Efficient management and compliance check of HVAC information in the building design phase using semantic web technologies

Ali Kücükavci[a,*], Mikki Seidenschnur[a,b], Pieter Pauwels[d], Mads Holten Rasmussen[c], Christian Anker Hviid[a]

[a]*Department of Civil Engineering, Technical University of Denmark, Copenhagen, Denmark*
[b]*Ramboll, Copenhagen, Denmark*
[c]*Niras, Allerød, Denmark*
[d]*Department 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. (4) 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 Kücükavci), msei@ramboll.dk (Mikki Seidenschnur), 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, polylithic, 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].

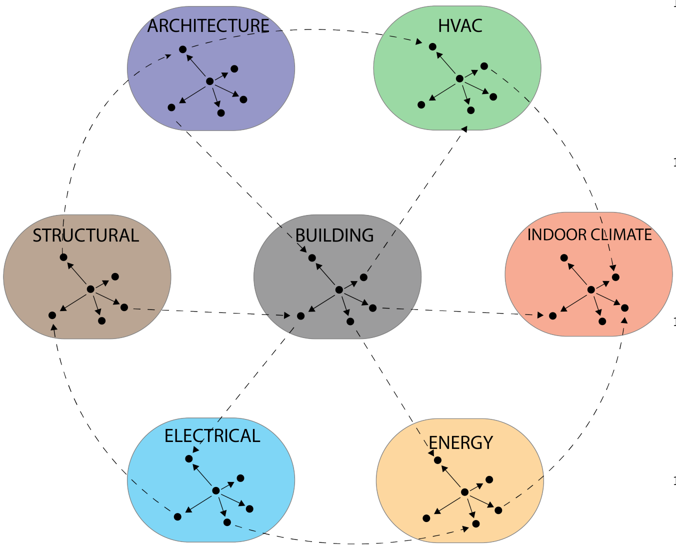

Figure 1: Interlinked domain-specific ontologies.

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. Backround

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 website[1] 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) ontology[2]. 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 submission[3]. 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:lengthValue_IfcQuantityLength 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).

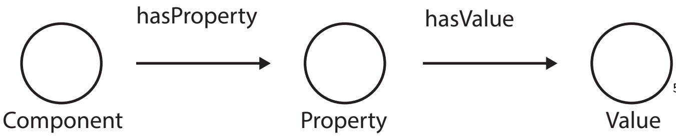

Figure 2: Relationship between components, properties, and property values.

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 GitHub[4].

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.

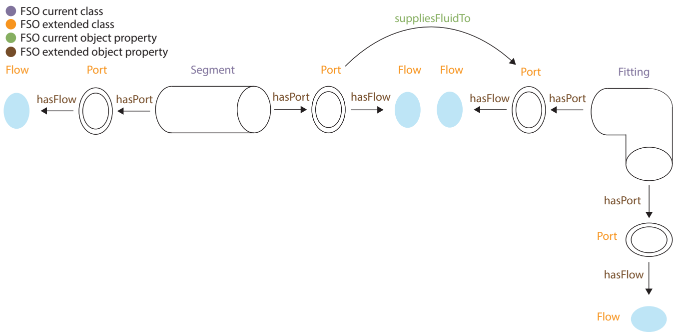

Figure 3: A segment partitioned with ports and flow connects to an fitting through its ports

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:ConnectPoint`. 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`

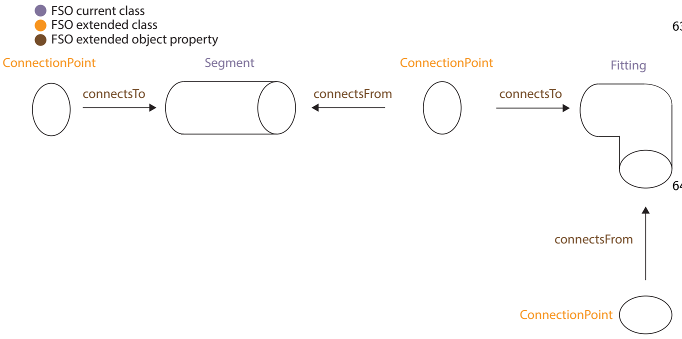

Figure 4: A segment connects to a fitting through connection points.

