Minimalistic Fuzzy Ontology Reasoning: An application to Building Information Modeling

Ignacio Huitzil[a,b,*], Miguel Molina-Solana[b,c], Juan Gómez-Romero[c], Fernando Bobillo[a,d]

[a] University of Zaragoza, Zaragoza, Spain
[b] Imperial College London, London, United Kingdom
[c] University of Granada, Granada, Spain
[d] Aragon Institute of Engineering Research (I3A), Zaragoza, Spain

Abstract

This paper presents a minimalistic reasoning algorithm to solve imprecise instance retrieval in fuzzy ontologies with application to querying Building Information Models (BIMs)—a knowledge representation formalism used in the construction industry. Our proposal is based on a novel lossless reduction of fuzzy to crisp reasoning tasks, which can be processed by any Description Logics reasoner. We implemented the minimalistic reasoning algorithm and performed an empirical evaluation of its performance in several tasks: interoperation with classical reasoners (Hermit and TrOWL), initialization time (comparing TrOWL and a SPARQL engine), and use of different data structures (hash tables, databases, and programming interfaces). We show that our software can efficiently solve very expressive queries not available nowadays in regular or semantic BIMs tools.

Keywords: Fuzzy ontologies, Flexible querying, Building Information Modeling

1. Introduction

Digitalization is a major innovation factor in the construction sector. The incorporation of new information management technologies is transforming how buildings are designed, planned and operated [1]. A key element to achieve this vision is the Building Information Model (BIM), a digital representation of a building for integrated design, modeling, planning and operation during its whole lifecycle [2], from inception to decommission. BIMs can help to optimize construction and maintenance costs, improve transparency and collaboration between different stakeholders, manage complex projects, and adapt to changing requirements quickly. Not surprisingly, the European BIM market was evaluated at 1.8 billion € in 2016 and it is expected to grow up to 2.1 billion € by 2023 [3].

The BIM concept brings together several pieces of interconnected information, including a 3D geometric model of the building elements and a description of the materials used and their properties. To encode these data, the buildingSMART[1] organization proposed the Industry Foundation Classes (IFC), a neutral and open ISO standard for

*Corresponding author

Email addresses: ihuitzil@unizar.es (Ignacio Huitzil), mmolinas@ic.ac.uk (Miguel Molina-Solana), jgomez@decsai.ugr.es (Juan Gómez-Romero), fbobillo@unizar.es (Fernando Bobillo)

[1] https://www.buildingsmart.org

BIM data [4]. The IFC specification defines a conceptual schema for BIM elements, encoded in the data modeling languages EXPRESS (ISO 10303-11) or XSD (XML Schema Definition), and file formats for specific building data, namely IFC-SPF (IFC STEP Physical Format) and ifcXML. Although these formats are light and easy to use, they lack the capabilities for sophisticated knowledge representation and reasoning offered by ontologies. Hence, there are several initiatives to evolve BIMs into semantic BIMs [5], powered by Semantic Web technologies (see Section 2.1 for details).

An ontology is "a formal, explicit specification of a shared conceptualization" [6], i.e., a definition of the vocabulary of a domain of interest consisting in axioms describing concepts, instances, and properties. By using a software called reasoner[2], one can infer facts which are implicitly contained in an ontology. The theoretical foundations of ontologies are Description Logics (DLs), a family of logics particularly well suited to represent structured knowledge [7]. OWL 2 (Ontology Web Language) [8] is the standard representation language and it is based on the RDF (Resource Description Framework) triple-based model [9].

Many real-world domains demand representing imprecise knowledge, vagueness, approximate reasoning, or flexible querying, for which classical ontologies do not provide support. To overcome this limitation, classical (i.e., *crisp*) ontologies have been extended with fuzzy logic [10] to create *fuzzy ontologies* [11]. In fuzzy ontologies, concepts and relations are modeled using fuzzy sets and fuzzy relations, respectively, and axioms and facts are not either true or false, but may hold to some degree of truth. Knowledge representation with fuzzy ontologies can be done with custom languages such as Fuzzy OWL 2 [12], while reasoning is supported by reasoning engines such as fuzzyDL [13] and DeLorean [14].

Objective. In a previous paper, we showed that fuzzy ontologies can accomplish information retrieval tasks not available in current BIM systems [15]; e.g., cross-domain information integration, flexible querying, and imprecise parametric modeling. Unfortunately, as highlighted in the conclusions, semantic BIM tools and fuzzy inference engines suffer some limitations in terms of scalability, efficiency and ease of use, which make them unsuitable for medium-scale models. In this paper, we address these problems by proposing and evaluating a new algorithm for efficient reasoning with fuzzy ontologies with application to the instance retrieval problem, arguably the most common one in BIMs and in many other domains. Our research approach is aligned to recent BIM research initiatives [16], which highlight the need for leveraging BIM data models and validating them on real use cases.

Contributions. More specifically, in the present work we developed a minimalistic-reasoning algorithm, where the term *minimalistic* refers to the fact that the algorithm cannot support any element of a fuzzy ontology, but only a selection of them useful for the instance retrieval task—namely, fuzzy datatypes and fuzzy concept assertions involving leaf concepts. These kind of queries is pervasive in real-world problems—and notably in BIMs—since they can be used to obtain the domain objects—i.e. building elements—that satisfy imprecise (and probably complex) constraints

[2] `http://owl.cs.manchester.ac.uk/tools/list-of-reasoners`

defined over their properties. However, solving them efficiently remains unexplored. The algorithm was implemented in a software prototype and its performance evaluated on sample fuzzy queries over a real-world BIM. Accordingly, the main contribution of the paper is the identification and optimization of a minimal set of reasoning tasks required for instance retrieval with restrictions in a fuzzy ontology, and particularly, in a fuzzy BIM. We also describe how these tasks can be expressed in terms of classical crisp inference, which can be solved by any classical (non-fuzzy) reasoning engine. Finally, we prove that the new algorithm can be useful to provide an efficient reasoning over some real BIM models.

Structure. This paper is structured as follows. In section 2, we overview some related works on reasoning. In section 3, we describe the bases and methods necessary for our research. In section 4, we propose a framework, describe the reasoning task and propose a reasoning algorithm. In section 5, we depict the implementation of the algorithm by means of an application that we tested with a real use case. Then, Section 6 discusses the pros and cons of our proposal. Section 7 concludes summarizing the main findings and pointing towards several pieces of future work.

2. Related work

2.1. Querying and reasoning over Semantic BIMs

The use of ontologies in the domains of architecture, engineering and construction (AEC) has notably increased over the last years, giving raise to the so-called semantic BIMs. Pauwels, Zhang and Lee stated that there are several motivations behind this interest [5]: (1) facilitating interoperability and information exchange between heterogeneous tools, (2) linking cross-domain information to exploit synergies of related domains, (3) equipping AEC data models with logic-based representation capacities. These authors concluded that are still many research gaps than remain unexplored, such as the combination of declarative and procedural techniques, and the automation of the data integration and retrieval procedures.

Recently, Mendes de Farias et al. explored the capabilities of rule-based reasoning in semantic BIMs [17]. They proposed the concept of *view* to represent a minimal usable sub-graph of elements extracted from an IFC file modeling a whole facility. The view is materialized as a knowledge graph based on the ifcOWL ontology [18], created by applying logical rules in SWRL (the Semantic Web Rule Language), and queried in the same language. The Stardog[3] triplestore is used to solve SPARQL [19] queries on RDF data and SWRL inferences. We perform a similar process to translate the heavyweight IFC files into a simpler OWL model, but we rely instead on the creation of modules based on the physical features of the building via a graphical user interface. We also leverage this interface to facilitate the creation of fuzzy queries over the IFC entities, instead of directly using SPARQL—which can be difficult for

[3] https://www.stardog.com/

non-expert users. Our algorithm also reduces the time required to solve the queries, which may take hours in their case.

Werbrouck et al. analyzed the limitations of IFC regarding modularity of BIM models and their support for query-solving [20]. Focusing on data represented as RDF triples, they presented a comparative of the usability and the performance of SPARQL against GraphQL-LD [21] and HyperGraphQL [22], two query languages based on the REST API language GraphQL. The transformation between IFC and RDF was done with IFCtoLBD, which we also use in this work. The authors showed that BIM models can exploit standard Linked Data languages for pattern-based query and data federation, but their expressivity is low: mostly simple RDF property-value and type-of queries on BIM elements are addressed. Our proposal supports instead a richer fuzzy extension of OWL 2, and at the same time, allows using existing reasoning engines.

Another proposal is that of Fahad et al., who focused on formal verification of IFC models by means of a Linked Data consistency checker—namely, the Semantic BIM Reasoner (SBIM-Reasoner) [23]. This issue was indeed mentioned in [20] (and also in [24], where the Shapes Constraint Language (SHACL) is suggested to address this issue). To that end, Fahad et al. developed a processing pipeline to extract geometry data from an IFC file, filter relevant information to reduce the model size, and create a resulting RDF graph. This model was managed with the Stardog triplestore via SPARQL queries and SWRL rules, in a similar way as in [17]. In contrast, our paper explores how fuzzy ontologies can be applied to define imprecise restrictions on data with the purpose of flexible querying. Fuzzy constraint satisfaction still remains as a future work.

To the best of our knowledge, the first approach to augment semantic BIMs with capabilities to manage imprecision and vagueness is our 2015 paper [15]. We used fuzzy ontologies and the fuzzy ontology reasoner *DeLorean* [14] to propose solutions to several AEC tasks: cross-domain knowledge linking (e.g. with partial concept inclusions and graded relationships), imprecise BIM queries (e.g. by using linguistic labels and imprecise topological relations) and fuzzy parametric modeling (e.g. by means of fuzzy axioms and maximization of their degree of fulfillment). In the current paper, we further develop these ideas and focus on one unsolved issue: the efficiency and the scalability of the reasoning algorithms. To that aim, we present a new algorithm for instance retrieval in large BIM models, which is evaluated on a real-world BIM.

Abualdenien and Borrmann highlighted that vague, imprecise, and incomplete information is frequent in the AEC industry, and acknowledged that is should be somehow incorporated into the BIM methodology [25]. These authors focused on the visualization of uncertain aspects of the building design, and particularly, vagueness of geometrical properties. In contrast to our work, they did not use a formal framework for the representation of uncertainty and imprecision. Our approach, based on Description Logics, allows us to guarantee the computational properties of the inference process and to use existing fuzzy and crisp reasoning engines.

Table 1 summarizes the main contribution and limitation of the previous approaches.

Reference	Main contribution	Main limitation
[17]	Semantic BIMs using RDF triples, OWL schema, SWRL rules, and SPARQL queries	Not scalable. No support for OWL reasoning tasks. No support for vagueness
[20]	Semantic BIM queries using HyperGraphQL and GraphQL-LD	Low expressivity. No support for vagueness
[23]	Consistency checking for Semantic BIMs using RDF triples and SWRL rules	No support for OWL reasoning tasks. No support for vagueness
[15]	Representation and reasoning with fuzzy Semantic BIMs	Not scalable algorithms
[25]	Visualization of vagueness in AEC	No formal model. No reasoning

Table 1: Related work on querying and reasoning over semantic BIMs

2.2. Efficient reasoning with fuzzy ontologies

Different families of reasoning algorithms for fuzzy ontologies can be found in the literature [26]. However, most of them are focused on showing the existence of an algorithm rather than on the efficiency in practice. For example, some reasoning algorithms are based on computing an equivalent crisp ontology, with a blowup in the size of the ontology [27]. DeLorean implements some of these algorithms. This is clearly not scalable and inappropriate to answer queries over real BIM models, with a very high number of individuals and axioms.

