de Lersundi, A. S. O. Gutiérrez, and C. F. Bandera (2021). Interoperability between building information modelling (Bim) and building energy model (bem). *Applied Sciences (Switzerland)* 11(5), 1–20.
- Rasmussen, M. H., M. Lefrançois, G. F. Schneider, and P. Pauwels (2021). BOT: The building topology ontology of the W3C linked building data group. *Semantic Web* 12(1), 143–161.
- Redmond, A., A. Hore, M. Alshawi, and R. West (2012). Exploring how information exchanges can be enhanced through Cloud BIM. *Automation in Construction* 24, 175–183.
- Soman, R. K. (2019). Modelling construction scheduling constraints using shapes constraint language (SHACL). *Proceedings of the 2019 European Conference on Computing in Construction* 1(2006), 351–358.
- Soman, R. K., M. Molina-Solana, and J. K. Whyte (2020). Linked-Data based Constraint-Checking (LDCC) to support look-ahead planning in construction. *Automation in Construction* 120(August), 103369.
- Stolk, S. and K. McGlinn (2020). Validation of IfcOWL datasets using SHACL. *CEUR Workshop Proceedings* 2636, 91–104.

6.10 Paper X - A roadmap toward a unified ontology for building service systems in the AECO industry: TSO and FSO

A roadmap toward a unified ontology for building service systems in the AECO industry: TSO and FSO

Nicolas Pauen¹, Ville Kukkonen^{2,3}, Ali Küçükavcı^{4,5}, Mads Holten Rasmussen⁶, Mikki Seidenschmur^{4,7}, Dominik Schlütter¹, Christian Anker Hviid⁴, and Christoph van Treeck¹

¹ RWTH Aachen University, Aachen, Germany
`pauen@e3d.rwth-aachen.de`

² Aalto University, Espoo, Finland

³ Granlund, Helsinki, Finland

⁴ Technical University of Denmark, Copenhagen, Denmark

⁵ COWI, Lyngby, Denmark

⁶ Niras, Allerød, Denmark

⁷ Ramboll, Copenhagen, Denmark

Abstract. Building service systems are complex structures consisting of different subsystems and components in varying relationships. Semantic Web Technologies (SWT) can be used to represent these systems in decentralized triplestores using ontologies. To describe interconnected building service systems and the flow of matter, energy and data between them over the whole life-cycle in the context of the Architecture, Engineering, Construction and Operation (AECO) industry, two recent contributions, TUBES System Ontology (TSO) and Flow System Ontology (FSO) have to be considered. This study thus supports the effort towards a future semantic web of building data by validating the given ontologies based on Competency Questions (CQs) and an application example, proposing an alignment of TSO v0.3.0 and FSO v0.1.0 and providing a roadmap to a unified representation of building service systems in the AECO industry.

Keywords: Linked Data · TSO · FSO · Building Service Systems · BIM.

1 Introduction

Building service systems, such as Heating, Ventilation, and Air Conditioning (HVAC) systems, form complex networks of components with varying kinds of relationships. During design, large quantities of information are produced about these systems, their components, and topological as well as functional relationships, which goes underutilized during operations and maintenance. While current data models such as Industry Foundation Classes (IFC) provide a standard format that most design tools can export, this file-based approach has its limitations. Recent research in data exchange for the Architecture, Engineering, Construction, and Operations (AECO) industry has seen an increasing application

of knowledge graphs and linked data [1]. To that end, two separate ontologies have been recently developed for describing building service systems and their flow: the TUBES System Ontology [2, 3] and the Flow Systems Ontology [4]. While the ontologies have similar aims, they have their differences in conceptualizations.

In order to align the two ontologies and pave the road towards a future unified representation of building service systems from design to operations, this paper compares the ontologies and describes the similarities and differences. An alignment between the concepts of the latest versions of the ontologies is proposed, and a roadmap for future unification is presented.

The paper is structured as follows. First, Section 2 reviews the state of the art. Following that, in Section 3, the ontologies are briefly introduced and then compared through a set of competency questions aimed at exercising the hierarchical, topological, and functional concepts of the ontologies, ending with the proposed alignments. Section 4 presents the roadmap for a unified representation of building service systems. Finally, Section 5 concludes the paper with closing remarks.

2 State of the Art

Several ontologies have been developed to improve interoperability within the AECO industry. A set of ontologies related to building service systems are described in this section.

The ifcOWL ontology is one of the first steps towards connecting BIM and semantic web technologies. It translates the IFC schema directly into the Web Ontology Language (OWL). Since the ifcOWL ontology is directly derived from the IFC schema, the relationships between different building components are intricate [5]. A complex data structure makes it difficult for AECO stakeholders and their tools to easily access building information [6]. Furthermore, ifcOWL includes too many domains, complicating any extension process [7]. To avoid the complexity and monolithic data structure of ifcOWL, a recent trend suggests a more modular and domain-specific approach to model building information with ontologies such as the Building Topology Ontology (BOT) [8, 9].

The Smart Energy Aware System (SEAS) ontology was developed through the EUREKA ITEA 12004 SEAS PROJECT to extend the Semantic Sensor Network (SSN) Ontology [10]. The ontology consists of three core modules: *seas:FeatureOfInterestOntology*, *seas:SystemOntology*, and *seas:EvaluationOntology*. Systems, relationships between systems, and the connection points between them can be expressed using *seas:SystemOntology*. The SEAS ontology focuses on electrical engineering and the supply of electrical energy and describes building service systems on a higher conceptual level. However, the Brick ontology and The Smart Appliances REference (SAREF) ontology can be used to represent the relationship of systems and components at a lower-conceptual level and a broader scope.

The Brick ontology defines the relationships among systems, components, sensors, and control parameters [11]. Furthermore, it has a schema definition that categorizes its classes into three types: *brick:Equipment*, *brick:Location*, and *brick:Point*. For example, we can state that an entity belonging to *brick:Air_Handling_Unit* is a subclass of *brick:Equipment*. This equipment can be located in a *brick:Room* which is a subclass of *brick:Location*. An air handling unit can contain a *brick:Supply_Air_Temperature_Sensor* which is a subclass of *brick:Point*.

The SAREF ontology was initially released in 2014, by the SmartM2M ETSI Technical Committee to describe smart devices in smart homes [12]. Currently, it consists of thirteen modules: SAREF, SAREF4SYST, and eleven extensions. SAREF is the core module and describes smart devices. SAREF4SYST has adopted the concepts of *seas:SystemOntology*, which defines systems, connections, and connections points. SAREF4BLDG is one of the eleven extensions that extend the building domain and describes the IFC taxonomy of building devices in OWL [13].