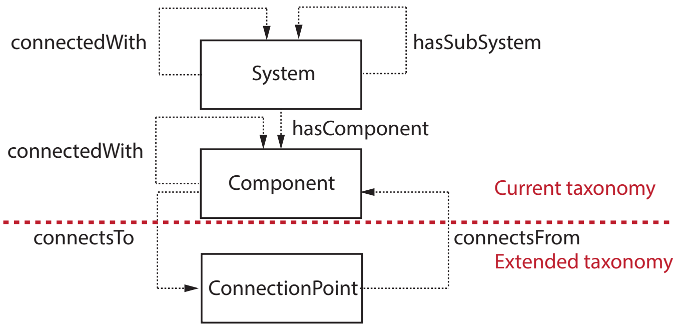

Figure 5: Current and extended taxonomy of FSO with connection points.

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.

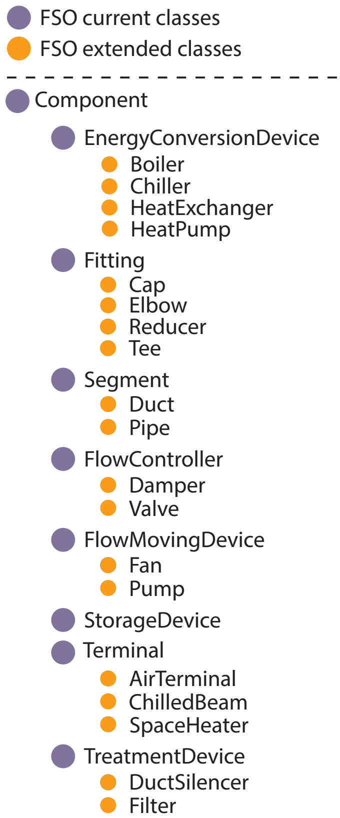

Figure 6: A class hierarchy of current and extended FSO components.

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 approach that is documented by Bonduel and Pauwels [61] and in principle follows a one-to-many pattern.

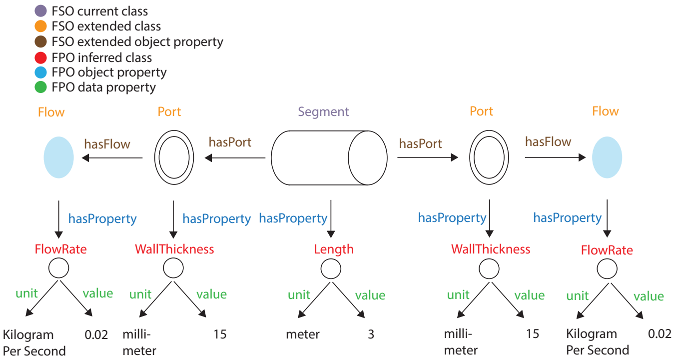

Figure 7: Describing the relationship between an `fso:Segment` and and its properties with FPO classes, object and data properties.

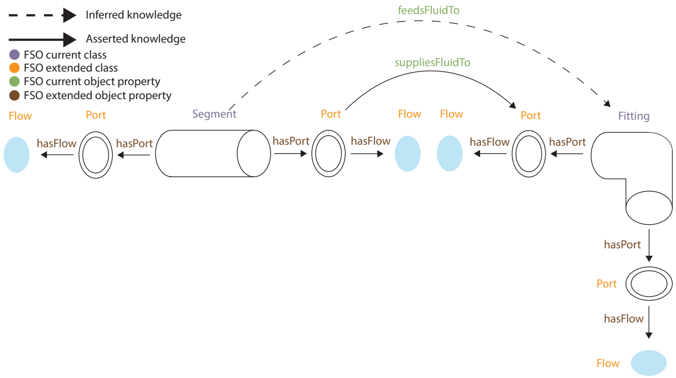

Figure 8: Deducing that the segment feeds fluid to the fitting as a port of the segment supplies fluid to a port of the fitting.