Because ontology languages provide a trade-off between expressive power and complexity of the reasoning, a first way to guarantee an efficient reasoning is to restrict the expressivity. In classical ontologies, the OWL 2 language has three sublanguages or profiles with tractable reasoning (i.e., the main reasoning tasks can be solved in a polynomial time), namely OWL 2 EL, OWL 2 QL, and OWL 2 RL [28]. In the fuzzy case, it has been showed that fuzzy extensions of tractable languages are not tractable in general [29], and they can even be undecidable [30]. Despite this fact, some fuzzy extensions of tractable DLs have been investigated, including fuzzy extensions of the logics behind OWL 2 EL [31, 32], OWL 2 QL [33], and OWL 2 RL [34].

We argue that this limited expressivity might not be enough to represent real BIM models. For example, OWL 2 EL does not support universal restrictions, which are important to represent that individuals of a class always have a certain property valued in a range. For example, we would want to represent that IfcStairFlight is a subclass of the set of elements which are related via the property riserHeight only with elements of the class IfcPositiveLengthMeasure (in Manchester syntax, riserHeight only IfcPositiveLengthMeasure).

Another approach is to develop specific optimization techniques to make reasoning more efficient in some common cases in practice. While many optimization techniques are known for classical DLs, optimizations for fuzzy DLs have not received such attention, but there are some exceptions. Haarslev et al. [35] proposed caching (to avoid repeating computations), lexical normalization (transforming concept expressions into a canonical form to detect inconsistencies earlier), simplifications of concept expressions, and ABox partitioning (splitting axioms about individuals—concept and property assertions—into disjoint sets). Simou et al. [36] proposed degrees normalization, to remove superfluous axioms when the same axioms is stated with different degrees of truth, and some optimizations of the algorithm to compute the best entailment degree of an axiom. Moreover, Bobillo and Straccia [37] proposed lazy unfolding, to delay the expansion of subclass axioms as much as possible, and an absorption algorithm to increase the applicability of lazy unfolding. fuzzyDL reasoner implements these and other optimization techniques, such as using different blocking strategies (adapted to the expressivity of the ontology) to guarantee the termination of the reasoning [13], or using some reasoning rules for some common special cases (such as n-ary conjunctions).

Finally, it is common to solve a reasoning task on fuzzy ontologies by reducing it to solving another one. For example, the instance retrieval problem can be solved by computing several entailment tests (one for each individual in the ontology). However, developing a specific reasoning algorithm is often more efficient. For example, we can mention a specific algorithm for the classification problem [26] and some recent algorithms to solve the realization and the instance retrieval problems [38]. In the present paper, we provide a new reasoning algorithm to solve a novel version of the instance retrieval problem. The main differences with the work in [38] is that we can reuse a classical DL reasoner, but imposing some restrictions on the language (for example, we only consider fuzzy concept assertions involving leaf concepts). Reusing classical reasoners is interesting because existing fuzzy ontology reasoners have limitations: most of them cannot completely support Fuzzy OWL 2 (e.g., fuzzyDL [13]) and the only current exception, DeLorean, implements a non-scalable algorithm [14].

Table 2 compiles the main contribution and limitation of the overviewed related work.

Reference	**Main contribution**	**Main limitation**
[27, 29]	Reasoning algorithms based on a reduction to crisp ontology reasoning. Classical reasoners can be reused	Not scalable (blow-up in the size of the computed crisp ontology)
[31, 32, 33, 34]	Fuzzy extensions of tractable languages (OWL 2 EL, OWL 2 QL, and OWL 2 RL)	Insufficient expressivity for a real fuzzy BIM
[13, 35, 36, 37]	Optimization techniques for fuzzy ontology reasoning	Classical reasoners cannot be reused
[26]	Specific algorithm for classification	Classical reasoners cannot be reused. Questionable scalability
[38]	Specific algorithms for realization and instance retrieval	Classical reasoners cannot be reused. Classical definition of the instance retrieval problem

Table 2: Related work on efficient reasoning in fuzzy ontologies

3. Background

This section overviews some basic notions on fuzzy logic (Section 3.1) and fuzzy ontologies (Section 3.2).

3.1. Fuzzy sets and fuzzy logic

Fuzzy logic is a generalization of classical logic proposed by Zadeh where statements are not necessarily either true or false, but hold to some degree of truth [10]. The cornerstone of fuzzy logic is the concept of fuzzy set, which is a generalization of a classical set where elements can have a partial membership. A fuzzy set A is characterized by a membership function $\mu_A(x)$ which associates with each object x a real number in $[0, 1]$ representing the membership degree of x in A. Figure 1 shows some examples of membership functions commonly used to build fuzzy

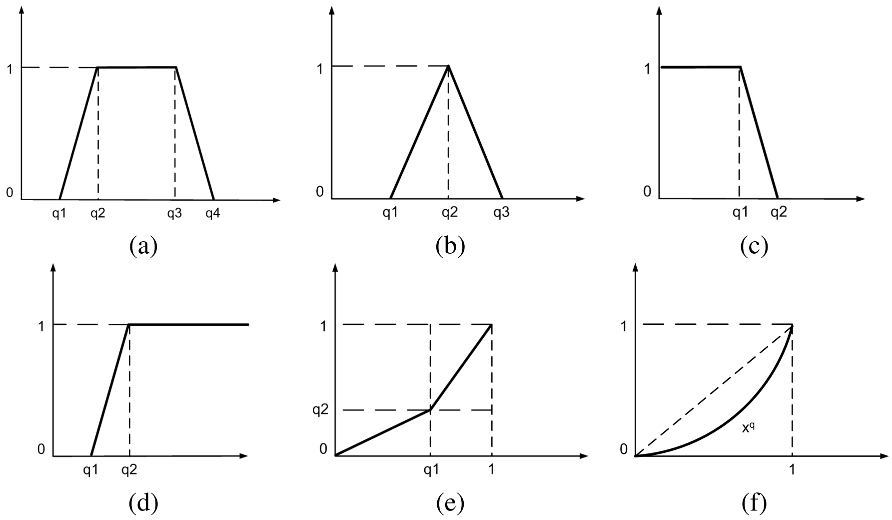

Figure 1: Typical fuzzy membership functions (borrowed from [13]): (a) Trapezoidal function; (b) Triangular function; (c) Left-shoulder function; (d) Right-shoulder function; (e) Linear function; (f) Power function

sets. For instance, TallWindow is a fuzzy set that contains tall windows. If a window window001 has a height of 1470 mm, we can evaluate the membership degree to TallWindow with a triangular function (Figure 1d) such as TallWindow(window001) $=$ (**triangular**$(1500, 1700, 2500))(1470) = 0.67$. This way, it becomes possible to represent imprecise information. In particular, the value of a property can be a *linguistic label* (represented as a fuzzy set) rather than a single numerical value.

Fuzzy logic enables approximate reasoning. Logical operations over classical sets are also generalized to the fuzzy case. To compute the conjunction, disjunction, complement and implication over fuzzy sets one can use different families of functions, namely a *t-norm*, a *t-conorm*, a *negation*, and a fuzzy *implication* (see [39] for details). For instance, the minimum (Min) is a t-norm and the maximum (Max) is a t-conorm.

Besides logical operations, there are other ways to combine fuzzy sets. For example, an aggregation operator is a function that takes n values in [0,1] (possibly representing the membership degrees to n fuzzy sets) and returns a single value in [0,1]. Some examples are the weighted mean (WMEAN) or the *Ordered Weighted Averaging* (OWA) operator [40]. Given a weighting vector $\mathbf{w} = [w_1, \ldots, w_n]$ such that $w_i \in [0, 1]$ and $\sum_{i=1}^{n} w_i = 1$, an OWA operator aggregates the values $x_1, \ldots, x_n \in [0, 1]$ into:

$$\sum_{i=1}^{n} w_i x_{\sigma(i)} \tag{1}$$

where σ is a permutation such that $x_{\sigma(1)} \geq x_{\sigma(2)} \geq \cdots \geq x_{\sigma(n)}$. Note that $x_{\sigma(i)}$ denotes the i-th largest value that we want to aggregate.

The problem of computing weighting vectors for OWA has been largely investigated. A popular solution is to use *quantifier-guided aggregation*. Given a Regular Increasing Monotone (RIM) quantifier [41], i.e., a function

$Q : [0, 1] \rightarrow [0, 1]$ satisfying some properties, each weight can be computed as:

$$w_i = Q(\frac{i}{n}) - Q(\frac{i-1}{n}) \tag{2}$$

For example, right-shoulder (Figure 1d), linear (Figure 1e), and power functions (Figure 1f) can be used to define RIMs.

To conclude this section, a *fuzzy modifier* (also called fuzzy hedge) modifies the shape of a fuzzy set by altering its membership function. Two common examples are the weakening modifier very, characterized by the function $\mathsf{very}(x) = x^2$, and the increasing modifier few, defined as $\mathsf{few}(x) = \sqrt{x}$. If we apply very to the fuzzy set $\mathsf{TallWindow}$, for each window x we can be compute the degree of being a very tall window as $\mu_{\mathsf{VeryTallWindow}}(x) = \mathsf{very}(\mu_{\mathsf{TallWindow}}(x)) = (\mu_{\mathsf{TallWindow}}(x))^2$. Fuzzy modifiers can also be defined, for example, using triangular (Figure 1b) or linear (Figure 1e) functions.

3.2. Fuzzy ontologies

Fuzzy ontologies are a generalization of classical ontologies based on fuzzy set theory and fuzzy logic [26]. They are useful in many real-world application domains to represent imprecise facts or axioms that are only partially true, and to enable approximate reasoning and flexible querying.

Fuzzy ontologies are a conceptualization of the world which can include the following elements:

- An *individual* is an object of the modeled domain, e.g., window001.
- A *data value* is a value from another domain, different to the one being modeled, such as an integer or a real number, a textual value, or a date. A *fuzzy datatype* is a generalization of crisp numerical values by using a fuzzy membership function instead (see e.g., Figure 1). For example, one can replace a crisp value 1700 mm with the fuzzy datatype HighOverallHeight, defined as **triangular**$(1500, 1700, 2500)$.
- A *fuzzy concept* (or fuzzy class) is a fuzzy set of individuals, e.g., TallWindow.
- A *fuzzy property* is a fuzzy binary relation between an individual and another individual or a data value.
  - An *object property* links two individuals, e.g., hasWindow links a building with a window.
  - A *data property* relates an individual and a data value, e.g., overallHeight links an individual with a real number.
- A *fuzzy axiom* states a restriction on the elements of the fuzzy ontology. A fuzzy axiom is not either true or false but might hold to some degree. In this paper, we will focus on the following types of fuzzy axioms:
  - A *fuzzy concept assertion* expresses a restriction on the membership degree of an individual to a fuzzy concept. For example, one can say that window001 belongs to the concept of TallWindow with at least degree 0.67, meaning that it is a quite tall window.

– A *fuzzy object property assertion* expresses that two individuals are partially related. For example, we can say that wall001 and window001 are related via hasWindow.

– A *fuzzy data property assertion* expresses that an individual is related to a data value, e,g,, one can state that a window has a height of 1634 mm by relating window001 and the number 1634 via overallHeight.

– A *fuzzy subclass* axiom ensures that a fuzzy concept is more specific than another one, i.e., it is a subclass of it. For instance, TallWindow is more specific than IfcWindow. A more complex example is that a TallWindow is related to the fuzzy datatype HighOverallHeight via the data property overallHeight.