As part of the building operation phase, SAREF and BRICK focus on active components and their relationships to sensor points. Passive components such as segments and fittings and the flow of matter, energy, and data are not represented in SAREF and BRICK, which is essential information for the Mechanical, Electrical, and Plumbing (MEP) and HVAC engineers over the whole life cycle and especially during the design phase of building service systems. TSO and FSO are introduced [2, 4] to fill this research gap and to describe the connectivity between systems and components, including active and passive components with the focus on the design phase.

3 TSO and FSO

TSO aims to explicitly define the hierarchical, structural, and functional aspects of interconnected building service systems in the AECO industry and their relationships to spatial entities throughout their whole life cycle. The version 0.2.0 was published in [3]. At the current time, TSO is available in version 0.3.0. It is documented and available according to best practice via its URI <https://w3id.org/tso> and the containing concepts are defined in the namespace <https://w3id.org/tso#>, which will be abbreviated as *tso:* in the following examples. TSO has 40 classes and 101 object properties. The main classes are *tso:Zone*, *tso:System* and *tso:State*.

tso:Zone has a strong alignment to *bot:Zone* and is defined as an *owl:equivalentClass* in the given alignment. *tso:System* is defined as a model of a whole which is isolated from the world or a supersystem, which consists of interconnected components or subsystems and has links between attributes such as inputs, outputs, and states. To represent different states of systems and add a level of abstraction, *tso:State* is defined as the internal condition of a planned or abstract system. This includes specific aspects as on, off, open or closed as well as general aspects such as outdoor-air-operation, mixed-air-operation or heating-

operation. These main concepts and some of the object properties between them are shown in Figure 1.

FSO aims to describe the matter and energy flow between systems and components, and the composition of such systems [4]. To that end, FSO consists of 14 classes and 23 object properties. The ontology is available and documented at <https://w3id.org/fso>, and defines concepts in the namespace <https://w3id.org/fso#>, later abbreviated as *fso:*. Central classes include the disjoint *fso:System* and *fso:Component*, and the rest are subclasses of these two.

An *fso:System* is defined as a collection of components that can have attributes such as design properties attached to it. Instances of *fso:Component*, on the other hand, are the tangible components that participate in the flow of energy or matter. The subclasses of *fso:System* include more specific nomenclature such as *fso:DistributionSystem*, while subclasses of *fso:Component* include IFC-derived abstract component classes such as *fso:EnergyConversionDevice* and *fso:Segment*.

The object properties of FSO enable the hierarchical decomposition of systems into subsystems and their components, expressing the connectivity and direction of fluids and heat, and designating components as "sources" and "sinks" of systems. FSO components are aligned with SAREF4BDLG. Further, the hierarchical composition of systems and high-level connectivity of components and systems is aligned with SAREF4SYST. While FSO is not explicitly restricted to describing flow systems *in buildings*, the current use cases are limited to those.

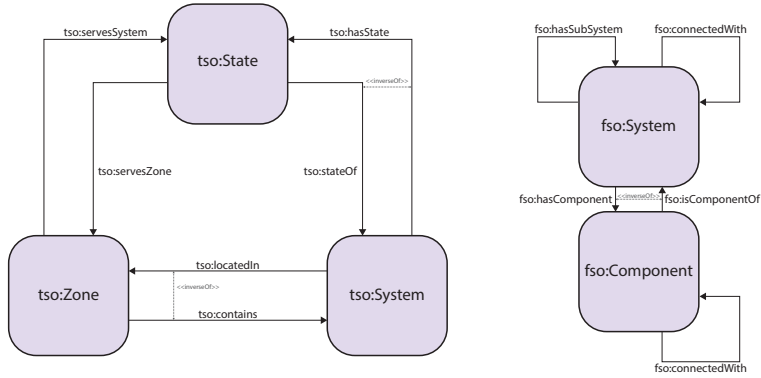


Fig. 1. Main concepts of TSO and FSO

To compare the given ontologies regarding the representation of building service systems in the AECO industry, the simplified application example involving a six-way valve to simulate a complex real life situation is given. The six-way valve was chosen as it demonstrates complex connectivity of components. The

valve is shown with context in Figure 2 and the following competency questions are used:

- CQ1: Which hierarchical concepts does a given component have?
- CQ1: Which supersystems does a component have?
- CQ2: Which topological concepts does a given component have?
- CQ2: What connections to other components does a component have?
- CQ3: Which is the flow of matter, energy or data through a given component?
- CQ3: What (matter/energy/data) can a component supply to downstream components?

The competency questions were chosen to exercise a set of core requirements as defined by the authors, covering hierarchical, topological, and functional aspects of systems. The example consists of two spaces: an office and a conference room. Both are served by an Active Chilled Beam (ACB) as part of the ventilation, heating and cooling systems. Upstream of the ACBs there are two six-way valves and pipes connecting those components. An IFC-based visualization of the application example, representations using TSO and FSO, as well as SPARQL queries to answer the CQs are given at <https://bs-visualizer.web.app>

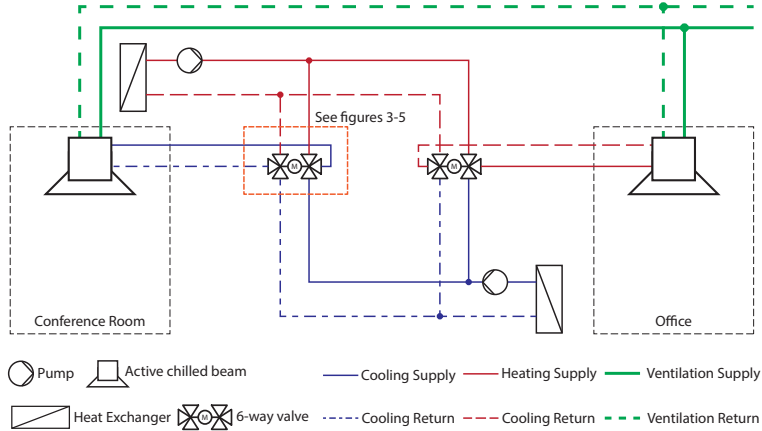


Fig. 2. Simplified application example

3.1 Hierarchical Concepts

To represent the hierarchical aspects of building service systems and answer CQ1, the four subclasses *tso:IntegratedSystem*, *tso:FunctionalSystem*,

