Managing interrelated project information in AEC Knowledge Graphs

Mads Holten Rasmussen[a,*], Maxime Lefrançois[b], Pieter Pauwels[c], Christian Anker Hviid[a], Jan Karlshøj[a]

[a] *Technical University of Denmark, Department of Civil Engineering, Denmark*
[b] *Mines Saint-Étienne, Univ Lyon, Univ Jean Monnet, IOGS, CNRS, UMR 5516, LHC, Institut Henri Fayol, F-42023 Saint-Étienne France*
[c] *Ghent University, Department of Architecture and Urban Planning, Belgium*

Abstract

In the Architecture, Engineering and Construction (AEC) industry stakeholders from different companies and backgrounds collaborate in realising a common goal being some physical structure. The exact goal is typically not known from the beginning, and throughout all design stages, new decisions are made - similarly to other design industries [1]. As a result, the design must adapt and subsequent consequences follow. With working methods being predominantly document-centric, highly interrelated and rapidly changing design data in a complex network of decisions, requirements and product specifications is primarily captured in static documents. In this paper, we consider a purely data-driven approach based on semantic web technologies and an earlier proposed Ontology for Property Management (OPM). The main contribution of this work consists of extensions for OPM to account for new competency questions including the description of property reliability and the reasoning logic behind derived properties. The secondary contribution is the specification of a homogeneous way to generate parametric queries for managing an OPM-compliant AEC Knowledge Graph (AEC-KG). A software library for operating an OPM-compliant AEC-KG is further presented in the form of an OPM Query Generator (OPM-QG). The library generates SPARQL 1.1 queries to query and manipulate construction project Knowledge Graphs represented using OPM. The OPM ontology aligns with latest developments in the W3C Community Group on Linked Building Data and suggests an approach to working with design data in a distributed environment using separate graphs for explicit facts and for materialised, deduced data. Finally, we evaluate the suggested approach using an open-source software artefact developed using OPM and OPM-QG, demonstrated online with an actual building Knowledge Graph. The particular design task evaluated is performing heat loss calculations for spaces of a future building using an AEC-KG described using domain- and project specific extensions of the Building Topology Ontology (BOT) in combination with OPM. With this work, we demonstrate how a typical engineering task can be accomplished and managed in an evolving design environment, thereby providing the engineers with insights to support decision making as changes occur. The application uses a strict division between the client viewer and the actual data model holding design logic, and can easily be extended to support other design tasks.

Keywords: Linked Data, Building Information Modeling, Complex design, Ontology, Inference, Information Exchange, BIM, AEC Knowledge Graph, Linked Building Data

1. Introduction

The architecture, engineering and construction (AEC) industry is a fragmented industry, with information spread between numerous, changing stakeholders, including architects, engineers, contractors, subcontractors, owners, and so forth. This is one of the reasons why Bertelsen [2] describes it as a complex industry. All these stakeholders have varying levels of proficiency in digital technologies. With the advent of Building Information Modeling (BIM) tools, these varying levels of proficiency have become more apparent. They are often referred to

*Corresponding author
Email address: mhoras@byg.dtu.dk (Mads Holten Rasmussen)

as levels of maturity or levels of BIM adoption in the industry, and there exist frameworks for quantifying these [3, 4].

1.1. Document Exchanges at the Heart of AEC Project Design Workflows

Winch [5] describes that construction project teams can, in essence, be considered as information processing systems. This definition emphasises why an unobstructed information exchange between project teams is essential – an observation which is also backed up in various analyses of the construction industry [6, 7, 1]. The situation in the industry is, however, not without obstacles. Notwithstanding the significant shift towards BIM tools and digital tools in general in the construction industry, many project participants are still working in a highly document-centric manner where data is stored in a static, fragmented fashion in multiple heterogenous formats [8, 9]. Also, BIM-based workflows are often heavily document- or model-based, as the focus is on the exchange of files for achieving interoperability (BIM Level 2 as described in the PAS 1192 Specification [4]).

To make matters worse, design changes occur rapidly during the design, engineering and construction phases, so substantial rework must be done. Tracking design changes becomes a complicated task as these are often only documented in meeting memos, mail correspondences, on a Post-it on the project manager's table, or solely in the memory of the project participants [10, 11, 9]. As a result, the effects of a design change are so opaque that the consequences are hard to judge, sometimes resulting in critical after-effects, also in current BIM-based workflows. Further, substantial information losses occur each time an employee leaves the project [12].

Current BIM implementations do improve matters somewhat, as data structure and standardisation challenges are addressed. However, the fact that the majority of the industry is entirely document-centric, still has quite a big impact as well. No matter how much standardisation is put into file formats and exchanges, document-centric approaches result in parsing, interpretation, serialisation and deserialisation workflows that are bound to bring about inefficiencies and errors.

1.2. Shifting from the Exchange of Documents to the Management of AEC Knowledge Graphs

Considering that most documents are simply representations of data, we focus on the following research question in this paper:

> *How can we effectively store design data in a structured way, allowing interrelated data to maintain their relations intact as the project progresses, without losing the history of properties' progression?*

Recent research efforts have proposed the use of semantic web technologies to overcome the document-based attitude and enhance interoperability [13]. One main component of the semantic web is the design of *ontologies* which are by Studer et al. [14] defined as *"a formal, explicit specification of a shared conceptualization."* 'Formal' refers to the fact that it must be machine-readable. 'Explicit' means that the concepts used and the constraints on their use are explicitly defined, and 'Shared' entails that it describes consensual knowledge which is accepted by a group. With this work, we investigate existing ontologies in order to satisfy the Data on the Web best practice that consists in reusing existing vocabularies where applicable [15]. We consider what additional terminology is needed to answer the above research question, thereby providing such shared conceptualisation – this terminology, together with the project-specific assertions of a particular project, form what we in this article refer to as the *AEC-KG*.

1.3. Running Example: Calculation of Heating Demands for Spaces in a Building

We define a typical design task, conducted as part of a building design process that the research question can be evaluated against. The particular task is the calculation of heating demands for spaces in a building. The heating demand of a space is calculated for a steady state winter condition specified for the region in which the building is located and it has two components being (1) the infiltration heat loss which is constituted by the undesired ventilation through leaks, and (2) the transmission heat loss through the building envelope. The magnitude of the infiltration heat loss is dependent on the temperature difference between the air in the space and the outdoor as well as the ventilation rate which is typically estimated by the engineer as

a function of the space volume. The total transmission heat loss of a space is the sum of the individual building envelope segments that face the space. Each transmission heat loss is dependent on the geometric properties of the segment, the thermal properties of the particular building element and the temperature difference over the segment. During the design stages, the building's geometry occasionally changes, and this has consequences for both components of the space heat loss. Further, the resulting heating demands define the boundary conditions for the devices that heat up the spaces. These devices further constitute the boundary condition for the heating distribution system and hence all its sub-components. Therefore, this design task involves multiple information exchanges, and in a design practice with successive design iterations, it is a challenge that these exchanges are not handled dynamically. As the project evolves, this leads to inconsistencies, and it becomes a labour intensive task to assess the consequences.

1.4. Overview of the Outline of the Article and the Main Contributions

In this article, we will first give a brief overview of the state of the art in the use of semantic web technologies in the AEC industry along with an introduction to this topic (Section 2). Then, in Section 3, we will explain the design of the Ontology for Property Management (OPM) and give brief examples and indications of how the ontology, in combination with existing ontologies, can be used to keep track of the history, reliability and provenance of a property of some Feature of Interest (FoI). With FoIs, we refer to anything of relevance in a building to an AEC expert. This includes either spatial elements (spaces, zones, storeys), physical elements (walls, windows, heaters, sensors) or abstract elements (interfaces, systems, concepts). The core of OPM dealing with property change management was already presented in [16], but we extend it here also to provide terminology for describing property reliability, derived properties and calculations for formalising reasoning logic (Section 3). The ontology extensions for describing property reliability and calculations along with best practice modelling examples are the first and main contribution of this work. The second contribution is the specification of a standardised way to generate parametric queries for managing an OPM-compliant AEC-KG. In Section 4, considerations concerned with this management is discussed in detail and in Section 5, a JavaScript-based Application Programming Interface (API) that facilitates the creation of uniform, reliable queries for retrieving, creating and updating OPM properties and calculations is presented. The API provides a crucial addition to the proposed OPM ontology. Namely, considering the expressiveness and complexity of OPM, end-user applications wishing to implement the proposed property management need a middeware that allows them to define the desired queries towards an OPM-compliant AEC-KG. The OPM Query Generator (OPM-QG) provides a reference implementation of such a middleware API. In Section 6, we demonstrate a Proof of Concept (PoC) open-source implementation built on top of the OPM infrastructure which assesses the practical design case described above. This software is demonstrated online with an actual building Knowledge Graph. The architecture and considerations in this regard are discussed in detail in this section. Finally, we conclude the work and present our visions for future work.

2. State of the Art

In this section, we first investigate existing research efforts and communities dealing with describing AEC knowledge in graphs. We then look into different approaches to property assignment and what benefits each of these possesses. Lastly, we look into existing efforts in dealing with the handling of property interdependencies and deduction of implicit knowledge from BIM models.

2.1. Web Ontologies for AEC Knowledge Graphs

Researchers in the linked data and semantic web domain have recently aimed at making building data available on the web, linking data rather than documents [17, 18]. This group of researchers in the AEC domain has been gathering behind the recent initiatives around linked data in architecture and construction, which includes the Linked Data Working Group (LDWG) in buildingSMART International (bSI)[1], and the Linked Building Data (LBD) Community Group[2] at the World Wide Web Consortium (W3C). In both standardisation bodies, ontologies are proposed for capturing building

[1] `http://www.buildingsmart-tech.org/future/linked-data`

[2] `https://w3.org/community/lbd/`

data using web technologies. The groups focus on the use of semantic web technologies, namely the Web Ontology Language (OWL) and the Resource Description Framework (RDF) [19, 20], thus creating smaller aligned ontologies and putting them on a track towards standardisation. Aligned in this regard meaning that they extend or comply with terminology from the other ontologies.

buildingSMART is the standardisation organ who maintains the standard exchange format for BIM data models: the Industry Foundation Classes (IFC) standardised by ISO 16739 [21].

The LDWG remains entirely in the buildingSMART realm, focusing mainly on the production of the ifcOWL ontology[3] [22], which was initially proposed by Beetz et al. [23]. The ifcOWL ontology is set up to be a direct translation from the IFC schema represented in the EXPRESS data modelling language [24] into an OWL representation. This has resulted in an extensive ontology, much unlike many of the existing ontologies in various other domains, that are typically more narrow scoped and depend on extensions enabled by linked data principles. For this reason, there have also been various attempts on simplifying this ontology. For example, Terkaj and Pauwels [25] worked on the modularisation of this ontology. Other approaches like IFCWoD (IFC Web of Data) [26], SimpleBIM [27], and BimSPARQL [28] aimed at providing simplified views over ifcOWL data. IFCWoD implements a set of rules within the triple store; SimpleBIM implements rewrite rules in code; and BimSPARQL simplifies ifcOWL data by extending the query language for RDF, SPARQL [29], with rewrite rules and geometry calculation algorithms. None of the simplification approaches proposes and defines an explicit OWL ontology. The BIM Shared Ontology (BIMSO) and BIM Design Ontology (BIMDO) together constitute another modularity approach with ontologies that are designed from scratch and, therefore, have no connection to IFC. BIMSO has a minimal core and builds on the UNIFORMAT II classification system and BIMDO provides specific terminology for design [30]. These are, unfortunately, not publicly available.

The W3C LBD Community Group, has focused mostly on the creation of ontologies for capturing building information from close to scratch. By starting from close to scratch, ontology engineering best practices can be more easily maintained. For example, it is possible to reuse existing ontologies and develop minimal extensions in a modular fashion. This is possible because RDF uses International Resource Identifiers (IRIs) to denote resources (i.e. something in the world) [20]. The Linked Data rules [31] further requires the use of HyperText Transfer Protocol (HTTP) IRIs like the ones used when browsing the web, thereby making it possible to provide useful information about a resource when someone looks it up in a web browser.