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 building 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 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 `s4bldg:flowCoefficient`, while `fpo:hasVolume` is an equivalent property to `s4bldg:volume`. Moreover, `fpo:hasDesignAirflowRate` is equivalent to `s4bldg: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.

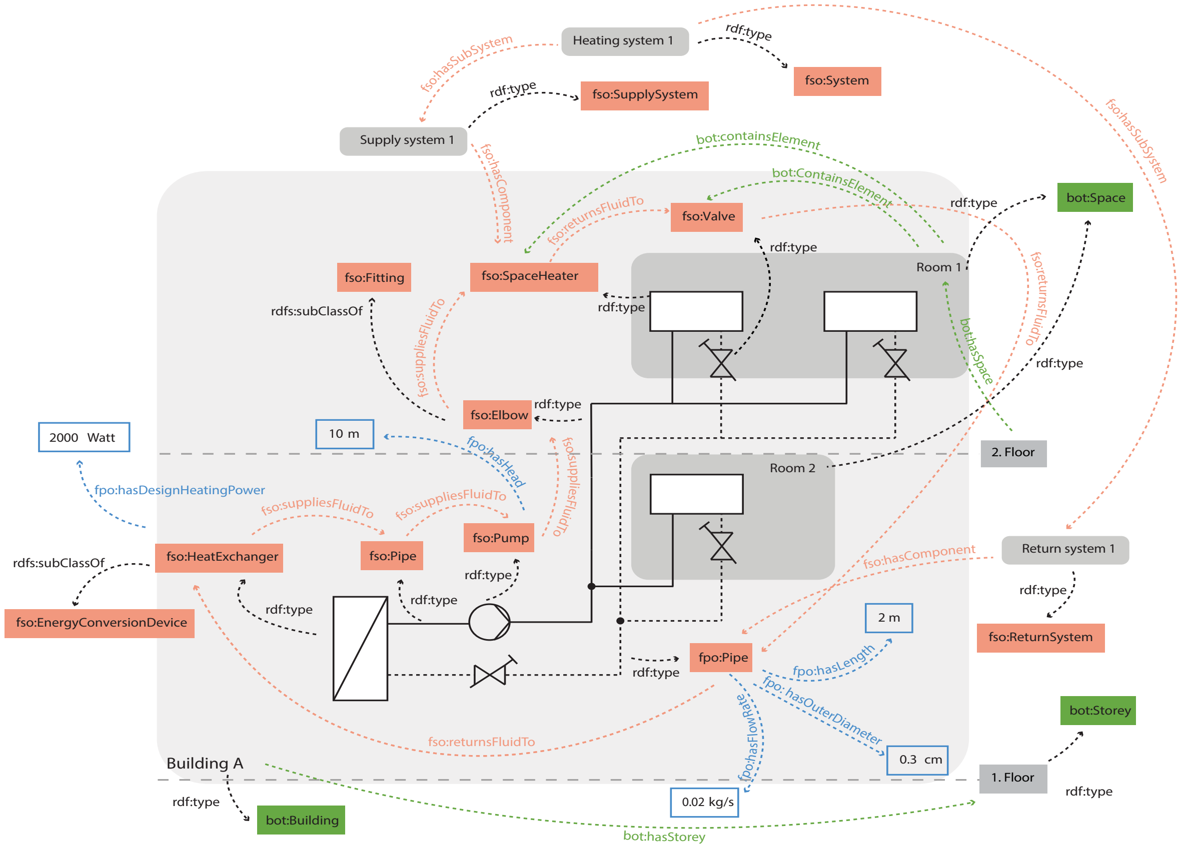

Figure 9: Combining multiple ontologies to represent building, spaces, systems, HVAC components, their properties and their relationships