tso:TechnicalSystem and *tso:Component* of *tso:System* are given in TSO. In the following the term “system” is used to describe all of them. *tso:IntegratedSystem* represents the coupling of different functional systems with independent inherent functions which are interconnected. *tso:FunctionalSystem* denotes a system that is defined by its overall inherent function. Typical examples would be *tso:HeatingSystem*, *tso:CoolingSystem* or *tso:VentilationSystem*. *tso:TechnicalSystem* is defined as a system with a coherent technical solution with which the inherent function (of the upper functional system) is fulfilled. Existing subclasses contain *tso:DistributionSystem* and *tso:ConversionSystem*. *tso:Component* denotes a system, for which the boundary that isolates it from the environment is defined by the manufacturer in terms of the product. Therefore, an air handling unit as well as the included rotary wheel heat exchanger or sensor could be instances of *tso:Component*. To classify components, TSO is aligned with ifcOWL and encourages the use of the international standard DIN EN IEC 81346-2 [14], which is yet to be implemented as an ontology. To link these systems, which represent different levels of hierarchy, the inverse object properties *tso:hasSubSystem* and *tso:subSystemOf*, as well as their eight subproperties (e.g. *tso:hasComponent*), can be used. The domain of this subproperty is defined to *tso:System* and the range to *tso:Component*

Given these concepts, the six-way valve would be an instance of *tso:Component* and be assigned to two instances of *tso:DistributionSystems* by *tso:hasComponent*. These systems are aggregated in two distinct instances of *tso:HeatingSystem* and *tso:CoolingSystem* which are both subsystems of the same instance of *tso:IntegratedSystem*.

In FSO, system hierarchy is modeled through the object properties *fso:hasSubSystem* (inverse *fso:isSubSystemOf*) linking systems to their subsystems, and *fso:hasComponent* (inverse *fso:isComponentOf*) linking systems to individual components. Depending on the use case, a model using FSO would likely have at least two top-level instances of *fso:System*: one for heating and once for cooling. These can be then decomposed to smaller subsystems via *fso:hasSubSystem*, for example, distribution systems as instances of *fso:DistributionSystem*. Finally, the valve could be modeled as an instance of *fso:Component* and assigned as a component of both the distribution systems using *fso:hasComponent*. The hierarchical concepts of TSO and FSO as well as the representation of CQ1 are shown in Figure 3.

In summary, TSO defines four levels of system hierarchy (*tso:IntegratedSystem*, *tso:FunctionalSystem*, *tso:TechnicalSystem* and *tso:Component*) as subclasses of an abstract *tso:System* class and corresponding object properties to describe the composition of building service systems. While in TSO the components are a subclass of system, FSO differentiates between the disjoint *fso:System* and *fso:Component* classes. FSO has no inherent system hierarchy to differentiate between functional or technical systems but uses object properties similar to TSO to decompose systems into subsystems and components.

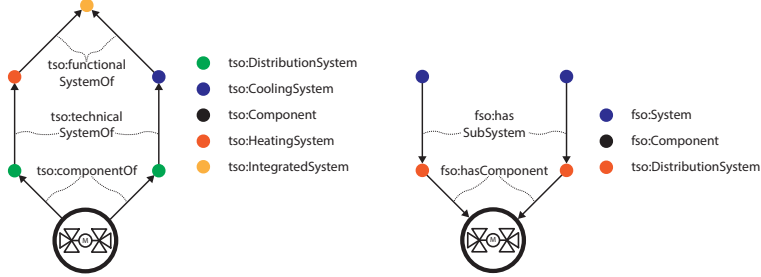


Fig. 3. Hierarchical concepts and CQ1 in TSO and FSO

3.2 Topological Concepts

To represent the topological aspects of building service systems and answer CQ2, TSO builds upon the concepts *tso:ConnectionPoint* and *tso:Connection* proposed in [10] and extends these by the subclasses *tso:InnerConnection* and *tso:OuterConnection* to differentiate between the connections inside a system and between different systems. *tso:ConnectionPoint* refers to an inlet or outlet of a system for a connection to other systems or within the same system, where some kind of matter, energy, or data can be transmitted. Using these concepts and the object properties which define the relationships between them, the symmetric object property *tso:connects* can be qualified to describe that two systems are connected. These concepts can be implemented to represent the system topology on every hierarchy level introduced in the last section.

For the example at hand, six instances of *tso:OuterConnections* need to be defined to represent the physical connections of the six-way valve. Each of these instances can be linked via *tso:connectsSystemAt* to a *tso:ConnectionPoint* which is connected via *tso:connectsAt* to the valve. To represent that not all of the six *tso:ConnectionPoints* are interlinked inside the six-way valve, two *tso:InnerConnections* can be defined and linked to the corresponding *tso:ConnectionPoints* using the described object properties.

Unlike TSO, the current version of FSO does not have concepts for connection points or connections. Instead, the top-level symmetric object property *fso:connectedWith* is used to communicate that two components or systems are connected so that they may exchange matter or energy. A central part of FSO is the tree of subproperties under *fso:connectedWith* that enables inferring more generic relationships from more specific ones. As such, the *fso:connectedWith* is not expected to be directly used, but rather inferred from more specific subproperties conveying further functional and logical relationships, such as *fso:suppliesFluidTo* and others, which are discussed in the next subsection with the functional concepts. A visual representation of the CQ2 for both TSO and FSO is given in Figure 4.

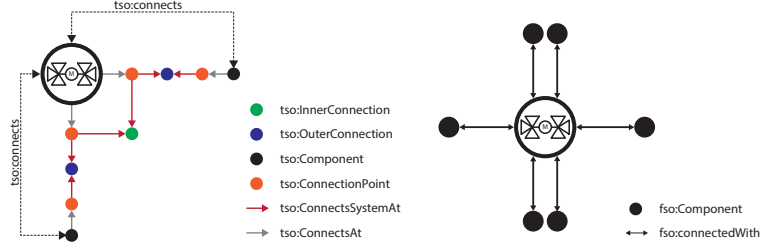


Fig. 4. Topological concepts and CQ2 in TSO and FSO

In summary, while both ontologies contain a symmetric object property to describe the connection between two systems, TSO further qualifies this connection using the concepts of *tso:ConnectionPoint*, *tso:InnerConnection* and *tso:OuterConnection*.

