P. Pauwels, W. Terkaj (2016) EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology.
Automation in Construction, Volume 63, Pages 100-133, ISSN 0926-5805.
https://doi.org/10.1016/j.autcon.2015.12.003
(https://www.sciencedirect.com/science/article/pii/S0926580515002435)

EXPRESS to OWL for construction industry: towards a recommendable and usable ifcOWL ontology

Pieter Pauwels[a,*], Walter Terkaj[b]

[a]*Department of Architecture and Urban Planning, Ghent University, J. Plateaustraat 22, B-9000 Ghent, Belgium*
[b]*Istituto di Tecnologie Industriali e Automazione (ITIA), Consiglio Nazionale delle Ricerche (CNR), Via Bassini, 15 - 20133 Milano, Italy*

Abstract

An increasing number of information management and information exchange applications in construction industry is relying on semantic web technologies or tools from the Linked Open Data (LOD) domain to support data interoperability, flexible data exchange, distributed data management and the development of reusable tools. These goals tend to be overlapped with the purposes of the Industry Foundation Classes (IFC), which is a standard for the construction industry defined through an EXPRESS schema. A connecting point between semantic web technologies and the IFC standard would be represented by an agreed ifcOWL ontology that allows to (1) keep on using the well-established IFC standard for representing construction data, (2) exploit the enablers of semantic web technologies in terms of data distribution, extensibility of the data model, querying, and reasoning, (3) re-use general purpose software implementations for data storage, consistency checking and knowledge inference. Therefore, in this paper we will look into existing efforts in obtaining an ifcOWL ontology from the EXPRESS schemas of IFC and analyse which features would be required in a usable and recommendable ifcOWL ontology. In making this analysis, we present our implementations of an EXPRES-to-OWL converter and the key features of the resulting ifcOWL ontology.

Keywords: IFC, OWL, building, construction, information technology, RDF, semantic web

1. Introduction

Building information modelling (BIM) can be named as one of the most notable efforts in recent years regarding information management in construction industry [1]. BIM environments allow to semantically describe any kind of information about the building in one 3D model, so that it can be better represented and more easily exchanged than in the case of traditional computer-aided design (CAD) tools. The Industry Foundation Classes (IFC) standard [2], developed by buildingSMART [3], aims at supporting these activities by providing a central *"conceptual data schema and an exchange file format for BIM data"* [2, scope]. In other words, using the IFC data model and file format, BIM data can be exchanged between software applications, which can in turn provide extra functionality (e.g. 4D planning, 5D cost calculation, CFD simulation, structural analysis, and so forth).

1.1. IFC and EXPRESS

Each IFC data model is represented as a schema in the EXPRESS data specification language defined in ISO 10303-11:1994 that *"consists of language elements which allow an unambiguous data definition and specification of constraints on the data defined and by which aspects of product data can be specified"* [4]. The EXPRESS language consists of the terms (e.g. types, entities, properties) and rules that must be used to build a specific EXPRESS schema (`.exp`). In the case of IFC, there are several available IFC EXPRESS schemas, including the most well-known IFC2X3.exp, IFC2X3_TC1.exp, IFC4RC4.exp, and IFC4_ADD1.exp (see [5] for an overview of the specifications). These schemas should be considered as chronologically ordered versions of one IFC schema, meaning that there is always *one most recent schema* available.

*Corresponding author: Tel. 0032 9 264 3880 - Mob. 0032 486 791488
Email addresses: `pipauwel.pauwels@ugent.be` (Pieter Pauwels), `walter.terkaj@itia.cnr.it` (Walter Terkaj)

Each of the EXPRESS schemas of IFC [5] enables a description of construction-related information where the represented objects have a well-defined and interrelated meaning and purpose. As a single window element can internally be described in very diverse ways by each BIM environment, the export/import possibilities to/from IFC should guarantee that each BIM environment is able to map its own descriptions to a generally understandable IFC format, thereby considerably improving (not solving) the interoperability of information.

We want to point out that IFC is not an isolated effort or standard. It should be used in combination with other standards, such as Model View Definitions (MVDs) and Information Delivery Manuals (IDMs), if an improved information exchange is targeted. Also Property Set Definition (PSD) releases [6] should be considered as important additions to the IFC schema. A PSD release provides a schema that defines how custom properties and property sets can be defined outside of the IFC specification. This schema is provided as an XML Schema Definition (XSD). PSD's, as well as MVDs and IDMs are considered out of the scope of this paper. Starting from a centrally agreed ifcOWL ontology, however, it is feasible to support PSD, MVD and IDM definitions in a semantic web representation.

1.2. RDF and OWL

1.2.1. The basics

The semantic web initiative [7] shares some of the goals of formal specification of information that IFC is targeting for the construction industry domain. The semantic web was conceived and presented as the successor of the existing World Wide Web (WWW) by describing all information in a language that could be understood by computer applications, i.e. machine-readable. Because the WWW contains information about almost any possible concept in the world, the language describing this information cannot follow one domain-specific schema. Instead, a flexible and generic language is needed that allows to describe and easily combine information from very different knowledge domains. Therefore, the semantic web was conceived as a semantic network [8] in which diverse semantic domains can be represented and combined using directed labelled graphs. Each node in such a graph thus represents a concept or object in the world and each labelled arc represents the logical relation between two of these concepts or objects. A graph can be constructed using the Resource Description Framework (RDF) [9], which has a basis in description logic (DL) [10]. The graph is thus formed by a set of logic-based declarative sentences and, in total, it represents a specific semantic domain, as it is understood and explained by Hennessy [11]. By describing information in a single directed labelled graph, a uniform representation of information is achieved, making information reusable by both humans and computer applications.

An RDF graph is constructed by applying a logical `AND` operator to a range of logical statements containing concepts or objects in the world and their relations. These statements are often referred to as *RDF triples*, consisting of a subject, a predicate and an object (Figure 1) and thus implying directionality in the RDF graph. In addition, each concept has a Unique Resource Identifier (URI), thereby making the RDF graph explicitly labelled. Every concept described in an RDF graph, whether this be an object, subject or predicate, is uniquely defined through this URI. When two identical URIs are found, their semantics are considered identical as well.

Figure 1: The triple form of an RDF statement: subject - predicate - object.

The resulting RDF graph can be represented using various syntaxes. Syntaxes used for RDF graphs are RDF/XML (`.RDF`), N-Triples (`.N_T`), Turtle (`.TTL` - [12]), and Notation-3 (`.N3`) [13]. RDF graphs can be given an improved semantic structure using RDF vocabularies or ontologies. The most basic elements to describe such ontologies are available in the RDF Schema (RDFS) vocabulary [14]. RDFS, for instance, enables the specification of classes, subclasses, comments, and data types. An RDFS interpreter is able to infer extra RDF statements that are implicitly available via the RDFS constructs. More expressive elements to describe ontologies are available within the Web Ontology Language (OWL) [15]. In short, OWL further enhances the RDFS concepts to allow making more complex RDF statements, such as cardinality restrictions, type restrictions, complex class expressions. The RDF graphs constructed with OWL concepts are called OWL ontologies.

1.2.2. OWL semantics and OWL profiles

As the expressiveness of OWL is a key element in this article, we will briefly outline what options are available in terms of building an OWL ontology. The semantic expressiveness of the OWL language is specified in

multiple W3C hosted specification documents. The first W3C Recommendation for OWL dates from 2004 [16]. This version is now superseded by the OWL2 language specification issued in 2012 [15]. Hence, any reference to OWL in this paper refers to the OWL2 specification. All relevant references to the exact semantics of OWL2 can be found in [17] (section about Semantics, including *OWL2 Direct Semantics* and *OWL2 RDF-Based Semantics*). Figure 2 provides an overview picture that we will use to explain the basics of OWL profiles.

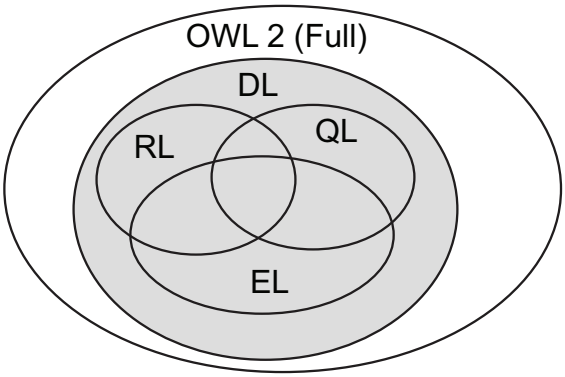

Figure 2: The OWL2 profiles EL, QL and RL each provide a certain kind of expressiveness, which determines to what level of detail information can be semantically represented and which impacts performance as a result (more information, less performing). Original figure in [18].

As pointed out in [17], *"the Direct Semantics assigns meaning directly to ontology structures, resulting in a semantics compatible with the model theoretic semantics of the $\mathcal{SROIQ}$ description logic – a fragment of first order logic with useful computational properties"*. This leads to a semantic expressiveness for OWL2 that is properly grounded in a particular description logic, namely $\mathcal{SROIQ}$. This semantic expressiveness is graphically displayed as the outer ellipse in Fig. 2 (OWL2 Full). However, *"some conditions must be placed on ontology structures in order to ensure that they can be translated into a $\mathcal{SROIQ}$ knowledge base"* [17]. For instance, transitive properties cannot be used in number restrictions. Whenever an OWL2 ontology satisfies these conditions, the expressiveness of the ontology is in the second ellipse in Fig. 2, namely OWL2 DL. An OWL ontology should remain within this boundary if it is to be used by $\mathcal{SROIQ}$-based tools, which are the tools typically supplied by the semantic web community.

As in the case of OWL, also OWL2 has a number of so-called *profiles*, namely OWL2 EL, OWL2 QL and OWL2 RL [19]. Figure 2 displays the relationships between these three key profiles. As outlined in Motik et al. [19], an OWL2 profile *"is a trimmed down version of OWL2 that trades some expressive power for the efficiency of reasoning"*. In short, in each of the given OWL2 profiles, a number of statements that can be used in OWL2 DL is not allowed. By not allowing these statements, and thus sacrificing some expressiveness, important improvements can be made in terms of performance. Namely, inference engines do not need to check a number of restrictions as they are not allowed (and thus not considered) in particular profiles. More information about the expressiveness of each of the profiles can be found in [19]. The following summarised descriptions can be used as a reference:

- OWL2 EL
  This profile is to be used in applications that use ontologies with many properties and/or classes. Basic reasoning can be performed in time that is polynomial with respect to the size of the ontology (`PTime`). Important constructs that are not allowed in OWL2 EL are, among others, universal restrictions (`allValuesFrom`), cardinality restrictions (`maxCardinality`, `minCardinality`, `exactCardinality`), disjunction (`unionOf`), enumerations (`oneOf`), and several particular property-related expressions (`inverseOf`, `disjoint`, `functionalProperty`, `symmetricProperty`).

- OWL2 QL
  This profile is recommended for applications with a large volume of instance data, and where query performance is most important. If done properly, query answering can be performed in LOGSPACE with respect to the size of the data. The expressive power of this profile is quite limited as it excludes constructs such as existential and universal restrictions (`someValuesFrom`, `allValuesFrom`), cardinality restrictions (`maxCardinality`, `minCardinality`, `exactCardinality`), disjunction (`unionOf`), property inclusions (`subPropertyOf`) and enumerations (`oneOf`). In comparison with OWL EL, some property-related expressions are allowed (`inverseOf`, `disjoint`, `symmetricProperty`).

- OWL2 RL
  This profile is meant to be used for applications that require scalable reasoning without sacrificing too much expressive power. Whenever reasoning is involved, it is a good choice to adopt ontologies in this profile. *"The ontology consistency, class*

expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering problems can be solved in time that is polynomial with respect to the size of the ontology." [19]. The expressive power of OWL2 RL is quite close to OWL2 DL. A few syntactic restrictions (see [19]) need to be taken into account in order for the ontology to be in the OWL2 RL profile. Moreover, all axioms of OWL2 are supported in OWL2 RL, except for disjoint unions of classes and reflexive object property axioms.

1.2.3. Closed World Assumption (CWA) versus Open World Assumption (OWA)

Two distinct approaches to knowledge representation are relevant when dealing with IFC and OWL: Closed World Assumption (CWA) and Open World Assumption (OWA) [20]. According to CWA, any statement that is not known to be `true`, must be considered as `false`. When applied to an IFC model or a BIM model, one can conclude that whenever something is not specified, it is most definitely *not there*. On the other hand, according to OWA a statement that is not known to be `true`, is not necessarily `false`, nor `true`, but `unknown`. In other words, it might be `true` or `false` in the future, when more information is supplied, but no conclusion can be drawn until then.

Many traditional software applications, including BIM tools, database systems and the IFC data model, adopt a Closed World Assumption (CWA). Semantic web technologies, however, generally rely on an Open World Assumption (OWA) because the technologies are supposed to be used on the Web, which is a system with incomplete information. One cannot conclude that something is not `true` simply because no one specified it on the web. Hence, an OWA needs to be adopted. At least, that is the original reason behind this decision. The difference between CWA and OWA plays a key role when an ontology is used to represent an IFC model or a BIM model, because if something is not specified, then one cannot conclude much, except that it might still be `true` or `false`. A whole different kind of information usage and inference becomes available.

Mapping information representations in CWA to information representations in OWA is not that hard; the main difference lies in the usage of the information that is presented in both. Furthermore, it is even possible to run a CWA-based validation of an OWL ontology (see [21, 22]). However, the OWA of semantic web technologies is still something different from the traditional CWA features in current software applications. In many cases, both types of assumptions have their value (e.g. [23]). If adopted properly, the usage of semantic web technologies is a fruitful *addition to* (and not replacement of) existing technologies, such as BIM software environments and the IFC specification in EXPRESS.

1.3. Current status of IFC and RDF

1.3.1. The parallels between IFC and RDF

There is a considerable parallel between the buildingSMART effort towards the specification and standardisation of IFC for construction industry, and the W3C effort towards the specification and standardisation of RDF for web data. The purpose of the EXPRESS language is similar to the purpose of the OWL language, and the semantic structure of an IFC file is to some extent comparable to the semantic structure of an RDF graph. However, Barbau et al. [24] emphasised the lack of formal semantics in EXPRESS, arguing that a logic-based language, such as OWL, brings certain modelling advantages in knowledge representation and semantic data sharing. Indeed, by adopting any of the given OWL profiles for specifying building information, one can rely on corresponding model theoretic semantics to interpret the information (see Sect. 1.2.2). A number of generally available tools beyond construction industry can then be used, including generic formally grounded query engines, reasoning engines, and so forth. Such formally grounded generic tools are not generally available for EXPRESS. Additionally, Beetz et al. [25] stressed the limits of EXPRESS with respect to the reuse of existing ontologies and interoperability with semantic web tools. With this statement, Beetz et al. [25] most likely refer to the OWA basis of semantic web technologies (see Sect. 1.2.3), which makes it possible to add new information without violating any of the conclusions that were inferred previously (cf. monotonic reasoning).

The main differences between the buildingSMART effort and the W3C effort thus lie in (1) the domain that is to be represented, and (2) the language/technology that is used to represent that domain. Currently, both IFC and RDF are mainly used within their respective domains. IFC is commonly used for exchange of construction-related information, notwithstanding its limitations, whereas RDF is used to represent web data. The combination of IFC and RDF led to, among other applications, an online Linked Open Data (LOD) cloud [26, 27, 28, 29] that collects a considerable number of open RDF datasets that are linked together in one open data cloud (1014 datasets in 2014). As there are so many parallels between the approaches behind IFC

and RDF, it should be possible to attempt publishing IFC data as part of this LOD cloud, or at least as RDF graphs.

1.3.2. Why porting IFC data into the RDF data model?

The formalisation of IFC data as an RDF graph requires to focus first on the conversion of the IFC schema, defined as an EXPRESS schema, into an OWL ontology. As soon as such an ontology is available, it is relatively straightforward to build RDF graphs that are compliant with that OWL ontology. Various research initiatives addressed the problem of converting an EXPRESS schema to an OWL ontology and some of them focused in particular on the specific IFC case.