The W3C group currently focuses on the development and maintenance of a Building Topology Ontology (BOT) [32] and demonstrating best practices for publishing building products and associated properties on the web using Linked Building Data principles.

BOT was originally proposed by Rasmussen et al. [33] who examined existing ontologies in the scope of buildings and found that they all redefined the same basic elements (e.g. spaces, storeys and elements) and topological relationships between these. Therefore, a minimal, extensible ontology for this sole purpose was proposed. Later it has been developed as a community effort by the W3C LBD Community Group [32].

2.2. Three Levels of Complexity for Design Property Descriptions

When assigning properties to some FoI, there are different considerations to illuminate. Is there a need to assign a physical unit? Is it necessary to capture metadata such as which property set a certain property belongs to? Is there a need to capture provenance data, such as, who created the property or what other properties or processes it was derived from? Should it be possible to change the value of this property, and in this case, should some record of state changes be maintained?

Bonduel [34] provides a visual presentation, thereby comparing different modelling approaches for property assignment. Figure 1 compares the ifcOWL and SimpleBIM approach to describing that some slab is load bearing. SimpleBIM uses the most straightforward approach by simply describing properties defined as OWL Data Properties [19, §5.4] with a literal value assigned. ifcOWL is noticeably more complex since (1) the property is assigned through a property set which is assigned through a relational node, (2) the property is not directly assigned to the property set but requires

[3] `http://www.buildingsmart-tech.org/ifcOWL/IFC4#`

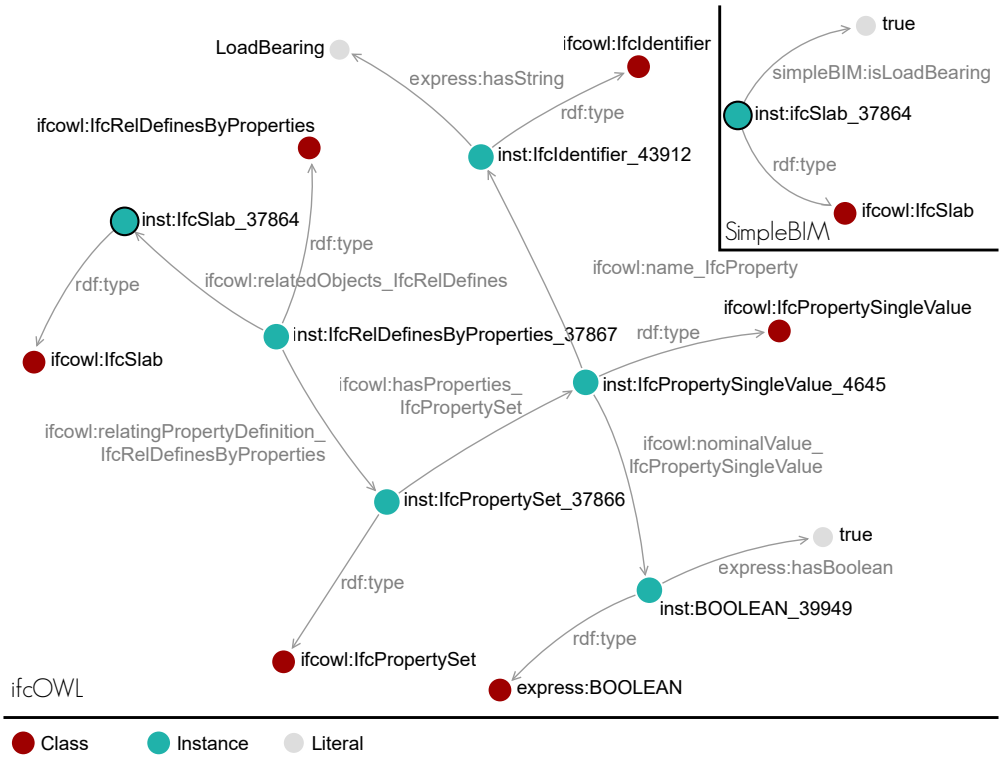

Figure 1: Visual complexity comparison of representing property assignment using ifcOWL and simpleBIM.[4]

an intermediate node which refers to two different nodes holding the name and the value of the property, and (3) the literal is described using an EXPRESS datatype.

Listing 1 shows how SimpleBIM properties encoded in the Turtle [35] serialisation format for RDF data models. In RDF, all statements are described in triples consisting of a *subject*, a *predicate* and an *object*. The first triple in the listing describes the *subject* `inst:ifcSlab_37864` which by the *predicate* `rdf:type` (often abbreviated 'a') is asserted to be an instance of the class (the *object*) `ifcowl:IfcSlab`. A period '.' marks the end of a triple. Semicolon ';' also marks the end of a triple, and further denotes that subject is not repeated in the subsequent triples. A comma ',' denotes that both the subject and predicate are not repeated in the next triples. All IRIs are prefixed but the namespaces associated with the prefixes are not shown in the listings. In this article, `inst:` is a generic prefix used to describe instances and all other prefixes relate to ontologies that are either described in this section or discoverable at http://prefix.cc/{prefix}. The object of the second triple is a literal value and the IRI after '^^' specifies the datatype of the literal. In this case a boolean as described by the ontology version of the XSD schema.

Listing 1: Property assignment in Turtle syntax.

```
# SimpleBIM
inst:ifcSlab_37864 rdf:type ifcowl:IfcSlab ;
    simpleBIM:isLoadBearing "true"^^xsd:boolean .
```

Examining the different approaches for property assignment reveals a high variance in complexity. The most simplistic form, direct assignment of datatype properties, reduces the complexity of the queries and thereby makes it easier to navigate the graph. Typed literals can encode information such as the unit of measure as proposed by Lefrançois and Zimmermann [36] with the Unified Code of Units of Measures datatype. However, no additional metadata can be included.

Rasmussen et al. [16] describes three levels of complexity, where each level refers to the number of steps between the FoI and the actual object (literal

[4] Visualization obtained using SPARQL visualizer `https://github.com/MadsHolten/sparql-visualizer`

or individual) that encodes the value of its property. L1 is equal to the approach used by SimpleBIM. L2 (used by IFCWoD and BIMDO) describes the property as an object, and thereby allows attaching metadata such as a unit of measure using a dedicated ontology like Quantities, Units, Dimensions and Types (QUDT) [37] or provenance data using the PROV Ontology (PROV-O) [38]. L3 is used by OPM and is inspired by the Smart Energy-Aware Systems (SEAS) evaluations [39], which is aligned with the joint W3C and OGC SOSA/SSN standard specifying the semantics of sensors, observations, sampling, and actuation [40, 41, 42]. By assigning multiple property states to one property, L3 property assignment allows the property value to change over time while keeping a record of how it evolved. The property assignment complexity in ifcOWL exceeds L3 by magnitudes but does not add functionality other than backward compatibility with IFC.

2.3. Handling Interdependent Properties

Isaac et al. [43] suggest handling the complexity of construction projects by modelling the projects' topology in graphs. Other recent research projects have further illustrated how relationships in building data beyond the geometrical ones can be handled by the use of semantic web technologies. This further introduces the capability of using reasoning engines to infer implicit information that is not directly asserted in the AEC-KG, but that can be deduced from the facts that are present, thereby making them explicit facts. This technology is heavily used to make machines capable of 'reading between the lines' to provide enhanced results from search engines and so forth.

Deducing implicit facts from prose text using description logic is not remarkably different from the work of an engineer who puts various inputs into equations in order to generate new outputs. This is also in accordance with the analogy by Winch [5] which compares construction project teams to information processing systems. The problem is that the information processing systems are today constituted by the knowledge workers and the software tools they use. This challenge was studied by Pauwels et al. [44] who considers the use of rule-checking environments to formalise the knowledge of the human workers. In particular, semantic web technologies are used, to establish an AEC-KG consisting of (1) explicit building information parsed from an IFC file, (2) an ontology parsed from the IFC schema, and (3) a set of rule-sets describing implicit engineering knowledge. The concrete case of performing compliance checking of acoustic performance is presented with this work as a proof of concept. This novel approach demonstrates how a typical task for a knowledge worker can be explicitly described in reuseable rule-sets, and thereby it represents an entirely different data-driven approach to a labour-intensive job. A more recent study by Zhang et al. [28] uses a similar approach with the purpose of (1) providing shortcuts to make an ifcOWL graph easier to query and (2) deduce geometric information from 3D geometry. The baseline of both studies is a final BIM model, and thereby it differs from the situation investigated with this work.

Both Pauwels et al. [44] and Zhang et al. [28] use rules to infer results at run-time. Providing derived properties at run-time rather than materialising them in the graph has the benefit that outdated or redundant information is avoided. Zamanian and Pittman [45], however, argues that since AEC projects are performed by distributed teams, it is often desirable to have some consistent redundancies that are designed and managed to provide more efficient means to access and manipulate the information. Some party might wish to refer to an intermediate result of a derived property, and this is not immediately possible if that information is only described in a rule. Also, with OPM, the intention is that interdependencies are explicitly stated so that the knowledge workers are provided with insights. The initial work presented by Rasmussen et al. [16] suggests that every state of a property is saved in order to evaluate design changes, and this is only possible when materialising reasoning results.

3. An Ontology for Property Management (OPM)

The Ontology for Property Management (OPM) was initially introduced by Rasmussen et al. [16]. It provides terminology for modelling complex properties in a design environment. Complexity, in this regard, entails that they (1) change over time (2) can be assumptions and hence comprise varying reliability, and (3) can be dependent on the value of other properties. The first contribution demonstrated how property assignment modelled according to this ontology can be used to answer a set of competency questions that are all concerned with managing evolving properties. Sec-

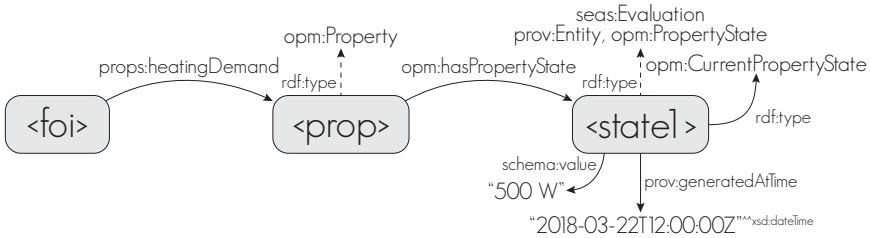

Figure 2: L3 property assignment using an opm:PropertyState for assigning the latest value and metadata describing when it was created [16].

tion 3.1 first overviews the concepts of the OPM ontology that was presented in the initial OPM paper. Then subsequent sections define additional sets of competency questions that deal with reliability (Section 3.2) and handling of interdependencies between properties (Section 3.3). We propose a set of new concepts extending the initial version of OPM to provide terminology to answer these questions. Like the original OPM ontology, the extensions depend on concepts from the SEAS [39], schema.org and PROV-O ontologies and align well with the BOT, PROPS and PRODUCT ontologies of the W3C LBD Community Group. OPM uses the namespace `https://w3id.org/opm##` and the documentation is provided when this IRI is visited in a web browser.

3.1. Property History

Rasmussen et al. [16] described seven competency questions, all dealing with property history modelling are listed below.

CQ 1.1 How to semantically describe a property such that its value is changeable while its historical record is maintained?

CQ 1.2 How to revise a property value?

CQ 1.3 How to delete a property while still being able to retrieve the history of it and not break all the links to derived properties that depend on it?

CQ 1.4 How to restore a deleted property?

CQ 1.5 How to retrieve the full history of how the value of a property has evolved over time?

CQ 1.6 How to retrieve only the latest value of a property?