3.3 Functional Concepts

To represent the functional aspects of building service systems and answer CQ3, TSO relies heavily on the concept of states. A *tso:State* is defined as the internal condition of a (planned) system. It can be used to add a level of abstraction to represent systems where the function, and therefore the flow of matter, energy or data, can change due to given states. Hence, a system can have multiple potential states assigned via the *tso:hasState* object property. To represent “what” is exchanged between different systems or inside a certain system, the classes *tso:Matter*, *tso:Energy* and *tso:Data*, as well as multiple subclasses, are defined. These classes can be linked via *tso:hasInput* and *tso:hasOutput* to the states of different systems. Based on these concepts the inverse object properties *tso:supplies* and *tso:suppliedBy*, which link the states directly, can be inferred, as well as their subproperties to denote what is supplied. To represent the flow inside a system, the classes *tso:Matter*, *tso:Energy* and *tso:Data* can be connected to the given state by using the *tso:hasInnerExchange* object property. To qualify the exchange the considered matter, energy or data can be linked to a *tso:Connection* via *tso:transmitsThrough* or to the *tso:ConnectionPoints* via *tso:transmitsFrom* and *tso:transmitsTo*.

To answer CQ3 regarding what the six-way valve can supply to the downstream components and the active cooling beam, two *tso:States* need to be defined and assigned to the six-way valve. Each state is linked to an instance of *tso:Liquid* via *tso:hasInnerExchange* to represent the intended transfer of warm and cold liquid inside the valve through inner connections. As described in the last section these connections range from the connection points linking the pipes of the heating and cooling system to the connection point linking the downstream pipe. The different liquids can be specified using attributes to represent

their temperature and further aspects. To represent the transfer to the downstream pipe the instances of *tso:Liquid* are linked to the states of the six-way valve via the *tso:hasOutput* object property and to the state of the downstream pipe via the *tso:inputOf* object property. Therefore, *tso:suppliesLiquid* which links those states directly can be inferred. To answer CQ3 you can follow this property path or, since there is no conversion process between the six-way valve and the active chilled beam, the two liquids can be linked directly to the beam via the *tso:inputOf* object property. Hence, given the state of the six-way valve or of the upper *tso:TechnicalSystem* it can be distinguished between the intended flow of warm or cold liquid which is supplied to the downstream active cooling beam and thus the supply of the conference room.

Using FSO to answer CQ3 and describe what the six-way valve can supply to the downstream ACB requires the use of the more specific subproperties of *fso:connectedWith*, such as *fso:suppliesFluidTo*. The connectivity in itself only describes that there is an *intended* flow of fluid between the components. One way to deduce further what is supplied downstream of the six-way valve would be to look at what systems and components supply the valve. That is, the question could be rephrased as "what is supplied to the valve", which would then be what the valve could supply downstream. Further adding the concept of *source* and *sink* components of systems with *fso:hasSourceComponent* and *fso:hasSinkComponent*, it could be inferred that both the heating and cooling systems have sources that can supply fluid to the ACB. Additionally, the use of component classes such as *fso:FlowController* and *fso:EnergyConversionDevice* can be used to indicate and deduce the intended function of connected components to an extent. An abbreviated representation of the functional concepts regarding the 6-way valve from both TSO and FSO is shown in Figure 5.

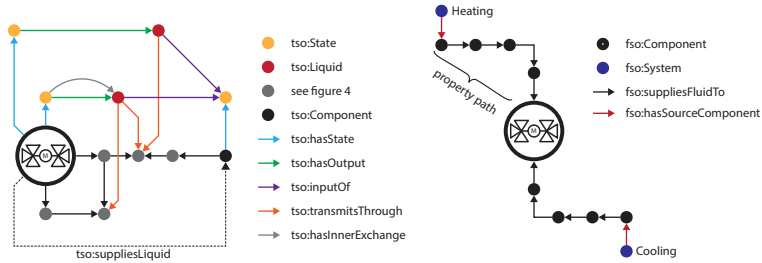


Fig. 5. Functional concepts and CQ3 in FSO and TSO

In summary, the ontologies strongly differ in regards to the representation of the functional concept. To describe the intended flow, TSO introduces the concept of states and defines matter, energy, and data as classes, which can be linked to the states. The object properties, which link the states of the systems

directly, can be inferred. On the other hand, FSO represents the intended transfer of fluids, heat, or electric charge between different systems via object properties that link these systems directly. As such, FSO does not consider the fact that the internal connectivity and function of systems and therefore the flow of matter, energy, and data can vary depending on aspects such as control values, while TSO enables expressing this with states.

3.4 Alignment

In order to concretely evaluate and describe the current compatibility of the two ontologies, an alignment of them is presented to the extent possible. Several concepts can be used to set up an alignment for TSO v0.3.0 and FSO v0.1.0. Distribution Systems, Supply Systems, and Return Systems are described as equivalent since they represent a nomenclature for systems on a certain level of hierarchy in both ontologies. *fso:System* is defined as a subclass of the abstract *tso:System* with *fso:isSubSystemOf* and *fso:isComponentOf* as subproperties of *tso:subSystemOf* - the inverse properties *fso:hasSubSystem* and *fso:hasComponent* being subproperties of *tso:hasSubSystem*. A Component in FSO is equivalent to the one in TSO, but a *tso:Component* is always an abstract *tso:System* in itself, while *fso:Component* and *fso:System* are disjoint classes.

A topological connection is categorized with *tso:connects* and *fso:connectedWith* respectively, hence they are defined as equivalent properties. Further topological concepts cannot be matched, since they do not exist in both TSO and FSO. The functional concepts are too diverse to propose a direct alignment between the two ontologies. The alignment is summarized in table 1.

Table 1. Alignment between FSO v0.1.0 and TSO v0.3.0

FSO		TSO
<i>fso:System</i>	<i>rdfs:subClassOf</i>	<i>tso:System</i>
<i>fso:DistributionSystem</i>	<i>owl:equivalentClass</i>	<i>tso:DistributionSystem</i>
<i>fso:SupplySystem</i>	<i>owl:equivalentClass</i>	<i>tso:SupplySystem</i>
<i>fso:ReturnSystem</i>	<i>owl:equivalentClass</i>	<i>tso:ReturnSystem</i>
<i>fso:Component</i>	<i>owl:equivalentClass</i>	<i>tso:Component</i>
<i>fso:isSubSystemOf</i>	<i>rdfs:subPropertyOf</i>	<i>tso:subSystemOf</i>
<i>fso:isComponentOf</i>	<i>rdfs:subPropertyOf</i>	<i>tso:subSystemOf</i>
<i>fso:hasSubSystem</i>	<i>rdfs:subPropertyOf</i>	<i>tso:hasSubSystem</i>
<i>fso:hasComponent</i>	<i>rdfs:subPropertyOf</i>	<i>tso:hasSubSystem</i>
<i>fso:connectedWith</i>	<i>rdfs:equivalentProperty</i>	<i>tso:connects</i>

4 Roadmap to a unified representation

A unified representation for interconnected building service systems which combines the concepts of TSO and FSO needs the expressiveness to model those systems throughout their whole life cycle and the simplicity to make them usable in day-to-day operations. Since the complexity and general structure of building service systems differ widely regarding the various disciplines and building types that are served by the systems, a modular ontology structure needs to be implemented.