Most of the initiatives to formalise IFC in an ontology language have been motivated with the aim of providing a semantically rich and platform independent framework that can support the integration of software tools and exchange of data in a knowledge-based system that is both human readable and usable by machines. Some early example applications were provisionally suggested in Pauwels et al. [30], Abdul-Ghafour et al. [31], Yurchyshyna et al. [32], Yurchyshyna and Zarli [33] for the construction domain and in Kadar et al. [34], Terkaj et al. [35] for the manufacturing domain. The potentiality and the presence of technologies to develop a semantic linking of building information models were presented by Törmä [36, 37]. In addition, Schevers and Drogemuller [38], as well as Zhang and Issa [39], argued that the conversion of IFC into OWL, besides enabling the exploitation of semantic web technologies for building information models, even facilitates the linkage between different IFC models and databases. Many more example applications have emerged by now, covering a myriad of use case scenarios in architectural design, construction industry, smart cities applications, built heritage applications, the factory and manufacturing domain, building regulation management, and facility management. From these use cases, it is clear that the focus of development does not really lie in the replacement of existing technologies, but rather in the combination of building information with relevant information in other domains. This can be considered as a useful OWA addition to the set of currently available tools that provide crucial CWA functionality.

Most of the semantic tools previously mentioned have relatively limited functionalities and applicability. In many cases, the reason for this limitation is the absence of a real standard ifcOWL ontology, even though several proposals were presented. Therefore, the developed applications are all based on slightly different ontologies, making them work as isolated examples. As a result, it is not really possible to build applications with a realistic scope. A recommendable and usable ifcOWL ontology would allow to test applications of semantic web technologies in more realistic settings and to let them mature into usable, trustworthy and helpful tools.

1.3.3. Previous proposals in the conversion of the EXPRESS schema of IFC into an OWL ontology

A number of EXPRESS to OWL conversion procedures has been proposed so far. Schevers and Drogemuller [38] proposed a first unidirectional conversion map from EXPRESS to OWL, taking IFC as a reference example and highlighting some of the key issues to be addressed. Agostinho et al. [40] proposed a mapping between EXPRESS and OWL within a broader Model Morphism initiative. Beetz et al. [25] proposed a semi-automatic method for converting EXPRESS schemas to OWL ontologies in order to enhance the applicability and re-usability of the IFC standard. Barbau and colleagues specified a set of rules for the automated conversion from EXPRESS to OWL within the OntoSTEP initiative [41, 24] and implemented the system as a plug-in for the Protégé software tool [42]. Another conversion tool was presented by Pauwels and Van Deursen [43]. Terkaj et al. [44] proposed an OWL version for a fragment of IFC aiming at facilitating the extension of IFC and the integration with other data models that are relevant in the scope of the industrial domain, in particular for the design and management of factories. Finally, Terkaj and Sojic [23] presented how some of the EXPRESS rules included in an IFC schema can be represented in OWL as well to enhance the previous ifcOWL ontologies.

Many of these proposals have been documented only in relatively short scientific articles that are not detailed enough. For example, Schevers and Drogemuller [38], Beetz et al. [25] and Pauwels and Van Deursen [43] presented the key ideas of their approach, without publishing the generated ontology that would provide the actual details of the conversion procedure. Therefore, anyone willing to adopt any of these proposals would be required to implement the conversion procedure, inevitably leading to slightly different versions (e.g. object property renaming undecided, presence/absence of inverse properties, different handling of `ENUM` and `SELECT` data types). In addition, these three earlier proposals were developed when the OWL2 specification by Hitzler et al. [15] was not yet published. Instead, there was the distinction between the OWL profiles OWL Full, OWL DL and OWL Lite, which are different from the OWL2 profiles that exist nowadays. Nevertheless, most of the presented proposals were in

OWL DL, which can be compared to OWL2 DL. Hence, the general outlines presented in these three early proposals are still of high value for this article and have informed many of the general decisions made here. As an example, class wrapping of EXPRESS simple data types has been proposed by Schevers and Drogemuller [38] and Beetz et al. [25] for good reasons, therefore this approach has been adopted here as well for the conversion of EXPRESS simple data types. Other than that, most of the details (cardinality restrictions, domain and range restrictions, class and property naming) had to be thoroughly reviewed.

Of the later proposals, the most relevant resources are the conversion procedure presented and documented within the OntoSTEP initiative [41, 24], and the conversion procedure presented by Hoang [45], Hoang and Törmä [46]. The OntoSTEP research initiative aims at providing a general purpose conversion mechanism for any EXPRESS schema to an OWL ontology, not only of the EXPRESS schema of IFC. In addition, the conversion procedure is well-documented and the resulting ontology file can be generated and checked. Of all the available resources, only the conversion procedure by Hoang [45], Hoang and Törmä [46] explicitly and appropriately takes into account the existence of the OWL2 profiles EL, QL and RL, thereby making a case for a conversion procedure that results in a layered ifcOWL ontology. Three layers are proposed: ifcOWL Simple, which only includes what can be specified in the intersection of OWL2 EL, QL and RL (see Fig. 2); ifcOWL Standard, which is an ontology in the OWL2 RL profile; and ifcOWL Extended, which is an ontology in OWL2 DL.

The proposal that we make in the following sections, only differs in the details compared to the proposals by Krima et al. [41], Barbau et al. [24] and by Hoang [45], Hoang and Törmä [46]. Many of the decisions made by Krima et al. [41], Barbau et al. [24] are motivated by the aim of developing a *general-purpose* converter for EXPRESS. In other words, the conversion procedure is not tailored to the case of construction industry or IFC, more specifically. As a result, quite verbose constructs are often used in order to take into account the many possible declarations in EXPRESS. A less verbose and thus better usable conversion can be used for several concepts employed in IFC.

The proposal by Hoang [45] and Hoang and Törmä [46] identified two main criteria for the conversion: (1) to be able to switch between different OWL2 profiles, and (2) to make it easy to generate RDF graphs as linked data, not necessarily tied to an overly restricted ontology. As a result, considerable compromises are made in the three proposed ifcOWL ontologies (Simple, Standard and Extended), in order to make them somewhat compatible. By choosing to not implement OWL class domain and range restrictions in the Simple and Standard versions of the ontology, mainly because of criterion 2, these ifcOWL versions tend to become underspecified. In other words, these ifcOWL versions consider only a fraction of the semantic richness of the original EXPRESS schema. The ifcOWL Extended ontology can be compared with what we propose in the following sections. However, by aiming to make this ontology compatible with ifcOWL Simple and ifcOWL Standard, it becomes hard to add all the restrictions that are originally defined in the EXPRESS schema.

1.3.4. Targeted criteria

As highlighted in the previous section, it is important to set appropriate criteria that underpin the proposed EXPRESS-to-OWL conversion procedure. Considering the reasons for targeting one standardised (or at least recommended) version of the ifcOWL ontology (see Sect.1.3.2), we decided to stick to the following three criteria, in order of importance:

1. The ifcOWL ontology must be in OWL2 DL.
2. The ifcOWL ontology should match the original EXPRESS schema as closely as possible.
3. The ifcOWL ontology primarily aims at supporting the conversion of IFC instance files into equivalent RDF files. Thus, herein it is of secondary importance that an instance RDF file can be modelled from scratch using the ifcOWL ontology and an ontology editor.

We decided to remain in OWL2 DL, and not in any of the three OWL2 profiles because an OWL2 DL ontology provides full expressiveness. Note that it is possible to develop an ifcOWL ontology in the EL, QL or RL profile from the OWL2 DL version, since these profiles are subsets of the OWL2 DL version (Fig. 2). The disadvantage behind this decision is that the resulting ifcOWL ontology will imply sacrifices in terms of performance, in particular reasoning performance. The advantage is that we have the complete freedom to build an ifcOWL ontology that is as close as possible to the original IFC EXPRESS schema (cf. our second criterion). If an ifcOWL ontology is used as a recommended version, then this ontology should be as close as possible to the original IFC standard. These two first criteria are thus very important here, which is not the case in the recent proposal by Hoang [45], Hoang and Törmä [46]. The proposal by Krima et al. [41], Barbau et al. [24]

takes our first two criteria in account as well, but additionally aims at supporting the conversion of EXPRESS schemas other than IFC (more general purpose).

The third criterion is considerably less important than the first two criteria. Essentially, we targeted first at supporting the conversion of IFC instance files into equivalent RDF files, and not at the creation of RDF graphs from scratch, using only the ifcOWL ontology. In most cases, IFC models will likely be built still in BIM software environments, as they are available and spread in the industry. The common case will thus remain to be a one-directional conversion process from BIM environment into an ifcOWL-compliant RDF graph. Considering this common case, most instance RDF graphs will already be checked for consistency in the native BIM tools. As a result, we can opt to not convert all procedural `RULE` and `FUNCTION` declarations in the native EXPRESS schema, which are in any case better handled in a native CWA BIM tool. If one would choose to set the third criterion the other way round (i.e. focusing on the creation of ifcOWL-compliant RDF graphs from scratch), then the ifcOWL ontology *needs* to include as many as possible `RULE` and `FUNCTION` declarations. This is to some extent possible, considering the proposals made in Terkaj and Sojic [23].

1.4. Paper outline

Relying on previous conversion proposals, we have looked in close details at the various options of converting EXPRESS schemas into OWL ontologies, so that we are able to propose and recommend the required ifcOWL ontology. The following outline is used to document this research.

- Sect. 2 will first outline the key characteristics of an EXPRESS schema, providing all the details that are relevant to take the conversion decisions.
- Section 3 presents the proposed EXPRESS to OWL conversion procedure. This section is similarly detailed to point out also the exceptions to the general conversion procedure.
- In Sect. 4, we briefly present our implementation(s) of the conversion procedure in executable software tools. The focus of this section (and of this article) is on the conversion of EXPRESS to OWL, i.e. only about the ifcOWL TBox. The section, however, also includes a brief indication of how this TBox can be exploited to generate ABox data.
- Section 5, finally, compares the proposed conversion procedure with the results of previous proposals, which were only briefly touched in Sect. 1.3.3.

2. The structure of EXPRESS

In this section, we will first look into the key characteristics and content of an EXPRESS schema. We will use the IFC4_ADD1.exp schema [2] as a reference example for this purpose. This brief introduction to EXPRESS will allow to better appreciate the diverse possible conversion routines presented in Sect. 3.

In EXPRESS, a number of declarations can be made. A distinction is hereby made between the declarations enumerated below.

- `TYPE` declarations [4, p.38]
- `ENTITY` declarations [4, p.40]
- `SCHEMA` declarations [4, p.62]
- `CONSTANT` declarations [4, p.63]
- `ALGORITHM` declarations [4, p.64] (cf. `FUNCTION` and `PROCEDURE` declarations)
- `RULE` declarations [4, p.72]

An EXPRESS schema contains exactly one `SCHEMA` declaration that covers the entire file, thereby assigning the body of this file to the particular schema declaration. An EXPRESS schema may import definitions from other schemas using the `REFERENCE FROM` keywords. The IFC schema is self-contained and there is no import of other schemas (see Fragment 1).

```
SCHEMA IFC4;
...
END_SCHEMA;
```

Fragment 1: Printout of the SCHEMA declaration present in IFC4_ADD1.exp.

At the end of the IFC4_ADD1.exp schema, there are two `RULE` declarations and 45 `FUNCTION` declarations, which will be briefly covered further on in this section. All the other declarations make use of the `TYPE` and `ENTITY` keywords that represent and reference a number of EXPRESS *data types*. A distinction is hereby made between the following data types:

- Simple data types [4, p.20], i.e. `NUMBER`, `REAL`, `INTEGER`, `LOGICAL`, `BOOLEAN`, `STRING`, `BINARY`
- Aggregation data types [4, p.23], i.e. `ARRAY`, `LIST`, `BAG`, `SET`

- Named data types [4, p.28], i.e. entity data types and defined data types
- Constructed data types [4, p.29], i.e. enumeration data types and select data types
- Generalised data types [4, p.35]

These data types will be briefly documented in the next subsections by following the order they appear in an IFC schema and skipping only the *generalised data types* since they are not regularly employed in an IFC schema (reference schema IFC4_ADD1.exp).

2.1. Simple data type declarations

Simple data types (i.e. NUMBER, REAL, INTEGER, LOGICAL, BOOLEAN, STRING, BINARY) are commonly used in an IFC schema. These data types are referenced by the other data type declarations as shown in the following subsections.

2.2. Defined (named) data type declarations

Apart from the simple data type declarations, the *defined (named) data type declarations* are the most basic elements that are used in an EXPRESS schema. These declarations are typically very short statements, eventually providing the building blocks used by more complex data types in the EXPRESS schema. Diverse defined data types can be found in an IFC EXPRESS schema and most declarations are similar to the one given in Fragment 2 (IfcAreaDensityMeasure), where a type name is given (e.g. IfcAreaDensityMeasure) together with a reference to a simple data type (e.g. REAL).

```
TYPE IfcAreaDensityMeasure = REAL;
END_TYPE;
```

Fragment 2: Printout of the defined data type declaration IfcAreaDensityMeasure.

Some defined data type declarations refer to other defined data types instead of a simple data type. For instance, the data type IfcBoxAlignment (see Fragment 3) refers to another defined type (i.e. IfcLabel). Fragment 3 provides also an example WHERE rule restriction, namely WR1. This is a local rule that only applies to the type in which it is declared. In the case of Fragment 3, the local rule specifies that the data type can be instantiated only using the values provided in the list.

```
TYPE IfcBoxAlignment = IfcLabel;
  WHERE
    WR1 : SELF IN ['top-left', 'top-middle', 'top-
      right', 'middle-left', 'center', 'middle-right
      ', 'bottom-left', 'bottom-middle', 'bottom-
      right'];
END_TYPE;

TYPE IfcLabel = STRING(255);
END_TYPE;
```

Fragment 3: Example of the defined data type declaration IfcBoxAlignment, which refers to another defined data type (IfcLabel), which in turn refers to a simple data type STRING of maximum 255 characters.

2.3. Aggregation data type declarations

Some of the defined data type declarations refer to an *aggregation data type*. These aggregation data types are declared locally and can make use of one of the following containers: ARRAY, LIST, BAG, and SET.

The LIST and ARRAY data types represent an ordered collection of elements, whereas the BAG and SET data types represent unordered collections [4, p.23]. Three examples are given in Fragment 4, namely IfcCompoundPlaneAngleMeasure, IfcLineIndex, and IfcComplexNumber. For each of these examples, cardinality restrictions are declared between brackets. As an example, the data type IfcCompoundPlaneAngleMeasure is defined as a LIST with minimum 3 and maximum 4 INTEGER elements. In addition, it can be noticed that the values of these INTEGER elements are restricted using WHERE rules.

```
TYPE IfcCompoundPlaneAngleMeasure = LIST [3:4] OF
  INTEGER;
  WHERE
    MinutesInRange : ABS(SELF[2]) < 60;
    SecondsInRange : ABS(SELF[3]) < 60;
    MicrosecondsInRange : (SIZEOF(SELF) = 3) OR (ABS
      (SELF[4]) < 1000000);
    ConsistentSign :
      ((SELF[1] >= 0) AND (SELF[2] >= 0) AND (SELF
        [3] >= 0) AND ((SIZEOF(SELF) = 3) OR (SELF
        [4] >= 0)))
      OR
      ((SELF[1] <= 0) AND (SELF[2] <= 0) AND (SELF
        [3] <= 0) AND ((SIZEOF(SELF) = 3) OR (SELF
        [4] <= 0)));
END_TYPE;

TYPE IfcLineIndex = LIST [2:?] OF IfcPositiveInteger
  ;
END_TYPE;

TYPE IfcComplexNumber = ARRAY [1:2] OF REAL;
END_TYPE;
```

Fragment 4: Example of three data type declarations referring to aggregation data types (LIST, ARRAY).