CQ 1.7 How to simplify a complex OPM property (using states) for easier and faster querying?

To answer these questions, the authors describe three different levels of property assignment with varying expressivity. The level in this regard refers to *"the number of steps/relations between the FoI and the actual object (literal or individual) that encodes the value of its property."* The most expressive level, L3, is used in OPM to capture property changes over time. It uses the concept of property evaluations from SEAS. Figure 2 illustrates how the opm:PropertyState (subclass of seas:Evaluation) can be used to capture a state of a property. It is assigned to a property using the opm:hasPropertyState (sub-property of seas:evaluation) predicate, and the rdfs:range of this predicate implies that it belongs to the opm:PropertyState class. OPM depends on schema:value, schema:minValue and schema:maxValue to assign a value to a property state, and, as a minimum, the state must further have a prov:generatedAtTime predicate. Retrieving the most recent state of a property can be achieved by querying for the highest prov:generatedAtTime value, but this (1) increases the query complexity and (2) reduces query performance [16]. Therefore, the opm:CurrentPropertyState class is always assigned to the most recent state and removed from the previous state when performing SPARQL [29] update queries on the AEC-KG. The opm:OutdatedPropertyState class can, in this case, be assigned to the outdated property state.

A different design pattern which is also supported by OPM is property assignment by classification. In this case, a generic property such as `props:hasProperty` is used as predicate and the object (the property) is classified according to the type of property. For example: `<foi> props:hasProperty <prop> . <prop> rdf:type props:NominalUA .`

As illustrated in Figure 3, changing a property's value is handled by assigning a new property state holding the new value and other metadata such as provenance. Thereby, it is possible to retrieve the

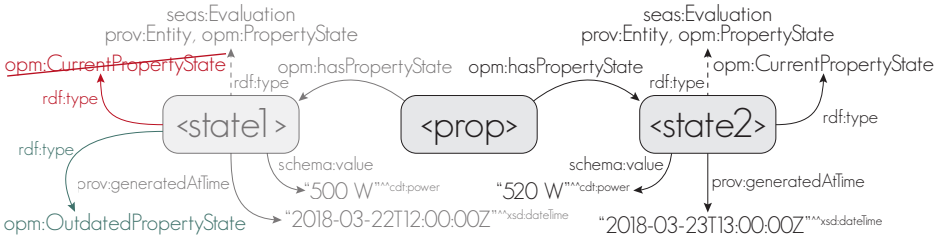

Figure 3: Changing a property is handled by assigning a new property state to the property and removing the opm:CurrentPropertyState from the state that was previously defined as the current property state. Assigning the previous state as opm:OutdatedPropertyState is optional [16].

full history of a property and restore previous properties if necessary. Changing and restoring properties can be handled with standardised SPARQL update queries that are executed against the AEC-KG by client applications.

In order to maintain the history of the project evolution and to be able to revert to an earlier stage, data should never be removed from the AEC-KG. Marking a deleted property state as an instance of both opm:CurrentPropertyState and opm:Deleted provides filtering options, and thereby the state can be stored while hidden from end users. A deletion is reverted by inferring a new state that inherits the properties of the most recent state with a value assigned to it. Having both the initial state, the deleted state and the restored state available allows for tracking when and by whom the different changes were conducted.

3.2. Property Reliability

The AEC industry is a complex industry where each project is constantly changing during the design and planning stages. As part of this work, we had discussions with industry professionals who described working methodologies that are dependent on being able to assess the quality of any data in the project. As a result, we defined the following set of additional competency questions for property reliability:

CQ 2.1 How to describe that the value of some property is defined temporarily until the actual value is known?

CQ 2.2 How to document that some property has been confirmed and can hence be trusted not to change in the future?

CQ 2.3 How to describe that some property is derived from, and hence dependent on the value of some other property?

CQ 2.4 How to describe a property requirement?

All these questions require terminology that was not initially part of OPM, but in the following, each question will be answered by adding a specific class to the ontology.

CQ 2.1, How to describe that the value of some property is defined temporarily until the actual value is known?. OWL class opm:Assumed
In the early stages of any construction project, it is common practice to make temporary assumptions in order to progress with the design. Assumptions are typically managed in assumption lists, conducted as simple documents such as a spreadsheet, and it is the project manager's job to make sure that all assumptions are later clarified and confirmed by a person in charge of the specific domain. An assumption is thus defined here as a non final value with a high probability of changing over time as the project evolves. Classifying a property state as opm:Assumed describes that the value is temporary and must later be confirmed. A querying for all the states that belong to this class reveals which properties are yet to be confirmed.

CQ 2.2, How to document that some property has been confirmed and can hence be trusted not to change in the future?. OWL class opm:Confirmed
A confirmed value is approved by a person who has the authorisation to do so. Classifying a property state as opm:Confirmed indicates that its value can be trusted not to change in the future. For legal documentation purposes, a digital signature can be

assigned to the property. Also, a link to documentation such as a mail, a scanned contract or similar can be attributed to the property state using the opm:documentation predicate. As it is common practice to set milestones in the course of a construction project, at which certain parameters are locked, these are obvious opportunities to mark all related object properties as confirmed. An example of a milestone is the date where the layout of the superstructure is frozen and concrete elements are ordered from the manufacturer. Changing the design after this date is possible, but it will likely lead to a cost penalty that someone needs to pay.

CQ 2.3, How to describe that some property is derived from, and hence dependent on the value of some other property?. OWL Class opm:Derived
Engineering is chiefly a discipline of gathering information, processing that information, typically by applying math and physics, and thereby deducing new information. A simple example is the area of a window, which is either directly deduced from its geometrical definition or by a product of the height and width properties. If the value of a property is dependent on other properties, it should have the opm:Derived class applied. Keeping derived properties up to date can be automated, but in many occasions it is desirable for the engineer to keep the design as is, knowing that the property is no longer valid. Only when the consequence of the change is significant, the design is revised. It might also be that the consequences are not acceptable, and in this case, the party who made the initial change must instead be asked to revert the change. The engineer can be supported in this decision by dynamically deriving the new result while calculating the deviation from the static result. Also, it can be explicitly stated that the state belongs to the class opm:OutdatedPropertyState. Next section deals specifically with the handling of derived properties.

CQ 2.4, How to describe a property requirement?. OWL class opm:Required
Requirements can be assigned to abstract product or space models holding the prerequisites for a design. Use cases for this include space schemas holding functional requirements of the spaces of a future building, design models holding functional requirements for mechanical equipment, and so forth. Rasmussen et al. [46] implements this feature to compare client requirements to a building with the actual design. It is thereby possible to query the AEC-KG for all properties that do not fulfil the requirements that have been defined. OPM allows both requirements and design values to change over time, and therefore, the consequence of a violated requirement must ultimately be that either the requirement itself or the violating design value needs to adapt.

3.3. Property Interdependence

The opm:Derived class enables to describe that some property is derived from one or more other properties. There are, however, more things to consider when dealing with interdependent properties such as traceability and quality assurance. We defined the following set of additional competency questions to capture these:

CQ 3.1 How to associate a derived property to the properties from which it was derived?

CQ 3.2 How to identify that a derived property is outdated?

CQ 3.3 How to formally describe a calculation that can be applied to infer derived properties?

CQ 3.4 How to associate a derived property to the calculation or algorithm that formalises how it was derived?

CQ 3.5 How to check for circular dependencies in derived properties?

CQ 3.6 How to define the reliability of a derived property?

CQ 3.7 How to check which derived properties will be affected if a specific property is changed?

OPM does not restrict how derived properties are inferred and whether this is accomplished with runtime inference or by materialising the derived properties in the graph. It does, however, define a best practice approach for modelling interdependencies in a way that satisfies the answering of above competency questions. A derived property is modelled like any other OPM property, but it is classified as opm:Derived and its value is inferred instead of being typed.

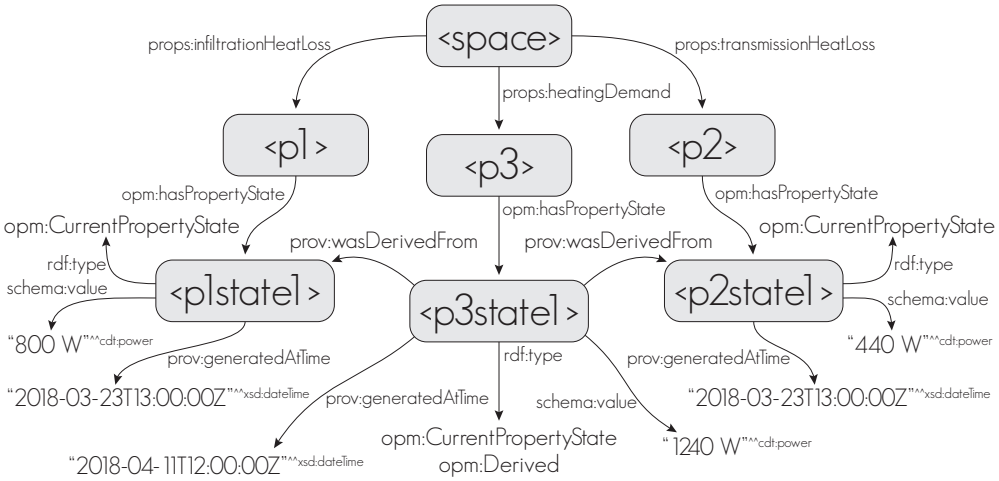

Figure 4: When inferring derived properties, they should be connected to the property states from which they are derived using a prov:wasDerivedFrom predicate.

CQ 3.1, How to associate a derived property to the properties from which it was derived?.
In order to associate a derived property with the properties from which it was derived each state of the derived property must be linked to the states of the properties from which it was derived (its arguments). Figure 4 illustrates how this is achieved using the prov:wasDerivedFrom predicate.

CQ 3.2, How to identify that a derived property is outdated?.
The most recent state of a derived property is classified as opm:CurrentPropertyState. Since this state is related to the states of the properties from which it was derived using the prov:wasDerivedFrom predicate it is possible to check whether these states are also classified as opm:CurrentPropertyState. If this is not the case, the derived property is outdated.

CQ 3.3, How to formally describe a calculation that can be applied to infer derived properties?. OWL class opm:Calculation
OPM includes the concept of calculations which formalises the specification of the reasoning logic. An instance of the opm:Calculation class holds such specification. As a minimum, a calculation must describe the IRI of the inferred property using predicate opm:inferredProperty, an expression using predicate opm:expression and a path from the FoI to each of the arguments using predicate opm:argumentPaths. The latter is described as an RDF list where each path is stated as a literal (See example in Listing 2). Figure 5 shows the calculation that inferred the derived props:heatingDemand property from Figure 4. The calculation describes the heating demand as the sum of the infiltration heat loss and the transmission heat loss and since both these properties are directly assigned to the space itself, the calculation is relatively simple. The opm:expression defines the result as the sum of the two variables `?htr` and `?inf`. The paths from the space to the two arguments are defined as `"?foi props:transmissionHeatLoss ?htr"` and `"?foi props:infiltrationHeatLoss ?inf"`. The number of argument paths must correspond to the number of variables, but the paths can be extended to restrict the results further. For example it can be specified that the FoI must be an instance of bot:Space by extending the first path to `"?foi a bot:Space ; props:transmissionHeatLoss ?htr"`. Listing 2 shows how Figure 5 is described using the Turtle syntax for RDF.