The (lightweight) core module(s) need to contain hierarchical, topological, and functional aspects, which are valid for all disciplines. This includes classes such as System and State as well as object properties as connected and supplied. The degree to which these aspects should be included is yet to be defined based on application examples ranging various trades and levels of complexity and in communication with approaches such as SEAS [10], BRICK [11], and SAREF [12].

Further hierarchical, topological, and functional aspects which are valid for all disciplines but should not be included in the core module(s) for the sake of usability could be defined in a hierarchical, topological and functional ontology pattern, which enhances the expressiveness of the unified representation. The hierarchical pattern could include different levels of system hierarchy and the object properties to link these. Concepts such as connection points and connections could be contained in the topological pattern and the definition of different forms of matter, energy, and data as well as properties describing the input and the output of systems could be included in the functional ontology pattern.

Classifications of systems and concepts which are necessary to describe specific aspects of disciplines could be defined in separate domain ontologies. These should not describe concepts out of the scope of building service systems but to be aligned with existing approaches such as BOT [9] which reached a high level of shared conceptualization in the context of linked data in the AECO industry.

5 Conclusion

This paper presented a structured comparison of the two recent ontologies aimed at describing building service systems and their flow of matter, energy, and data: the TUBES System Ontology (TSO) and the Flow Systems Ontology (FSO). The ontologies were compared in terms of their overall goal and size and through the lenses of hierarchical, topological, and functional concepts. It was shown that while the ontologies have similar goals, their conceptualizations differ substantially. TSO is a considerably larger ontology, enabling more detailed descriptions of interconnected systems and their functional relationships based on their states. FSO offers a more limited set of concepts and properties and lacks particularly in terms of qualifying connections through concepts such as connections and connection points. Further, FSO does not enable the description of flow states that may vary, such as in the example of a six-way valve. On the other hand,

the expressivity of TSO comes at the cost of complexity, and a more modular approach could make it more approachable.

High-level alignments for the ontologies were proposed, which are admittedly rather superficial. This is primarily due to the incompatible functional concepts of the two ontologies, and is further complicated by the disjointness of *fso:System* and *fso:Component*. Finally, the roadmap towards future unification was presented, with ideas for further development and structuring of the ontologies to support shared conceptualizations.

The structured comparison of the two ontologies highlighting their respective strengths and weaknesses is useful for future efforts to refine the ontologies or build new ones. Aligning the ontologies and showing the friction points for the alignment is useful for future refinements to make the ontologies more compatible. In the end, supports the development of ontologies that better serve use cases related to building service systems.

References

1. P. Pauwels, A. Costin, M. H. Rasmussen, Knowledge Graphs and Linked Data for the Built Environment, in: M. Bolpagni, R. Gavina, D. Ribeiro (Eds.), *Industry 4.0 for the Built Environment: Methodologies, Technologies and Skills, Structural Integrity*, Springer International Publishing, Cham, 2022, pp. 157–183. doi:10.1007/978-3-030-82430-3_7.
2. N. Pauen, D. Schlütter, J. Siwiecki, J. Frisch, C. van Treeck, Integrated representation of building service systems: Topology extraction and TUBES ontology, *Bauphysik* 42 (6) (2020) 299–305. doi:10.3217/978-3-85125-786-1-59.
3. N. Pauen, D. Schlütter, J. Frisch, C. van Treeck, TUBES System Ontology: Digitalization of building service systems, in: *Proceedings of the 9th Linked Data in Architecture and Construction Workshop*, 2021.
4. V. Kukkonen, A. Küçükavci, M. Seidenschnur, M. H. Rasmussen, K. M. Smith, C. A. Hviid, An ontology to support flow system descriptions from design to operation of buildings, *Automation in Construction* 134 (December 2021) (2022) 104067. doi:10.1016/j.autcon.2021.104067.
URL <https://doi.org/10.1016/j.autcon.2021.104067>
5. J. Beetz, J. Van Leeuwen, B. De Vries, IfcOWL: A case of transforming EXPRESS schemas into ontologies, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM* 23 (1) (2009). doi:10.1017/S0890060409000122.
6. J. Flore, T. Djuedja, Integration of environmental data in BIM tool & Linked Building Data, in: *Proceedings of the 7th Linked Data in Architecture and Construction Workshop*, 2019.
7. A.-H. Hamdan, M. Bonduel, R. J. Scherer, An ontological model for the representation of damage to constructions, in: *Proceedings of the 7th Linked Data in Architecture and Construction Workshop*, 2019.
8. M. Niknam, S. Karshenas, A shared ontology approach to semantic representation of bim data, *Automation in Construction* 80 (2017) 22–36. doi:<https://doi.org/10.1016/j.autcon.2017.03.013>.
URL <https://www.sciencedirect.com/science/article/pii/S0926580517302364>
9. M. H. Rasmussen, M. Lefrançois, G. F. Schneider, P. Pauwels, Bot: the building topology ontology of the w3c linked building data group, *Semantic Web* 12 (1) (2021) 143–161. doi:10.3233/SW-200385.

10. M. Lefrançois, Planned ETSI SAREF extensions based on the W3C & OGC SOSA/SSN-compatible SEAS ontology patterns, *CEUR Workshop Proceedings* 2063 (December 2016) (2017).
11. B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Bergés, D. Culler, R. K. Gupta, M. B. Kjærgaard, M. Srivastava, K. Whitehouse, Brick: Metadata schema for portable smart building applications, *Applied Energy* 226 (September 2017) (2018) 1273–1292. doi:10.1016/j.apenergy.2018.02.091.
URL <https://doi.org/10.1016/j.apenergy.2018.02.091>
12. L. Daniele, F. den Hartog, J. Roes, Created in Close Interaction with the Industry: The Smart Appliances REference (SAREF) Ontology, in: *Lecture Notes in Business Information Processing*, Vol. 225, 2015. doi:10.1007/978-3-319-21545-7₉.
13. M. Poveda-Villalón, R. García-Castro, Extending the SAREF ontology for building devices and topology?, *CEUR Workshop Proceedings* 2159 (2018) 16–23.
14. IEC: International Electrotechnical Commission, DIN EN IEC 81346-2: Industrial systems, installations and equipment and industrial products – structuring principles and reference designations – Part 2: Classification of objects and codes for classes (2020).