The IfcLineIndex in Fragment 4 refers to an ordered LIST of at least 2 elements. No upper boundary is declared for the LIST size. The IfcComplexNumber type refers to a fixed-size, indexed list (ARRAY). In this case, the indices between brackets indicate the *"lowest and highest index which are valid for an array value of this data type"* [4, p.24]. Therefore, an instance of IfcComplexNumber will be an array of exactly two REAL simple data type instances, indexed 1 and 2.

An appropriate example of a SET aggregation data type declaration is available in Fragment 5. In this Fragment, an ENTITY IfcArbitraryProfileDefWithVoids is declared with one of its attributes (namely InnerCurves) pointing towards a SET of IfcCurve data types. This SET is an unordered aggregation of different IfcCurve instances [4, p.27].

```
ENTITY IfcArbitraryProfileDefWithVoids
  ...
  InnerCurves : SET [1:?] OF IfcCurve;
  ...
END_ENTITY;
```

Fragment 5: Printout of the IfcArbitraryProfileDefWithVoids entity data type declaration in EXPRESS including an attribute declaration that refers to a SET aggregation data type declaration.

The IFC4_ADD1 schema does not include BAG aggregation data types, so we do not present any example here. The definition of a BAG aggregation data type is comparable to the definition of a SET aggregation data type. A BAG data type can, however, contain the same instances more than once, whereas SET data types can only contain different elements [4, p.26].

2.4. Constructed data type declarations

Two types of *constructed data types* can be declared in an EXPRESS schema, namely ENUMERATION (e.g. IfcDistributionSystemEnum) and SELECT (e.g. IfcActorSelect).

2.4.1. Enumeration data type declarations

An *enumeration data type* declaration is identified by the keyword ENUMERATION, as shown in Fragment 6. Any instantiation of such data type should refer to one of the values listed in the enumeration.

```
TYPE IfcAddressTypeEnum = ENUMERATION OF
  (OFFICE
  ,SITE
  ,HOME
  ,DISTRIBUTIONPOINT
  ,USERDEFINED);
END_TYPE;
```

Fragment 6: Printout of the ENUMERATION data type declaration IfcAddressTypeEnum, which refers to a list of the allowed values within this enumeration (OFFICE, SITE, HOME, DISTRIBUTIONPOINT and USERDEFINED).

2.4.2. Select data type declarations

A *select data type* declaration is identified by the keyword SELECT, as shown in Fragment 7. Any instantiation of such data type should refer to one instance of the listed types or entities. It must be noted that a select data type declaration might include various other EXPRESS data types, including defined (named) data types, other select data types, and entity (named) data types. The declaration of the IfcMetricValueSelect select data type represents an example of such a mixture, since it lists one defined (named) data type (i.e. IfcValue) and five entity (named) data types (i.e. IfcAppliedValue, IfcMeasureWithUnit, IfcReference, IfcTable, and IfcTimeSeries).

```
TYPE IfcMetricValueSelect = SELECT
  (IfcAppliedValue
  ,IfcMeasureWithUnit
  ,IfcReference
  ,IfcTable
  ,IfcTimeSeries
  ,IfcValue);
END_TYPE;
```

Fragment 7: Printout of the SELECT data type declaration IfcMetricValueSelect, which refers to a number of allowed data type instantiations: IfcAppliedValue, IfcMeasureWithUnit, IfcReference, IfcTable, IfcTimeSeries, and IfcValue.

2.5. Entity (named) data type declarations

Entity (named) data type declarations are identified by the keyword ENTITY and form most of the content of an EXPRESS schema in the case of IFC. An example declaration is given in Fragment 8, namely IfcBSplineCurve. After the name of the entity data type, a specification can be given of the hierarchical structure through statements that make use of the keywords SUPERTYPE and SUBTYPE. In this case, IfcBSplineCurve is an abstract supertype of IfcBSplineCurveWithKnots, implying that only IfcBSplineCurveWithKnots can be explicitly instantiated (i.e. no IfcBSplineCurve instances are allowed) while inheriting all the properties from IfcBSplineCurve. The IfcBSplineCurve entity is also a SUBTYPE of IfcBoundedCurve. An *entity data* type can have multiple supertypes and multiple subtypes [4].

```
ENTITY IfcBSplineCurve
 ABSTRACT SUPERTYPE OF (ONEOF
    (IfcBSplineCurveWithKnots))
 SUBTYPE OF (IfcBoundedCurve);
  Degree : IfcInteger;
  ControlPointsList : LIST [2:?] OF
    IfcCartesianPoint;
  CurveForm : IfcBSplineCurveForm;
  ClosedCurve : IfcLogical;
  SelfIntersect : IfcLogical;
 DERIVE
  UpperIndexOnControlPoints : IfcInteger := (SIZEOF(
    ControlPointsList) - 1);
  ControlPoints : ARRAY [0:UpperIndexOnControlPoints
    ] OF IfcCartesianPoint := IfcListToArray(
    ControlPointsList,0,UpperIndexOnControlPoints);
 WHERE
  SameDim : SIZEOF(QUERY(Temp <* ControlPointsList |
     Temp.Dim <> ControlPointsList[1].Dim)) = 0;
END_ENTITY;
```

Fragment 8: Printout of the entity (named) data type declaration `IfcBSplineCurve`.

The declaration of the entity attributes follows the declaration of the hierarchical structure. It is possible to declare explicit, derived, and inverse attributes [4, p.41]. These attributes are defined exclusively within the scope of the entity data type under which they are declared, and they establish a relationship with other data types that are declared elsewhere in the EXPRESS schema (i.e. simple data types, named data types, aggregation data types, constructed data types, entity data types). It is also possible to refer to two-dimensional aggregations, such as the `LIST` of `LIST` elements that can be found in IFC4_ADD1.exp (cf. the attribute `ControlPointsList` of entity `IfcBSplineSurface`).

A specific kind of attribute that can be defined for an entity data type is a derived attribute, identified by the keyword `DERIVE`. These are typically attributes that can be derived from the other entity data type attributes through simple rules or more complex routines. In the case of the `ControlPoints` attribute in Fragment 8, for instance, a reference is made to the declared `FUNCTION` `IfcListToArray()` in order to calculate the value of the `ControlPoints` attribute.

A second example of an entity data type declaration is given in Fragment 9 for `IfcObject`. This Fragment shows the usage of the `OPTIONAL` keyword, implying that all attributes without this annotation are required attributes in the definition of an instance of such an entity data type.

```
ENTITY IfcObject
 ABSTRACT SUPERTYPE OF (ONEOF
    (IfcActor
    ,IfcControl
    ,IfcGroup
    ,IfcProcess
    ,IfcProduct
    ,IfcResource))
 SUBTYPE OF (IfcObjectDefinition);
  ObjectType : OPTIONAL IfcLabel;
 INVERSE
  IsDeclaredBy : SET [0:1] OF IfcRelDefinesByObject
    FOR RelatedObjects;
  Declares : SET [0:?] OF IfcRelDefinesByObject FOR
    RelatingObject;
  IsTypedBy : SET [0:1] OF IfcRelDefinesByType FOR
    RelatedObjects;
  IsDefinedBy : SET [0:?] OF
    IfcRelDefinesByProperties FOR RelatedObjects;
 WHERE
  UniquePropertySetNames : IfcUniqueDefinitionNames(
    IsDefinedBy);
END_ENTITY;
```

Fragment 9: Printout of the entity (named) data type declaration `IfcObject`.

Furthermore, Fragment 9 shows the usage of inverse attributes with the keyword `INVERSE`. Any such `INVERSE` statement declares an attribute that is inverse to an attribute that has been declared elsewhere and in the opposite direction. In the case of the `IsDeclaredBy` attribute in Fragment 9, for example, the attribute going in the inverse direction can be found in Fragment 10, namely the `RelatedObjects` attribute of the `IfcRelDefinesByObject` entity data type. Finally, the usage of local `WHERE` rule declarations within an entity data type declaration is also displayed in Fragment 9.

```
ENTITY IfcRelDefinesByObject
 SUBTYPE OF (IfcRelDefines);
  RelatedObjects : SET [1:?] OF IfcObject;
  RelatingObject : IfcObject;
END_ENTITY;
```

Fragment 10: Printout of the entity (named) data type declaration `IfcRelDefinesByObject`.

2.6. FUNCTION declarations

The IFC4_ADD1.exp schema includes 45 `FUNCTION` declarations with a variable level of complexity. Fragment 11 shows a simple example `FUNCTION` declaration that is used in the declarations of the `IfcElementQuantity` and `IfcPhysicalComplexQuantity` entity data types. The `FUNCTION` `IfcUniqueQuantityNames` defines (1) what input data it expects, namely a set of `IfcPhysicalQuantity` instances, and (2) what output it generates, namely one simple `LOGICAL` data type instance. The calculation from input data to output data is given in the body of the `FUNCTION` declaration. In the case of the `IfcUniqueQuantityNames`, the number of

elements in the argument is counted and the result is returned.

```
FUNCTION IfcUniqueQuantityNames
(Properties : SET [1:?] OF IfcPhysicalQuantity)
:LOGICAL;

LOCAL
  Names : SET OF IfcLabel := [];
END_LOCAL;

REPEAT i:=1 TO HIINDEX(Properties);
  Names := Names + Properties[i].Name;
END_REPEAT;
RETURN (SIZEOF(Names) = SIZEOF(Properties));
END_FUNCTION;
```

Fragment 11: Printout of the FUNCTION declaration IfcUniqueQuantityNames.

2.7. RULE declarations

The EXPRESS language allows the declaration of *rules* that *"permit the definition of constraints that apply collectively to the entire domain of an entity data type, or to instances of more than one entity data type."* [4]. As an example, Fragment 12 shows one of the two RULE declarations made in the IFC4_ADD1.exp schema. The head of this RULE declaration indicates which entity data types are affected (i.e. IfcProject), whereas the body of the declaration indicates what restrictions apply to these entity data types (i.e. the number of IfcProject instances in a model may not be larger than one).

```
RULE IfcSingleProjectInstance
  FOR (IfcProject);

  WHERE
    WR1 : SIZEOF(IfcProject) <= 1;
END_RULE;
```

Fragment 12: Printout of the RULE declaration IfcSingleProjectInstance.

3. The EXPRESS to OWL conversion pattern

This section presents the proposed EXPRESS to OWL conversion pattern that can be used to generate an ifcOWL ontology. We will outline a reasonable number of conversion options whenever they are available and relevant in the overall conversion strategy. Section 5 will summarise a comparison of the different conversion patterns available in the literature. The structure of the previous section will be followed by matching the EXPRESS declaration examples (Fragments 1 to 12) with corresponding OWL declarations (Fragments 13 to 43) using the Turtle syntax [12].

3.1. OWL header

We propose to adopt a unique URI (namespace) for each ifcOWL ontology corresponding to a distinct EXPRESS schema of IFC. Currently, this results in four URIs for IFC2X3.exp, IFC2X3_TC1.exp, IFC4RC4.exp, and IFC4_ADD1.exp, respectively. We propose to use the following URI design for these ontologies: `http://www.buildingsmart-tech.org/ifcOWL/[schemaName]`, with `[schemaName]` being one of the four schema names. Fragment 13 shows what this results in OWL together with a few Dublin Core (dce) metadata annotations added to the ontology declaration, indicating details about the origins of the OWL ontology. The vann metadata annotations indicate preferred namespace URI and namespace prefix, and the `cc:license` property indicates which license is associated to the ontology.

```
<http://www.buildingsmart-tech.org/ifcOWL/IFC4_ADD1>
  rdf:type owl:Ontology ;
  dce:creator "Pieter Pauwels (pipauwel.
    pauwels@ugent.be)" ;
  dce:creator "Walter Terkaj (walter.terkaj@itia.cnr
    .it)" ;
  dce:date "2015/07/30" ;
  dce:contributor "Aleksandra Sojic (aleksandra.
    sojic@itia.cnr.it)" ;
  dce:title "IFC4_ADD1" ;
  dce:description "OWL ontology for the IFC4_ADD1
    conceptual data schema and exchange file format
    for Building Information Model (BIM) data" ;
  dce:format "ttl" ;
  dce:identifier "IFC4_ADD1" ;
  dce:language "en" ;
  vann:preferredNamespacePrefix "ifc" ;
  vann:preferredNamespaceUri "http://www.
    buildingsmart-tech.org/ifcOWL/IFC4_ADD1" ;
  cc:license <http://creativecommons.org/licenses/by
    /3.0/> .
```

Fragment 13: Definition of the ifcOWL ontology.

Alternatively, one could decide to use one version-independent URI for the ifcOWL ontology. Such a URI would then correspond to an equivalent ifcOWL namespace that is used as base URI for the definition of the key ontology entities (i.e. classes, object properties and data properties). In this case, the ifcOWL namespace would not change in time when the ifcOWL is updated after a new release of the IFC standard, thus supporting backward compatibility and re-use of data sets generated according to previous IFC releases. In this case, the piece of information about the specific IFC release that was used to generate the ifcOWL ontology can be stored using an additional annotation `owl:versionIRI`.

In theory, the IFC schema in EXPRESS is backward compatible, thus the version-independent alterna-

tive should be implementable. However, as the IFC schema changes quite a lot from version to version, this backward compatibility is hard to securely maintain using only OWL constructs in one singular ifcOWL namespace. Therefore, we decided to opt for separate namespaces for the ontology URI (one namespace for each IFC schema). To ensure backward compatibility, one could add relations between the (four) different ifcOWL ontologies afterwards. This will likely allow a safer implementation of the backward compatibility.

At the outset of any OWL ontology, a number of other ontologies are referenced, thereby declaring the appropriate prefixes. In the proposed ifcOWL ontology, the namespace prefixes listed in Fragment 14 are declared. We recommend the usage of a separate namespace for classes and properties that are specific to EXPRESS, namely `http://purl.org/vocab/express/`. This was also proposed by Hoang [45], Hoang and Törmä [46]. In this namespace, we can put 'helper' class and property declarations that are not specific to IFC, but rather to EXPRESS itself. We will see examples of how this separate namespace is used in the following subsections.

```
@base <http://www.buildingsmart-tech.org/ifcOWL/
  IFC4_ADD1> .
@prefix : <http://www.buildingsmart-tech.org/ifcOWL/
  IFC4_ADD1#> .
@prefix ifc: <http://www.buildingsmart-tech.org/
  ifcOWL/IFC4_ADD1#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema
  #> .
@prefix dce: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-
  syntax-ns#> .
@prefix vann: <http://purl.org/vocab/vann/> .
@prefix cc: <http://creativecommons.org/ns#> .
@prefix express: <http://purl.org/vocab/express/> .
```

Fragment 14: Printout of the ontologies that are used by the proposed ifcOWL ontology.

3.2. Simple data type declarations

The notion of a data type in EXPRESS is considerably different from a data type in OWL, since *all* declarations made in EXPRESS are data type declarations. In OWL, however, a distinction is made between `owl:Class` and `rdfs:Datatype` declarations. In other words, the EXPRESS data type declarations could map both to `owl:Class` and `rdfs:Datatype` declarations in OWL, depending on which kind of EXPRESS data type is considered. For example, the simple data type declaration `STRING` would map very well to the `xsd:string` data type in OWL. On the other hand, the entity data type declaration given in Fragment 9 maps far more naturally to an `owl:Class` declaration in OWL. This is a conversion decision that should be well-considered, as it has many implications further on in using the RDF instances according to the ifcOWL ontology.