Listing 2: Calculation from Figure 4 in Turtle syntax.

```
inst:c1  rdf:type                opm:Calculation .
inst:c1  opm:expression          "?htr+?inf" .
inst:c1  opm:inferredProperty    props:heatingDemand .
inst:c1  opm:argumentPaths (
           "?foi props:transmissionHeatLoss ?htr"
           "?foi props:infiltrationHeatLoss ?inf" ) .
```

The opm:expression is preferably defined in SPARQL 1.1 syntax, which includes methods that are sufficient for defining the simple calculations that are extensively used in engineering. Since the

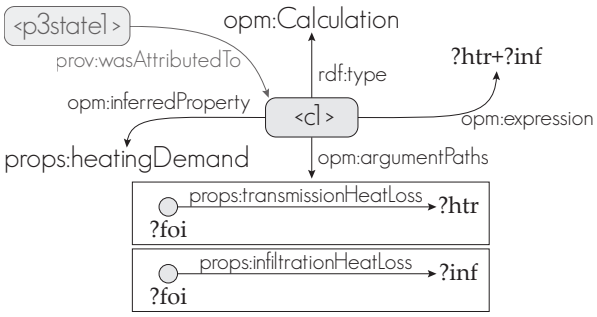

Figure 5: Describing a calculation with OPM. The dimmed part in the upper left corner shows how states inferred by the calculation can be attributed to the calculation from which they were inferred. `<p3state1>` refers to the inferred state illustrated in Figure 4.

expression is assigned as an OWL Data Property, it is also possible to describe more complex expressions. Expressions are expected to be encoded in the SPARQL 1.1 syntax, but other languages such as Javascript can be used as well. In this case, the datatype of the expression should be the JavaScript mediatype IRI iana:application/javascript[5].

The algorithm in Figure 6 shows the intended use of OPM calculations to generate reasoning results. The first step is to retrieve the calculation data. If the expression is not defined using a special datatype, it is expected to be described in SPARQL syntax. In this case, the expression is analysed, and if it contains an aggregation function such as sum, min, max, avg or count, an aggregation sub-query is constructed based on the single argument given by the opm:argumentPaths property. If it does not contain an aggregation function, the latest state of each argument is retrieved, and the expression is applied to the query using a `BIND` form, which allows the assignment of a value to a variable[6]. The naming of the variables used in the argument paths must match the arguments used in the expression (e.g. `?htr` and `?inf` in Listing 2). Listing 3 shows some examples of SPARQL 1.1 expressions that can be used in expressions. The result of the three first expressions can be bound to a `?result` variable using a `BIND` form. The latter requires an aggregation sub-query which can bind the result to a `?result` variable.

If the expression is some JavaScript procedural code, the arguments must first be retrieved using a query, and afterwards, the result can be calculated. This is the only valid approach if more complex calculations like simulations are needed.

Listing 3: Four example expressions for the opm:expression datatype property.

```
# Math expression
"math:sqrt( math:pow(?arg1, 2) + math:pow(?arg1, 2) )"
# String operation
"STRAFTER(?arg1,'/')"
# Conditional expression
"IF(?arg1 > ?arg2, 'true', 'false')"
# Aggregation expression
"SUM(?area)"
```

CQ 3.4, How to associate a derived property to the calculation or algorithm that formalises how it was derived?.
In order to associate a derived property with the calculation from which it was derived, a derived property should be linked to the particular opm:Calculation instance using the prov:wasAttributedTo predicate. This is illustrated in Figure 5.

CQ 3.5, How to check for circular dependencies in derived properties?.
Properties can be derived from properties which are themselves derived, and so forth. Since all property states are interlinked with the prov:wasDerivedFrom predicate, it is possible to retrieve the full set of property states from which a property is derived. A property cannot be derived from itself, and in order to check that no circular dependencies exist in the graph, the query from Listing 4 can be used. This query uses a SPARQL property path (OneOrMorePath, [29]) to ask for all arguments from which a property state is derived, and a filter to only return results where the state is dependent on itself.

Listing 4: Query to check for circular dependency.

```
SELECT ?propSt
WHERE {
    ?propSt prov:wasDerivedFrom+ ?arg .
    FILTER(?propSt = ?arg)
}
```

CQ 3.6, How to define the reliability of a derived property?.
The reliability of a derived property should also not be specified manually, but should instead be inferred from the properties from which it is derived

[5] https://www.iana.org/assignments/media-types/application/javascript
[6] https://www.w3.org/TR/sparql11-query/#bind

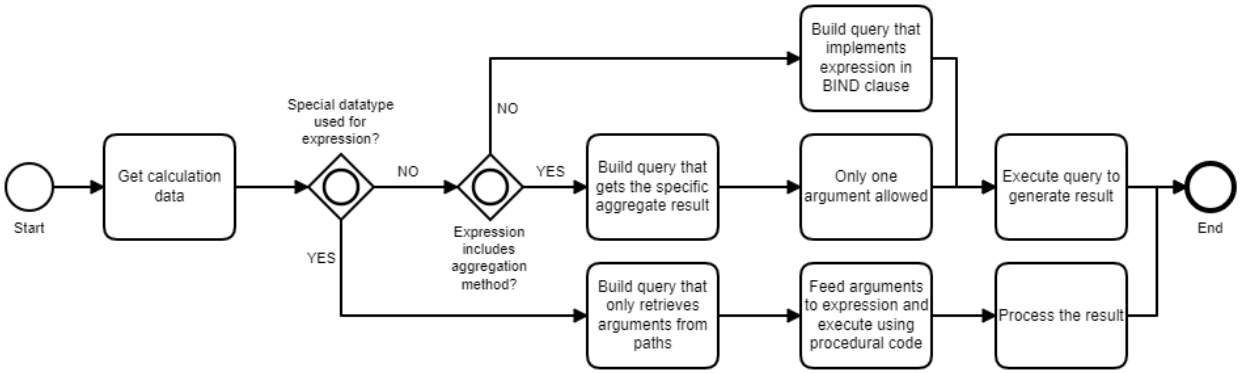

Figure 6: Algorithm to yield results from a opm:Calculation.

(the arguments). If one or more of these belong to the class opm:Assumed, the same is the case for the derived property. Likewise, if an argument is of type opm:Deleted, the derived property is also no longer valid (i.e. deleted). Inheritance of the opm:Confirmed class to a derived property requires all the arguments to be confirmed.

CQ 3.7, How to check which derived properties will be affected if a specific property is changed?.
In a construction project, it is practically impossible to know the derived properties that other parties might have created. With OPM, however, the derived properties are related to the property from which they are derived, and hence it is possible to check what derived properties are affected when changing a property.

4. Reasoning with the OPM Ontology

This section describes the use of two main components of the semantic web in relation to OPM: reasoning (Section 4.1), and use of named graphs (Section 4.2). It further describes the use of SPARQL queries for inferencing (Section 4.3).

4.1. Materialising Derived Properties

Dealing with derived properties can be achieved either by inferencing upon request (deduce results at runtime) or by materialisation (saving the results) in the AEC-KG. Below, we justify why we recommend the latter approach for OPM. Table 1 compares the two approaches qualitatively.

Table 1: Runtime vs. materialised inferencing.

	Runtime	Materialised
Validity	●	◑
Performance	○	●
Traceability	○	●

Validity. When materialising the results of derived properties, there will at some point exist data that is no longer valid. Inferencing upon request will always provide the correct result based on the most recent state of all arguments. It is, however, possible to check whether a derived property is outdated. This check could be automated in order to infer the opm:OutdatedPropertyState class to the property states that are no longer valid using the logics described in *CQ 3.2*.

Performance. As properties can be derived from other derived properties, the dependency chain can become long, and hence the reasoning engine needs to loop over the data several times to saturate the graph. The resulting performance drop can be a significant drawback for inferencing upon request. As complexity grows, performance in materialising derived triples will also decrease, but this task can be performed in the background, thereby not sacrificing the user experience.

Traceability. Materialising every single state of a derived property and the specific states of the properties from which it was derived, provides valuable insights. This increases transparency and allows for more in-depth analyses of embedded consequences of particular changes. It is easy to

imagine that the management of interdependencies will eventually become a chaotic task when everything is dynamic and automatically updating as the design changes. In a construction project, it might for some reason be desired to stick to the value as it is, knowing that the inputs have changed slightly for some reason not known to the reasoner. The missing traceability when having a calculation performed at runtime furthermore entails some legal implications concerning responsibility.

4.2. Separation of Explicit and Inferred Triples

To distinguish inferred triples from explicit triples, they can be stored in separate named graphs in the same database. Carroll et al. [47] describes several purposes for named graphs. With this work, we suggest that they are used here to separate explicit triples from derived triples. The triples that are inferred from those in graph `IRId` are then stored in a second named graph denoted `IRId-I`. This provides a mechanism to remove all inferred triples and re-establish them from the most recent state of all arguments in cases where the history of derived properties is not important.

4.3. Inferencing with SPARQL queries

Materialising derived properties based on calculation data can be achieved with SPARQL update queries. Listing 5 shows a SPARQL update query that will append derived properties based on the calculation shown in Listing 2. The query appends the derived property props:heatingDemand in the graph of inferred triples <https://host/project−I> for all FoIs that satisfy the argument paths specified in Listing 2, but only if the property is not already assigned. The query further adds the opm:Derived and opm:Assumed classes to the newly generated property state, and the generation time.

As a matter of fact, this query may be generated automatically from the opm:Calculation instance of Listing 2 using the algorithm shown in Figure 6.

Materialised derived properties may become outdated, as they depend on arguments that could potentially change after materialisation. A similar query may be constructed to update derived properties where one or more of the arguments have changed.

These challenges may be addressed using incremental reasoning [48], or defeasible reasoning approaches [49].

5. An API to interact with OPM Data

In the description of CQ 3.3 in Section 3.3 we described the algorithm illustrated on Figure 6 for automatically generating reasoning results from opm:Calculation instances. Listing 5 illustrates that parametric SPARQL queries can be used to work with OPM properties in general. This section introduces an API to interact with OPM data using parametric SPARQL queries.

Listing 5: SPARQL query to append a calculation.

```
PREFIX opm: <https://w3id.org/opm#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX schema: <http://schema.org/>
PREFIX props: <https://w3id.org/props/>
INSERT {
  # insert in the graph of inferred triples
  GRAPH <https://host/project-I> {
    ?foi props:heatingDemand ?propertyIRI .
    ?propertyIRI opm:hasPropertyState ?stateIRI .
    ?stateIRI a opm:CurrentPropertyState ,
                    opm:Derived , opm:Assumed ;
        schema:value ?res ;
        prov:generatedAtTime ?now ;
        prov:wasDerivedFrom ?state1 , ?state2 .
  }
}
# using both the explicit and the inferred triples
USING NAMED <https://host/project>
USING NAMED <https://host/project-I>
WHERE {
    GRAPH ?g1 { # get argument 1
        ?foi props:transmissionHeatLoss ?htr_ .
        ?htr_ opm:hasPropertyState ?state1 .
        ?state1 a opm:CurrentPropertyState, opm:Assumed ;
            schema:value ?htr .
    }
    GRAPH ?g2 {   # get argument 2
        ?foi props:infiltrationHeatLoss ?inf_ .
        ?inf_ opm:hasPropertyState ?state2 .
        ?state2 a opm:CurrentPropertyState, opm:Assumed ;
            schema:value ?inf .
    }
    # check that property state is not already inferred
    MINUS {
        GRAPH <https://host/project-I> {
            ?foi ?inferredProperty ?prop
        }
    }
    # perform calculation
    BIND((?htr+?inf) AS ?res)
    # create state and property IRIs
    BIND(IRI( CONCAT("https://host/project/",
        "states/", STRUUID() ) ) AS ?stateIRI)
    BIND(IRI( CONCAT("https://host/project/",
        "properties/", STRUUID() ) ) AS ?propertyIRI)
    # get current time
    BIND(now() AS ?now)
}
```

5.1. Interacting with Properties through Parametric Query Generation

The OPM Query Generator (OPM-QG)[7] is a JavaScript library for simplifying the task of writing queries for doing Create, Read, Update and Delete (CRUD) operations on an OPM-compliant AEC-KG. This library eases the access to the concepts introduced in Sections 3 and 4 for people not so familiar with RDF. Further, it provides standardised methods for creating the complex queries (Listing 5), thereby ensuring that no unintended operations are performed on the graph.