Appendices

APPENDIX A

UML Diagram

This appendix provides the UML diagrams developed during this thesis

A.1 THERM UML diagram

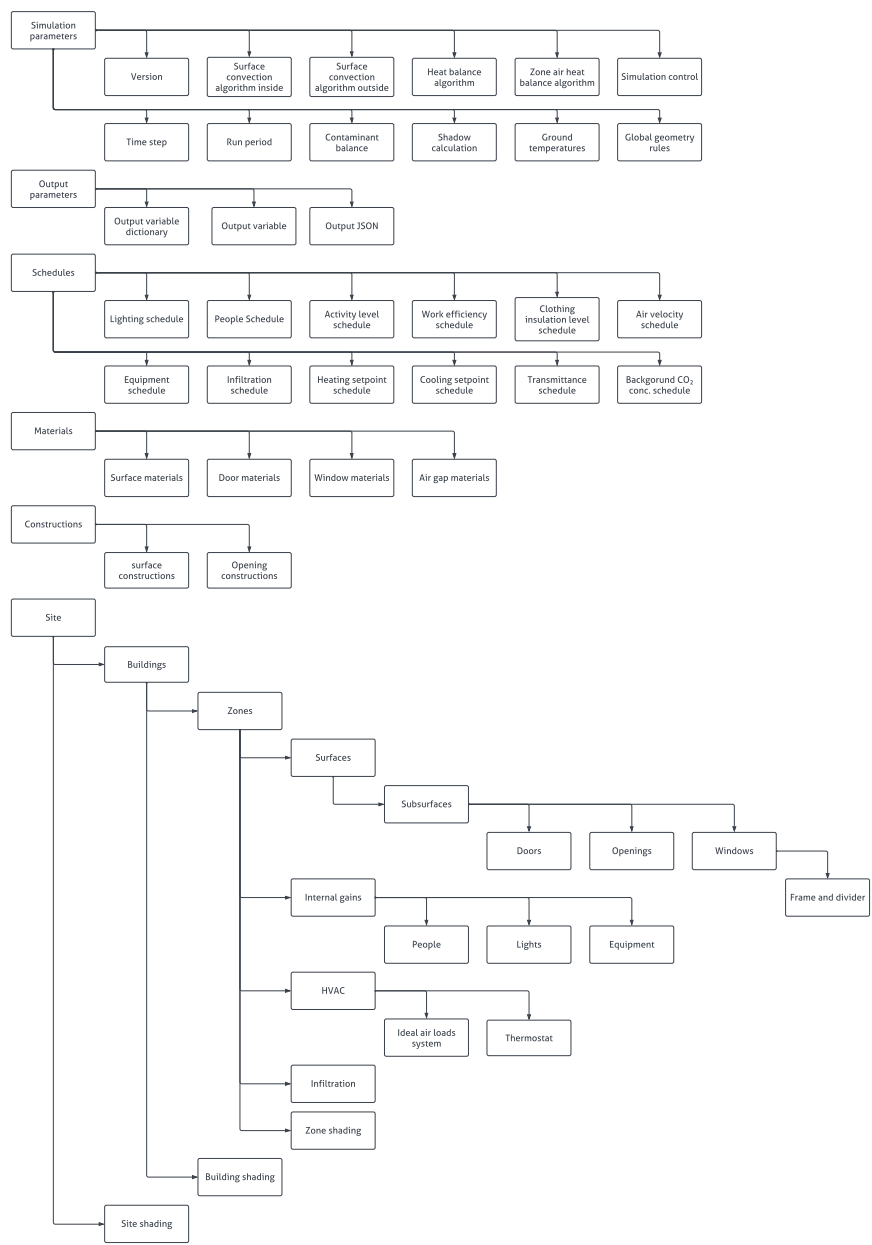


Figure A.1. Illustration of the THERM object model, from Paper II

Bibliography

- [1] F. Barbosa, J. Woetzel, J. Mischke, M. J. Ribeirinho, M. Sridhar, M. Parsons, N. Bertram, and S. Brown, “Reinventing Construction: A Route To Higher Productivity,” *Mckinsey Global Insititute*, no. February, p. 20, 2017.
- [2] H. Nassereddine, M. E. Jazzar, and M. Piskernik, “Transforming the AEC Industry: A Model-Centric Approach,” in *Proceedings of the Creative Construction e-Conference (2020) 076*, pp. 13–18, 2020.
- [3] Y. Rezgui and A. Zarli, “Paving the Way to the Vision of Digital Construction: A Strategic Roadmap,” *Journal of Construction Engineering and Management*, vol. 132, no. 7, pp. 767–776, 2006.
- [4] P. De Wilde, “The gap between predicted and measured energy performance of buildings : A framework for investigation,” *Automation in Construction*, vol. 41, pp. 40–49, 2014.
- [5] M. Jradi, K. Arendt, F. C. Sangogboye, C. G. Mattera, E. Markoska, M. B. Kjærgaard, C. T. Veje, and B. N. Jørgensen, “ObepME: An online building energy performance monitoring and evaluation tool to reduce energy performance gaps,” *Energy and Buildings*, vol. 166, pp. 196–209, 2018.
- [6] A. Menezes, Carolina, A. Cripps, D. Bouchlaghem, and R. Buswell, “Predicted vs . actual energy performance of non-domestic buildings : Using post-occupancy evaluation data to reduce the performance gap,” *Applied Energy*, vol. 97, pp. 355–364, 2012.
- [7] M. S. De Wit, “Uncertainty analysis in building thermal modelling,” in *SAMO95 Proceedings*, pp. 324–330, 1995.
- [8] B. Bordass, R. Cohen, M. Standeven, and A. Leaman, “Assessing building performance in use 3: Energy performance of the Probe buildings,” *Building Research and Information*, vol. 29, no. 2, pp. 114–128, 2001.