Table 3: Alignments between FPO and s4bldg.

owl:Class	rdfs:subClassOf
owl:ObjectProperty	rdfs:subPropertyOf
	owl:equivalentClass
fpo:hasDesignAirflowRate	s4bldg:airFlowRateMin
fpo:hasCrossSectionalArea	s4bldg:faceArea
fpo:hasKv	s4bldg:flowCoefficient
fpo:hasHeight	s4bldg:height
fpo:hasOuterDiameter	s4bldg:inletConnectionSize
fpo:hasDesignVolume	s4bldg:waterStorageCapacity
fpo:hasPressure	s4bldg:openPressureDrop
fpo:hasOuterDiameter	s4bldg:outletConnectionSize
fpo:hasDesignHeatingPower	s4bldg:outputCapacity
fpo:hasOuterDiameter	s4bldg:outerDiameter
fpo:hasRoughness	s4bldg:roughness
fpo:hasThermalConductivity	s4bldg:thermalConductivity
fpo:hasVolume	s4bldg:volume
fpo:hasLength	s4bldg:length

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 GitHub[5]. 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.

[5] `https://github.com/Semantic-HVAC-Tool/Rule-Service/tree/main/Public/Shapes/fsosh.ttl`

The shape fsosh: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 GitHub[6]. 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.

```
fsosh:Pipe
    a sh:NodeShape;
    sh:nodeKind sh:IRI ;
    sh:targetClass fso:Pipe ;
    sh:property[
        sh:path fso:hasPort ;
        sh:dataType xsd:anyURI;
        sh:minCount 2;
        sh:maxCount 2;
        sh:message "A pipe must have exactly two flow ports"
    ]; #... 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

```
    BIND (((?capacity/?demand)-1)*10 as ?oversizing) .
    FILTER (?demand > ?capacity || ?oversizing > 10 )
    } """ ;]  .
```

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.