Since the API is built in JavaScript, queries can be constructed and executed either directly from a web client application or a NodeJS[8]-based server-side application.

OPM-QG is divided into two interfaces, where one deals with properties and the other one with calculations: OPMProp and OPMCalc (See Figure 7, as well as Sections 5.2 and 5.3). Each interface contains a set of methods that return SPARQL queries (e.g. from Listing 2 to Listing 5).

	Create	Read	Update	Delete
OPMProp	postProp() postClassProp()	getProps()	putProp() restoreProp()	setReliability()
OPMCalc	postCalcData() postCalc()	getCalcData() getOutdated() getSubscribers()	putCalc()	
Interfaces	Methods			

Figure 7: OPM-QG interfaces and methods.

OPM-QG supports the separation of explicit and inferred triples in two named graphs as described in Section 4.2, but can also construct queries that operate only on the main graph. When instantiating one of the two interfaces from Figure 7, it is defined what `host` IRI is to be used. This information is necessary when constructing IRIs for new resources. The two `BIND` forms in the query illustrated in Listing 5 (lines 52-55) show how OPM-QG creates new IRIs as a concatenation of `{host}/{type}/{UUID}` where variable `{type}` is typically either *state* or *property*.

All read queries can be generated either as `SELECT` or `CONSTRUCT` queries and create, update and delete queries can be generated either as `INSERT` or `CONSTRUCT` queries. The API can thereby be used for both runtime inferencing and materialising derived triples.

[7] Query Generator - `https://www.npmjs.com/package/opm-qg`

[8] https://nodejs.org/

The contained methods answer to most of the competency questions described in Section 3.

In the following subsections, it is described how OPM-QG can be used to generate parametric queries for accessing and manipulating properties and calculations respectively (Sections 5.2 and 5.3).

5.2. Properties

The OPMProp interface provides methods for dealing with OPM properties. In the following, the methods illustrated in Figure 7 are described in detail.

Create. Properties can be assigned either as instance properties or as property restrictions to an owl:Class instance. The latter approach was used by Rasmussen et al. [46] to specify space requirements at type level that would then be inherited by all instances of that class. Listing 6 shows an example where a property restriction is applied to a project-specific wall class. The property restriction is for the U-value (props:thermalTransmittance), and it restricts its value to a specific property, `inst:heavyWall_r1_s1`. This property is inherited by all instances of the wall class and it can be changed using the general OPM principles, since the property is described with an OPM property state.

Listing 6: OWL property restriction.

```
# project-specific class with property restriction
inst:heavyWall rdfs:subClassOf prod:Wall ;
    rdfs:subClassOf inst:heavyWall_r1 .
# property restriction
inst:heavyWall_r1 rdf:type owl:Restriction ;
    owl:onProperty props:thermalTransmittance ;
    owl:hasValue inst:heavyWall_r1_s1 .
# first state of property restriction
inst:heavyWall_r1_s1 rdf:type opm:CurrentPropertyState ,
        opm:Required ;
    schema:value "0.21 W/(m2.K)"^^cdt:ucum ;
    prov:generatedAtTime "2018-10-31T.."^^xsd:dateTime .
```

OPM-QG includes two methods `postProp()` and `postClassProp()` that generates queries for assigning a new property and an associated property state to some FoI. The methods take the IRI of the new property and the value as arguments. For both methods, the instance/class to which the property should be assigned, can be specified by providing the IRI of a specific FoI or by providing a triple path that must return a match. The path is described like the opm:argumentPaths from Listing 2. Optionally, a `reliability`, a `userIRI` and a `comment` can be provided.

Providing the object shown in Listing 7 to the `postProp()` method returns the query shown in

Listing 8. It is important to note that the path is commented out. Either a `foiIRI` or a `path` must be provided. Also, the last three attributes are optional, and leaving them out would have omitted lines 5, 7-8 and 20-21 from the query. Supplying a `path` instead of the `foiIRI` would have omitted line 19 and replaced line 24 with the `path`. Since it is a create method, it should only apply the new property to the space if it does not already have the property assigned.

Listing 7: Input object to `postProp()` or `postClassProp()`.

```
{
    foiIRI:     'https://host/project/space_1',
    //path:     '?foi a bot:Space',         //alternative
    property:   'props:area',
    value:      '"20 m2"^^cdt:area',
    comment:    'Just a test',              //optional
    userIRI:    'https://niras.dk/mhra',    //optional
    reliability: 'assumed'                  //optional
}
```

Read. The generic method, `getProps()`, can be used to get all properties in the AEC-KG. The result can, however, also be restricted by providing a specific `foiIRI` and/or a specific `propertyType` and/or a specific `propertyIRI`. The results can also be restricted to only include the latest property state or property states with a specific restriction, such as all deleted properties.

Listing 8: Result when providing Listing 7 to `postProp()` (main graph).

```
CONSTRUCT {
    ?foi props:area ?propertyIRI .
    ?propertyIRI a opm:Property ;
        opm:hasPropertyState ?stateIRI .
    ?stateIRI a opm:Assumed .
    ?stateIRI a opm:CurrentPropertyState ;
        prov:wasAttributedTo ?userIRI ;
        rdfs:comment ?comment ;
        schema:value ?val ;
        prov:generatedAtTime ?now .
}
WHERE {
    # create state and property iris
    BIND( IRI( CONCAT( "https://host/project/",
        "states/", STRUUID() ) ) AS ?propertyIRI )
    BIND( IRI( CONCAT( "https://host/db/architect/",
        "properties/", STRUUID() ) ) AS ?propertyIRI )
    BIND(now() AS ?now)
    BIND(<https://host/project/space_1> AS ?foi)
    BIND(<https://niras.dk/employees/mhra> AS ?userIRI)
    BIND("Just a test" AS ?comment)
    BIND("20 m2"^^cdt:area AS ?val)
    # foi must exist
    ?foi ?p ?o .
    # the foi cannot have the property assigned already
    MINUS { ?foi props:area ?prop . }
}
```

Update. The `putProp()` method for updating a property by assigning a new state is comparable to `postProp()`, but separate methods exist for setting the reliability or restoring a specific property (as described in [16]). Restoring a property (method `restoreProp()`) or setting the reliability (method `setReliability()`) requires the IRI of a specific property (`propertyIRI`) as argument. The `putProp()` method, however, accepts also a set of the IRI of a specific FoI (`foiIRI`) and a `propertyType` or a `path`.

The query generated by the `putProp()` method is comparable with the query in Listing 8. It contains `MINUS` clauses so that it will only create a new property if the previous state is not an instance of opm:Confirmed, opm:Derived or opm:Deleted. The first because a confirmed property should not be changed, the second because a derived property should be changed by the algorithm which it was generated by and the latter because a deleted property should first be restored using the `restoreProp()` method which restores the previous state. If the existing value is equal to the new one, it will also not be updated.

Delete. Creating a query to delete a property is achieved by using the `setReliability()` method to set the reliability to deleted. A helper method, `deleteProperty()` also exists. This method only takes the `propertyIRI` as an argument and preferably a `userIRI` and a `comment`.

5.3. Calculations

The OPMCalc interface provides methods for dealing with OPM calculations. Similar to OPMProp, it contains methods to generate queries for CRUD operations on the AEC-KG. When dealing with calculations, however, there are both the management of opm:Calculation instances and the operations to be performed on the AEC-KG in order to infer derived properties.

Create. The `postCalcData()` method returns a query for creating a new opm:Calculation instance. As a minimum, it takes a `label`, an `expression`, a set of `argumentPaths` and the type of the inferred property (`inferredProperty`) as arguments. The number of variables used in the `expression` is compared to the number of `argumentPaths` to ensure that the two match and it is checked that the variable names match. For the parametric queries to function, the name used for the first variable in each argument path is replaced by `?foi`. This means that `?s props:area ?a`

is automatically changed to `?foi props:area ?a`. Optional arguments include `userIRI`, a FoI restriction (opm:foiRestriction) which will restrict the calculation to only be applied to a specific FoI and a path restriction (opm:pathRestriction) which will restrict the calculation only to be applied where a specific path is matched. The generated query creates a resource with the above properties assigned (similar to Listing 2).

The `postCalc()` method takes all the arguments that are available on an opm:Calculation instance including the `calculationIRI`, and performs the same validation of the arguments and generates a query like the one illustrated in Listing 5. As described in Section 3.3, lines 18-29 and 31-42 retrieve the arguments. OPM-QG generates these according to the number of arguments given in the calculation. Each argument path is first appended with an underscore suffix for the argument variable name. This is because the variable name is instead used to describe the value of the most recent property state. The state is itself saved in a variable, and all these are appended in line 10 where it is specified what property states the specific derived property state was derived from.

If the expression contains an aggregation function, the structure of the calculation is quite different. OPM-QG will recognise either of these and instead generate a query like the one shown in Listing 9. For aggregation functions, it is further checked that the list of opm:argumentPaths only contains one item.

The query in Listing 9 is a `CONSTRUCT` query, so it will generate all the new derived properties and return the full graph without materialising it in the AEC-KG. This enables the end user to evaluate the results before making a final change and thereby provides insights for the engineers to compare and assess changes continuously without having everything being updated automatically.

The sub-query in lines 19-25 assigns the FoI to variable `?foi` and all the latest states of the properties that match the path to variable `?state1`. This sub-query is needed in order to get the individual states for assigning the prov:wasDerivedFrom predicate to the derived property (line 10). The actual sum is calculated in the next sub-query. This query generates a result for each `?foi` and creates IRIs for the new derived properties while doing so. At line 55 the expression is applied. For this particular query, it would be enough to just assign the value of `?htr` directly, but this approach allows for post-processing such as formatting the result or adding 10 % (`sum(?htr)*1.1`). OPM-QG takes care of removing the `sum()` function from the expression since `?htr` already holds the sum. The property will not be assigned to FoIs already having the property assigned.

Listing 9: SUM query returned by `postCalc()`.

```
1  CONSTRUCT {
2      ?foi ?inferredProperty ?propertyIRI .
3      ?propertyIRI a opm:Property ;
4          opm:hasPropertyState ?stateIRI .
5      ?stateIRI a opm:CurrentPropertyState ,
6                          opm:Derived , ?reliability ;
7          schema:value ?res ;
8          prov:generatedAtTime ?now ;
9          prov:wasAttributedTo <https://host/project/c2> ;
10         prov:wasDerivedFrom ?state1 .
11 }
12 USING NAMED <https://host/project>
13 USING NAMED <https://host/project-I>
14 WHERE {
15     BIND(props:transmissionHeatTransferRate AS
16                                 ?inferredProperty)
17     # GET THE MOST RECENT STATES OF THE ARGUMENTS
18     GRAPH ?g {
19         { SELECT ?foi (?state AS ?state1) WHERE {
20             ?foi a ice:ThermalEnvironment ;
21                 ^ice:surfaceInterior ?i .
22             ?i props:totalHeatTransferRate ?htr_ .
23             ?htr_ opm:hasPropertyState ?state .
24             ?state a opm:CurrentPropertyState
25         }}
26         # CALCULATE THE SUM
27         { SELECT ?foi (SUM(?res_) AS ?htr)
28             (IRI(CONCAT("https://host/project/",
29                 "states/", STRUUID())) AS ?stateIRI)
30             (IRI(CONCAT("https://host/project/",
31               "properties/", STRUUID())) AS ?propertyIRI)
32             (now() AS ?now)
33           WHERE {
34             ?foi a ice:ThermalEnvironment ;
35                 ^ice:surfaceInterior ?i .
36             ?i props:totalHeatTransferRate ?htr_ .
37             ?htr_ opm:hasPropertyState ?state1 .
38             ?state1 schema:value ?htr__ .
39             BIND(?htr__ AS ?res_)
40           } GROUP BY ?foi
41         }
42         # INHERIT CLASS OPM:ASSUMED OR OPM:DELETED
43         OPTIONAL {
44             ?state1 a ?reliability .
45             FILTER( ?reliability = opm:Assumed ||
46                     ?reliability = opm:Deleted )
47         }
48     }
49     # DO NOT APPEND IF PROPERTY ALREADY DEFINED
50     MINUS {
51         GRAPH <https://host/project-I> {
52             ?foi ?inferredProperty ?prop }
53     }
54     # APPLY EXPRESSION
55     BIND((?htr) AS ?res)
56 }
```

Read. Getting calculation data is achieved with the `getCalcData()` method. If no arguments are provided, it will return a query to get all calculations. Providing a `calculationIRI` will return the data for

that specific calculation. Providing a `label` will return the calculation matching that particular label. Providing a `foiIRI` and a `propertyType` will return data on the calculation which inferred the particular derived property.