- [9] M. Way and B. Bordass, "Making feedback and post-occupancy evaluation routine 2: Soft Landings - Involving design and building teams in improving performance," *Building Research and Information*, vol. 33, no. 4, pp. 353–360, 2005.
- [10] C. Demanuele, T. Tweddell, and M. Davies, "Bridging the gap between predicted and actual energy performance in schools," *Open Journal of Energy Efficiency*, vol. 5, no. September, pp. 1–6, 2010.
- [11] J. Bloomberg, "Digitization, digitalization, and digital transformation: confuse them at your peril," *Forbes*, vol. 29, April, pp. 1–6, 2018.
- [12] J. Axley, "Multizone airflow modeling in buildings: History and theory," *HVAC and R Research*, vol. 13, no. 6, pp. 907–928, 2007.
- [13] E. M. Ryan and T. F. Sanquist, "Validation of building energy modeling tools under idealized and realistic conditions," *Energy and Buildings*, vol. 47, pp. 375–382, 2012.
- [14] K. Katsigarakis, G. N. Lilis, and D. Rovas, "A cloud IFC-based BIM platform for building energy performance simulation," *Proceedings of the 2021 European Conference on Computing in Construction*, vol. 2, pp. 350–357, 2021.
- [15] D. L. Macumber, B. L. Ball, and N. L. Long, "A graphical tool for cloud-based building energy simulation," in *2014 ASHRAE/IBPSA-USA Building Simulation Conference*, no. January 2014, pp. 87–94, ASHRAE, 2014.
- [16] E. Hale, L. Lisell, D. Goldwasser, D. Macumber, J. Dean, I. Metzger, A. Parker, N. Long, B. Ball, M. Schott, E. Weaver, and L. Brackney, "Cloud-Based Model Calibration Using OpenStudio," in *eSim*, no. March, pp. 1–14, 2014.
- [17] N. Gu and K. London, "Understanding and facilitating BIM adoption in the AEC industry," *Automation in Construction*, vol. 19, no. 8, pp. 988–999, 2010.
- [18] Dansk Standard, *DS418 - Beregning af bygningers varmetab*. 2020.
- [19] E. Djunaedy, K. Van Den Wymelenberg, B. Acker, and H. Thimmana, "Oversizing of HVAC system: Signatures and penalties," *Energy and Buildings*, vol. 43, no. 2-3, pp. 468–475, 2011.
- [20] Y. Sun, L. Gu, C. F. Wu, and G. Augenbroe, "Exploring HVAC system sizing under uncertainty," *Energy and Buildings*, vol. 81, pp. 243–252, 2014.

- [21] M. M. Ouf, W. O'Brien, and H. B. Gunay, "Improving occupant-related features in building performance simulation tools," *Building Simulation*, vol. 11, no. 4, pp. 803–817, 2018.
- [22] E. Djunaedy, K. Van Den Wymelenberg, B. Acker, and H. Thimmanna, "Rightsizing: Using simulation tools to solve the problem of oversizing," in *Proceedings of Building Simulation*, pp. 14–16, 2011.
- [23] M. Wetter, "Modelica-based modelling and simulation to support research and development in building energy and control systems," *Journal of Building Performance Simulation*, vol. 2, no. 2, pp. 143–161, 2009.
- [24] A. Andriamamonjy, D. Saelens, and R. Klein, "An automated IFC-based workflow for building energy performance simulation with Modelica," *Automation in Construction*, vol. 91, no. February, pp. 166–181, 2018.
- [25] A. Andriamamonjy, R. Klein, and D. Saelens, "Automated grey box model implementation using BIM and Modelica," *Energy and Buildings*, vol. 188–189, pp. 209–225, 2019.
- [26] J. Cao, T. Maile, J. O'Donnell, R. Wimmer, and C. van Treeck, "MODEL TRANSFORMATION FROM SIMMODEL TO MODELICA FOR BUILDING ENERGY PERFORMANCE SIMULATION," in *BauSIM2014: 5th German-Austrian Conference of the International Building Performance Simulation Association, Aachen University, Germany, 22-24 September 2014*, 2014.
- [27] V. Bazjanac and T. Maile, "GENERATION OF BUILDING GEOMETRY FOR ENERGY PERFORMANCE SIMULATION USING MODELICA," in *Proceedings of the CESBP/BauSIM 2016 conference, Dresden*, pp. 361–368, 2016.
- [28] A. Nicolai and A. Paepcke, "Co-Simulation between detailed building energy performance simulation and Modelica HVAC component models," *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, vol. 132, pp. 63–72, 2017.
- [29] IBPSA, "IBPSA Project 1."
- [30] M. Wetter, W. Zuo, T. S. Nouidui, and X. Pang, "Modelica Buildings library," *Journal of Building Performance Simulation*, vol. 7, no. 4, pp. 253–270, 2014.
- [31] D. Müller, M. Lauster, A. Constantin, M. Fuchs, and P. Remmen, "Aixlib - an Open-Source Modelica Library Within the IEA-EBC Annex 60 Framework," in *Proceedings of the CESBP Central European Symposium on Building Physics and BauSIM 2016*, no. September, pp. 3–9, 2016.

- [32] C. Nytsch-Geusen, J. Huber, M. Ljubijankic, and J. Rädler, "Modelica BuildingSystems - eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme," *Bauphysik*, vol. 35, no. 1, pp. 21–29, 2013.
- [33] F. Jorissen, G. Reynders, R. Baetens, D. Picard, D. Saelens, and L. Helsen, "Implementation and verification of the ideas building energy simulation library," *Journal of Building Performance Simulation*, vol. 11, no. 6, pp. 669–688, 2018.
- [34] M. Wetter, K. Benne, A. Gautier, T. S. Nouidui, A. Ramle, A. Roth, H. Tummescheit, S. Mentzer, and C. Winther, "LIFTING THE GARAGE DOOR ON SPAWN, AN OPEN-SOURCE BEMCONTROLS ENGINE," in *Building Performance Modeling Conference and SimBuild*, no. July, pp. 518–525, 2020.
- [35] Michael Wetter, Kyle Benne, and Baptiste Ravache, "Software Architecture and Implementation of Modelica Buildings Library Coupling for Spawn of EnergyPlus," in *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*, vol. 181, pp. 325–334, 2021.
- [36] C. Nytsch-Geusen, J. Rädler, M. Thorade, and C. Ribas Tugores, "BIM2Modelica - An open source toolchain for generating and simulating thermal multi-zone building models by using structured data from BIM models," *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, vol. 157, pp. 33–38, 2019.
- [37] W. Yan, M. Clayton, J. Haberl, W. Jeong, J. B. Kim, S. Kota, J. L. B. Alcocer, and M. Dixit, "INTERFACING BIM WITH BUILDING THERMAL AND DAYLIGHTING MODELING," in *13th Conference of International Building Performance Simulation Association*, no. Proceedings of BS2013, pp. 3521–3528, 2013.
- [38] J. Beetz and N. Gu, "BIMserver.org - an Open Source IFC Model Server," in *Proceedings of the CIB W78 2010*, pp. 1–9, 2009.
- [39] K. Jaskula, D. Kifokeris, D. E. Papadonikolaki, and D. Rovas, "Common Data Environments in construction: State-of-the-art and challenges for practical implementation," *SSRN Electronic Journal*, pp. 1–32, 2022.
- [40] Atlassian, "Microservices vs Monolithic Architecture."
- [41] D. T. Ross and J. E. Rodriguez, "Computer-Aided Design System," in *AFIPS '63 (Spring): Proceedings of the May 21-23, 1963, spring joint computer conference*, vol. 33, (Massachusetts), 1969.
- [42] buildingSMART, "Industry Foundation Classes (IFC) - An Introduction," 2022.