```
fsosh:PipePressureDrop
    a sh:NodeShape;
    sh:nodeKind sh:IRI ;
    sh:targetClass fpo:Pipe ;
    sh:sparql  [
        a sh:sh:SPARQLConstraint  ;
        sh:message "The pressure drop of a fso:Pipe shall
↪   not exceed 100 Pa/m";
        sh:prefixes (fpo: fso: inst:) ;
        sh:select """PREFIX fso: <http://w3id.org/fso#>
        PREFIX fpo: <http://w3id.org/fpo#>
        PREFIX inst: <https://example.com/inst#>
        SELECT ?this ?value
        WHERE {
        ?this a fso:Pipe .
        ?this fpo:hasLength ?length .
        ?length fpo:hasValue ?lengthvalue .
        ?this fso:hasPort ?port .
        ?port fpo:hasFlowDirection ?flowDirection .
        ?flowDirection fpo:hasValue "Out" .
        ?port fpo:hasPressureDrop ?pressureDrop .
        ?pressureDrop fpo:hasValue ?pressureDropValue .
        bind ((?pressureDropValue / ?lengthvalue) AS
↪   ?value) .
        FILTER (?value > 100)} """ ; ] .
```

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 PEX[7] 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-is[8]. 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 GitHub[9]. 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`

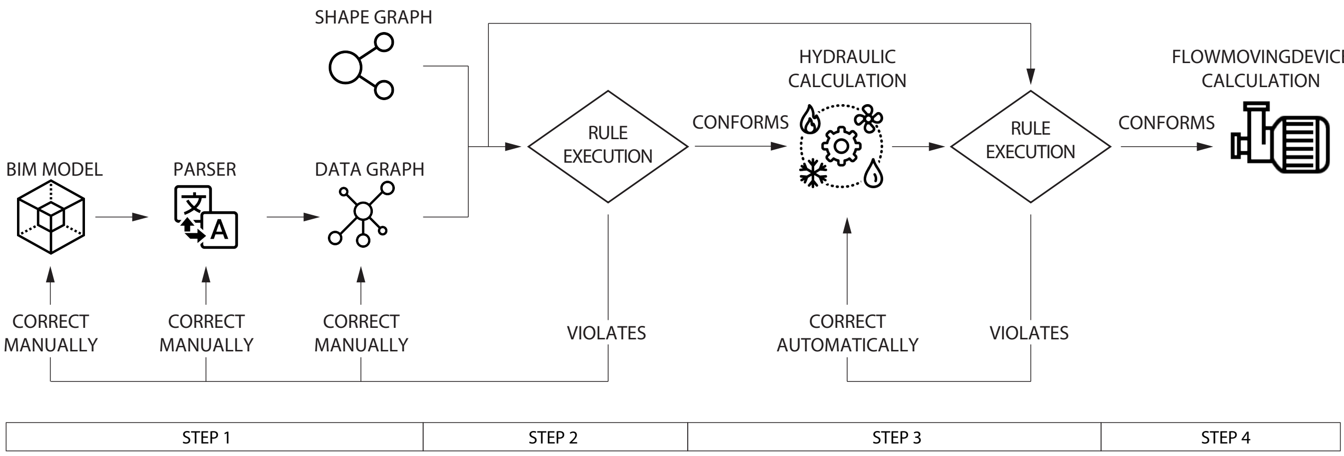

Figure 10: The process of performing conformance checking and design calculations for an HVAC model.

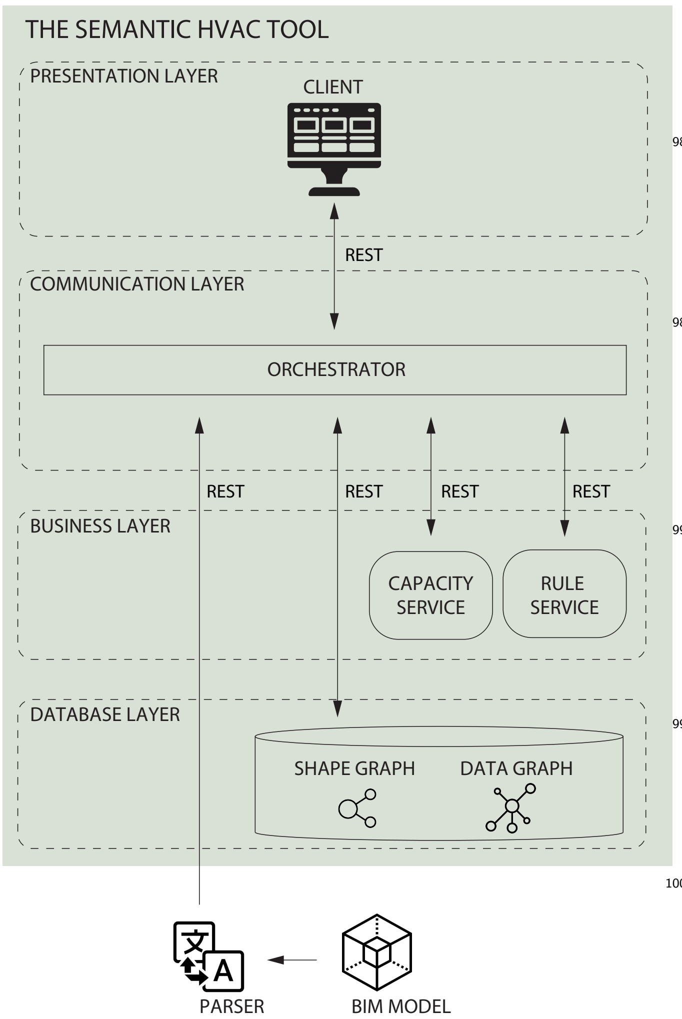

Figure 11: The system architecture of the Semantic HVAC Tool.

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 queries[10]. 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 parser[11] and the BIM model[12] 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 GitHub[13]. 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`

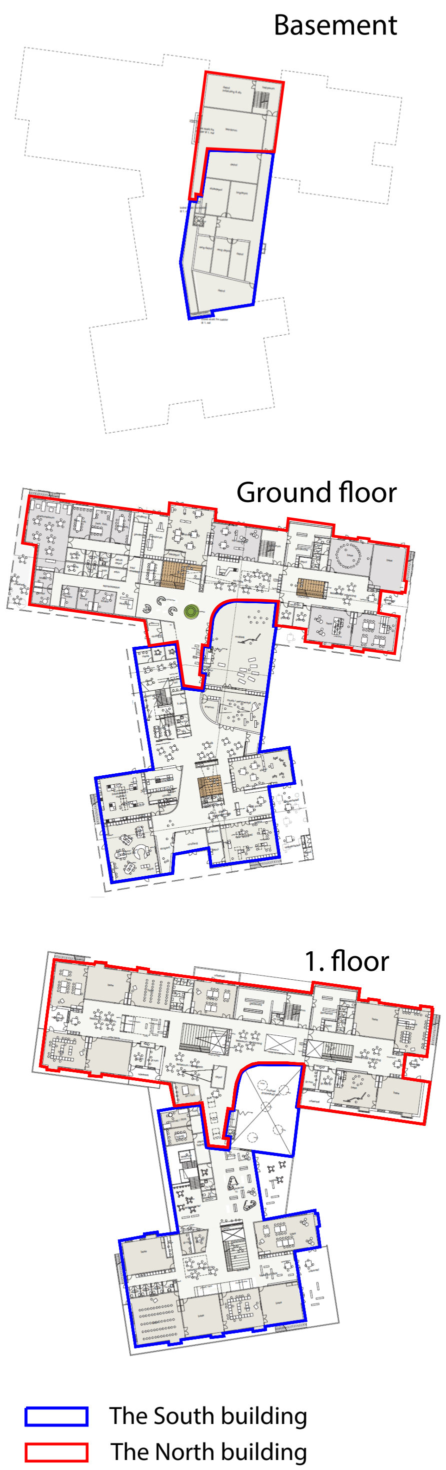

Figure 12: The illustration shows the floor plans of Frederiksberg Skole in Sorø, Denmark. The south building is marked with red, while the north building is marked with blue [73]

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.J

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 properties.