Figure 2 illustrates a simple ontology with four classes (denoted with a yellow circle), one individual (purple rhombus), one data property (green rectangle), and a fuzzy datatype (red circle). Solid lines denote that a class is a subclass of another one. Dashed lines denote other axioms, namely a fuzzy concept assertion and a complex subclass axiom involving a data property and a fuzzy datatype. Figure 3 shows how to encode this fuzzy ontology using a fuzzy ontology editor.

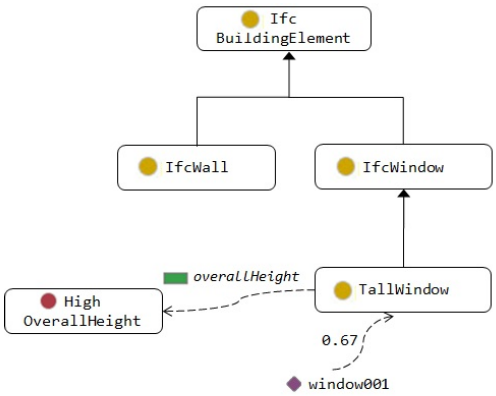

Figure 2: Fragment of the model of a fuzzy ontology

Several fuzzy ontology languages have been proposed in the literature. Among them, the most used one is *Fuzzy OWL 2* [12], which extends OWL 2 ontologies [8] with OWL 2 annotations encoding the fuzzy information that cannot be represented in standard OWL 2. Such annotations are represented using a special annotation property fuzzyLabel. To avoid dealing with the syntax of the language, there is a Protégé plug-in to develop Fuzzy OWL 2 ontologies[4].

Many reasoning tasks for fuzzy ontologies have been studied in the literature, and some reasoners have been implemented, such as fuzzyDL [13] and DeLorean [14]. In this work, we will define a new task similar to the *instance retrieval* problem [38] (i.e., retrieving all the instances of a fuzzy concept and their minimal degrees of membership).

[4] http://www.umbertostraccia.it/cs/software/FuzzyOWL/index.html

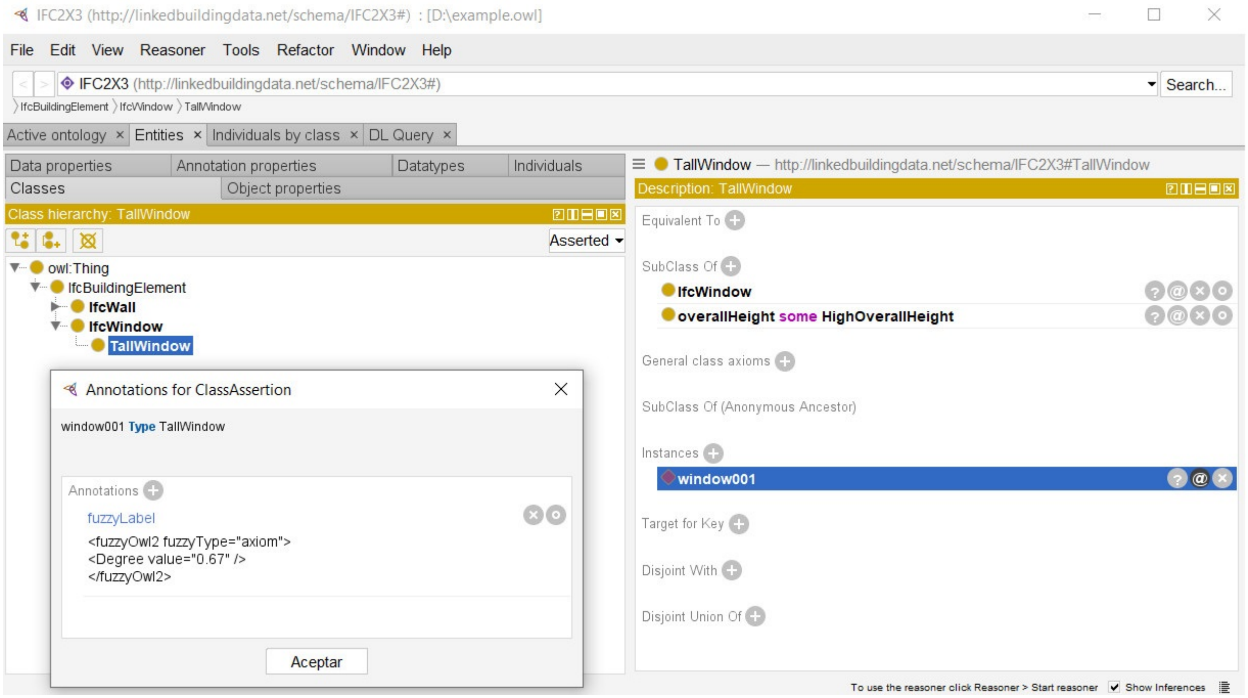

Figure 3: Snapshot of a fuzzy ontology edited in Protégé with the FuzzyOWL 2 plugin

4. Minimalistic fuzzy ontology reasoning

To solve flexible queries over a BIM model, we propose a framework composed of the following steps:

- Representation of the BIM model using an OWL 2 ontology. This involves using a conversion tool and defining a BIM schema.
- Split of the BIM model in several subontologies with smaller size which fit our hardware requirements. The reason is that representing real BIM data usually leads to very large OWL 2 ontologies.
- Fuzzification of the ontology. On the one hand, one can define linguistic labels using fuzzy datatypes. On the other hand, one can state partial membership of individuals to classes using class assertion axioms.
- Retrieval of the instances of the ontology that satisfy a given (flexible) query.

In this section we will focus on the last step of the process, in which the minimalistic fuzzy reasoning is applied. In particular, we will define a novel reasoning task, namely flexible faceted instance retrieval (Section 4.1), discuss a more specific case obtained after some simplifications (Section 4.2), and propose a reasoning algorithm (Section 4.3). The other steps will be discussed in Section 6.

4.1. Flexible faceted instance retrieval

Let us describe the reasoning task *flexible faceted instance retrieval*. The idea is to extend classical fuzzy instance retrieval to narrow down the query results by imposing some conditions on the attribute values. In particular, given a fuzzy ontology O, our aim is to retrieve the instances of a fuzzy concept C such that the values of n functional data properties p_i with a numerical range are compatible with a fuzzy datatype D_i. For example, retrieving all the instances of IfcWindow (or its subclasses) such that their overallHeight and overallWidth, representing the height and the width of a window, are Low and High, respectively. Furthermore, the intermediate degrees of truth can be combined using a combination function F_c (a t-norm, a t-conorm, or an aggregation operator such as the weighted sum or OWA), and the final degree can be modified using a modifier function F_h (a fuzzy hedge), e.g., to ensure that the query is very much satisfied. Formally:

Definition 1 (Flexible faceted instance retrieval). *Given the sextuple $\langle O, C, [p_1, \ldots, p_n], [D_1, \ldots, D_n], F_c, F_h\rangle$, the solution to the flexible faceted instance retrieval is an ordered list of pairs $\langle i_i, \beta_i\rangle$ such that*

$$\begin{aligned}
&O \models \langle C(i_i), \alpha_i\rangle \,, \\
&O \models p_j(i_i, v_j),\, j \in \{1, \ldots, n\} \,, \\
&\beta_i = F_h(F_c(\max\{\alpha_i\}, D_1(v_1), \ldots, D_n(v_n))) > 0 \,, \\
&\beta_i \geq \beta_j,\, j > i \,.
\end{aligned} \tag{3}$$

Example 1. *Consider a BIM ontology O where the class IfcBuildingElement has 5 sibling subclasses, namely IfcWindow, IfcDoor, IfcColumn, IfcWall, and IfcStair. IfcWindow has some subclasses BasicWindow, HistoricWindow, SlidingWindow, and SpecialWindow. Such subclasses are non-direct in general, for example, InteriorBasicWindow is a direct subclass of BasicWindow, which is a direct subclass of IfcWindow. Note that IfcWindow and all its subclasses are crisp concepts, so all the $\alpha_i = 1$. There are 2 data properties overallWitdth and overallHeight and 5 individuals GUI_eYJ, GUI_jTl, GUI_pCt, GUI_rYh, and GUI_nVz. The following table shows for each window to which subclass of IfcWindow it belongs to, the width, and the height at millimeters:*

Window	Type	overallWidth	overallHeight
GUI_eYJ	InteriorBasicWindow	1200	2000
GUI_jTl	HistoricWindow	900	800
GUI_pCt	SpecialWindow	1430	2512
GUI_rYh	SlidingWindow	1000	2200
GUI_nVz	BasicWindow	940	1760

We want to retrieve the instances of IfcWindow such that the overallWidth is HighWidth and the overallHeight is HighHeight, using as a combination function the minimum t-norm $f_c(x_1, \ldots, x_k) = \min\{x_1, \ldots, x_k\}$ and using as a modifier function the fuzzy hedge very defined as $f_h(x) = x^2$. HighWidth (fuzzy datatype) is defined as a triangular fuzzy *function* **triangular**$(900, 1200, 2000)$ *and* HighHeight *is defined as a triangular fuzzy function* **triangular**$(1500, 1700, 2500)$. *Remember that* $\beta_i = F_h(F_c(\max\{\alpha_i\}, D_1(v_1), \ldots, D_n(v_n)) > 0$. *Thus:*

Window	HighWidth	HighHeight	β_i
GUI_eYJ	1	0.63	0.39
GUI_jTI	0	0	0
GUI_pCt	0.71	0	0
GUI_rYh	0.33	0.37	0.11
GUI_nVz	0.13	0.93	0.02

Therefore, the answer would be:

$$\Big\{\langle \mathit{GUI\_eYJ}, 0.39\rangle, \langle \mathit{GUI\_rYh}, 0.11\rangle, \langle \mathit{GUI\_rYh}, 0.02\rangle\Big\} \quad \square$$

This very general case could be simplified in different ways. For example, C can be a crisp concept, there can be a smaller number of properties (or even none), and F_h can be omitted assuming that it is the identity function. Also, it is trivial to extend the reasoning task to consider only the top-k results.

4.2. A more specific scenario

Our aim now will be to propose a reasoning algorithm for more specific, but still common in practice, cases:

Restriction 1 We assume that the only fuzzy elements that the fuzzy ontology can contain are fuzzy concept assertions and fuzzy datatypes.

Restriction 2 We only take into account those partial memberships that are stated via a fuzzy concept assertion $\langle C(i) \geq \alpha\rangle$ with $\alpha > 0$ (that will also be propagated to the named concepts that are superclasses of C).

Restriction 3 We assume that if an individual partially belongs to a concept, it does not fully belong to another concept (except to owl:Thing).

Example 2 shows an example of an implicit fuzzy concept assertion that is excluded by Restriction 2.