- [43] K. Afsari, C. M. Eastman, and D. Castro-Lacouture, "JavaScript Object Notation (JSON) data serialization for IFC schema in web-based BIM data exchange," *Automation in Construction*, vol. 77, pp. 24–51, 2017.
- [44] M. Elagiry, N. Charbel, P. Bourreau, E. D. Angelis, and A. Costa, "IFC To Building Energy Performance Simulation: a Systematic Review of the Main Adopted Tools and Approaches," no. September, 2020.
- [45] I. V. Ivanova, *Comparison Between Semi- Automated Building Energy Simulation Processes Using BIM- Generated gbXML and IFC Formats in EnergyPlus*. PhD thesis, 2014.
- [46] R. J. Hitchcock and J. Wong, "Transforming IFC architectural view BIMs for energy simulation: 2011," in *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association*, pp. 1089–1095, 2011.
- [47] T. Laine and A. Karola, "Benefits of Building Information Models in Energy Analysis," in *Proceedings of Clima 2007 WellBeing Indoors*, pp. 1–8, 2007.
- [48] P. Pauwels and K. McGlinn, *Buildings and Semantics*, vol. 1. CRC Press/Balkema, 2023.
- [49] H. Lange, A. Johansen, and M. B. Kjærgaard, "Evaluation of the opportunities and limitations of using IFC models as source of building metadata," *BuildSys 2018 - Proceedings of the 5th Conference on Systems for Built Environments*, pp. 21–24, 2018.
- [50] W. Solihin, C. Eastman, and Y. C. Lee, "Toward robust and quantifiable automated IFC quality validation," *Advanced Engineering Informatics*, vol. 29, no. 3, pp. 739–756, 2015.
- [51] J. Werbrouck, P. Pauwels, J. Beetz, and L. v. Berlo, "Towards a decentralised common data environment using linked building data and the solid ecosystem," in *Advances in ICT in Design, Construction and Management in Architecture, Engineering, Construction and Operations (AECO) : Proceedings of the 36th CIB W78 2019 Conference.*, pp. 113–123, 2019.
- [52] N. Guarino, D. Oberle, and S. Staab, "What is an ontology," in *Handbook on Ontologies*, pp. 1–17, Springer, 2004.
- [53] T. Gruber, "Collective knowledge systems: Where the Social Web meets the Semantic Web," *Web Semantics*, vol. 6, no. 1, pp. 4–13, 2008.

- [54] J. Beetz, J. Van Leeuwen, and B. De Vries, “IfcOWL: A case of transforming EXPRESS schemas into ontologies,” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, vol. 23, no. 1, pp. 89–101, 2009.
- [55] G. F. Schneider, M. H. Rasmussen, P. Bonsma, J. Oraskari, and P. Pauwels, “Linked building data for modular building information modelling of a smart home,” *eWork and eBusiness in Architecture, Engineering and Construction - Proceedings of the 12th European Conference on Product and Process Modelling, ECPPM 2018*, no. 2016, pp. 407–414, 2018.
- [56] M. H. Rasmussen, M. Lefrançois, G. F. Schneider, and P. Pauwels, “BOT: The building topology ontology of the W3C linked building data group,” *Semantic Web*, vol. 12, no. 1, pp. 143–161, 2020.
- [57] M. H. Rasmussen, J. Karlshøj, C. A. Hviid, and P. Pauwels, “Proposing a central AEC ontology that allows for domain specific extensions,” in *LC3 2017: Volume I - Proceedings of the Joint Conference on Computing in Construction (JC3)*, no. July, pp. 237–244, 2017.
- [58] D. Beckett, T. Berners-Lee, E. Prud’hommeaux, and G. Carothers, “Turtle - Terse RDF Triple Language,” Tech. Rep. February 2014, 2014.
- [59] V. Kukkonen, A. Küçükavci, M. Seidenschnur, M. H. Rasmussen, K. M. Smith, and C. A. Hviid, “An ontology to support flow system descriptions from design to operation of buildings,” *Automation in Construction*, vol. 134, no. November 2020, p. 104067, 2022.
- [60] M. Ngilinga De Carvalho, “Improving Interoperability in Increasingly Fragmented AEC Workflows,” *Springer*, no. September, 2020.
- [61] M. H. Rasmussen, *Digital Infrastructure and Building Information Modeling in the Design and Planning of Building Services*. PhD thesis, Technical University of Denmark, 2019.
- [62] M. Seidenschnur, A. Küçükavci, E. Visby, K. Michael, P. Pauwels, and C. Anker, “A common data environment for HVAC design and engineering,” *Automation in Construction*, vol. 142, no. July, p. 104500, 2022.
- [63] E. V. Fjerbæk, M. Seidenschnur, A. Küçükavci, K. M. Smith, and C. A. Hviid, “From BIM databases to Modelica - Automated simulations of heating systems,” in *REHVA 14th HVAC World Congress*, pp. 1–7, 2022.
- [64] E. V. Fjerbæk, M. Seidenschnur, A. Küçükavci, K. M. Smith, and C. A. Hviid, “Coupling Modelica simulations and a Common Data Environment for BIM,” 2023.

DTU Construct
Section of Thermal Energy
Technical University of Denmark

Koppels Allé, Bld. 403
DK-2800 Kgs. Lyngby
Denmark
Phone: +45 4525 4131
Fax: +45 4525 1961

www.construct.dtu.dk

February 2023

ISBN: 978-87-7475-720-7

DCAMM
Danish Center for Applied Mathematics
and Mechanics

Koppels Allé, Bld. 404
DK-2800 Kgs. Lyngby
Denmark
Phone (+45) 4525 4250
Fax (+45) 4525 1961

www.dcammm.dk

DCAMM Special Report No. S328

ISSN: 0903-1685