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.

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 second 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.

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 Ramboll Foundation; and COWI A/S. We thank Sorø municipality for providing the BIM model for Frederiksberg Skole.

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.

[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/

[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#

[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.

[7] A. Hogan, The Web of Data, 2020. doi:10.1007/978-3-030-51580-5.

[8] J. Flore, T. Djuedja, Integration of environmental data in BIM tool & Linked Building Data, Tech. rep.
URL http://www.enit.fr

[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.

[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. Karray, 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. 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

[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 LBDcall 20180312.

[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 interoperatibility 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. Afsari, 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. Mavrokapnidis, 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. Ardakanian, 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

```
DELETE {
  ?system a ?systemType .
  ?system ?systemPred ?systemObj .
  ?system fso:hasFlow ?flow .
  ?flow ?flowPred ?flowObj .
  ?flow fpo:hasTemperature ?temperature .
  ?temperature ?tempPred ?tempObj
  }
WHERE {
  VALUES ?systemType {fso:ReturnSystem fso:SupplySystem}
  ↪  ?system a ?systemType .
  ?system ?systemPred ?systemObj .
  ?system fso:hasFlow ?flow .
  ?flow ?flowPred ?flowObj .
  ?flow fpo:hasTemperature ?temperature .
  ?temperature ?tempPred ?tempObj
  FILTER NOT EXISTS {?system fso:hasComponent ?component}
  ↪  .
  }
```

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