`getOutdated()` is a method for retrieving all derived properties where one or more of the arguments is no longer the opm:CurrentPropertyState. It can be restricted to only return derived properties of a specific FoI.

`getSubscribers()` returns a list of derived properties that are dependent on a specific property. This can be used to evaluate whether other parties will be influenced before a change to the property is conducted. Instead of providing an IRI for the property, it is also possible to provide a `foiIRI` and the `propertyType`.

Getting the latest state or all the states of a particular derived property is not different from getting a typed property. The OPMProp interface contains methods for this purpose.

Update. Calculations do not use OPM for managing the specifications, and hence the calculations themselves cannot be updated. It is, however possible to support this by simply assigning property states to the expression, argument paths and so forth (`putCalc()` method). This will, however, increase complexity, since a derived property will not only be outdated when one of its arguments has changed, but also if the calculation which it was attributed to has changed.

Similar to the `postCalc()` method, the `putCalc()` method takes all the arguments that are available on an opm:Calculation instance, including the `calculationIRI`, and generates an update query. The only difference is that this query will apply new states to existing derived properties where at least one argument has changed.

Delete. A derived property automatically inherits the opm:Deleted state from any of its arguments and thereby automatically becomes deleted itself.

6. Proof of Concept

In order to demonstrate the capabilities of the OPM architecture, a Proof of Concept (PoC) application was developed internally at the danish consulting engineering company Niras. The application performs the particular design task of calculating heating demand as described in the introduction. It uses a generic approach; however, that is transferable to other design tasks in the future. An open-source web application[9] with an accompanying OPM-REST backend[10] was developed to demonstrate most of the operations that the PoC application performs on the AEC-KG.

6.1. System Architecture

Since the AEC-KG was stored in a triplestore that exposes a SPARQL 1.1 endpoint [50], it would have been possible for a client application to perform queries directly through HTTP requests. However, it was decided to make a middleware on a backend server that handles communication with a SPARQL 1.1 endpoint. The Stardog triplestore was originally used, but any other triplestore that implements the SPARQL 1.1 Protocol standard may be used instead (e.g., Jena Fuseki, RDF4J Sesame). No reasoning needs to be performed by the triplestore. In fact, the queries generated by OPM-REST API do contain complex constructs for OWL2 RL axioms [51]. Therefore, the SPARQL engine does operate all the OWL RL inferences, which has sufficient expressivity for our needs.

Using the OPM-REST API, the frontend application can be developed by developers with no knowledge of RDF and OPM, and they are hence protected from the complex queries demonstrated in the previous sections. This is particularly practical for tasks that require several queries to the triplestore and further entails that complicated requests can be used across several client applications.

The backend is built as a Representational state transfer (REST) API which exposes a set of routes to which client applications can send HTTP request in order to do CRUD operations on the AEC-KG (see Section 5). Some of the routes are generic and will, for example, return all properties assigned to a FoI, change a class assigned to a FoI, update a property or return a full list of calculations. Others are provided for a particular application and will, for example, return all the parts of the building envelope that face a specific room.

The frontend of both applications is built with the Angular[11] and is structured so that a service component takes care of the communication with the backend whereas a set of controllers take care

[9] Demo: `https://madsholten.github.io/OPM-REST/`, sources `https://github.com/madsholten/OPM-REST/`

[10] Installation instructions: `https://github.com/MadsHolten/OPM-REST/blob/master/readme.md`

[11] `https://angular.io/` JavaScript framework

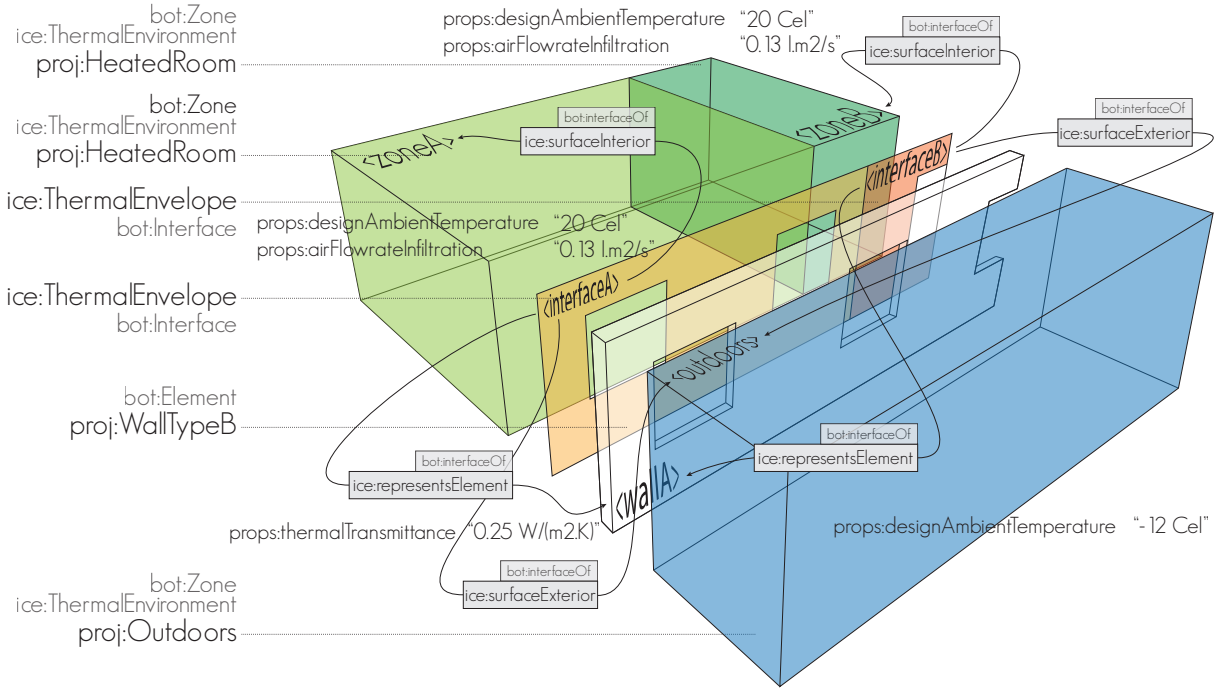

Figure 8: Visualization of the AEC-KG model for heat loss calculation.

of building the view based on the data returned by the service.

In the next sections, the functionalities of OPM-REST related to OPM are described.

6.2. Interface to the Architect

Typical practice in engineering companies is that a BIM model is received from the architect on a weekly basis. The architectural model is then mainly used as a geometrical reference model for the mechanical and structural design models. With this work, we suggest an approach where all the valuable information is extracted for further use, thereby relying on the set of ontologies and methods proposed within the W3C LBD Community Group. For example, using the Revit-BOT-exporter [12] [52] with an extension for extracting the building envelope and for communicating with the REST API instead of writing triples to a file, it was possible to establish direct communication between the native BIM tool and the backend. All topological relationships and properties are extracted and sent via an OPM batch upload route to the server through this setup. A set of definitions for the graphical programming tool Dynamo for Revit that directly communicate with OPM-REST were further developed to demonstrate how a connection to legacy models can be established. These are included in the OPM-REST repository and it is our hope that similar interfaces can in the future be created to other BIM and simulation tools.

When receiving triples at the OPM batch upload routes, all the zone and element instances and their topological relationships are written to the AEC-KG. The state of class instances and topological relationships are not managed with OPM but properties such as space names and numbers, areas, volumes, 2D space boundaries and object mesh models are. All properties are received in complexity level L1 (see Section 2.2), and these are loaded into a temporary named graph in the AEC-KG (see Section 4.2). Then, a comparison is conducted between what is already in the AEC-KG and what new properties have been received. Finding the new properties is handled by searching the temporary graph for matches to the triple path `?foi ?prop ?val` while leaving out results in the project graph meeting the triple pattern `?foi ?prop ?x`. New properties and states are created using the OPM-QG API discussed in Section 5, and the triples are added to

[12] `https://github.com/MadsHolten/revit-bot-exporter`

the project graph. Next, the backend checks for updated triples by finding the value of the latest property state and comparing it to the new value. Only if states are different, they are added to the project graph in the form of new property states for specific updated properties.

6.3. Appending Engineering Properties

With the initial graph in place, starting from the architectural design model, the engineer then defines and assigns a set of project-specific classes for element and zone types with OWL property restrictions similar to what was shown in Listing 6. For heat loss calculations, most spaces are simply seen as heated or unheated, and the exterior can be either ground or air. The data model and the inherited properties are illustrated in Figure 8. It is based on BOT with Indoor Climate and Energy (ICE) specific extensions and even more specific project extensions. The two rooms are instances of the project-specific `proj:HeatedRoom` ($\sqsubseteq$ `ice:ThermalEnvironment` $\sqsubseteq$ bot:Zone) and inherit a props:designAmbientTemperature of 20 $^{\circ}C$ as well as an props:airFlowRateInfiltration of 0.13 $l.m^2/s$. The outdoor environment is an instance of the project-specific `proj:Outdoors` class, thereby inheriting a props:designAmbientTemperature of -12 $^{\circ}$ C. The wall is an instance of the project-specific `proj:WallTypeB` class, thereby inheriting a props:thermalTransmittance of 0.25 W/m^2K. Each `ice:ThermalEnvelope` ($\sqsubseteq$ bot:Element) has a props:heatTransferSurfaceArea assigned explicitly, and each individual room also has an area. Hence, all the necessary information for deriving the heating demand is available, and a set of opm:Calculations can be defined to do so.

Pre-defined calculations. Table 2 lists all the calculations that were defined in the project, including their opm:inferredProperty, opm:expression and opm:argumentPaths. A quick examination of the inferred properties and the argument paths reveals that there are some internal interdependencies. As long as there are no circular dependencies, it is not a problem, and therefore a check needs to be performed on the backend to assure that no circular dependencies will be inferred by any new calculation before it is created.

Appending calculations. When sending a `POST` request to the IRI of a calculation, new derived properties are appended. The `getCalcData()` method

Table 2: Predefined opm:Calculations.

props:nominalUA	
`?u*?a`	'?foi props:heatTransferSurfaceArea ?a', '?foi ice:representsElement ?el . ?el props:thermalTransmittance ?u'
props:designTemperatureDifference	
`?te-?ti`	'?foi ice:surfaceInterior ?si . ?si props:designAmbientTemperature ?ti', '?foi ice:surfaceExterior ?se . ?se props:designAmbientTemperature ?te'
props:totalHeatTransferRate	
`?ua*?dt`	'?foi props:designTemperatureDifference ?dt', '?foi props:nominalUA ?u'
props:transmissionHeatTransferRate	
`sum(?htr)`	'?foi a ice:ThermalEnvironment ; ^ice:surfaceInterior ?int', '?int props:totalHeatTransferRate ?htr'
props:infiltrationHeatTransferRate	

`?a*?inf*`
`1.166*1.0075`
`*(?ti-(-12))`
	'?sp props:netFloorArea ?a',
	'?sp props:designAmbientTemperature ?ti',
	'?sp props:airFlowrateInfiltration ?inf'

props:heatingDemand	
`?tr+?inf`	'?foi props:transmissionHeatTransferRate ?tr', '?foi props:infiltrationHeatTransferRate ?inf'

of the OPM-QG is used to get the calculation data and subsequently, the `postCalc()` method is used to infer the derived properties where the arguments are matched but the derived property is not already appended.