Example 2. *Let us assume* $\{\{i\} \sqsubseteq (A \sqcup A)\} \in \mathcal{O}$ *under Łukasiewicz family of fuzzy operators. Therefore, according to the usual semantics (see for example [27]), for each element* x *of the domain,* $(\{i\})^{\mathcal{I}}(x) \leq (A \sqcup A)^{\mathcal{I}}(x)$ *holds. In particular,* $x = i^{\mathcal{I}}$ *implies* $1 \leq (A \sqcup A)^{\mathcal{I}}(i^{\mathcal{I}})$, *so* $A^{\mathcal{I}}(x) \oplus A^{\mathcal{I}}(x) = \min\{A^{\mathcal{I}}(i^{\mathcal{I}}) + A^{\mathcal{I}}(i^{\mathcal{I}}), 1\} \geq 1$, *and thus* $2 \cdot A^{\mathcal{I}}(i^{\mathcal{I}}) \geq 1$, *so* $A^{\mathcal{I}}(i^{\mathcal{I}}) \geq 0.5$ *holds. Therefore, the fuzzy ontology entails a fuzzy concept assertion* $\langle A(i) \geq 0.5\rangle$ *that is not explicitly represented in* $\mathcal{O}$. $\square$

Therefore, if $\langle i, \alpha \rangle$ is in the solution of the flexible faceted instance retrieval of a fuzzy concept C and $\alpha < 1$, there is at least a fuzzy concept assertion of the form $\langle C'(i) \geq \alpha \rangle$ in the ontology, for some subclass C' of C. Formally:

$$\begin{aligned} O &\models C' \sqsubseteq C \, , \\ O &\ni \langle C'(i) \geq \alpha \rangle \, . \end{aligned} \tag{4}$$

Note in particular that the case $C' = C$ is possible. The rationale behind this restriction is to avoid computing the membership degrees of individuals to classes using an ontology reasoner. We instead assume that there is one (or more) fuzzy concept assertions and propagate the membership degrees to the superclasses of the concept. If there is more than one fuzzy concept assertion involving subclasses of C, we can take the maximum of the membership degrees $\max\{\alpha_i\}$.

Example 3. *Assume that b is a* SpecialWindow *with degree* 0.9 *and a* BasicWindow *with degree* 0.8*. Then, the membership degree to the common superclass* IfcWindow *is* $\max\{0.9, 0.8\} = 0.9$. □

Note also that we do not restrict to a specific family of fuzzy operators (Zadeh, Gödel, Łukasiewicz, or Product). Because we only consider fuzzy datatypes and the propagation of fuzzy concept assertions to their superclasses, where the subclasses axioms are fully true, our algorithm does not depend on the choice of the fuzzy operators.

To efficiently retrieve the degrees α_i without using a reasoner, we retrieve them from the OWL 2 annotations and store them in an appropriate data structure (such as a noSQL database storing triples) for an efficient data access. In particular, for each fuzzy concept assertion $\langle C(i) \geq \alpha \rangle \in O$, we add a tuple $\langle i, C, \alpha \rangle$ to the data structure. Note that it is possible to visit all fuzzy concept assertions in a Fuzzy OWL 2 ontology, by looping over all existing annotation assertions involving the fuzzyLabel property.

We avoid adding to the data structure individuals which fully belong to a concept. That is, for each classical concept assertion $C(i)$ we do not add to the data structure a tuple $\langle i, C, 1 \rangle$. The reason is that it is not efficient to retrieve each concept C' such that $O \models C'(i)$ but $C'(i) \notin O$; in particular, we would need to use an ontology reasoner.

Note that if there is a classical assertion stating that an individual belongs to a class, there is no annotation assertion. Therefore, individuals fully belonging to a class are not stored in the data structure. Given a flexible faceted instance retrieval over a concept C, it is fine to have an individual partially belonging to several fuzzy concepts that are subclasses of C (i.e., appearing in more than one fuzzy concept assertion) and it is fine to have an individual fully belonging to several fuzzy concepts that are subclasses of C. Restriction 3 forbids having both cases at the same time, and it is needed to propagate a membership degree to a class C' to a (possibly non-direct) superclass C without having to check if the individual fully belongs to C.

4.3. An algorithm

We call our approach minimalistic reasoning because it is restricted to a more specific case and imposes some assumptions to reuse classical ontology reasoners, providing an incomplete solution (i.e., if a fuzzy ontology does not satisfy our constraints, some inferences can be missed). Algorithm 1 shows how to compute the flexible faceted instance retrieval of a fuzzy ontology under the restrictions enumerated in the previous section.

The first part (Lines 2–7) is an initialization that can be computed just once, and can be reused by future queries. Firstly, we load the ontology (Line 2), classify the ontology by computing the hierarchy of concept names that fully subsumes their subclasses (Line 3), and store the degrees of the fuzzy concept assertions in an auxiliary data structure *DS* (Lines 4–7). The rest of the code (Lines 9–35) implements the proper query answering. The next steps are retrieving all instances of C, noting that some of them might partially belong to C (Line 9) and retrieving all subclasses of C (Line 10). Then, we will look in the data structure if each retrieved instance appears in the data structure (partial membership) or not (fully membership). In particular, Lines 13–23 compute the maximum of the degrees in the data structure (1 if there is none). The next step is to compute the satisfaction degrees of the linguistic labels associated to the attributes of the instance. Therefore, Lines 24–29 retrieve the values (which must be unique because the properties are functional) of each data property and compute the membership degrees to the respective fuzzy datatypes (0 if the value of the property is unknown). All the obtained degrees are aggregated in Lines 30–32 using a combination functions and a fuzzy hedge, and then added to a list of solutions. Finally, the list is ordered and returned.

One of the key points of the algorithm is that Lines 3, 9, 10, and 25 can be obtained using a classical ontology reasoner and, therefore, rather efficiently.

5. Implementation and evaluation

In this section we describe a prototype implementation and an evaluation of our tool on a fuzzy ontology obtained from a real BIM model. Firstly, we discuss the reuse of classical ontology reasoners (Section 5.1). Then, we describe the implementation of the tool (Section 5.2). Next, we describe the dataset, taken from a real use case (Section 5.3), and the results of an empirical evaluation (Section 5.4).

5.1. Reuse of classical ontology reasoners

Line 9 of Algorithm 1 requires solving the instance retrieval concept, Line 10 requires solving the classification problem, and Line 25 requires retrieving the values of data property, possibly not explicitly stored in the ontology. While the two former tasks are relatively well supported by a number of reasoners, this is not the case of last one. HermiT reasoner is one of the few exceptions, as it has indeed a method `getDataPropertyValues` to solve Line 25. TrOWL is a reasoner for the OWL 2 EL profile and by means of the OWL API it is possible to access to the data properties values.

There is another way to get the real values: using a SPARQL query. Figure 4 illustrates the results of a simple query to obtain the overallHeight and overallWidth values of the instances of IfcWindow class (for the third floor ontology obtained using IFC-to-RDF converter [42]). It is clear that if we need to infer knowledge or to classify the ontology, SPARQL is not appropriate.

Algorithm 1 Algorithm to compute the flexible faceted instance retrieval assuming Restrictions 1–3.

Input: A sextuple $\langle uri, C, [p_1, \ldots, p_n], [D_1, \ldots, D_n], F_c, F_h \rangle$ composed by a fuzzy ontology URI uri, a concept C, a list of functional numerical data properties $[p_1, \ldots, p_n]$, a list of fuzzy datatypes $[D_1, \ldots, D_n]$, a combination function F_c, and a fuzzy hedge F_h

Output: A list of pairs with individuals and membership degrees to C $\left\{\langle i_i, \beta_i \rangle\right\}$

// Initialization
$O \leftarrow loadOntology(uri)$
$O \leftarrow crispClassify(O)$
$DS \leftarrow \emptyset$
for all $\langle C(i) \geq \alpha \rangle \in O$ **do**
    $DS \leftarrow DS \cup \langle i, C, \alpha \rangle$
end for
// Query answering
$I \leftarrow$ Retrieve all instances i of C in O
$S \leftarrow$ Retrieve all subclasses of C in O
$Sol \leftarrow \emptyset$
for all $i \in I$ **do**
    $A \leftarrow \emptyset$
    for all $s \in S$ **do**
        if $\langle i, s, \alpha \rangle \in DS$ **then**
            $A \leftarrow A \cup \alpha$
        end if
    end for
    if $A = \emptyset$ **then**
        $\alpha \leftarrow 1$
    else
        $\alpha \leftarrow \max(A)$
    end if
    for all data property p_i **do**
        $v \leftarrow$ Retrieve the value of the data property p_i for i in O
        if v $\neq$ null **then**
            $D \leftarrow D \cup D_i(v)$
        end if
    end for
    $auxDegree \leftarrow F_c(\alpha, D)$
    $\beta \leftarrow F_h(auxDegree)$
    $Sol \leftarrow Sol \cup \langle i, \beta \rangle$
end for
$Sol \leftarrow sort(Sol)$ in decreasing order of degrees of truth
return Sol

Figure 4: SPARQL query for IFC-to-RDF converter file

5.2. *Implementation*

We developed a prototype tool which is available online[5]. It implements Algorithm 1 and a graphical interface to submit queries. It is a Java (1.8) implementation using the OWL API[6] to manage OWL 2 ontologies represented in Fuzzy OWL 2 language. The classical semantic reasoner used is TrOWL 3.4. To reduce the time to access the ontology, we stored the fuzzy concept assertions using a hash table and a NoSQL database (MongoDB 4.0.10). As a baseline, we also considered direct calls to the OWL API. A graphical user interface (for desktop computers) makes it possible to submit queries about building elements. Appendix A shows some snapshots of our tool.

The general functionality of this software is shown next. The tool contains three tabs:

- The first one (see Figure A.8) specifies the path of the ontology and the base URI (it corresponds to the IFC2X3 schema) by default. The converter software uses the base URI `http://linkedbuildingdata.net/schema/IFC2X3#`. Sometimes that URI could change, as it depends on the converter or the version schema. The fuzzy ontology file can have .owl or .ttl extensions. The user also needs to select the IFC element (a class) from the schema, such as IfcWindow.

  In this tab the user also needs to select the operator to combine the values and a fuzzy modifier. Possible operators include minimum (T-norm Min), maximum (T-conorm Max), weighted mean (WMEAN), and OWA. Figure A.12 shows an example of OWA operator built using quantifier-guided aggregation. Possible modifiers are none, very, few, linear, and triangular. Figure A.8 shows as an example the definition of very.

[5] `http://webdiis.unizar.es/~ihvdis/fuzzyBIMgui.html`
[6] `http://owlapi.sourceforge.net`

- The second tab shows all the data properties in the ontology and the user has the possibility to select some of them. Figure A.9 shows an example where overallHeight and overallWidth properties are checked.
- The third tab allows to select or create the fuzzy datatypes for the chosen data properties (see Figure A.10). One way is to select fuzzy datatypes already defined in the ontology file. It is also possible to create a new fuzzy datatype, using labels like VeryLow, Low, Neutral, High, and VeryHigh, and membership functions such as left-shoulder, triangular, trapezoidal, and right-shoulder.

Initially, the Run button is disabled until all necessary parameters are specified. When the run button is clicked, a process is executed to solve the query. Eventually, a dialog with a sorted list of instances is displayed, as shown in Figure A.11.