In the case of the simple data type declarations in EXPRESS (e.g. `REAL`, `INTEGER`, and so forth), these can be converted to OWL data type or class declarations. Fragments 15 and 16 give an example of both options for the case of the simple data type `REAL`.

```
express:REAL
  rdf:type rdfs:Datatype ;
  owl:equivalentClass xsd:double .
```

Fragment 15: Printout of the RDF graph representation for one of the simple data types (`REAL`) in the ifcOWL schema, illustrating the option to convert it into an `rdfs:Datatype`.

```
express:REAL
  rdf:type owl:Class ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:allValuesFrom xsd:double ;
      owl:onProperty express:hasDouble
    ] .
```

Fragment 16: Printout of the RDF graph representation for one of the simple data types (`REAL`) in the ifcOWL schema, illustrating the option to convert it into an `owl:Class`.

If the `rdfs:Datatype` conversion option is adopted (see Fragment 15), then an instance of an EXPRESS simple data type (e.g. `REAL`) is converted to an RDF typed literal (e.g. `xsd:double`). If the `owl:Class` conversion option (see Fragment 16) is chosen, then an instance of an EXPRESS simple data type is converted to an individual of a class (e.g. `express:REAL`) and this individual is in turn linked to an RDF typed literal via a functional `owl:DatatypeProperty`. The required `owl:DatatypeProperty` must be properly declared by defining its domain, range and label (e.g. see Fragment 17 for `express:hasDouble`). Note that the `rdfs:Datatype` or `owl:Class express:REAL` and the property `hasDouble` are declared within the `express` namespace, as they are not concepts coming from IFC, but from EXPRESS. Such concepts are placed in a separate namespace, in order to be able to distinguish them from the concepts specific to IFC, but also to allow reuse of such concepts in OWL ontologies for EXPRESS schemas other than IFC. The domain declaration is done in the form of an `owl:unionOf` construct, because, in some cases, several EXPRESS sim-

ple data types (e.g. REAL and NUMBER) map to the same xsd datatype (xsd:double). An overview of the proposed mapping between the simple data types in EXPRESS and the XSD data types in OWL is given in Table 1.

```
express:hasDouble
  rdf:type owl:DatatypeProperty ;
  rdf:type owl:FunctionalProperty ;
  rdfs:label "hasDouble" ;
  rdfs:domain
    [
      rdf:type owl:Class ;
      owl:unionOf ( express:REAL express:NUMBER )
    ] ;
  rdfs:range xsd:double .
```

Fragment 17: Printout of the RDF graph representation of the express:hasDouble datatype property as it would be required in combination with the owl:Class declaration given in Fragment 16.

EXPRESS	OWL
NUMBER	xsd:double
REAL	xsd:double
INTEGER	xsd:integer
LOGICAL	xsd:boolean
BOOLEAN	xsd:boolean
STRING	xsd:string
BINARY	xsd:hexBinary

Table 1: Overview of the simple data types defined in EXPRESS and the corresponding XSD datatypes used in OWL.

If a conversion routine via an owl:Class declaration and an accompanying owl:DatatypeProperty is adopted, then an extra restriction with universal quantifier (owl:allValuesFrom) can be added to the owl:Class declaration for the corresponding datatype property. An example of such a restriction can be seen in Fragment 16 for the property express:hasDouble.

Table 1 includes one peculiar element, namely the simple data type LOGICAL. This simple data type is similar to the BOOLEAN simple data type, except that it can have three values instead of two: TRUE, FALSE, and UNKNOWN. There is no XSD data type that has these three values. This is normal, because, setting this third value would actually make little sense in the (semantic) web domain. Namely, the OWA states exactly that anything that is *not* set is by default UNKNOWN. Hence, it is possible to consider this LOGICAL data type as equal to BOOLEAN, as is also proposed in Table 1. Whenever an IFC file happens to include this UNKNOWN value, it can simply be discarded. We have opted for this option in our proposal, as this stays true to the original EXPRESS schema and makes good sense in the OWA of OWL2 DL.

Alternatively, the LOGICAL simple data type could be explicitly converted into a separate OWL class that enumerates the three named individuals express:TRUE, express:FALSE, and express:UNKNOWN (see Fragment 18). This can be further constrained by adding an owl:oneOf constraint to the express:LOGICAL class, which indicates that the express:LOGICAL class contains exactly the three given individuals express:TRUE, express:FALSE, and express:UNKNOWN.

```
express:LOGICAL
  rdf:type owl:Class ;
  owl:equivalentClass
    [
      rdf:type owl:Class ;
      owl:oneOf
      (
        express:TRUE
        express:FALSE
        express:UNKNOWN
      )
    ] .

express:TRUE
  rdf:type express:LOGICAL ;
  rdf:type owl:NamedIndividual .

express:FALSE
  rdf:type express:LOGICAL ;
  rdf:type owl:NamedIndividual .

express:UNKNOWN
  rdf:type express:LOGICAL ;
  rdf:type owl:NamedIndividual .
```

Fragment 18: Printout of the RDF graph representation of the express:LOGICAL class and its three named individuals.

In summary, for EXPRESS simple data types, we can distinguish the conversion options that are listed in Table 2. Option 1 was documented in Fragment 15. In this option, a LOGICAL will have to be converted into a datatype that is equivalent to xsd:boolean. Option 2 was documented in Fragment 16 and 17, with LOGICAL eventually pointing again to the xsd:boolean. Option 3 was documented in Fragment 16 and 17, with LOGICAL exceptionally converted as in Fragment 18. Herein we propose to follow option 2.

option 1	rdfs:Datatype	LOGICAL same as BOOLEAN
option 2	owl:Class (incl. unionOf)	LOGICAL same as BOOLEAN
option 3	owl:Class (incl. unionOf)	LOGICAL as ENUM

Table 2: Overview of the key available conversion options for simple data types, with the here proposed option marked in grey..

3.3. Defined (named) data type declarations

As shown in Fragment 2 and 3, a defined (named) data type declaration in EXPRESS can refer either to a simple data type or to another defined (named) data type. The conversion of these defined (named) data type declarations thus depends in part on the way in which simple data types are converted, as `rdfs:Datatype` declarations or as `owl:Class` declarations. Both conversion options are given for the `IfcAreaDensityMeasure` declaration in Fragments 19 and 20, respectively. These conversion options can also be used for the conversion of defined (named) data type declarations that refer to other defined (named) data types, such as `IfcBoxAlignment` (Fragment 3).

```
ifc:IfcAreaDensityMeasure
  rdf:type rdfs:Datatype ;
  owl:equivalentClass express:REAL .
```

Fragment 19: Printout of the RDF graph representation for the `IfcAreaDensityMeasure` defined in Fragment 2, when converting an EXPRESS simple data type to an `rdfs:Datatype` declaration.

```
ifc:IfcAreaDensityMeasure
  rdf:type owl:Class ;
  rdfs:subClassOf express:REAL .
```

Fragment 20: Printout of the RDF graph representation for the `IfcAreaDensityMeasure` defined in Fragment 2, when converting an EXPRESS simple data type to an `owl:Class` declaration.

Herein we decided to choose the `owl:Class` conversion option (see Fragments 16 and 20) because, even if it leads to an overhead in terms of triples in an RDF graph, this is the only solution that guarantees a safe conversion from EXPRESS to OWL, as it will be better explained in Sect. 3.5.2.

3.4. Aggregation data type declarations

In the following subsections we present the conversion pattern for aggregation data types (i.e. `BAG`, `LIST`, `SET`, `ARRAY`), while referring to the relevant EXPRESS declaration examples given before (i.e. `IfcCompoundPlaneAngleMeasure`, `IfcLineIndex`, `IfcComplexNumber`, and the `InnerCurves` attribute of `IfcArbitraryProfileDefWithVoids`). The `BAG` data type does not appear in the IFC4_ADD1 schema, so we decided not to devote a separate section on the conversion of this aggregation data type. If necessary, the general-purpose conversion pattern for `BAG` proposed by Barbau et al. [24] can be adopted.

3.4.1. *LIST* aggregation data types

Lists (or any kind of ordered sequence) cannot be easily represented in an RDF graph, because RDF relies on a triple structure that inherently allows to link *only two* concepts, not collections of multiple concepts (Fig. 1). Lists are thus typically represented in RDF by linking each concept to the next using `rdf:List`, `rdf:first` and `rdf:rest` declarations (see [14] and [47]). When using this construct in an OWL ontology, however, the ontology goes beyond OWL2 DL expressiveness (see Fig. 2), thus impeding the use of generally available semantic web tools. This would violate the first criterion that we have set in Sect. 1.3.4 (i.e. to keep ifcOWL in OWL2 DL). Therefore, this conversion routine is not a viable option.

There have been suggestions to represent ordered lists in a fashion alternative to the `rdf:List`, `rdf:first` and `rdf:rest` declarations [47]. Most of these alternative suggestions rely on additional statements that correctly define the relations between the elements of a list to remain in OWL2 DL expressiveness, so that the first conversion criterion can be met. Herein, our suggested solution relies on the class `express:List` to represent the list elements as proposed by Drummond et al. [47]. Each element of the list is related to the following element in the list by using the object property `express:hasNext`, that is a sub-property of the transitive property `express:isFollowedBy`. Finally, each list element is linked with its actual content via the object property `express:hasContents`. These definitions (see Fragment 21) are declared in the `express` namespace so that they can be used in other ontologies in different contexts, since they are not specific to IFC but rather to EXPRESS.

```
express:List
  rdf:type owl:Class ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:isFollowedBy ;
      owl:allValuesFrom express:List
    ] ,
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasNext ;
      owl:allValuesFrom express:List
    ] .

express:hasContents
  rdf:type owl:ObjectProperty ;
  rdf:type owl:FunctionalProperty ;
  rdfs:label "hasContents" ;
  rdfs:domain express:List .

express:isFollowedBy
```

```
  rdf:type owl:ObjectProperty ;
  rdf:type owl:TransitiveProperty ;
  rdfs:label "isFollowedBy" ;
  rdfs:range express:List ;
  rdfs:domain express:List .

express:hasNext
  rdf:type owl:ObjectProperty ;
  rdf:type owl:FunctionalProperty ;
  rdfs:label "hasNext" ;
  rdfs:range express:List ;
  rdfs:domain express:List ;
  rdfs:subPropertyOf express:isFollowedBy .
```

Fragment 21: Printout of the additional OWL statements that allow to represent ordered lists, including the `express:List` class declaration and the three object property declarations `express:hasContents`, `express:hasNext`, and `express:isFollowedBy`.

In order to be fully compliant with the list proposal by Drummond et al. [47], one could also include an `express:EmptyList` class declaration, as shown in Fragment 22. Instances of this class, which is declared as a subclass of `express:List`, can then be used to mark the end of the list. We did not include this `express:EmptyList` in our proposal, as it does not seem to bring much added value.

```
express:EmptyList
  rdf:type owl:Class ;
  rdfs:subClassOf express:List ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasContents ;
      owl:maxQualifiedCardinality "0"^^xsd:
        nonNegativeInteger ;
    ] ,
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasNext ;
      owl:maxQualifiedCardinality "0"^^xsd:
        nonNegativeInteger ;
    ] .
```

Fragment 22: Printout of the `express:EmptyList` class declaration that could be added to the statements in Fragment 21 in order to be fully compliant with Drummond et al. [47].

Any specific ordered list can be defined as a subclass of `express:List` while specialising the relevant restrictions. Fragment 23 shows how the class `express:INTEGER_List` is used to define an item in a list of `INTEGER` instances, while restrictions are used on object properties `express:hasContents` and `express:hasNext` to specify that the content of the list item must be an `INTEGER` instance and the next item in the list must be an `INTEGER_List` instance, respectively. All these class declarations are declared within the `express` namespace.

```
express:INTEGER_List
  rdf:type owl:Class ;
  rdfs:subClassOf express:List ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasContents ;
      owl:allValuesFrom express:INTEGER
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasContents ;
      owl:someValuesFrom express:INTEGER
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:isFollowedBy ;
      owl:allValuesFrom express:INTEGER_List
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasNext ;
      owl:allValuesFrom express:INTEGER_List
    ] .
```

Fragment 23: Printout of the RDF graph representation for class `INTEGER_List`, which relies on the definitions presented earlier in Fragment 21.

`IfcCompoundPlaneAngleMeasure` is defined as a list of `INTEGER` instances (see Fragment 4) and, therefore, it is converted as a subclass of `express:INTEGER_List` (see Fragment 24).

```
ifc:IfcCompoundPlaneAngleMeasure
  rdf:type owl:Class ;
  rdfs:subClassOf express:INTEGER_List ;
```

Fragment 24: Printout of the RDF graph representation for the `IfcCompoundPlaneAngleMeasure` defined data type, which refers to a `LIST` aggregation data type declaration, as displayed earlier in Fragment 4.

The size limits of aggregation data types can be bounded or unbounded. For instance, `IfcCompoundPlaneAngleMeasure` has a lower bound equal to three and an upper bound equal to four, whereas `IfcLineIndex` has a lower bound equal to two and an unbounded upper limit. The constraints on the size of the list could be well defined via cardinality restrictions (`owl:maxQualifiedCardinality`, `owl:minQualifiedCardinality`, or `owl:qualifiedCardinality`) on the transitive object property `express:isFollowedBy`, as shown in Fragment 25 for `IfcCompoundPlaneAngleMeasure`. Because the cardinality restrictions are set on the transitive object property `express:isFollowedBy`, one should only start counting *after* the first instance of *express:List*. As a result,

if an EXPRESS LIST has at least three elements and at most four elements, it means that that first instance of *express:List is followed by* at least two instances of *express:List* and at most three instances of *express:List*.

```
ifc:IfcCompoundPlaneAngleMeasure
  rdf:type owl:Class ;
  rdfs:subClassOf express:INTEGER_List ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:maxQualifiedCardinality "3"^^xsd:
        nonNegativeInteger ;
      owl:onClass express:INTEGER_List ;
      owl:onProperty express:isFollowedBy
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:minQualifiedCardinality "2"^^xsd:
        nonNegativeInteger ;
      owl:onClass express:INTEGER_List ;
      owl:onProperty express:isFollowedBy
    ] .
```

Fragment 25: Printout of the RDF graph representation for the IfcCompoundPlaneAngleMeasure defined data type, which refers to an aggregation data type, as displayed earlier in Fragment 4.

However, the declaration of a cardinality restriction on a transitive object property would cause ifcOWL to be outside of OWL2 DL expressiveness [48, Sect.8.2], again violating our first criterion. An alternative way in which cardinality restrictions for EXPRESS LISTs might by expressed, is given in Fragment 26. In this proposal, cardinality restrictions are not set on the transitive property express:isFollowedBy, but instead, universal (owl:allValuesFrom) and existential quantifiers (owl:someValuesFrom) are used together with the non-transitive property express:hasNext, which is allowed in OWL2 DL and which allows to indirectly set the maximum and minimum cardinality constraints defined in EXPRESS schemas. The number of nested restrictions that is added depends on the cardinality that needs to be set. In the case of IfcCompoundPlaneAngleMeasure, three universal quantifiers are added (see Fragment 26) to specify that the fourth item in the list (express:INTEGER_List) cannot be followed by another item by using an owl:qualifiedCardinality of 0. Furthermore, two existential quantifiers are used in Fragment 26 to define a restriction on property express:hasNext specifying that the first list item must be followed by an express:INTEGER_List instance, which in turn must be followed by another express:INTEGER_List instance.

```
ifc:IfcCompoundPlaneAngleMeasure
  rdf:type owl:Class ;
  rdfs:subClassOf express:INTEGER_List ,
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasNext ;
      owl:allValuesFrom
        [
          rdf:type owl:Restriction ;
          owl:onProperty express:hasNext ;
          owl:allValuesFrom
            [
              rdf:type owl:Restriction ;
              owl:onProperty express:hasNext ;
              owl:allValuesFrom
                [
                  rdf:type owl:Restriction ;
                  owl:onProperty express:hasNext ;
                  owl:onClass express:INTEGER_List ;
                  owl:qualifiedCardinality "0"^^xsd:
                    nonNegativeInteger
                ]
            ]
        ]
    ] ,
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasNext ;
      owl:someValuesFrom
        [
          rdf:type owl:Restriction ;
          owl:onProperty express:hasNext ;
          owl:someValuesFrom express:INTEGER_List
        ]
    ] .
```

Fragment 26: Alternative definition of cardinality restrictions on LIST classes, using a specific number of universal (owl:allValuesFrom) and existential quantifiers (owl:someValuesFrom) on the non-transitive property express:hasNext.