Each time a new BIM model is received from the architect, there are potentially new matches to the calculations, and, therefore, the newly derived properties must be appended. Because of the interdependencies between derived properties, it might require several loops for all the derived properties to be inferred, so some pre-processing is done. The whole network of interdependencies is explicitly stated in the graph, so it is possible first to calculate an execution order. As a result, the server load is reduced dramatically.

A dedicated route on the backend calculates the full tree of all calculations using the algorithm illustrated in the BPMN diagram in Figure 9. The depth of a calculation denotes its position in the dependency chain. A calculation having depth 0 has no dependencies and can hence be executed directly. Table 3 shows the depths of all the properties inferred by the calculations in Table 2. The first three are independent on output from the other

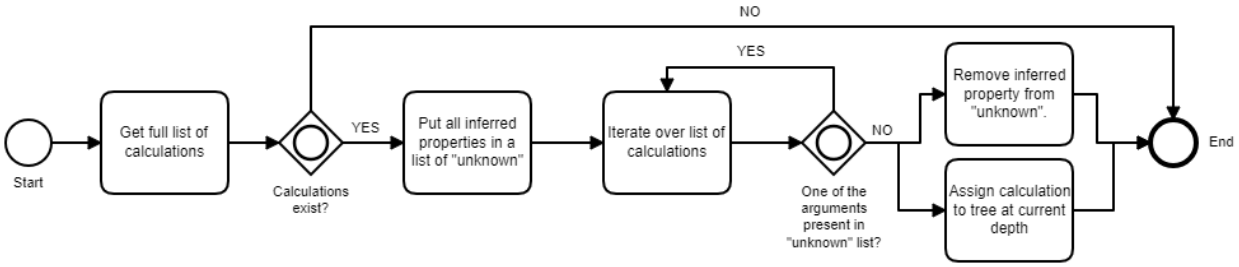

Figure 9: Algorithm to calculate complete execution tree for calculations.

calculations, but three other properties must first be inferred by calculations in order to calculate the props:heatingDemand. All calculations located at the same depth can be executed in parallel.

Table 3: Calculation depths for calculations in Table 2.

Inferred property	depth
props:infiltrationHeatTransferRate	0
props:designTemperatureDifference	0
props:nominalUA	0
props:totalHeatTransferRate	1
props:transmissionHeatTransferRate	2
props:heatingDemand	3

Updating calculations. When sending a `PUT` request to the IRI of a calculation, existing derived properties are updated. The approach is similar to appending calculations, but instead uses the `putCalc()` method to update the derived properties where at least one of its arguments has changed.

The task of updating derived properties is not automated since the engineer must make the decision of conducting a change. The engineer is, however, provided with an overview of the outdated properties along with insights concerning the consequences of conducting a particular change. Thereby, the engineer is provided with supporting tools for decision making.

In the user interface, an outdated derived property is highlighted, and clicking an icon will request the backend for a full dependency tree of the particular outdated property. The algorithm is a bit different from what is illustrated in Figure 9 and it uses a recursive "follow your nose" approach for retrieving a full list of arguments. Figure 10 shows an example of a derived property with a rather long dependency chain inferred by the calculations shown in Table 2. The figure reveals that the props:heatingDemand is outdated because the props:heatTransferSurfaceArea of one of the thermal envelope segments facing the particular space is no longer valid. This consequently means that the props:nominalUA and hence also the props:totalHeatTransferRate of that particular segment are outdated, which in turn means that the props:transmissionHeatTransferRate of the space is also outdated.

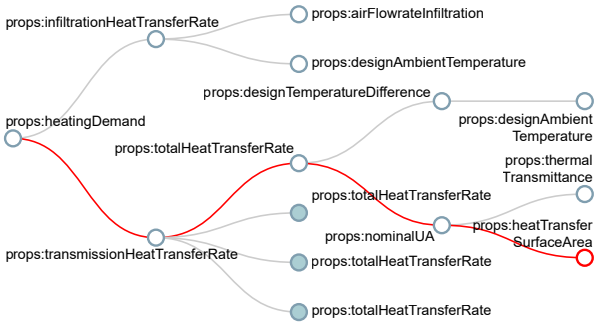

Figure 10: When the user clicks an outdated derived property, the full calculation tree is shown to provide insights.

Performing a `PUT` request on the derived props:heatingDemand will not have any influence as long as the derived props:transmissionHeatTransferRate has not been revised. Therefore, updating the props:heatingDemand will require that all the intermediate derived properties are also updated starting with the one closest to the typed property (props:heatTransferSurfaceArea). Since all these properties belong to the engineer's design, she can decide to update the whole tree, which can be achieved by sending a `PUT` request to the calculation IRI with a query parameter specifying that the whole chain should be updated. If one or more properties had belonged to another stakeholder, the decision of updating the whole chain would require involving this 3rd party.

Checking for outdated. It was previously described how an outdated property is identified by checking if any of the arguments (or arguments' arguments) are no longer an instance of opm:CurrentPropertyState. This can be achieved with a SPARQL property path and a `MINUS` clause: `?prop prov:wasDerivedFrom+ ?arg . MINUS{?arg a opm:CurrentPropertyState}`. This, however, causes problems when depending on OWL restrictions for property inheritance. If a wall instance is, for example, changed from `proj:WallHeavy` to `proj:WallLight`, the inherited U-value will change accordingly. The derived property for props:nominalUA is dependent on the U-value, but, since the latest state of this property is related to the most recent state of the U-value of `proj:WallHeavy` (by prov:wasDerivedFrom), the property path above will not recognise that the derived property is no longer valid. Re-calculating the result with a `PUT` request to the calculation IRI will retrieve the new result, and comparing this with the current result will reveal that the property is outdated. The comparison approach is more resource-intensive than the one that looks for states that are not classified as opm:CurrentPropertyState. This resource-intensive query can, however, be performed on the server as a scheduled job, thereby explicitly inferring the opm:OutdatedPropertyState class. Also, it is possible to do this check each time the class of an instance is changed.

6.4. Interface to Other Engineers

The props:heatingDemand of each space sets the boundary conditions for the heater serving that space. Generating heaters can be automated with a SPARQL update query which searches for any space classified as a `HeatedSpace` with a heating demand above a certain threshold that does not already have a heater assigned. Calculations specific to the flow system can then be set up by the HVAC engineer similar to the procedure demonstrated for the ICE engineer in Section 6.3.

7. Conclusions and Future Work

In this article we demonstrated how semantic web technologies can be used to cope with the highly interrelated and rapidly changing design decisions when developing a construction project. We used the preliminary Ontology for Property Management (OPM) [16], and tripled the number of competency questions it answers to account for property reliablity, and the description of how Features of Interest property values may be inter-dependent by laws of physics. We proposed an API to homogeneously query and manage OPM-compliant AEC Knowledge Graphs (AEC-KG). We demonstrated how these contributions can be used with an open-source and reusable Proof of Concept implementation. This implementation exposes the benefits of having all interconnections between project properties explicitly connected in an AEC-KG. With such traceability and consequence analysis of a property change, we demonstrated that AEC softwares, even proprietary softwares, could use semantic web technologies to better support the design engineers' decision inter-operating throughout the development of construction projects.

We developed a reuseable open-source implementation which can be used to store design data in a structured way, allowing interrelated data to maintain their relations intact as the project progresses. This system answers our initial research question:

> *How can we effectively store design data in a structured way, allowing interrelated data to maintain their relations intact as the project progresses, without losing the history of properties' progression?*

With our proof of concept implementation, the full history of the project persists, and it is hence possible at any point in time to analyse the background of a specific design change. It was endeavoured to use terminology which is already widely adopted to describe the AEC-KG. Property inferencing is based on standard OWL reasoning, and the widely adopted ontologies PROV-O and schema.org are used for describing properties.

The PoC shows an alternative approach for working with building data. It is a general experience in the industry that valuable data is trapped in proprietary BIM models and, with this work, we demonstrate how data from the other stakeholder's (architect's) model can be made accessible as RDF triples, directly from within the designer's toolchain and not through an intermediate file format.

7.1. With the Engineer Hat on

The presented infrastructure is fundamentally different from how engineers currently work. The intention is that calculations and inferencing, which

are currently done in different tools by different people, is described explicitly and in an interoperable manner. As soon as all the arguments for a particular calculation are available, so is the result. In such a setup, the primary task of an engineer is to make sure that all arguments are provided by the other practitioners. Further, since consequence analyses can be performed much faster, the engineer could potentially work with multiple parallel concepts for each design until sufficient knowledge is available for making a final choice.

Making calculations and hence design tasks reusable entails that the knowledge of the company as a whole grows over time in contrast to today where it is mainly the knowledge of the employees that evolves. Over time, the growing AEC-KG could potentially overcome the still existing challenge that companies lose access to large quantities of critical knowledge as employees turn over as it was implied by O'Leary [12].

7.2. Future Outlook

In the PoC, all calculations were performed at the backend using an approach similar to the one illustrated in Figure 6. It would be interesting to investigate how some calculations could be performed by the client application while still following the OPM principles described in Section 3 when writing the resulting triples to the AEC-KG. It would also be interesting to investigate how more complex calculations such as thermal simulations could be implemented in practice.

Future work also includes proof of concept integration of the presented work in various proprietary softwares. In fact, designers usually rely on such softwares to perform their work; for example, structural designers use analysis and design software when designing a multistory building. The approach presented in this paper is a first step towards the demonstration that Linked Data will facilitate the parallel development of various analysis over the same building model in these different heterogeneous softwares.

Acknowledgements

Special thanks to the NIRAS ALECTIA Foundation and Innovation Fund Denmark for funding. Also thanks to Niras for allowing open distribution of the developed artifacts[13]. It is a fundamental necessity for the future growth of the proposed standards that they are adopted and further developed by the community.

References

1. McKinsey GI. Reinventing construction: A route to higher productivity. 2017. URL: https://www.mckinsey.com/industries/capital-projects-and-infrastructure/our-insights/reinventing-construction-through-a-productivity-revolution; accessed Nov. 2018.
2. Bertelsen S. Construction as a complex system. In: *Proceedings of the 11th Annual Conference of the International Group for Lean Construction*. 2003:143–68. URL: http://www.iglc.net/papers/details/231; accessed Sep. 2019.
3. Succar B. Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction* 2009;18(3):357–75. doi:10.1016/j.autcon.2008.10.003.
4. BSI . PAS 1192-3: 2014. Specification for information management for the operational phase of assets using Building Information Modelling. 2014. URL: https://shop.bsigroup.com/Sandpit/PAS-old-forms/PAS-1192-3/; accessed Dec. 2018.
5. Winch GM. Managing construction projects. John Wiley & Sons; 2010. ISBN 978-1-405-18457-1.
6. Gallaher MP, O'Connor AC, John L. Dettbarn J, Gilday LT. Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry. Tech. Rep.; National Institute of Standards and Technology; 2004. doi:10.6028/nist.gcr.04-867.
7. Egbu C, Hayles C, Quintas P, Hutchinson V, Anumba C, Ruikar K. Knowledge Management for Sustainable Construction Competitiveness. *Knowledge Management for Sustainable Construction Competitiveness Project, Partners in Innovation (CI 39/3/709) Work Package* 2004;URL: http://www.knowledgemanagement.uk.net/resources/kmfinal.pdf; accessed Sep. 2019.
8. Isikdag U, Aouad G, Underwood J, Wu S. Building information models: a review on storage and exchange mechanisms. In: *Proceedings of the 24th CIB W78 Conference*. 2007:URL: http://itc.scix.net/data/works/att/w78-2007-020-068b-Isikdag.pdf; accessed Dec. 2018.
9. Deshpande A, Azhar S, Amireddy S. A framework for a BIM-based knowledge management system. *Procedia Engineering* 2014;85:113–22. doi:10.1016/j.proeng.2014.10.535.
10. Kiviniemi A. Requirements Management Interface to Building Product Models. Ph.D. thesis; Stanford, CA, USA; 2005. doi:10.25643/bauhaus-universitaet.242; aAI3162341.
11. Tserng HP, Lin YC. Developing an activity-based knowledge management system for contractors. *Automation in Construction* 2004;13(6):781–802. doi:10.1016/j.autcon.2004.05.003.