Example 4. *Assume we need to retrieve a set of windows with high width and very high height from the fuzzy ontology. So, we use the desktop tool and ask for a IFC building element called* IfcWindow*. We consider two data properties of a window, namely* overallWidth *and* overallHeight*. We define two fuzzy datatypes* HighOverallWidth*, using a triangular fuzzy function* **triangular**(900, 1200, 2000)*, and* VeryHighOverallHeight*, using a right-shoulder fuzzy function* **right**(1700, 2500)*. We choose the maximum t-conorm operator (auxDegree) and the fuzzy modifier* very *(β_i). Table 3 shows the evaluation of 12 window instances. α_i denotes the degree used in the fuzzy concept assertion, and was added randomly to each window in the fuzzy ontology. Figure A.11 shows the result: a sorted list of windows (colored using the satisfaction degree of the query).* □

Window	overallWidth	overallHeight	tri	right	α_i	auxDegree	β_i
GUID_kMl	1430	2512	1	0.71	0.20	1	1
GUID_O8D	940	1760	0.13	0.75	0.70	0.70	0.48
GUID_7nV	940	1760	0.13	0.07	0.10	0.13	0.01
GUID_S27	940	1760	0.13	0.07	0.60	0.60	0.36
GUID_eYJ	1430	2512	0.71	1	1	1	1
GUID_RyI	916	1760	0.05	0.07	0.50	0.50	0.25
GUID_wCu	940	1760	0.13	0.75	0.40	0.40	0.16
GUID_jtL	1430	2512	0.71	1	0.10	1	1
GUID_hhq	1430	2512	0.71	1	0.90	1	1
GUID_pct	916	1760	0.05	0.07	0.80	0.80	0.64
GUID_41F	940	1760	0.13	0.07	0.30	0.30	0.09
GUID_lLC	916	1760	0.05	0.07	0.20	0.20	0.04

Table 3: Set of individuals from IFCWindow

5.3. Use case: a fuzzy ontology for a real BIM model

We evaluated our proposal using the Schependomlaan public BIM dataset[7]. This project was developed and built by Hendriks Bouw en Ontwikkeling[8] and comprises 10 apartments located in Nijmengen, Netherlands. The dataset

[7] https://github.com/openBIMstandards/DataSetSchependomlaan
[8] https://www.hendriksbouwenontwikkeling.nl/en

Tool	Classes	Data Properties	Object Properties	Individuals
IFC-to-RDF	1085	929	1502	10127

Table 4: Statistics of the conversion of the third floor.

contains a design model in IFC, extract, suppliers, point clouds, schedules and construction log files. Figure 5 shows the 3D model visualised on the academic version of Archicad 22[9].

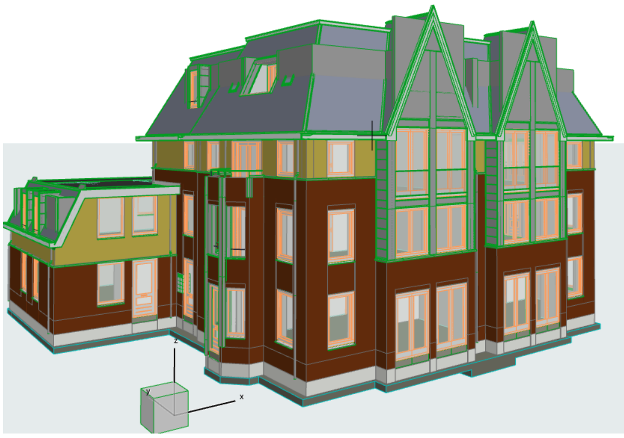

Figure 5: Use case: 3D representation

For our purpose, we need to obtain an ontology from the IFC model of the use case. The ontology that we need should consider classes, individuals and relationships (data and object properties). For example, the class IfcDoor has the instance IfcDoor_01 with a data property overallHeight equals to 2282 mm and an object property representation linking it to the b179 instance. After testing four IFC converters (IFC-to-RDF Version 1.0[10] [42], IFC2LD[11] [43], IFCtoRDF[12], and IFCtoLBD[13] [44]) we selected IFC-to-RDF.

In order to reduce both the file size and the reasoning time, we divided the use case in six submodules (the six storeys of the original IFC building model) that correspond to foundation, ground floor, first floor, second floor, third floor, and roof. The fragmentation task was manually done with the help of the graphical environment Archicad. Next, we exported to IFC format and then used the converter to obtain the ontology.

We focused on the third floor, because it has the smaller .ifc and .ttl files, and that makes reasoning more feasible without loss of generality. Table 4 shows some statistical data about the ontology representing the third floor.

Furthermore, we defined a *modified* version by making the following changes:

[9] https://www.graphisoft.es/archicad

[10] Not available online anymore. Latest version (1.5) is called Ifc2Rdf and is available at https://github.com/Web-of-Building-Data/Ifc2Rdf/tree/master/software

[11] https://github.com/Web-of-Building-Data/ifc2ld.git

[12] https://github.com/pipauwel/IFCtoRDF

[13] http://github.com/jyrkioraskari/IFCtoLBD

1. We removed the graphic elements that do not have property values that are needed for our queries. For example, walls or columns that do not have a height and a width. The priority is a high number of windows. The complete ontology has 12 windows, and 8 of them have values. The reduced ontology was updated to have the 12 windows (we used the tool Measure of Archicad to obtain the missing sizes).
2. We modified in the schema file the range of the data properties overallHeight and overallWidth, to make it xsd:double.
3. We added some new classes representing specific styles defined in Archicad, and created some new instances of them (via concept assertions). For the IfcWindow class we added 9 subclasses, namely BasicWindow (with 8 instances), DormersAndSkylights, EmptyWindowsOpenings, HistoricWindow, SingleDoubleHungWindow, SindingWindow, SpecialWindow (with 4 instances), StoreFronts, and TerraceDoors. For IfcDoor class we added 8 subclasses, namely Bed, EmptyDoorOpenings, GarageDoor, HingedDoor(2 instances), SidingFoldingDoor. For IfcWall class we added 5 subclasses: GenericWall, ExteriorWall, InteriorWall, PartitionalWall, and StructuralWall.

Finally, we fuzzified the ontology representing the third floor for testing our novel algorithm. We firstly defined a fuzzy ontology (called *Fuzzy1*) using the plugin Fuzzy OWL 2 for Protégé 4.3. In particular, we added 12 fuzzy concepts assertion, adding a degree of truth to some axioms at BasicWindow and SpecialWindow classes. The degree values in (0,1) were chosen in random way. 10 fuzzy datatypes were created based on our experience about size windows [12]. The definition of the window labels is shown in Figure 6; Section 6 discusses how to build them.

We also created another version (*Fuzzy2*) by adding more individuals to the fuzzy ontology (in particular, 6498 individuals, with 1400 windows and 100 doors).

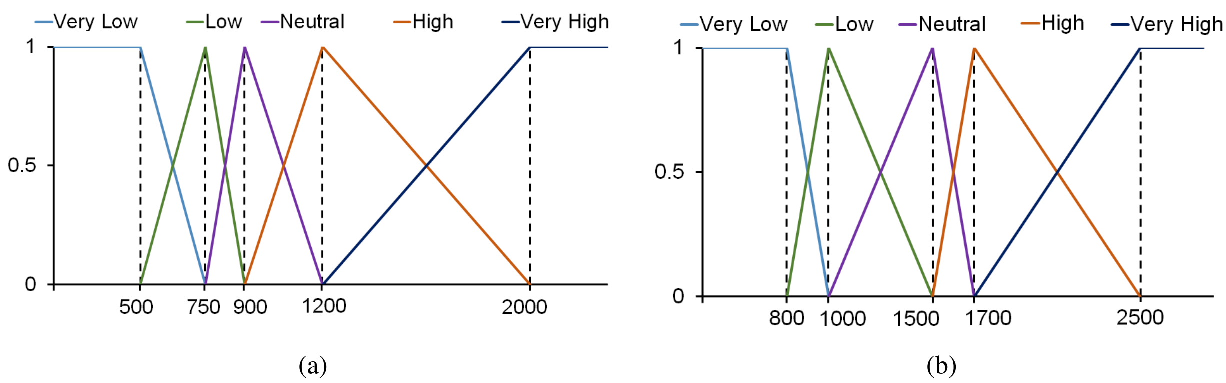

Figure 6: Linguistic labels for a) overallWidth and b) overallHeight

The fuzzy ontology and schema were saved using OWL/XML syntax. The ontology lost some valuable data (such as graphic placement and anonymous nodes) but this does not affect the result of our queries.

Ontology	File size (MB)	Reasoner	Time (s)	Individuals
Original	27.8	HermiT	OutOfMemoryError	10127
Modified	15.1	HermiT	OutOfMemoryError	5498
Fuzzy1	56.4	HermiT	OutOfMemoryError	5498
Original	27.8	TrOWL	OutOfMemoryError	10127
Modified	15.1	TrOWL	80.38	5498
Fuzzy1	56.4	TrOWL	83.28	5498

Table 5: Time (s) to load and classify the ontology

Ontology	Hash table	DB	OWL API
Fuzzy1	0.07	1.09	0.01
Fuzzy2	0.32	2.04	6.52

Table 6: Time (s) to create the data structures

5.4. Evaluation

Firstly, we evaluated the initialization time of our tool, which includes loading the ontology, computing the classification, and the initialization of a data structure with the degrees of truth. Secondly, we evaluated the proper query time, as well as the time to retrieve the values of the data properties. The evaluation was performed on a Intel Core i7-8550U 1.8 GHz, 16 GB RAM (7 GB were allocated for the JVM) laptop running Windows 7 64-bits.

Initialization time. Before describing the evaluation of the initialization time, it is worth to recall that it must be computed just once. Firstly, we tested two classical reasoners (Hermit 1.3.8[14] and TrOWL 3.4[15]) to measure the load and classification times for the original, modified, and fuzzy ontologies of the third floor. Table 5 shows the ontology, file size, reasoner used, time and number of named individuals. Time includes the time to load the ontology, to classify it by precomputing the class hierarchy and the class assertions, and to perform a consistency test. Note that HermiT run out of memory in all cases, after approximately 20 minutes. TrOWL also run out of memory for the original ontology, but the modified versions could be successfully processed.

We also evaluated the use of the auxiliary data structures to reduce the answering time (Lines 4–7). The results are shown in Table 6. For ontology Fuzzy1, OWL API method was slightly faster than the hash table, so it seems to be the best option to avoid the cost of maintaining the data structure. In particular, for such ontologies with a small number of fuzzy concept assertions, the database performs worse than the OWL API. For the ontology *Fuzzy2*, hash table is clearly faster than the other two methods.

Query time. Next, we evaluated the time to obtain the values from the data properties (height and width) of each individual (Lines 24–29). We used TrOWL reasoner and SPARQL for the modified and *Fuzzy1* versions. For the SPARQL queries we used the server Apache Jena Fuseki 3.14 and Jena Java API.[16]. It is worth to note, however, that

[14] http://www.hermit-reasoner.com
[15] http://trowl.org/download-page
[16] http://jena.apache.org

Ontology	Reasoner	Loading + classification time (s)	Query time (s)
Modified	Jena	4.40	1.160
Modified	TrOWL	80.38	0.007
Fuzzy1	TrOWL	83.28	0.007

Table 7: Time (s) to get the data properties values in the IfcWindow class

a SPARQL query cannot be used in general to solve a query to the ontology, but only to retrieve the data property values. Table 7 shows the results (the average of five executions) for the fuzzy ontology. For the first query, the SPARQL query is solved faster than using the reasoner because it only needs to load the ontology, but the reasoner performs a more complex preprocessing including classification. However, for the next queries the reasoner is faster.

Then, we evaluated the full query time (Lines 9–35). Starting from the ontology *Fuzzy1* (with 5498 individuals where 12 are windows and 2 doors), we created a set of 6 queries, 5 of them about IfcWindow class and 1 about IfcDoor class. To get the query time, the 6 queries were executed in a sequential way on an instance of the tool. Queries were solved 5 times and we computed the average values. Table 8 summarizes the queries and the results. The first columns include the query ID, and the parameters of the query: the class, the data property, the label (fuzzy datatype), the aggregation operator, and the modifier. The final columns include the query time when using a hash table, a NoSQL Database Mongo DB, or only calls to OWL API methods. As already discussed, hash table is slightly preferable.