The proposal in Fragment 26 can be generalised to constrain the list size for any data type (e.g. *DTypeX*) defined as a LIST aggregation of another data type (e.g. *DTypeY*). The pseudocode of Algorithm 1 (see function *ListMinSize*) can be used to generate a restriction for the minimum list size (*MinSize*) if it is greater than one, whereas Algorithm 2 (see function *ListMaxSize*) generates a restriction for the maximum list size (*MaxSize*) if it is greater than one and not unbounded. If the minimum and maximum size are equal (*Size*), then Algorithm 3 (see function *ListExactSize*) can be employed instead of Algorithms 1 and 2.

The presented conversion procedure for an EXPRESS LIST is not ideal or highly performing. As outlined in Pauwels et al. [49] and de Farias et al. [50], more elegant options are available by replacing the verbose and complex LIST constructs with (1) semantically more meaningful and more direct object properties or (2) using datatype properties pointing to Well-Known Text (WKT) values. Examples of these two alternative approaches can be found in Pauwels et al. [49]. Because

Algorithm 1 Generation of a restriction to constrain the minimum size of a LIST aggregation data type.

```
1:  function ListMinSize(DTypeX,DTypeY,MinSize)
2:    print DTypeX, " rdfs:subClassOf"
3:    GenRestrMinSize(DTypeY,MinSize)
4:    print "."
5:  end function

6:  function GenRestrMinSize(DTypeY,MinSize)
7:    if (MinSize-2) >0 then
8:      for i=1 to (MinSize-2) do
9:        print "[ "
10:       print "rdf:type owl:Restriction ;"
11:       print "owl:onProperty express:hasNext ;"
12:       print "owl:someValuesFrom "
13:     end for
14:   end if
15:   print "["
16:   print "rdf:type owl:Restriction ;"
17:   print "owl:onProperty express:hasNext ;"
18:   print "owl:someValuesFrom ", DTypeY, "_List"
19:   print "]"
20:   if (MinSize-2) >0 then
21:     for i=1 to (MinSize-2) do
22:       print "]"
23:     end for
24:   end if
25: end function
```

Algorithm 2 Generation of a restriction to constrain the maximum size of a LIST aggregation data type.

```
1:  function ListMaxSize(DTypeX,DTypeY,MaxSize)
2:    print DTypeX, " rdfs:subClassOf"
3:    GenRestrMaxSize(DTypeY,MaxSize)
4:    print "."
5:  end function

6:  function GenRestrMaxSize(DTypeY,MaxSize)
7:    if (MaxSize-1) >0 then
8:      for i=1 to (MaxSize-1) do
9:        print "[ "
10:       print "rdf:type owl:Restriction ;"
11:       print "owl:onProperty express:hasNext ;"
12:       print "owl:allValuesFrom "
13:     end for
14:   end if
15:   print "["
16:   print "rdf:type owl:Restriction ;"
17:   print "owl:onProperty express:hasNext ;"
18:   print "owl:onClass ", DTypeY, "_List ;"
19:   print "owl:qualifiedCardinality "0"^^xsd:nonNegativeInteger"
20:   print "]"
21:   if (MaxSize-1) >0 then
22:     for i=1 to (MaxSize-1) do
23:       print "]"
24:     end for
25:   end if
26: end function
```

they cannot be generally applied and they diverge from the original EXPRESS schema, however, these alternatives are not appropriate options for the general purpose ifcOWL ontology targeted in this article. In our proposal, we therefore chose to adopt the definition given in Fragment 26, which is as rich as the original EXPRESS representation and remains in OWL2 DL, and thus fits best to our conversion criteria.

Algorithm 3 Generation of a restriction to constrain the exact size of a LIST aggregation data type.

```
1:  function ListExactSize(DTypeX,DTypeY,Size)
2:    print DTypeX, " rdfs:subClassOf"
3:    GenRestrExactSize(DTypeY,Size)
4:    print "."
5:  end function

6:  function GenRestrExactSize(DTypeY,Size)
7:    if (Size-1) >0 then
8:      for i=1 to (Size-1) do
9:        print "[ "
10:       print "rdf:type owl:Restriction ;"
11:       print "owl:onProperty express:hasNext ;"
12:       print "owl:someValuesFrom "
13:     end for
14:   end if
15:   print "["
16:   print "rdf:type owl:Restriction ;"
17:   print "owl:onProperty express:hasNext ;"
18:   print "owl:onClass ", DTypeY, "_List ;"
19:   print "owl:qualifiedCardinality "0"^^xsd:nonNegativeInteger"
20:   print "]"
21:   if (Size-1) >0 then
22:     for i=1 to (Size-1) do
23:       print "]"
24:     end for
25:   end if
26: end function
```

3.4.2. `ARRAY` aggregation data types

`ARRAY` data type declarations (see `IfcComplexNumber` in Fragment 4) are not much different from `LIST` data type declarations, the one difference being that an `ARRAY` is fixed in size whereas a `LIST` is not fixed in size [4, p.24]. We propose to use the conversion routine for `LIST` data type declarations also for `ARRAY` data type declarations (see Fragment 27), adding also a restriction to set the size of the array. Such restriction can be generated by exploiting the pseudocode in Algorithm 3.

```
ifc:IfcComplexNumber
  rdf:type owl:Class ;
  rdfs:subClassOf express:REAL_List ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:onProperty express:hasNext ;
      owl:someValuesFrom
        [
          rdf:type owl:Restriction ;
          owl:onProperty express:hasNext ;
          owl:onClass express:REAL_List ;
          owl:qualifiedCardinality "0"^^xsd:
            nonNegativeInteger
        ]
```

```
  ] .
```

Fragment 27: Printout of the RDF graph representation for the IfcComplexNumber defined data type, which refers to an ARRAY aggregation data type declaration, as displayed earlier in Fragment 4.

3.4.3. *SET aggregation data types*

The SET aggregation data types are *unordered* aggregations of instances that are supposed to be different from each other [4, p.27]. This is naturally represented in OWL through a common non-functional object property that can be assigned an unlimited number of times to the same instance. Cardinality restrictions for this non-functional property allow to represent possible constraints on the size of the set, as defined in the EXPRESS schema. An example of how this can be converted is given in Fragment 28 for the IfcArbitrary-ProfileDefWithVoids entity data type declaration given earlier in Fragment 5.

```
ifc:IfcArbitraryProfileDefWithVoids
  ...
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:minQualifiedCardinality "1"^^xsd:
        nonNegativeInteger ;
      owl:onClass ifc:IfcCurve ;
      owl:onProperty ifc:InnerCurves
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:allValuesFrom ifc:IfcCurve ;
      owl:onProperty ifc:InnerCurves
    ] .
```

Fragment 28: Partial printout of the owl:Class declaration for the IfcArbitraryProfileDefWithVoids entity data type declaration displayed in Fragment 5, including a cardinality restriction on the minimum size of the set (cf. SET[1:?] of IfcCurve).

The SET aggregation data type declaration in Fragment 5 can be represented as in Fragment 28 because it is declared within an EXPRESS *attribute declaration*. This attribute declaration maps well to an OWL property declaration for which value and cardinality restrictions can be declared (see Sect. 3.6 for further details). However, in the IFC4_ADD1 schema, the SET aggregation data type is also used in the declaration of the *defined data type* IfcPropertySetDefinition-Set. Unfortunately, in this case, no attribute is involved and, therefore, it is not possible to convert the relation between IfcPropertySetDefinitionSet and IfcPropertySetDefinition by means of a restriction on a non-functional OWL object property that maps an attribute declaration (since no such property is available). The best possibility to cope with this one case is likely to modify the original IFC schema in EXPRESS and replace the defined data type IfcPropertySetDefinitionSet with an attribute declaration that points to a SET of specific EXPRESS data types. This would allow to use the conversion procedure just explained, using non-functional OWL object properties.

As such a modification of the EXPRESS schema lies beyond our capacities, however, we propose to convert this exception as displayed in Fragment 29, namely by defining IfcPropertySetDefinition-Set as an owl:Class with a universal restriction (owl:allValuesFrom) on the predefined object property express:hasSet. This object property is not directly converted from the EXPRESS schema and, therefore, it represents an overhead that is required to cope with the misalignment between EXPRESS and OWL. The constraint on the minimum size of the SET is converted using an owl:minQualifiedCardinality restriction (see Fragment 29).

```
ifc:IfcPropertySetDefinitionSet
  rdf:type owl:Class ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:allValuesFrom ifc:IfcPropertySetDefinition
        ;
      owl:onProperty express:hasSet
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:minQualifiedCardinality "1"^^xsd:
        nonNegativeInteger ;
      owl:onProperty express:hasSet ;
      owl:onClass ifc:IfcPropertySetDefinition
    ] .
```

Fragment 29: Printout of the RDF graph representation for the IfcPropertySetDefinitionSet defined data type, which refers to a SET aggregation data type in EXPRESS.

3.5. *Constructed data type declarations*

The ENUMERATION data type and SELECT data type declarations are quite peculiar of the EXPRESS language and have no immediate equivalent in OWL. These data types could be converted into owl:Class or rdfs:Datatype declarations. We propose to choose the owl:Class option, as the following subsections will illustrate.

3.5.1. *Enumeration data type declarations*

A first conversion option (option 1) consists in converting any ENUMERATION data type into an owl:Class

that is equivalent to *one of* a limited set of named individuals (`owl:NamedIndividual`), each individual representing one enumeration item. Therefore, the `ENUMERATION` data type declaration of `IfcAddressTypeEnum` given earlier in Fragment 6 is converted into the representation reported in Fragment 30.

```
ifc:IfcAddressTypeEnum
  rdf:type owl:Class ;
  owl:equivalentClass
    [
      rdf:type owl:Class ;
      owl:oneOf
      (
        ifc:OFFICE_of_IfcAddressTypeEnum
        ifc:SITE_of_IfcAddressTypeEnum
        ifc:HOME
        ifc:DISTRIBUTIONPOINT
        ifc:USERDEFINED_of_IfcAddressTypeEnum
      )
    ] ;
  rdfs:subClassOf express:ENUMERATION .
```

Fragment 30: Printout of the RDF graph representation for the `IfcAddressTypeEnum` enumeration data type declaration displayed earlier in Fragment 6.

Since the enumeration items are defined within the scope of the `ENUMERATION`, it may happen that the same name is used for items belonging to different enumerations. Because an `owl:NamedIndividual` does not have such a local scope, ambiguity might occur and result in inconsistencies. Therefore, while converting the enumeration to an `owl:Class`, we propose to uniquely name its set of named individuals by adding a suffix according to the following naming convention: `[NameOfNamedIndividual]_of_[EnumerationName]`. This naming convention is followed if and only if such an ambiguity indeed occurs. For the example in Fragment 30, `ifc:HOME` and `ifc:DISTRIBUTIONPOINT` are not renamed, whereas the other `owl:NamedIndividual` declarations are renamed to guarantee a unique URI.

As a slight variation (option 2), one could opt to follow this naming convention for *all* enumeration items, but this would result in notably more changes compared to the original IFC schema, which would violate the second and third criteria given in the introduction (keep as closely as possible to the original IFC schema and to the corresponding IFC instance file). Anyhow, the original name of the enumeration item, as defined in the EXPRESS schema, is preserved by the annotation `rdfs:label` added to each `owl:NamedIndividual` (see Fragment 31). Finally, each enumeration class is defined as a subclass of the predefined class `express:ENUMERATION` (see Fragment 30).

```
ifc:SITE_of_IfcAddressTypeEnum
  rdf:type ifc:IfcAddressTypeEnum ;
  rdf:type owl:NamedIndividual ;
  rdfs:label "SITE" .
```

Fragment 31: Printout of the RDF graph representation for the `SITE_of_IfcAddressTypeEnum` named individual referenced by the `IfcAddressTypeEnum` class declaration in Fragment 30.

Two alternative conversion options were suggested by Krima et al. [41], Barbau et al. [24] (option 3) and Hoang [45], Hoang and Törmä [46] (option 4). Both options enforce no renaming, and the proposal by Krima et al. [41], Barbau et al. [24] (option 3) additionally avoids the use of the `owl:oneOf` restriction. Indeed, named individuals (such as `ifc:USERDEFINED`) may belong to multiple classes. This is feasible in OWL and results in a less specific, but also less restricted ifcOWL ontology, as one can see in Fragment 32. Note that we only list 10 of the 161 classes in Fragment 32 of which `ifc:USERDEFINED` would in this case be an instance.

```
ifc:IfcAddressTypeEnum
  rdf:type owl:Class ;
  rdfs:subClassOf express:ENUMERATION .

ifc:OFFICE
  rdf:type ifc:IfcAddressTypeEnum , ifc:
    IfcCrewResourceTypeEnum ;
  rdf:type owl:NamedIndividual ;
  rdfs:label "OFFICE" .

ifc:SITE
  rdf:type ifc:IfcAddressTypeEnum , ifc:
    IfcCrewResourceTypeEnum , ifc:
    IfcAssemblyPlaceEnum ;
  rdf:type owl:NamedIndividual ;
  rdfs:label "SITE" .

ifc:HOME
  rdf:type ifc:IfcAddressTypeEnum ;
  rdf:type owl:NamedIndividual ;
  rdfs:label "HOME" .

ifc:DISTRIBUTIONPOINT
  rdf:type ifc:IfcAddressTypeEnum ;
  rdf:type owl:NamedIndividual ;
  rdfs:label "DISTRIBUTIONPOINT" .

ifc:USERDEFINED
  rdf:type ifc:IfcElectricTimeControlTypeEnum ,
    ifc:IfcTaskTypeEnum ,
    ifc:IfcOpeningElementTypeEnum ,
    ifc:IfcAirTerminalBoxTypeEnum ,
    ifc:IfcProjectionElementTypeEnum ,
    ifc:IfcElectricFlowStorageDeviceTypeEnum ,
    ... ,
    ifc:IfcActionTypeEnum ,
    ifc:IfcDuctSegmentTypeEnum ,
    ifc:IfcRoofTypeEnum ,
    ifc:IfcStructuralCurveActivityTypeEnum ;
  rdf:type owl:NamedIndividual ;
  rdfs:label "USERDEFINED" .
```

Fragment 32: Printout of the RDF graph representation for the `IfcAddressTypeEnum` enumeration data type declaration displayed earlier in Fragment 6, including the declaration of its individuals, without renaming of named individuals and without the `oneOf` restriction. Note that we only list 10 of the 161 classes of which `ifc:USERDEFINED` is an instance.

The four conversion options documented above are summarised in Table 3. Herein we suggest to adopt the first conversion option, i.e. converting any `ENUMERATION` data type into an `owl:Class` that is equivalent to *one of* a limited set of named individuals (`owl:NamedIndividual`), while minimising the number of renaming, as shown in Fragment 30.

	renaming	`oneOf`
option 1	partial	yes
option 2	all	yes
option 3	none	no
option 4	none	yes

Table 3: Overview of the key available conversion options for `ENUMERATION` data types, with the proposed option marked in grey.

3.5.2. Select data type declarations

We propose to convert any `SELECT` data type into an `owl:Class` that is equivalent to a *union of* a limited set of `owl:Class` statements (option 1). Thus, the `SELECT` data type declaration given earlier in Fragment 7 is converted into the representation given in Fragment 33.

```
ifc:IfcMetricValueSelect
  rdf:type owl:Class ;
  owl:equivalentClass
    [
      rdf:type owl:Class ;
      owl:unionOf
        (
          ifc:IfcAppliedValue
          ifc:IfcMeasureWithUnit
          ifc:IfcReference
          ifc:IfcTable
          ifc:IfcTimeSeries
          ifc:IfcValue
        )
    ] ;
  rdfs:subClassOf express:SELECT .
```

Fragment 33: Printout of the RDF graph representation for the `IfcMetricValueSelect` *select data type* declaration displayed earlier in Fragment 7.