[13] https://github.com/w3c-lbd-cg/opm

12. O'Leary D. Enterprise knowledge management. *Computer* 1998;31(3):54–61. URL: https://doi.org/10.1109%2F2.660190. doi:10.1109/2.660190.
13. Santos R, Costa AA, Grilo A. Bibliometric analysis and review of Building Information Modelling literature published between 2005 and 2015. *Automation in Construction* 2017;80:118–36. doi:10.1016/j.autcon.2017.03.005.
14. Studer R, Benjamins V, Fensel D. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering* 1998;25(1-2):161–97. doi:10.1016/s0169-023x(97)00056-6.
15. Lóscio BF, Burle C, et al. NC. Data on the Web Best Practices. *W3C recommendation* 2012;URL: https://www.w3.org/TR/2017/REC-dwbp-20170131/; accessed Nov. 2018.
16. Rasmussen MH, Lefrançois M, Bonduel M, Hviid CA, Karlshøj J. OPM: An ontology for describing properties that evolve over time. In: Poveda-Villalón M, Pauwels P, Roxin A, eds. *Proceedings of the 6th Linked Data in Architecture and Construction Workshop*; vol. 2159 of *CEUR Workshop Proceedings*. CEUR-WS.org; 2018:24–33. URL: http://ceur-ws.org/Vol-2159/03paper.pdf; accessed Sep. 2018.
17. Curry E, O'Donnell J, Corry E, Hasan S, Keane M, O'Riain S. Linking building data in the cloud: Integrating cross-domain building data using linked data. *Advanced Engineering Informatics* 2013;27(2):206–19. doi:10.1016/j.aei.2012.10.003.
18. Pauwels P, McGlinn K, Törmä S, Beetz J. Linked data. In: Borrmann A, Knig M, Koch C, Beetz J, eds. *Building Information Modeling*. Springer. ISBN 978-3-319-92862-3; 2018:181–97. doi:10.1007/978-3-319-92862-3.
19. W3C OWL Working Group . OWL 2 Web Ontology Language Document Overview (Second Edition). *W3C recommendation* 2012;URL: http://www.w3.org/TR/2012/REC-owl2-overview-20121211/; accessed Nov. 2018.
20. Cyganiak R, Wood D, Lanthaler M. RDF 1.1 concepts and abstract syntax. *W3C recommendation* 2014;URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/; accessed Nov. 2018.
21. ISO16739 . Industry foundation classes (ifc) for data sharing in the construction and facility management industries. Standard; International Organization for Standardization; Geneva, CH; 2013. URL: https://www.iso.org/fr/standard/51622.html; accessed Sep. 2019.
22. Pauwels P, Terkaj W. EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction* 2016;63:100–33. doi:10.1016/j.autcon.2015.12.003.
23. Beetz J, van Leeuwen J, de Vries B. IfcOWL: A case of transforming EXPRESS schemas into ontologies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 2008;23(01):89. doi:10.1017/s0890060409000122.
24. ISO10303-11 . Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual. Standard; International Organization for Standardization; Geneva, CH; 2004. URL: https://www.sis.se/api/document/preview/905433/; accessed Sep. 2019.
25. Terkaj W, Pauwels P. A method to generate a modular ifcOWL ontology. In: Sanfilippo EM, Daniele L, Colombo G, eds. *Proceedings of the 8th International Workshop on Formal Ontologies meet Industry*; vol. 2050 of *CEUR Workshop Proceedings*. CEUR-WS.org; 2017:URL: http://ceur-ws.org/Vol-2050/FOMI_paper_3.pdf; accessed Sep. 2019.
26. Mendes de Farias T, Roxin A, Nicolle C. IfcWoD, semantically adapting IFC model relations into OWL properties. In: *Proceedings of the 32nd CIB W78 Conference on Information Technology in Construction*. 2015:URL: https://arxiv.org/abs/1511.03897; accessed Mar. 2018.
27. Pauwels P, Roxin A. SimpleBIM : From full ifcOWL graphs to simplified building graphs. In: Christodoulou SE, Scherer R, eds. *eWork and eBusiness in Architecture, Engineering and Construction (ECPPM 2016)*. Limassol, Cyprus: CRC Press. ISBN 9781138032804; 2016:11–8. doi:10.1201/9781315386904.
28. Zhang C, Beetz J, de Vries B. BimSPARQL: Domain-specific functional SPARQL extensions for querying RDF building data. *Semantic Web* 2018;9(6):829–55. doi:10.3233/sw-180297.
29. Harris S, Seaborne A, Prudhommeaux E. SPARQL 1.1 query language. *W3C recommendation* 2013;21(10). URL: https://www.w3.org/TR/sparql11-query/; accessed Dec. 2018.
30. Niknam M, Karshenas S. A shared ontology approach to semantic representation of BIM data. *Automation in Construction* 2017;80:22–36. doi:10.1016/j.autcon.2017.03.013.
31. Berners-Lee T. Linked data. *W3C recommendation* 2006;URL: https://www.w3.org/DesignIssues/LinkedData.html; accessed Nov. 2018.
32. Holten Rasmussen M, Pauwels P, Lefrançois M, Ferdinand Schneider G. Building Topology Ontology. W3C Draft Community Group Report; W3C; 2019. URL: https://w3c-lbd-cg.github.io/bot/; accessed Sep. 2019.
33. Rasmussen MH, Pauwels P, Hviid CA, Karlshøj J. Proposing a central AEC ontology that allows for domain specific extensions. In: Bosché F, Brilakis I, Sacks R, eds. *Proceedings of the Joint Conference on Computing in Construction*; vol. 1. Heriot-Watt University. ISBN 978-0-9565951-6-4; 2017:doi:10.24928/jc3-2017/0153.
34. Bonduel M. Towards a PROPS ontology. 2018. URL: https://github.com/w3c-lbd-cg/lbd/blob/gh-pages/presentations/props/presentation_LBDcall_20180312_final.pdf; accessed Nov. 2018.
35. Prud'hommeaux E, Carothers G, Beckett D, Berners-Lee T. RDF 1.1 Turtle - Terse RDF Triple Language. *W3C recommendation* 2014;URL: http://www.w3.org/TR/2014/REC-turtle-20140225/; accessed Nov. 2018.
36. Lefrançois M, Zimmermann A. The unified code for units of measure in rdf: cdt:ucum and other ucum datatypes. In: Gangemi A, Gentile AL, Nuzzolese AG, Rudolph S, Maleshkova M, Paulheim H, Pan JZ, Alam M, eds. *The Semantic Web: ESWC 2018 Satellite Events*. Cham: Springer International Publishing. ISBN 978-3-319-98192-5; 2018:196–201. doi:10.1007/978-3-319-98192-5_37.
37. Hodgson R, Keller PJ. QUDT -quantities, units, dimensions and data types in OWL and XML. 2011. URL:

http://www.qudt.org; accessed Dec. 2018.
38. Lebo T, Sahoo S, McGuinness D, Belhajjame K, Cheney J, Corsar D, Garijo D, Soiland-Reyes S, Zednik S, Zhao J. PROV-O: The PROV ontology. *W3C recommendation* 2013;URL: https://www.w3.org/TR/prov-o/; accessed Dec. 2018.
39. Lefrançois M. Planned ETSI SAREF Extensions based on the W3C&OGC SOSA/SSN-compatible SEAS Ontology Patterns. In: Fensel A, Daniele L, eds. *Proceedings of Workshop on Semantic Interoperability and Standardization in the IoT, SIS-IoT,*; vol. 2063 of *CEUR Workshop Proceedings*. CEUR-WS.org; 2017:URL: http://ceur-ws.org/Vol-2063/sisiot-paper2.pdf; accessed Dec. 2018.
40. Haller A, Janowicz K, Cox SJD, Le Phuoc D, Taylor K, Lefrançois M. Semantic Sensor Network Ontology. W3C Recommendation; World Wide Web Consortium; 2017. URL: https://www.w3.org/TR/vocab-ssn/; accessed Sep. 2019.
41. Haller A, Janowicz K, Cox S, Lefranois M, Taylor K, Le Phuoc D, Lieberman J, Garca-Castro R, Atkinson R, Stadler C. The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. *Semantic Web* 2019;10:9–32. doi:10.3233/SW-180320.
42. Janowicz K, Haller A, Cox SJ, Le Phuoc D, Lefrançois M. Sosa: A lightweight ontology for sensors, observations, samples, and actuators. *Journal of Web Semantics* 2019;56:1–10. doi:doi.org/10.1016/j.websem.2018.06.003.
43. Isaac S, Sadeghpour F, Navon R. Analyzing Building Information Using Graph Theory. In: *Proceedings of the 30th International Symposium on Automation and Robotics in Construction and Mining (ISARC 2013): Building the Future in Automation and Robotics*. 2013:doi:10.22260/isarc2013/0111.
44. Pauwels P, Deursen DV, Verstraeten R, Roo JD, Meyer RD, de Walle RV, Campenhout JV. A semantic rule checking environment for building performance checking. *Automation in Construction* 2011;20(5):506–18. doi:10.1016/j.autcon.2010.11.017.
45. Zamanian M, Pittman JH. A software industry perspective on AEC information models for distributed collaboration. *Automation in Construction* 1999;8(3):237–48. doi:10.1016/s0926-5805(98)00074-0.
46. Rasmussen MH, Bonduel M, Hviid CA, Karlshøj J. Managing Space Requirements of New Buildings Using Linked Building Data Technologies. In: *eWork and eBusiness in Architecture, Engineering and Construction (ECPPM 2018)*. CRC Press. ISBN 978-1-138-58413-6; 2018:399–406.
47. Carroll JJ, Bizer C, Hayes P, Stickler P. Named graphs, provenance and trust. In: Ellis A, Hagino T, eds. *Proceedings of the 14th international conference on World Wide Web*. ACM Press; 2005:doi:10.1145/1060745.1060835.
48. Barbieri DF, Braga D, Ceri S, Valle ED, Grossniklaus M. Incremental Reasoning on Streams and Rich Background Knowledge. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg; 2010:1–15. doi:10.1007/978-3-642-13486-9_1.
49. Antoniou G, Bikakis A. DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web. *IEEE Transactions on Knowledge and Data Engineering* 2007;19(2):233–45. doi:10.1109/tkde.2007.29.
50. Ogbuji C. SPARQL 1.1 Protocol. *W3C recommendation* 2013;URL: http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/; accessed Nov. 2018.
51. Bischof S, Krötzsch M, Polleres A, Rudolph S. Schema-agnostic query rewriting in sparql 1.1. In: Mika P, Tudorache T, Bernstein A, Welty C, Knoblock C, Vrandečić D, Groth P, Noy N, Janowicz K, Goble C, eds. *The Semantic Web – ISWC 2014*. Cham: Springer International Publishing. ISBN 978-3-319-11964-9; 2014:584–600.
52. Rasmussen MH, Hviid CA, Karlshøj J. Web-based topology queries on a bim model. In: *Proceedings of the 5th Linked Data in Architecture and Construction Workshop*. 2017:URL: https://orbit.dtu.dk/ws/files/140019661/Untitled.pdf; accessed Sep. 2019.