#	Class	Property	Label	Operator	Modifier	Time (s)		
						Hash table	**DB**	**OWL API**
1	IfcWindow	overallWidth overallHeight	High VeryHigh	T-conorm Max	Very $f_m(x) = x^2$	0.14	0.18	0.18
2	IfcWindow	overallWidth overallHeight	Neutral High	T-norm Min	Few $f_m(x) = \sqrt{x}$	0.11	0.12	0.11
3	IfcWindow	overallWidth overallHeight	Low Neutral	OWA	Linear (0.3)	0.06	0.10	0.06
4	IfcWindow	overallWidth overalHeight	Low Neutral	WMEAN	tri (1000, 1500, 2000)	0.05	0.11	0.05
5	IfcWindow	overallWidth overallHeight	Neutral Low	T-norm Min	None	0.05	0.09	0.03
6	IfcDoor	overallWidth overallHeight	High High	T-conorm Max	Few $f_m(x) = \sqrt{x}$	0.10	0.09	0.08

Table 8: Queries and query time (s) for ontology *Fuzzy1*

We also repeated the same queries for the ontology *Fuzzy2*. Figure 7 shows the result of the query times. We can see that using the best data structure, query time is very fast (less than 0.62 s), making our algorithm acceptable for such models.

The previous query times assume that the system has already been initialized. Table 9 shows the total time for the first query. Likewise for the query time, OWL API performs similarly to the hash table version for *Fuzzy1*, but hash table performs clearly better for *Fuzzy2*.

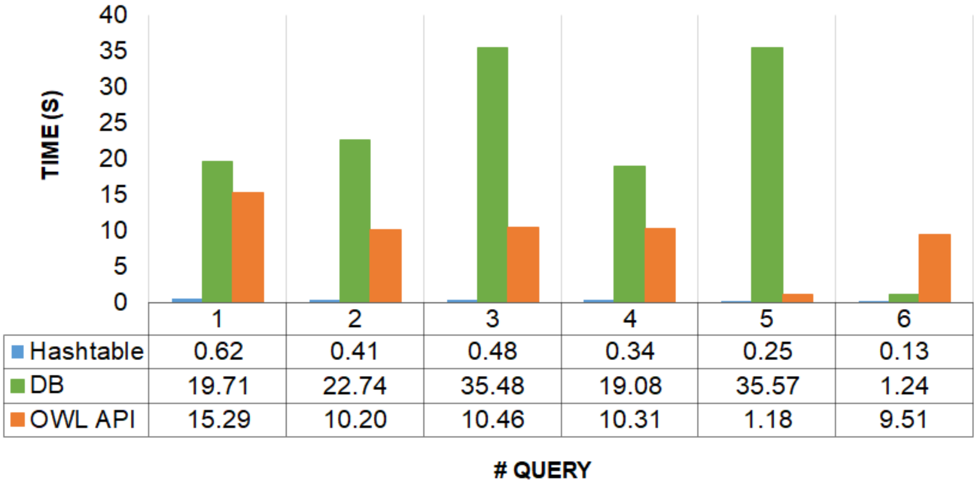

Figure 7: Query time (s) for ontology *Fuzzy2*

Ontology	**Task**	**Time (s)**		
		Hash table	**DB**	**OWL API**
Fuzzy1	Loading + Classification	83.28	83.28	83.28
	Data structure	0.07	1.09	0.01
	Query	0.08	0.11	0.1
	Total	83.43	84.48	83.39
Fuzzy2	Loading + Classification	112.63	112.63	112.63
	Data structure	0.32	2.04	6.52
	Query	0.37	22.30	9.49
	Total	113.32	136.97	128.64

Table 9: Total time (s) for the first query

6. Discussion

This section summarizes the pros and cons of our approach, debates possibilities of our framework, discusses how to learn its elements, and examines the verification of the results.

Pros and cons. Our framework provides a solution to a problem identified in [15]: the need to improve scalability of the reasoning with fuzzy semantic BIMs, i.e.. of the reasoning algorithms for fuzzy ontologies representing extensions of semantic BIMs with fuzzy logic. In this way, the practical use cases envisioned in that previous work are feasible even for large BIMs—i.e., integration of cross-domain knowledge, imprecise BIM query, and flexible parametric modeling. In particular, more efficient querying over BIM elements and geometric relations is the most straightforward application of our framework.

A notable contribution with respect to the existing work by other authors is that we support more expressive queries. In particular, we support answering flexible queries thanks to the linguistic labels of the fuzzy ontology. This unique service is provided to users by a publicly available software. Furthermore, our experiments show that our framework can reduce the query time. While we were able to answer queries over real BIM data in a reasonable time, some previous works needed hours to solve the queries.

On the negative side, our approach also has some limitations. As already mentioned, some steps of the framework require manual intervention so far, in particular splitting a large file into subontologies. Moreover, only some of the features of fuzzy ontologies (fuzzy datatypes and fuzzy concept assertions) are supported by our minimalistic reasoning algorithm.

BIM ontology. Our framework requires a representation of our BIM model using an OWL 2 ontology. Firstly, this involves a conversion from IFC to RDF. In this work we used the IFC-to-RDF tool [42], but using more sophisticated parsers could be possible. Secondly, it involves using an OWL schema to categorize BIM elements. In this work, we used the ifcOWL ontology. Another option is the Building Typology Ontology (BOT)[17], or the BIM schemas used by other conversion tools.

In general, when dealing with real data, one needs to split the ontology into smaller subontologies. In this work, we did it manually. It would be possible to study methods to compute a split automatically given some restrictions. In particular, one could consider using a method to reduce the geometrical data (e.g., position and orientation of the building elements) which are not necessary unless one wants to reason with spatial semantics [45]. This makes it possible to reduce the size of the ontology while having a more efficient representation for some queries, e.g., those involving intersections of building elements.

Learning. A common problem in ontology development is how to obtain the linguistic labels, i.e., the concrete definitions of the fuzzy datatypes. We did it manually in the example discussed in this paper but it would be recommendable

[17] http://www.student.dtu.dk/~mhoras/bot/index-en.html

to use supporting tools, such as Datil [46] or Fudge [47]. Datil makes it possible to learn the definitions from numerical data: it uses a clustering algorithm and uses the centroids as the parameters of the membership functions. Fudge instead builds the fuzzy datatypes as a consensual definition of the individual definitions given by several domain experts.

So far, we assumed that the fuzzy datatypes were learnt offline. However, it is entirely possible to use online learning to incrementally build the fuzzy membership functions defining the fuzzy datatypes. For example, in Datil, it suffices to use incremental clustering algorithms [48] and then update the definitions of the fuzzy datatypes in the fuzzy ontology.

Let us also mention that the relation between symbolic reasoning and deep learning has been recently studied [49], opening the door to future research on ontology reasoning not based on logical deduction.

Verification. Our reasoning algorithm is correct, i.e., all retrieved instances satisfy the query. However, the solution is only complete if the fuzzy ontologies satisfies some restrictions. Regarding the quality of the solutions, they depend on the quality of the linguistic labels. In this regard, it is worth mentioning that Datil's algorithm to learn fuzzy datatypes has been evaluated in the field of beer recommendation, showing that it provides similar results to a human expert [50].

7. Conclusions and future work

This paper proposed a novel algorithm to perform minimalistic fuzzy ontology reasoning based on the reuse of classical reasoners. The objective is to be able to reason with larger Building Information Modeling files, closer to those used in real-world applications. We developed a desktop application (in Java) and evaluated our proposal.

We considered a real BIM model as a case of study. The model was converted from IFC to OWL (RDF syntax) using an existing tool. We showed that such a big model could not be supported by two classical reasoners (Hermit and TrOWL). Hence, the final ontology was fragmented; for operativeness, we restricted the tests and evaluations to just the third floor of the dataset. Also, with this sub module we built a fuzzy ontology updated with new property assertions, concept assertions and fuzzy datatypes.

We evaluated the performance of our proposal by measuring the times of retrieval of the data property values of each individual. We conclude that TrOWL give us satisfactory results. We also found that the query times can be reduced when using additional data structures (an extra hash table). Another finding is that our tool requires a considerable initial time to classify the ontology, but following queries require less time.

A future research line is to improve the size of the fragments that can be supported, as the whole ontology is not currently supported by the classical reasoners and could not be evaluated. For example, a possible strategy would be using a preprocessing step to filter the ontology (or fragments) and reduce the sizes with specific classes.

Another future work could be improving the implementation to automatically split the ontology into subontologies, or using more sophisticated parsers to translate the BIM model into OWL. Finally, it would be interesting to test a set of use cases with a high-level digital representation of a real building.

Acknowledgments

This work was funded by Imperial College's Centre for Systems Engineering and Innovation under its 2019 call for small projects. I. Huitzil was partially supported by a grant from Universidad de Zaragoza – Santander Universidades (Ayudas de Movilidad para Latinoamericanos – Estudios de Doctorado). I. Huitzil and F. Bobillo were partially supported by the project TIN2016-78011-C4-3-R (AEI/ FEDER, UE). J. Gómez-Romero was partially supported by the Spanish Ministry of Science, Innovation and Universities (TIN2017-91223- EXP). M. Molina-Solana was supported by European Union's H2020 MSCA-IF (ga. No. 743623) and Athenea3i (ga. No. 754446) programmes.

References

[1] A. Sawhney, M. Riley, J. Irizarry, Construction 4.0: An Innovation Platform for the Built Environment, Routledge, 2020. doi:10.1201/9780429398100.

[2] A. Borrmann, M. König, C. Koch, J. Beetz, Building information modeling: Why? what? how?, in: Building Information Modeling: Technology Foundations and Industry Practice, Springer International Publishing, 2018, pp. 1–24. doi:10.1007/978-3-319-92862-3_1.

[3] MagiCAD, BIM adoption in Europe: Current state, challenges and a vision of tomorrow, Tech. rep., visited on April 2020 (2020).

[4] M. Poljansek, Building information modelling (BIM) standardization, Tech. Rep. JRC109656, European Joint Research Centre (JRC) (2017). doi:10.2760/36471.

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

[6] S. Staab, R. Studer (Eds.), Handbook on Ontologies, International Handbooks on Information Systems, Springer, 2004.

[7] F. Baader, D. Calvanese, D. L. McGuiness, D. Nardi, P. F. Patel-Schneider, The Description Logic handbook, 2nd Edition, Cambridge University Press, 2010.

[8] B. Cuenca-Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, U. Sattler, OWL 2: The next step for OWL, Journal of Web Semantics 6 (4) (2008) 309–322. doi:10.1016/j.websem.2008.05.001.

[9] P. J. Hayes, P. F. Patel-Schneider, RDF 1.1 Semantics (2014).
URL http://www.w3.org/TR/rdf11-mt

[10] L. A. Zadeh, Fuzzy sets, Information and Control 8 (1965) 338–353. doi:10.1016/S0019-9958(65)90241-X.

[11] T. Lukasiewicz, U. Straccia, Managing uncertainty and vagueness in Description Logics for the Semantic Web, Journal of Web Semantics 6 (4) (2008) 291–308. doi:10.1016/j.websem.2008.04.001.

[12] F. Bobillo, U. Straccia, Fuzzy ontology representation using OWL 2, International Journal of Approximate Reasoning 52 (7) (2011) 1073–1094. doi:10.1016/j.ijar.2011.05.003.