As an alternative (option 2), it is possible to replace the `owl:unionOf` restriction with a simple hierarchical `rdfs:subClassOf` declaration, which would result in a less restricted, but also less specific ifcOWL ontology (there would be no distinction between the conversion of `SUBCLASSOF` declarations and `SELECT` data type declarations in EXPRESS). This alternative option is adopted by Krima et al. [41], Barbau et al. [24] and Hoang [45], Hoang and Törmä [46]. These two main conversion options are summarised in Table 4. Herein we suggest to adopt the first conversion option, i.e. converting any `SELECT` data type into an `owl:Class` that is equivalent to a *union of* a limited set of OWL classes.

	`unionOf`	`subClassOf`
option 1	yes	no
option 2	no	yes

Table 4: Overview of the key available conversion options for `SELECT` data types, with the proposed option marked in grey.

In both options summarised in Table 4, the conversion of `SELECT` data type declarations is rather straightforward, but it is in both cases of crucial importance because it affects the whole EXPRESS to OWL conversion strategy. As we anticipated in Sect. 2.4.2, a `SELECT` data type declaration can refer to *any* other data type declared in the EXPRESS schema (i.e. simple data types, named data types, aggregation data types, and entity data types), as shown in the following critical examples of IFC4_ADD1.exp:

- `IfcColourOrFactor` includes the entity data type `IfcColourRgb` and the defined data type `IfcNormalisedRatioMeasure`.
- `IfcTrimmingSelect` includes the entity data type `IfcCartesianPoint` and the defined data type `IfcParameterValue`.
- `IfcPresentationStyleSelect` includes the enumeration data type `IfcNullStyle` and the entity data types `IfcTextStyle`, `IfcFillAreaStyle`, `IfcCurveStyle`, and `IfcSurfaceStyle`.

However, the data types included in a `SELECT` data type should all be converted in the same way to avoid inconsistencies in the ifcOWL ontology, because an `owl:unionOf` statement (see Fragment 33) *cannot* include references to both `owl:Class` and `rdfs:Datatype` instances. As we saw in Sect. 3.2 and 3.3, a simple data type and a defined data type declaration can be represented both as an `owl:Class` and `rdfs:Datatype` declaration, whereas aggregation data type and entity data type declarations (see next Sect. 3.6) must be converted into `owl:Class` declarations. If the simple data type and defined data type declarations were converted into `rdfs:Datatype` declarations, it would result in

an owl:unionOf statement involving both owl:Class and rdfs:Datatype.

Therefore, as anticipated in Sect. 3.3, we propose to convert all simple data type and defined (named) data type declarations as owl:Class declarations, thus choosing for the options given in Fragments 16 and 20. This decision is consistent with the literature as already suggested by Schevers and Drogemuller [38] (referring to the conversion as *"objectifying the EXPRESS types into OWL classes"*) and Barbau et al. [24] (referring to the conversion as *"data wrapping"*). Finally, each select class is defined as a subclass of the custom, predefined class express:SELECT (see Fragment 33).

3.6. Entity (named) data type declarations

There is a common agreement among previous approaches in the literature to convert EXPRESS entity (named) data type declarations into owl:Class declarations. Indeed, most of the elements that are part of an entity data type declaration in EXPRESS, if not all, have direct parallels in owl:Class declarations.

3.6.1. Class hierarchy statements

The SUBTYPE OF declaration is converted into a declaration using rdfs:subClassOf. If the EXPRESS entity is an ABSTRACT SUPERTYPE OF, then the corresponding owl:Class is declared as a subclass of the union (owl:unionOf) of its subclasses. Moreover, if the subclasses must be disjoint (see declaration ONE OF in EXPRESS), then an OWL axiom involving owl:disjointWith is added to the conversion. Fragment 34 shows the conversion of these core owl:Class statements, which correspond to the first four lines of the IfcBSplineCurve entity (see Fragment 8).

```
ifc:IfcBSplineCurve
  rdf:type owl:Class ;
  rdfs:subClassOf ifc:IfcBoundedCurve ;
  rdfs:subClassOf
    [
      rdf:type owl:Class ;
      owl:unionOf
        (
          ifc:IfcBSplineCurveWithKnots
        )
    ] ;
  owl:disjointWith
      ifc:IfcPolyline,
      ifc:IfcIndexedPolyCurve,
      ifc:IfcCompositeCurve,
      ifc:IfcTrimmedCurve .
```

Fragment 34: Partial printout of the RDF graph representation for the IfcBSplineCurve entity (named) data type declaration displayed earlier in Fragment 8 (class header).

In the other lines (besides the first four) listed in Fragment 8, five regular attributes and two DERIVE attributes are declared. Herein, for sake of conciseness we avoid to list the conversion result for all the attributes that are declared for the entity IfcBSplineCurve (see Fragment 8), but we limit ourselves to two key reference examples, namely the Degree and ControlPointsList attributes.

3.6.2. Regular attributes

We suggest to convert any regular attribute declared within an entity (named) data type declaration as an object property declaration in OWL with the addition of a proper set of restrictions to the declaration of the owl:Class. Only object properties (and no data properties) are employed to convert the attributes because all simple data type and defined (named) data type declarations are converted into owl:Class declarations, as explained in Sect. 3.5.2.

The suggested conversion for the Degree and ControlPointsList attributes are shown in Fragments 35 and 36, respectively. Any regular attribute is converted into an owl:ObjectProperty, while explicitly adding also the declaration of the domain and range. In these two cases, the properties are declared as owl:FunctionalProperty because the maximum cardinality of the attribute is equal to one. Moreover, we propose to guarantee that each object property is characterised by one rdfs:domain and rdfs:range.

```
ifc:Degree
  rdfs:label "Degree" ;
  rdfs:domain ifc:IfcBSplineCurve ;
  rdfs:range ifc:IfcInteger ;
  rdf:type owl:FunctionalProperty, owl:
    ObjectProperty .
```

Fragment 35: Printout of the RDF graph representation for the object property ifc:Degree, which is used to convert the entity (named) data type declaration IfcBSplineCurve displayed earlier in Fragment 8.

```
ifc:ControlPointsList_of_IfcBSplineCurve
  rdfs:label "ControlPointsList" ;
  rdfs:domain ifc:IfcBSplineCurve ;
  rdfs:range ifc:IfcCartesianPoint_List ;
  rdf:type owl:FunctionalProperty, owl:
    ObjectProperty .
```

Fragment 36: Printout of the RDF graph representation for the object property ifc:ControlPointsList_of_IfcBSplineCurve, which is used to convert the entity (named) data type declaration IfcBSplineCurve displayed earlier in Fragment 8.

Because EXPRESS attributes are defined in the scope of the entity (see Sect. 2.5), it can occur that multiple attributes in different entity scopes have the same name,

thus being just homonyms. The uniqueness of the object property name can be enforced by following a similar approach adopted to rename enumeration items with identical names (cf. Sect. 3.5.1). In this case, the renaming convention consists in adding a suffix that recalls the name of the entity where the attribute is defined, i.e. `[NameOfAttribute]_of_[NameOfEntity]`. This convention is different compared to the work by Krima et al. [41], where the renaming is enforced for all properties, even in case of no ambiguity, by adding a prefix that recalls the name of the entity, i.e. `[NameOfEntity]_has_[NameOfAttribute]`.

Anyhow, as a novelty with respect to the state of the art, we propose to keep track of the original name of the attribute by adding an annotation `rdfs:label` to each object property. An example is represented by the conversion of the attribute `ControlPointsList` into the `ifc:ControlPointsList_of_IfcBSplineCurve` object property (Fragment 36), which has the original name `ControlPointsList` as its `rdfs:label` property. Note that the range of the corresponding object property is not the class `ifc:IfcCartesianPoint`, but the class `ifc:IfcCartesianPoint_List`, because the EXPRESS attribute `ControlPointsList` refers to a `LIST`.

The range type and cardinality constraints of relevance for each attribute in EXPRESS are added as restrictions to the OWL class that acts as the domain of the property. Fragments 37 and 38 illustrate the declaration of such restrictions for the object properties `ifc:Degree` and `ifc:ControlPointsList_of_IfcBSplineCurve` of class `ifc:IfcBSplineCurve`. A universal quantifier restriction (`owl:allValuesFrom`) is added in all cases. For the examples in Fragments 37 and 38, an additional `owl:qualifiedCardinality` restriction is added, with its value set to one (‘1’), because the original entity attributes are strictly required and have a maximum cardinality equal to one (‘1’). The following examples will show the use of the `owl:maxQualifiedCardinality` and `owl:minQualifiedCardinality` restrictions (Fragment 40 and 43).

```
ifc:IfcBSplineCurve
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:allValuesFrom ifc:IfcInteger ;
      owl:onProperty ifc:Degree
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:qualifiedCardinality "1"^^xsd:
        nonNegativeInteger ;
      owl:onProperty ifc:Degree ;
      owl:onClass ifc:IfcInteger
    ] .
```

Fragment 37: Partial printout of the RDF graph representation for the `IfcBSplineCurve` entity (named) data type declaration displayed earlier in Fragment 8 (restrictions on property `ifc:Degree`).

```
ifc:IfcBSplineCurve
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:allValuesFrom ifc:IfcCartesianPoint_List ;
      owl:onProperty ifc:
        ControlPointsList_of_IfcBSplineCurve
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:qualifiedCardinality "1"^^xsd:
        nonNegativeInteger ;
      owl:onProperty ifc:
        ControlPointsList_of_IfcBSplineCurve ;
      owl:onClass ifc:IfcCartesianPoint_List
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:onProperty ifc:
        ControlPointsList_IfcBSplineCurve ;
      owl:allValuesFrom
        [
          rdf:type owl:Restriction ;
          owl:onProperty express:hasNext ;
          owl:someValuesFrom ifc:
            IfcCartesianPoint_List
        ]
    ] .
```

Fragment 38: Partial printout of the RDF graph representation for the `IfcBSplineCurve` entity (named) data type declaration displayed earlier in Fragment 8 (restrictions on property `ifc:ControlPointsList_of_IfcBSplineCurve`).

Unfortunately, it is not possible to add explicit cardinality restrictions on the object property `ifc:ControlPointsList_of_IfcBSplineCurve` in order to represent also the `LIST` size limits (i.e. minimum two, maximum unbounded as defined in Fragment 8) because of the motivations explained in Sect. 3.4.1. As a workaround, we propose to add a further restriction using a correct combination of universal and existential qualifiers to set the minimum list size. In the example in Fragment 38, the last restriction enforces the first item of the list to be directly followed by another item, thus representing a minimum list size equal to two as required. A further restriction is not needed for the maximum size because the list is unbounded.

Also in this case the proposal in Fragment 38 can be generalized to constrain the LIST size for any en-

tity data type (e.g. *EntV*) characterised by an attribute (e.g. *AttrW*) defined as a LIST of another data type (e.g. *DTypeZ*). The pseudocode of Algorithm 4 (see function *AttrListMin*) can be used to generate a restriction for the minimum list size (*MinSize*) if it is greater than one, whereas Algorithm 5 (see function *AttrListMax*) generates a restriction for the maximum list size (*MaxSize*) if it is greater than one and not unbounded. If the minimum and maximum size are equal (*Size*), then Algorithm 6 (see function *AttrListExact*) can be employed instead of Algorithms 4 and 5.

Algorithm 4 Generation of a restriction to constrain the minimum size of an attribute defined as a LIST aggregation data type. Function *GenRestrMinSize* from Algorithms 1 is reused.

```
1: function AttrListMin(EntV,AttrW,DTypeZ,MinSize)
2:   print EntV, " rdfs:subClassOf"
3:   print "[ "
4:   print "rdf:type owl:Restriction ;"
5:   print "owl:onProperty ", AttrW, " ;"
6:   print "owl:allValuesFrom "
7:   GenRestrMinSize(DTypeZ,MinSize)
8:   print "] ."
9: end function
```

Algorithm 5 Generation of a restriction to constrain the maximum size of an attribute defined as a LIST aggregation data type. Function *GenRestrMaxSize* from Algorithms 2 is reused.

```
1: function AttrListMax(EntV,AttrW,DTypeZ,MaxSize)
2:   print EntV, " rdfs:subClassOf"
3:   print "[ "
4:   print "rdf:type owl:Restriction ;"
5:   print "owl:onProperty ", AttrW, " ;"
6:   print "owl:allValuesFrom "
7:   GenRestrMaxSize(DTypeZ,MaxSize)
8:   print "] ."
9: end function
```

Algorithm 6 Generation of a restriction to constrain the exact size of an attribute defined as a LIST aggregation data type. Function *GenRestrExactSize* from Algorithms 3 is reused.

```
1: function AttrListExact(EntV,AttrW,DTypeZ,Size)
2:   print EntV, " rdfs:subClassOf"
3:   print "[ "
4:   print "rdf:type owl:Restriction ;"
5:   print "owl:onProperty ", AttrW, " ;"
6:   print "owl:allValuesFrom "
7:   GenRestrExactSize(DTypeZ,Size)
8:   print "] ."
9: end function
```

3.6.3. `DERIVE` attributes

Since the `DERIVE` attributes (e.g. `ControlPoints` of `IfcBSplineCurve`, cf. Fragment 8) depend directly on the content and structure of other attributes, the conversion of these attributes should be paired with additional specific rules, maybe in the form of SWRL [51], that constrain their values to the attributes they depend on in order to avoid inconsistencies. However, the addition of rules to the ifcOWL was considered out of scope, because the attention is focused mainly on the declarations that are needed to support the conversion of an IFC file into an RDF graph (cf. criterion n.3 at the end of Sect.1.3). Therefore, the herein proposed conversion pattern neglects both the `DERIVE` attributes and the `WHERE` rules that can be found in some of the EXPRESS entity declarations (e.g. rule `SameDim` in `IfcBSplineCurve`, cf. Fragment 8). On the other hand, if the goal was to instantiate an RDF graph from scratch while following the ifcOWL ontology and remaining consistent with the original IFC schema, then it would be required to include these rules in the ifcOWL ontology to check the consistency of the graph.

3.6.4. The `OPTIONAL` keyword

The `IfcObject` entity (named) data type declaration (see Fragment 9) is converted in a similar fashion as shown for `IfcBSplineCurve`, even if two differences need to be mentioned. First of all, if the `OPTIONAL` keyword is used (cf. `ObjectType` attribute of `IfcObject`), then the attribute is not strictly required and an `owl:maxQualifiedCardinality` restriction should be used. This restriction then replaces the `owl:qualifiedCardinality` restriction used otherwise (see previous Fragments 37 and 38). Fragment 39 shows the declaration of the object property `ifc:ObjectType_of_IfcObject` and Fragment 40 then finally shows the application of the `owl:maxQualifiedCardinality` restriction to this object property.

```
ifc:ObjectType_of_IfcObject
  rdfs:label "ObjectType" ;
  rdfs:domain ifc:IfcObject ;
  rdfs:range ifc:IfcLabel ;
  rdf:type owl:FunctionalProperty,
      owl:ObjectProperty .
```

Fragment 39: Printout of the RDF graph representation for the object property `ifc:ObjectType_of_IfcObject` that is used to convert the entity (named) data type declaration `IfcObject` displayed earlier in Fragment 9.

```
ifc:IfcObject
  rdf:type owl:Class ;
  rdfs:subClassOf
```

```
    [
      rdf:type owl:Restriction ;
      owl:allValuesFrom ifc:IfcLabel ;
      owl:onProperty ifc:ObjectType_of_IfcObject
    ] ;
  rdfs:subClassOf
    [
      rdf:type owl:Restriction ;
      owl:maxQualifiedCardinality "1"^^xsd:
        nonNegativeInteger ;
      owl:onProperty ifc:ObjectType_of_IfcObject ;
      owl:onClass ifc:IfcLabel
    ] .
```

Fragment 40: Partial printout of the RDF graph representation for the `ifc:IfcObject` entity (named) data type declaration displayed earlier in Fragment 9 (`owl:maxQualifiedCardinality` restriction).