[13] F. Bobillo, U. Straccia, The fuzzy ontology reasoner fuzzyDL, Knowledge-Based Systems 95 (2016) 12–34. doi:10.1016/j.knosys.2015.11.017.

[14] F. Bobillo, M. Delgado, J. Gómez-Romero, DeLorean: a reasoner for fuzzy OWL 2, Expert Systems with Applications 39 (2012) 258–272. doi:10.1016/j.eswa.2011.07.016.

[15] J. Gómez-Romero, F. Bobillo, M. Ros, M. Molina-Solana, M. D. Ruiz, M. J. Martín-Bautista, A fuzzy extension of the semantic building information model, Automation in Construction 57 (2015) 202–212. doi:10.1016/j.autcon.2015.04.007.

[16] European Construction Sector Observatory, Building Information Modelling in the EU construction sector, Tech. rep., visited on May 2019 (2019).

[17] T. Mendes de Farias, A. Roxin, C. Nicolle, A rule-based methodology to extract building model views, Automation in Construction (2018). doi:10.1016/j.autcon.2018.03.035.

[18] P. Pauwels, W. Terkaj, EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology, Automation in Construction 63 (2016) 100–133. doi:10.1016/j.autcon.2015.12.003.

[19] S. Harris, A. Seaborne, SPARQL 1.1 Query Language, http://www.w3.org/TR/sparql11-query (2013).

[20] J. Werbrouck, M. Senthilvel, J. Beetz, P. Bourreau, L. Van Berlo, Semantic query languages for knowledge-based web services in a construction context, in: P. Geyer, K. Allacker, M. Schevenels, F. De Troyer, P. Pauwels (Eds.), Proceedings of the 26th International Workshop on Intelligent Computing in Engineering (EG-ICE), Vol. 2394 of CEUR Workshop Proceedings, CEUR-WS.org, 2019, p. 13.
URL http://ceur-ws.org/Vol-2394/paper03.pdf

[21] R. Taelman, M. Vander Sande, R. Verborgh, GraphQL-LD: Linked Data querying with GraphQL, in: Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks, Vol. 2180 of CEUR Workshop Proceedings, CEUR-WS.org, 2018.
URL http://ceur-ws.org/Vol-2180/paper-65.pdf

[22] Semantic Integration Ltd., Hypergraphql, https://www.hypergraphql.org, accessed: 2020-06-16 (2020).

[23] M. Fahad, N. Bus, B. Fies, Semantic bim reasoner for the verification of ifc models, in: eWork and eBusiness in Architecture, Engineering and Construction, CRC Press, 2018, pp. 361–368. doi:10.1201/9780429506215-45.

[24] R. K. Soman, M. Molina-Solana, J. Whyte, Linked-Data based Constraint-Checking (LDCC) to support look-ahead planning in construction, Automation in Construction 120 (2020) 103369. doi:10.1016/j.autcon.2020.103369.

[25] J. Abualdenien, A. Borrmann, Vagueness visualization in building models across different design stages, Advanced Engineering Informatics 45 (2020). doi:10.1016/j.aei.2020.101107.

[26] U. Straccia, Foundations of Fuzzy Logic and Semantic Web Languages, CRC Studies in Informatics Series, Chapman & Hall, 2013.

[27] F. Bobillo, M. Delgado, J. Gómez-Romero, U. Straccia, Joining Gödel and Zadeh fuzzy logics in fuzzy Description Logics, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 20 (04) (2012) 475–508. doi:10.1142/S0218488512500249.

[28] W3C, OWL 2 web ontology language profiles, http://www.w3.org/TR/2009/REC-owl2-profiles-20091027 (2009).

[29] F. Bobillo, The role of crisp elements in fuzzy ontologies: The case of fuzzy OWL 2 EL, IEEE Transactions on Fuzzy Systems 24 (2016) 1193–1209. doi:10.1109/TFUZZ.2015.2505329.

[30] S. Borgwardt, M. Cerami, R. Peñaloza, The complexity of fuzzy EL under the łukasiewicz t-norm, International Journal of Approximate Reasoning 91 (2017) 179–201. doi:10.1016/j.ijar.2017.09.005.

[31] F. Bobillo, U. Straccia, Reasoning within fuzzy OWL 2 EL revisited, Fuzzy Sets and Systems 351 (2018) 1–40. doi:10.1016/j.fss.2018.03.011.

[32] T. P. Mailis, G. Stoilos, N. Simou, G. B. Stamou, S. D. Kollias, Tractable reasoning with vague knowledge using fuzzy $\mathcal{EL}^{++}$, Journal of Intelligent Information Systems 39 (2) (2012) 399–440. doi:10.1007/s10844-012-0195-6.

[33] J. Z. Pan, G. Stamou, G. Stoilos, E. Thomas, S. Taylor, Scalable querying service over fuzzy ontologies, in: Proceedings of the 17th International Conference on World Wide Web (WWW 2008), 2008, pp. 575–584. doi:10.1145/1367497.1367575.

[34] G. Stoilos, T. Venetis, G. Stamou, A fuzzy extension to the OWL 2 RL ontology language, The Computer Journal 58 (11) (2015) 2956–2971. doi:10.1093/comjnl/bxv028.

[35] V. Haarslev, H.-I. Pai, N. Shiri, Optimizing tableau reasoning in $\mathcal{ALC}$ extended with uncertainty, in: Proceedings of the 20th International Workshop on Description Logics (DL 2007), Vol. 250 of CEUR Workshop Proceedings, CEUR-WS.org, 2007, pp. 307–314.
URL http://ceur-ws.org/Vol-250/paper-29.pdf

[36] G. S. N. Simou, T. Mailis, G. Stamou, Optimization techniques for fuzzy description logics, in: Proceedings of the 23rd International Workshop on Description Logics (DL 2010), Vol. 573 of CEUR Workshop Proceedings, CEUR-WS.org, 2010.
URL http://ceur-ws.org/Vol-573/paper-25.pdf

[37] F. Bobillo, U. Straccia, Optimising fuzzy description logic reasoners with general concept inclusions absorption, Fuzzy Sets and Systems 292 (2016) 98–129. doi:10.1016/j.fss.2014.10.029.

[38] I. Huitzil, J. Bernad, F. Bobillo, Algorithms for instance retrieval and realization in fuzzy ontologies, Mathematics 8 (2) (2020) 154:1–16. doi:10.3390/math8020154.

[39] G. J. Klir, B. Yuan, Fuzzy sets and fuzzy logic: theory and applications, Prentice-Hall, Inc., 1995.

[40] R. R. Yager, On ordered weighted averaging aggregation operators in multicriteria decision making, IEEE Transactions on Systems, Man and Cybernetics 18 (1) (1988) 183–190. doi:10.1109/21.87068.

[41] R. R. Yager, Connectives and quantifiers in fuzzy sets, Fuzzy Sets and Systems 40 (1) (1991) 39–75. doi:10.1016/0165-0114(91)90046-S.

[42] P. P. S. Törma, N. Hoang, IFC-to-RDF conversion tool, visited on March 2019.
URL http://www.rymreport.com/pre/result/opening-bim-to-the-web-ifc-to-rdf-conversion-software

[43] N. M. Hoang, S. Törmä, Implementation and experiments with an ifc-to-linked data converter, in: Proceedings of the 32nd CIB W78 Conference, 2015, pp. 285–294.

[44] M. Bonduel, J. Oraskari, P. Pauwels, M. Vergauwen, R. Klein, The IFC to linked building data converter: current status, in: Proceedings of the 6th Linked Data in Architecture and Construction Workshop, Vol. 2159 of CEUR Workshop Proceedings, CEUR-WS.org, 2018, pp. 34–43.
URL http://ceur-ws.org/Vol-2159/04paper.pdf

[45] S. Daum, A. Borrmann, Processing of topological BIM queries using boundary representation based methods, Advanced Engineering Informatics 28 (4) (2014) 272–286. doi:10.1016/j.aei.2014.06.001.

[46] I. Huitzil, U. Straccia, N. Díaz-Rodríguez, F. Bobillo, Datil: Learning fuzzy ontology datatypes, in: Proceedings of the 17th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2018), Part II, Vol. 854 of Communications in Computer and Information Science, Springer, 2018, pp. 100–112. doi:10.1007/978-3-319-91476-3_9.

[47] I. Huitzil, F. Bobillo, J. Gómez-Romero, U. Straccia, Fudge: Fuzzy ontology building with consensuated fuzzy datatypes, Fuzzy Sets and Systems 401 (2020) 91–112. doi:10.1016/j.fss.2020.04.001.

[48] A. M. Bagirov, N. Karmitsa, S. Taheri, Incremental clustering algorithms, in: Partitional Clustering via Nonsmooth Optimization. Unsupervised and Semi-Supervised Learning, Springer, 2020, pp. 185–200. doi:10.1007/978-3-030-37826-4_7.

[49] P. Hohenecker, T. Lukasiewicz, Ontology reasoning with deep neural networks, Journal of Artificial Intelligence Research 68 (2020) 503–540. doi:10.1613/jair.1.11661.

[50] I. Huitzil, F. Alegre, F. Bobillo, GimmeHop: A recommender system for mobile devices using ontology reasoners and fuzzy logic, Fuzzy Sets and Systems 401 (2020) 55–77. doi:10.1016/j.fss.2019.12.001.

Appendix A. Snapshots of the implemented tool

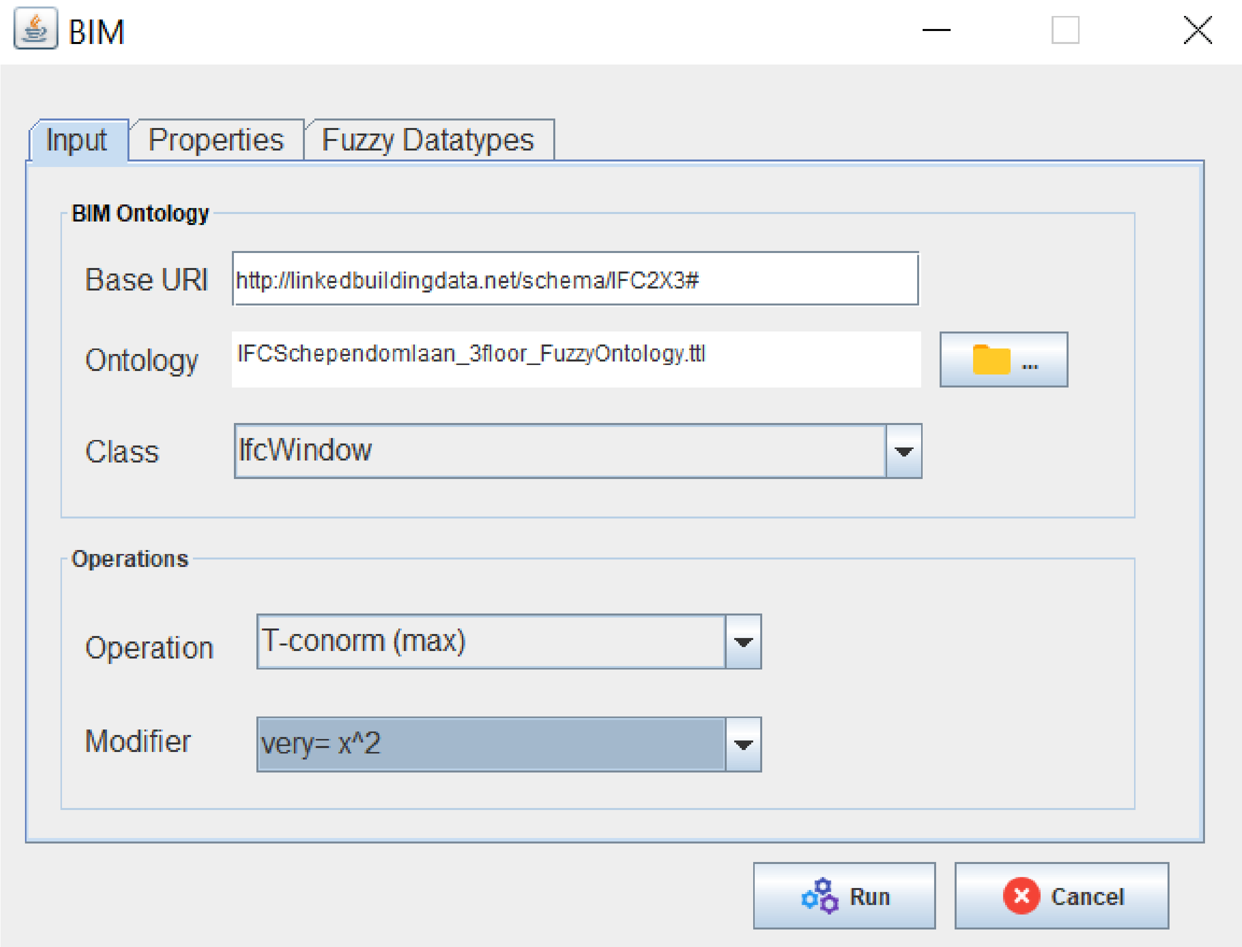

Figure A.8: User interface: loading fuzzy BIM ontology, selection of a class and fuzzy operators

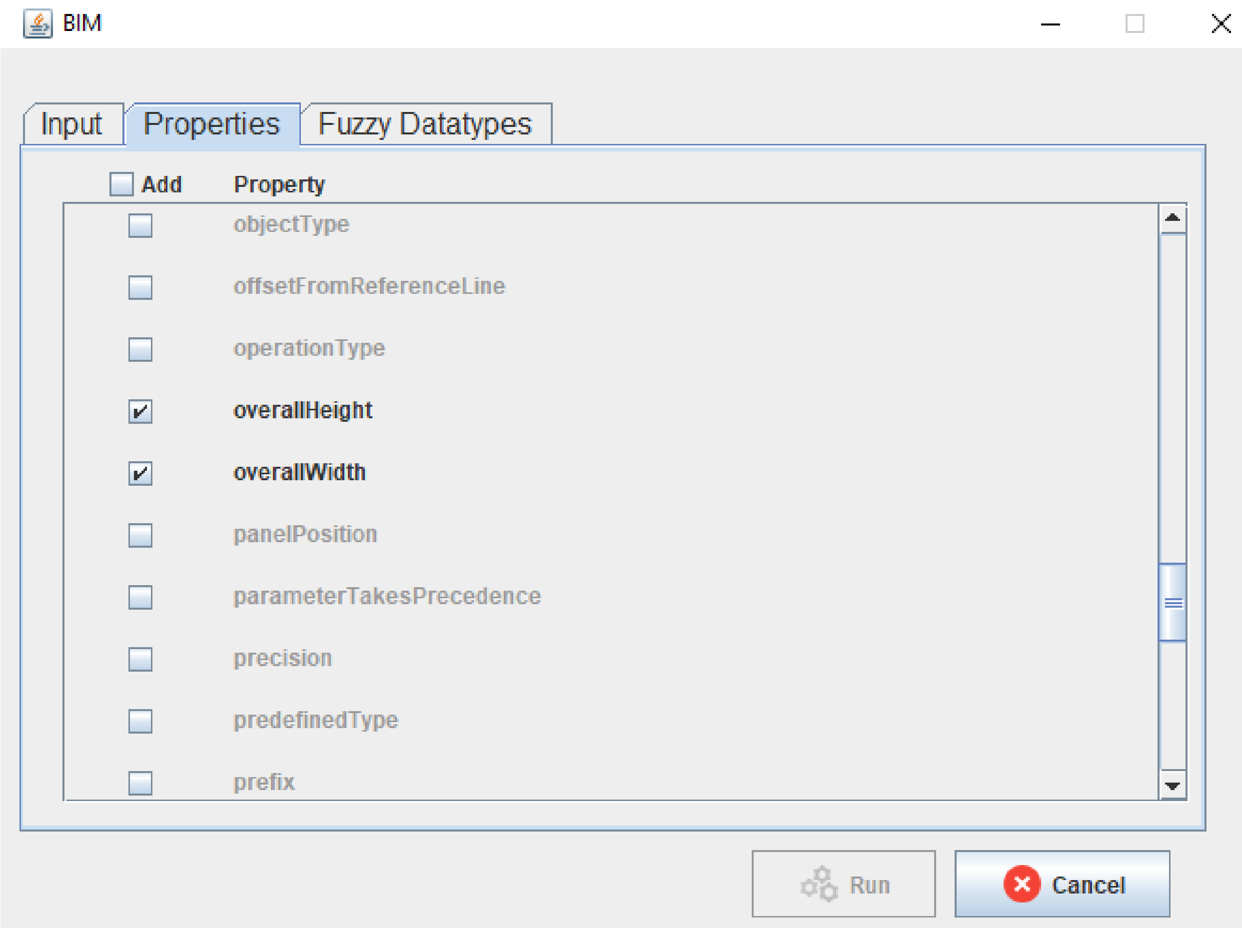

Figure A.9: User interface: selection of data properties

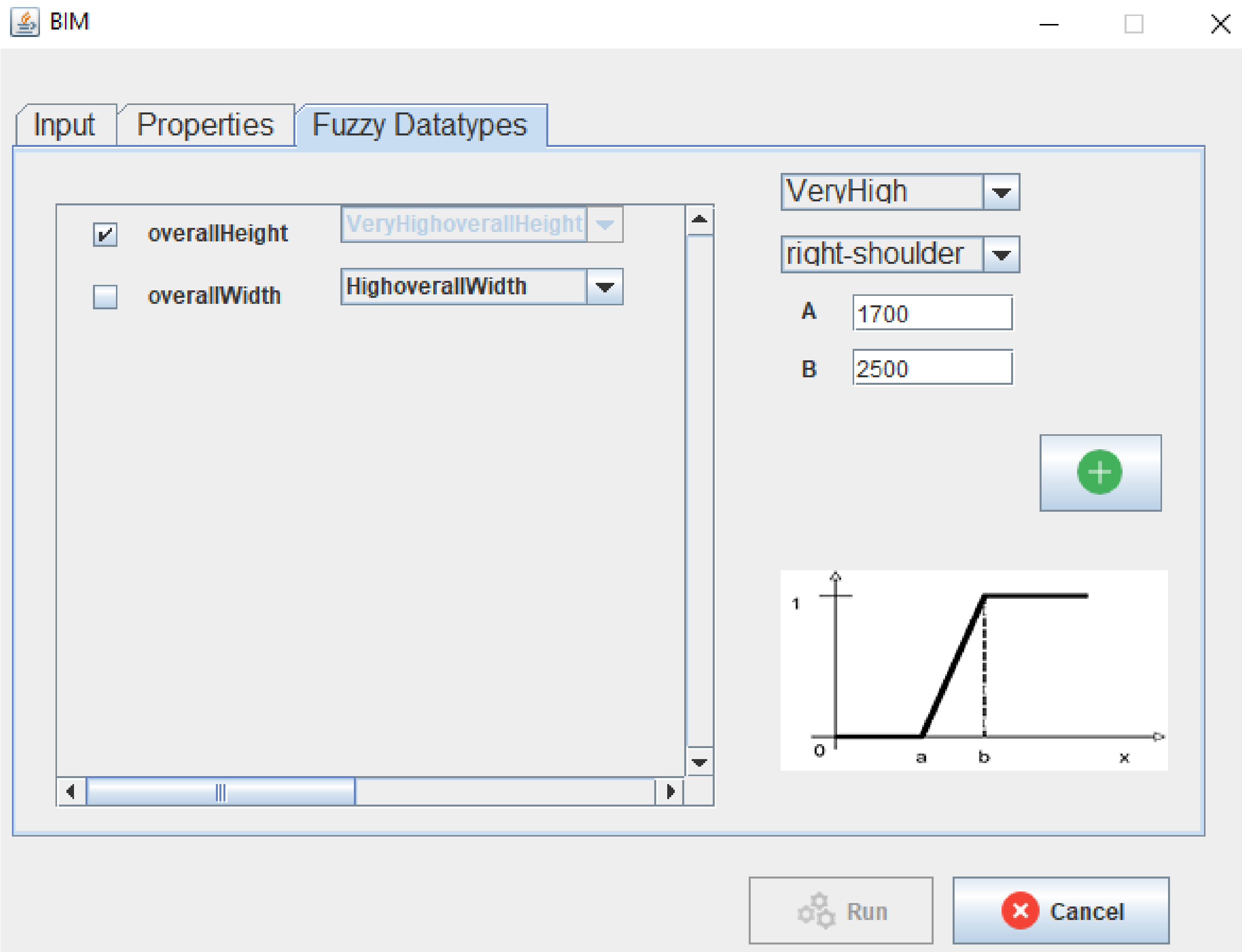

Figure A.10: User interface: selection or creation of fuzzy datatypes

Figure A.11: User interface: final result

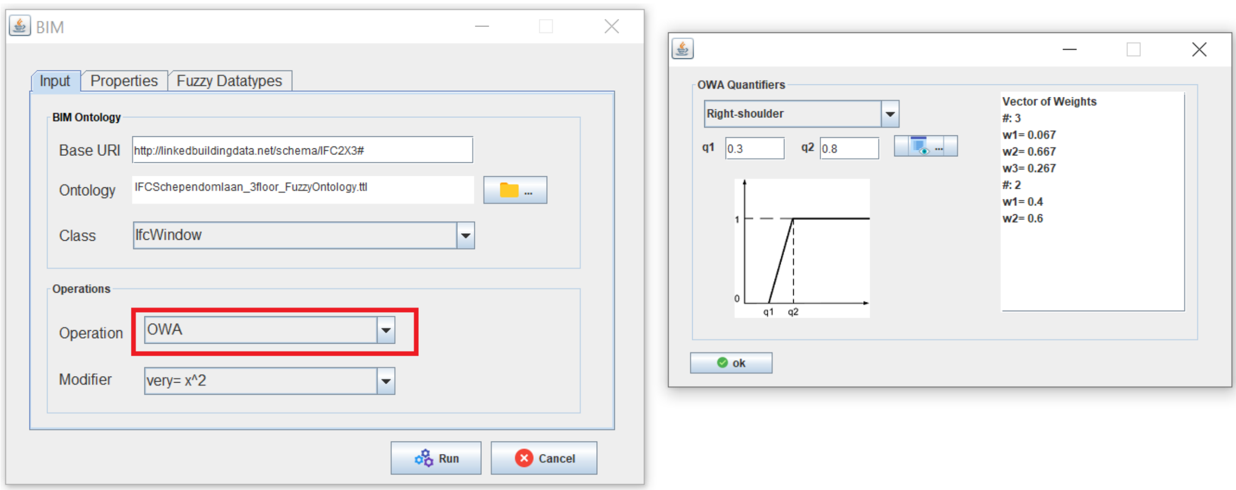

Figure A.12: User interface: use of a quantifier to get the parameters of the OWA aggregation operator