3.6.5. *INVERSE attributes*

We propose to convert `INVERSE` attributes (e.g. `IsDeclaredBy` attribute of `IfcObject`) as already shown for regular attributes, but with the addition of a statement expressing the inverse relation (see the conversion of `IsDeclaredBy` in Fragment 41).

```
ifc:IsDeclaredBy
  rdfs:label "IsDeclaredBy" ;
  rdfs:domain ifc:IfcObject ;
  rdfs:range ifc:IfcRelDefinesByObject ;
  owl:inverseOf
      ifc:RelatedObjects_of_IfcRelDefinesByObject ;
  rdf:type owl:FunctionalProperty,
      owl:ObjectProperty .
```

Fragment 41: Printout of the RDF graph representation for the `ifc:IsDeclaredBy` property, which parallels the `IsDeclaredBy` attribute declared in Fragment 9.

However, the outlined conversion procedure of the `INVERSE` attributes is not always safe. Indeed, there are two situations where these attributes must be ignored, because their conversion would lead to unwanted results in combination with a reasoning engine:

- An attribute has two or more `INVERSE` attributes. This is, for example, the case of attribute `RelatedDefinitions` of entity `IfcRelDeclares`. This attribute has two inverse attributes: `HasContext` of entity `IfcObjectDefinition` and `HasContext` of entity `IfcPropertyDefinition`. If all these `INVERSE` attributes were converted to object properties in ifcOWL, then a reasoning engine would infer that the two `HasContext` object properties are equivalent. Moreover, other inferences would lead to say that some classes are equivalent to `owl:Nothing`.
- A regular attribute or its `INVERSE` attribute has a `LIST` or an `ARRAY` as its range. Given the particular conversion pattern needed for ordered lists (see Sect. 3.4.1), if the `INVERSE` attributes were converted to object properties, then there would be a mismatch between the range of an object property and the domain of its inverse. Therefore, a reasoning engine would infer that the range of the object property is equal to the intersection of two disjoint classes. An example of this case is represented by attribute `Addresses` of entity `IfcPerson` and attribute `OfPerson` of entity `IfcAddress`.

As a final example, the conversion of the entity `IfcRelDefinesByObject` (cf. Fragment 10) and its attribute `RelatedObjects` shows the use of the `owl:minQualifiedCardinality` restriction to convert an attribute characterised by the use of a `SET` with a minimum size greater than zero. The conversion of the attribute `RelatedObjects` is shown in Fragment 42, and the partial conversion of entity `IfcRelDefinesByObject` is given in Fragment 43.

```
ifc:RelatedObjects_of_IfcRelDefinesByObject
  rdfs:label "RelatedObjects" ;
  rdfs:domain ifc:IfcRelDefinesByObject ;
  rdfs:range ifc:IfcObject ;
  owl:inverseOf ifc:IsDeclaredBy ;
  rdf:type owl:ObjectProperty .
```

Fragment 42: Printout of the RDF graph representation for the `ifc:Declares_of_IfcObject` property, which parallels the `Declares` attribute declared in Fragment 9.

```
ifc:IfcRelDefinesByObject
 rdf:type owl:Class ;
 rdfs:subClassOf
  [
   rdf:type owl:Restriction ;
   owl:allValuesFrom ifc:IfcObject ;
   owl:onProperty ifc:
     RelatedObjects_of_IfcRelDefinesByObject
  ] ;
  rdfs:subClassOf
  [
   rdf:type owl:Restriction ;
   owl:minQualifiedCardinality "1"^^xsd:
     nonNegativeInteger ;
   owl:onProperty ifc:
     RelatedObjects_of_IfcRelDefinesByObject ;
   owl:onClass ifc:IfcObject
  ] .
```

Fragment 43: Partial printout of the RDF graph representation for the `ifc:IfcRelDefinesByObject` entity (named) data type declaration displayed earlier in Fragment 10 (`owl:minQualifiedCardinality` restriction).

3.7. FUNCTION and RULE declarations

We did not handle FUNCTION and RULE declaration types in the conversion from the EXPRESS schema into an ifcOWL ontology, because they are most often used to restrict the content of an IFC file. In recall of our third criterion in the introduction, we aim first and foremost to use this ontology for the conversion of existing IFC files into equivalent RDF graphs, not at the creation from scratch of RDF graphs that follow the ifcOWL ontology. Since the restrictions represented by these FUNCTION and RULE declarations are normally already checked during the generation of an IFC file, then the converted RDF graphs should naturally comply with these FUNCTION and RULE declarations as well.

Nevertheless, some of these FUNCTION and RULE declarations might be converted into ontology equivalents. A proposal in this direction is made by Terkaj and Sojic [23]. Yet, most likely, the regular set of OWL class expressions will not suffice, but the use of an appropriate rule language from the semantic web domain (e.g. SWRL [51] or N3Logic [52]) might make the information still available within a semantic web context.

4. Towards the implemented software tools

4.1. Implementation of the EXPRESS to OWL converter

The proposed EXPRESS to OWL conversion pattern can be implemented in a number of different algorithms. In our research, we worked on two parallel converters, one in Java, one in C++. The Java converter does not rely on any existing RDF library for the conversion process itself. For basic checking of the consistency of the resulting ifcOWL ontology, the Java converter uses the Jena library [53]. The C++ converter makes use of the Redland C libraries [54].

The two converters provide identical ifcOWL ontologies when receiving as input the IFC4_ADD1.exp schema, thus demonstrating that the general conversion pattern is reproducible and can be implemented using different programming languages. The resulting ifcOWL ontology file can be found online, at [55], as it is produced by the C++ converter. Additionally, the source code of the Java converter is provided and maintained at [56, branch 'BS'].

Obviously, alternative conversion patterns can be defined depending on specific requirements. If the usage of highly performing reasoning engines is required in the application scenario, then an entirely different conversion pattern may be implemented, for instance excluding some of the most complex restrictions added to the ifcOWL ontology that would in this case not be used (e.g. cardinality restrictions). As a second example, if it is needed to generate RDF graphs directly from the ifcOWL ontology and not via an intermediate IFC file, then the ifcOWL must be enriched by additional rules and constraints (cf. DERIVE, WHERE, FUNCTION, RULE). Also ifcOWL ontologies in the OWL2 profiles EL, QL, or RL can be derived from the proposed ifcOWL ontology, in support of specific use cases.

4.2. Instantiating the TBox

The focus has been so far on the creation of an ifcOWL ontology from an EXPRESS schema of IFC (TBox). This is also the main contribution of this article. Yet, end users will eventually mainly use the instances following this ontology (ABox). There are a number of ways in which a TBox can be instantiated. Herein we do not intend to provide a full and exhaustive list of the options, including the advantages, disadvantages and additional remarks, but it is possible to briefly outline the key options and indicate which routines are more beneficial in each scenario. We distinguish between (1) the instantiation of the ifcOWL ontology from scratch and (2) the instantiation of the ifcOWL ontology from an original IFC-SPF (STEP Physical File).

4.2.1. ifcOWL instantiation from scratch

One of the most obvious but also work-intensive ways to instantiate an ontology is to do it from scratch, using only an OWL ontology (Tbox) as a resource. In this case, one has two options: either using an ontology editor, such as Protégé [42], or developing and using a dedicated software tool that relies on appropriate programming libraries (e.g. Jena [53] for Java and Redland [54] for C language). The first option is not recommended, because it costs a lot of work and time, which is justified for developing an ontology Tbox but not for creating the instances, and because it can easily result in errors and inconsistencies due to the manual nature of the work.

The second option is more scalable and reliable. In this case, an ontology-based application is developed to automate the procedures of parsing and generation/removal of individuals in the Abox, thus avoiding time-consuming operations to be performed manually using an ontology editor. Furthermore, sophisticated applications will be able to properly parse also the Tbox, thus retrieving the class hierarchy and the restrictions characterising each class. The analysis of the restrictions plays a key role to correctly generate new individuals and new relations between individuals, above all.

The developer of the ontology-based application can properly set the scope of the instance graph according

to the specific needs. For example, one might choose to create an RDF graph with only a dozen of the hundreds of the ifcOWL classes and properties. Alternatively, one might obviously also choose to extend the scope and build a full ifcOWL instance graph that is linked with concepts defined in other ontologies, such as a safety ontology [57], a material library ontology [58], energy performance related ontologies [59, 60, 61, 62], a built heritage ontology [63, 64, 65], or a geospatial ontology [66].

Our implementation work has not focused in this direction, but more details and examples can be found in Terkaj and Sojic [23]. In principle, as long as the ifcOWL ontology is correctly used while generating instances (using the approach outlined above), anyone should be able to parse the generated instance graph. It is thus not really necessary to agree on strict guidelines about the instantiation of the ifcOWL ontology. Nevertheless, there are a number of best practices and recommendations. As soon as the ifcOWL ontology is instantiated, it is typically recommended to be published so that others (not necessarily everyone) can access the data. In this regard, we suggest to follow the guidelines that are published for a particular case in building energy consumption by Radulovic et al. [67].

4.2.2. Conversion of IFC-SPF files into RDF graphs

The second scenario in which an ifcOWL ontology can be generated is closely tied to our third criterion. In this scenario, it is assumed that the regular AEC expert keeps working in existing BIM software for producing BIM models. These BIM models can then be exported into IFC-SPF files using regular IFC exporter plug-ins in the native software. In such a scenario, which is currently the most common in the AEC domain, IFC-SPFs are readily available for direct conversion into RDF graphs that comply with the provided ifcOWL ontology. What needs to be supplied in this case, is an out-of-the-box application that supports the submission of any IFC-SPF and returns an RDF graph that is compliant with the proposed ifcOWL ontology. Such an out-of-the-box demo application is temporarily publicly available at [68], providing the end user with an RDF graph in `TTL` and `RDF/XML` syntax after submission of the original IFC file.

5. Comparison between the output of previous converters and the proposed converter

Some of the key differences between the proposed conversion pattern and other solutions available in the literature have already been mentioned in Sect. 3. This section presents a more complete comparison by considering different criteria and aiming to outline the novel contributions. The alternative conversion patterns taken as a reference are listed below in reverse chronological order:

- the proposal by Hoang and Törmä [46], Hoang [45]
- OntoSTEP, 2009-2012 [24, 41]
- the first LDAC proposal, 2012 [43]
- the OWL/SWRL proposal by Zhao and Liu [69], 2008
- the early conversion proposal by Beetz et al. [70, 25], 2005-2009
- the early conversion proposal by Schevers and Drogemuller [38], 2005

Since the OntoSTEP tool [71] implementing the conversion pattern by Barbau et al. [24] is freely available, it is possible to run a detailed comparison between the ifcOWL generated using the OntoSTEP tool and the ifcOWL resulting from the conversion procedure herein proposed. A comparison can also be made with the ontology provided by Hoang and Törmä [46]. The comparison is reported in Table 5, where it can be noticed that our proposal is richer in terms of axioms (21306 axioms versus 17498 for Barbau et al. [24] and 11009 for Hoang and Törmä [46]. These additional axioms are mainly used for defining a larger number of inverse and functional properties, disjoint classes, property ranges, equivalent classes and individuals. This richness is actually the most important contribution of this paper, in comparison with existing approaches. By adding further restrictions and axioms, we aimed at building an ifcOWL ontology that is semantically closer to the original IFC schema in EXPRESS, in comparison to the proposals available in the literature.

Table 5 reports a quantitative evaluation of the differences between the three main ontologies considered here, but it is also useful to consider some fundamental differences between the available and documented approaches in terms of criteria and suitable application scenarios (see Table 6), and in terms of key features (see Table 7). The following subsections will delve into the key technical differences.

The line in Table 5 about DL expressivity might need a bit more explanation, although it would lead us too far to go in full detail here. More details about

Metrics	Krima et al. 2009 Barbau et al. 2012	Hoang et al. 2014	Pauwels and Terkaj 2015
Axioms	17498	11009	21306
Logical Axioms	7971	8591	13649
Classes	1348	1556	1230
Object properties	1778	854	1578
Data properties	4	9	5
Individuals	1155	1158	1627
DL expressivity	$\mathcal{ALUHN}(\mathcal{D})$	$\mathcal{ALCON}(\mathcal{D})$	$\mathcal{SROIQ}(\mathcal{D})$
SubClassOf axioms	4257	4991	4622
EquivalentClasses axioms	0	268	266
DisjointClasses axioms	0	2429	2429
SubObjectPropertyOf axioms	186	0	1
InverseObjectProperties axioms	0	0	94
FunctionalObjectProperty axioms	62	853	1441
TransitiveObjectProperty axioms	62	0	1
ObjectPropertyDomain axioms	1592	8	1577
ObjectPropertyRange axioms	174	6	1576
FunctionalDataProperty axioms	0	9	5
DataPropertyDomain axioms	7	10	5
DataPropertyRange axioms	4	10	5
ClassAssertion axioms	1627	3	1627
AnnotationAssertion axioms	5240	0	3210

Table 5: Comparison between the output of the OntoSTEP tool [71], the ontology made available by Hoang and Törmä [46] and the output of the procedure herein proposed. Statistics retrieved from Protégé software tool [42].

Criteria differences	Krima et al. 2009 Barbau et al. 2012	Hoang et al. 2014	Pauwels and Terkaj 2015
Criterium 1	general-purpose for all EXPRESS schemas	allow OWL2 DL as well as the EL, QL, RL profiles	remain first and foremost in OWL2 DL
Criterium 2	remain first and foremost in OWL2 DL	publication of flexible, less restricted linked data	stay true to original IFC schema
Most suitable application scenario	publication of all sorts of EXPRESS data as linked data and combine with other linked data sets	publication of flexible, less restricted linked data	publication of IFC data as linked data, thereby staying as close as possible to the original schema in EXPRESS

Table 6: Differences in criteria and application scenario focus between the existing conversion procedures and the procedure proposed here.

the different levels of DL expressivity can be found in Kontchakov and Zakharyaschev [73], more particularly around slide 18 of the presentation [74]. The DL expressivity ($\mathcal{ALUHN}(\mathcal{D})$, $\mathcal{ALCON}(\mathcal{D})$, $\mathcal{SROIQ}(\mathcal{D})$) captures which kind of DL statements are made in the ontology. $\mathcal{AL}$ stands for Attributive Language, and $\mathcal{ALC}$ stands for Attributive Language with Complements. Furthermore, the $\mathcal{H}$ in $\mathcal{ALUHN}(\mathcal{D})$, for example, indicates that the ontology includes role inclusions or role hierarchies (`subPropertyOf`); ($\mathcal{D}$) indicates that datatype properties, data values or data types are used; the *I* indicates that inverse properties are used; and so forth. The $\mathcal{S}$ in $\mathcal{SROIQ}$ is an abbreviation for $\mathcal{ALC}$. Instead of using these symbols to explain the differences between the three ontologies, we will directly refer to the actual type of statements made in the ontologies.

The ontology generated with the OntoSTEP tool [71], which was considered to fill in Tables 5 and 7, seems to show some differences with the conversion pattern documented in Krima et al. [41] and Barbau et al. [24]. For example, Krima et al. [41] proposed to convert `LIST` cardinality restrictions in a fashion that is similar to what we presented in Algorithms 4 and 5, but making use of `EmptyList`. However, this proposal is not implemented in the OntoSTEP tool [71].

5.1. OWL profile

The goal of obtaining an ifcOWL ontology in OWL2 DL ($\mathcal{SROIQ}(\mathcal{D})$) was defined as one of the key require-

Conversion differences	**Krima et al. 2009 Barbau et al. 2012**	**Hoang et al. 2014**	**Pauwels and Terkaj 2015**
Simple data type	option 2 in Table 2	option 3 in Table 2	option 2 in Table 2
Defined data type	OWL class	OWL class	OWL class
Defined data type as an aggregation `SET` data type	OWL class	- OWL class - subclassOf `Set` based on [72] - cardinality constraint on property `slot` [72] to define the set size	- OWL class - restriction on `owl:ObjectProperty express:hasSet` to define the list size
Defined data type as an aggregation data type `LIST` (or `ARRAY`)	OWL class	- OWL class - subclassOf `List` based on [72] - cardinality constraint on property `slot` [72] to define the list size	- OWL class - subclassOf `List` based on [47] - restrictions to define the list size (Algorithms 1,2,3)
Constructed `SELECT` *data type*	option 2 in Table 4	option 2 in Table 4	option 1 in Table 4
Constructed `ENUMERATION` *data type*	option 3 in Table 3	option 4 in Table 3	option 1 in Table 3
Entity data type	OWL class	OWL class	OWL class
Attribute of *entity data type*	- non-functional object property - property name always renamed - explicit domains, no explicit ranges - `owl:AllValuesFrom` restriction - `owl:maxCardinality` restriction	- functional object property - property name never renamed - no explicit domains and ranges - `owl:AllValuesFrom` restriction - `owl:maxCardinality` restriction	- functional object property - property name renamed if necessary - explicit domains and ranges - `owl:AllValuesFrom` restriction - `owl:qualifiedCardinality` or `owl:maxQualifiedCardinality` restriction
Attribute of *entity data type* as a `SET`	- non-functional object property - `owl:AllValuesFrom` restriction on a `Set` class	- functional object property - `owl:AllValuesFrom` restriction on subclass of `Set` - subclass of `Set` characterised by cardinality constraint on property `slot` [72] to define the set size	- non-functional object property with specified domain and range - `owl:AllValuesFrom` restriction - `owl:minQualifiedCardinality` and/or `owl:maxQualifiedCardinality` restriction or `owl:qualifiedCardinality` restriction
Attribute of *entity data type* as a `LIST` (or `ARRAY`)	- non-functional object property - `owl:AllValuesFrom` restriction on a `List` (`Array`)	- functional object property - `owl:AllValuesFrom` restriction on subclass of `List` (`Array`) - subclass of `List` (`Array`) characterised by cardinality constraint on property `slot` [72] to define the list (array) size	- functional object property with a subclass of `express:List` as its range - `owl:AllValuesFrom` restriction on subclass of `express:List` - restrictions to define the list size (Algorithms 4,5,6)
`INVERSE` attribute	N/A	N/A	- object property - `owl:inverseOf`
`DERIVE` attribute	N/A	N/A	N/A
`WHERE` rule	N/A	N/A	N/A
`FUNCTION`	N/A	N/A	N/A
`RULE`	N/A	N/A	N/A

Table 7: Fundamental differences between the existing conversion procedures and the procedure proposed here.

ments in this research and article. However, this goal was not shared by all previous efforts. For example, the research by Hoang and Törmä [46] aims at supporting the OWL EL, QL and RL profiles as well. As these

three are subsets of OWL2 DL, it should be possible to generate ifcOWL ontologies in these three last profiles as well, starting from the proposal made here. By first focusing on OWL2 DL and aiming to include as much as axioms as possible, we ensure that we have at least one 'maximal' version that is as close as possible to the original schema in EXPRESS.

Many of the earlier proposals, including Schevers and Drogemuller [38], Beetz et al. [25] aimed at an OWL DL profile, as was the case in our approach. Note that this is not the same as OWL2 DL, as OWL DL is based on $\mathcal{SHOIN}(\mathcal{D})$ and OWL2 DL is based on $\mathcal{SROIQ}(\mathcal{D})$. $\mathcal{SROIQ}(\mathcal{D})$ can hereby considered as a more expressive variant of $\mathcal{SHOIN}(\mathcal{D})$, with the $\mathcal{H}$ (role hierarchy) being subsumed by the more expressive $\mathcal{R}$ (role hierarchy, complex role inclusion axioms, reflexivity and irreflexivity, role disjointness), and with the $\mathcal{N}$ (cardinality restrictions) being subsumed by the more expressive $\mathcal{Q}$ (qualified cardinality restrictions). Many of the decisions in these early proposals are thus similar to the decisions made here, having mainly differences in the details and in a number of extensions that were not available in the earlier OWL DL.

Earlier approaches typically identified the conversion of EXPRESS simple data types (i.e. `REAL`, `INTEGER`, and so forth) into 'slots' or OWL datatype properties as problematic. As they pointed out [38], the resulting ontology would not be in OWL DL (cf. Sect. 3.5.2). Therefore, the authors adopted an alternative approach, which was also followed here (see Fragment 17), that consists in 'objectifying' the data properties and converting them into object properties with an additional value property.

Finally, the ifcOWL ontology generated according to OntoSTEP [24] comes in the OWL DL profile and its DL expressivity is defined as $\mathcal{ALUHN}(\mathcal{D})$, that is less expressive than the $\mathcal{SROIQ}(\mathcal{D})$ expressivity of the ifcOWL proposed in this paper. Also the $\mathcal{ALCON}(\mathcal{D})$ is less expressive than the $\mathcal{SROIQ}(\mathcal{D})$ targeted here. Nevertheless, and most importantly, the three types of DL expressivities outlined in Table 5 ($\mathcal{ALUHN}(\mathcal{D})$, $\mathcal{ALCON}(\mathcal{D})$, and $\mathcal{SROIQ}(\mathcal{D})$) are all within DL expressiveness (and not in OWL Full or one of the OWL DL subsets (EL, QL, RL) reported in Fig. 2).

5.2. Inverse attributes

The explicit conversion of the `INVERSE` attributes into inverse object properties is a novel contribution with respect to the previous works in the literature (cf. InverseObjectProperties axioms in Table 5 and the $\mathcal{I}$ in $\mathcal{SROIQ}(\mathcal{D})$), probably because of the associated complexity and the need of detecting unsafe conditions as highlighted in Sect.3.6.5. The availability of inverse properties gives more flexibility in the instantiation and exploration of an RDF graph based on ifcOWL. Moreover, the number of inferences that can be generated is increased. Finally, the availability of inverse properties is particularly relevant in the case of ifcOWL because it enables the generation of restrictions converting some of the `WHERE` rules [23].

5.3. Domain and range restrictions for object properties

The way in which object properties and their domain and range restrictions are represented is a very important feature of an ontology. This can be done in a number of ways, thereby defining what the ontology can eventually be used for. Interestingly, the conversion of entity attributes into object properties with domain and range restrictions is handled differently in each of the three approaches outlined in Table 5. This can be seen in the row that includes the ObjectPropertyDomain and ObjectPropertyRange axiom counts, showing values of 1592, 8, 1577 and 174, 6, 1576, respectively.

We have proposed to rename the non-unique object properties (see Fragment 35 and 36), so that each attribute can be converted into an object property with exactly one domain and one range. So, in our proposal, all domains and ranges are explicitly included, coming forth from our effort to stay as close as possible to the original IFC schema in EXPRESS (criterion n.2 in Sect.1.3). This decision explains the nearly equal number of domain and range restrictions (1577 and 1576) in Table 5 (cf. ObjectPropertyDomain and ObjectPropertyRange axioms count); the only object property without an explicit domain is `express:hasContents` (see Sect.3.4.1).

The ontology proposed by Hoang and Törmä [46], however, explicitly avoids renaming of attributes as well as the definition of domain and range (although they are included in the OWL class declarations, as was also proposed here in Fragment 37). This explains the low number (8 and 6) in Table 5. The result is an ifcOWL ontology that is very flexible (the same object property can be reused in various contexts), but not that specific since it is not possible to provide a unique definition for some of the object properties.

The proposal by Barbau et al. [24] defines the domain for all properties, but the range only for the object properties that are used to convert `LIST`, `ARRAY` and `SET` data types. All properties are always renamed, in order to allow setting one domain for each property. This results in 1592 domain definitions versus only 174 range definitions in Table 5.

5.4. *WHERE rules for defined data types*

Although we did not yet fully translate `WHERE` rule statements in EXPRESS, we prepared for such an extension by converting all simple data types in EXPRESS into `owl:Class` statements. Indeed, as `WHERE` rules in EXPRESS often act upon the simple data types referenced by any of the declared EXPRESS types or entities, most of these rules can be converted into class expressions acting upon the `owl:Class` representations of these simple types, instead of immediately having to rely on an additional rule language like SWRL or N3Logic, as is for instance suggested in [69] and [43]. Further `WHERE` rules defined for subclasses of `IfcRoot` can be converted into class expressions that are customized for the ifcOWL ontology, as proposed in Terkaj and Sojic [23].

5.5. *`OPTIONAL` statements and functional properties*

An important distinction between the RDF data model and the EXPRESS data model lies in the distinction between an open world assumption (OWA) and a closed world assumption (CWA) (see Sect. 1.2.3). This has an effect on the way in which EXPRESS `OPTIONAL` attributes should be interpreted and converted. Barbau et al. [24] suggest that *"in the case of an optional attribute, the '`ObjectAllValuesFrom`' construct is used to link the entity to the union of the attribute type and the class owl:Nothing. This solution is adopted to explicitly express the semantics of the OPTIONAL keyword: a value is not required for this attribute."*. Instead, we propose to convert any `OPTIONAL` property into a common `owl:ObjectProperty`, because *a property is by default optional in RDF*. All other properties are essentially required properties in EXPRESS (although many typically have an empty value), so they result in `owl:FunctionalProperty` statements combined with appropriate cardinality restrictions on `owl:ObjectProperty` (see Fragment 35).

Using an `owl:FunctionalProperty` statement results in a one-to-one relation between a class instance and its (functional) property. We can use this conversion option as a default, because this one-to-one relation is also the default in EXPRESS. If an attribute is meant to refer to multiple elements, it always relies on an aggregation data type declaration (`LIST`, `SET`, `BAG`, `ARRAY`). In [24], the use of the `owl:FunctionalProperty` statement is limited to the object properties that are used to convert lists, arrays and sets. This explains the considerable difference in the FunctionalObjectProperty axioms count in Table 5 (62 for Barbau et al. 2012, as opposed to 1441 for Pauwels and Terkaj 2015). Note that all object properties defined by Hoang and Törmä [46] (854) are defined as functional object properties (853), except for the property `expr:slot`, which is an object property used for the representation of `LIST`, `ARRAY`, and `SET` aggregation data types.

5.6. *`LIST` aggregation data types*

The attributes referring to a `LIST` aggregation data type in EXPRESS are essentially considered to be functional (or one-to-one), at least in the sense that the attribute links *one entity* with *one list*. This is maintained in our conversion of a `LIST` aggregation data type, thereby following criterion n.2 in the introduction (i.e. match the original EXPRESS schema as closely as possible), so we can rightfully maintain the use of an `owl:FunctionalProperty` also for required attributes referring to a `LIST`. This is an important decision, considering the options that are listed in Pauwels et al. [49] to convert `LIST` data types in EXPRESS into OWL equivalents.

Regarding the actual conversion of the `LIST` data type itself, various suggestions have already been made in the literature. The minority opts to use the common construct via `rdf:List`, `rdf:first`, and `rdf:rest`, which results in an OWL Full profile (not desired here). Schevers and Drogemuller [38] appear to rely on this construct, as well as the proposal in Pauwels and Van Deursen [43]. In order to retain an OWL DL profile, diverse variations were proposed on the theme suggested by Drummond et al. [47], as we discussed before. They aim to translate `LIST` data types into appropriate OWL class expressions, almost all of them relying on an artificially added `List` construct.

5.6.1. *Comparison with Barbau et al. [24]*

Our conversion pattern as well as the proposal by Barbau et al. [24] rely on the work by Drummond et al. [47]. There are slight differences in the actual implementation, however. For example, whereas we propose to use three additional properties only (see Fragment 21: `hasNext`, `isFollowedBy`, and `hasContents`), the proposal by Barbau et al. [24] includes more specific subproperties of these three (e.g. `array_of_real_is_followed_by`), which allows to set more strict range and domain restrictions (similar to Fragment 35 and 36). We propose to restrict domains and ranges using restrictions that are added to specific additional class declarations (see Fragment 23 and 26). This difference explains the difference in SubObjectPropertyOf axiom count in Table 5 (186 versus 0 versus

1), as well as the difference in object property count between Barbau et al. [24] (1778) and Pauwels and Terkaj 2015 (1578).

In addition, Barbau et al. [24] define the 62 subproperties of `has_next` as regular object properties; the 62 subproperties of `is_followed_by` as transitive properties; and the 62 subproperties of `has_content` as functional properties. This explains the numbers in Table 5 for FunctionalObjectProperty, TransitiveObjectProperty and SubObjectPropertyOf axiom count. Also the 174 ObjectPropertyRange axioms in Table 5 Barbau et al. [24] are all associated to the conversion of aggregation data types, since no range restrictions are added to regular properties by Barbau et al. [24].

5.6.2. Comparison with Hoang and Törmä [46]

The ontology proposed by Hoang and Törmä [46] relies on an entirely other ontology for the representation of EXPRESS aggregation data types, namely the *Ordered List Ontology (OLO)*, which was originally proposed by Abdallah and Ferris [72]. This OLO approach is a bit different from the proposal for ordered lists by Drummond et al. [47], as it includes an explicit index for each of the items in the list (`expr:slot`).

In the proposal by Hoang and Törmä [46], the statistics for Object property count (854), FunctionalObjectProperty axiom count (853), TransitiveObjectProperty axiom count (0), SubObjectPropertyOf axiom count (0), are all considerably lower. The only meaningful numbers here (Object property count (854), FunctionalObjectProperty axiom count (853)) are associated with the regular EXPRESS data types, not the aggregation data types. Indeed, the aggregation data types are all converted into `owl:Class` and `rdfs:subClassOf` constructions, for which a considerable number of restrictions is added using the two additional properties `expr:slot` and `expr:item` (which come from Abdallah and Ferris [72]). This explains the lower numbers related to property representations in Table 5 and the higher numbers related to class representations in Table 5 (class count (1556) and SubClassOf axiom count (4991)).

In conclusion, there are three proposals, each with a slightly different syntax. We propose here to generate restrictions according to the functions defined in Algorithms 1 to 6. These restrictions are partially similar to what was proposed by Krima et al. [41] and Barbau et al. [24], even though such proposal was not implemented in their converter [71]. These restrictions are considerably different from what is proposed by Hoang and Törmä [46] because they rely on the OLO ontology.

5.7. SET aggregation data types

Barbau et al. [24] and Hoang and Törmä [46] propose to convert the `SET` aggregation data type declarations using the complex constructs that are also used for `LIST` and `ARRAY` aggregation data types. This is significantly different from what we propose here. Instead, we propose to simply convert a `SET` into a non-functional object property, and appropriate cardinality restrictions are added to represent the constraints on the size of the `SET` aggregation data type. These cardinality restrictions are possible since the transitive object property `ifc:isFollowedBy` is not involved. This is similar to what is proposed in Schevers and Drogemuller [38].

5.8. Naming conventions for object properties

As reported in Barbau et al. [24], *"EXPRESS attributes are defined to be within the scope of the entity. In OWL, properties have a global scope"*. So, as soon as an attribute with the same name appears in multiple data type declarations, one has either the option to (1) assign multiple domains to this `owl:ObjectProperty` or `owl:DataTypeProperty` or to (2) rename the attribute so that it becomes unique and it is again *only* in scope of the original data type it was declared for in EXPRESS (see Fragment 8). The former option is proposed by Hoang and Törmä [46]. This explains the lower number of object properties in Table 5 for Hoang and Törmä [46] (854), as opposed to Barbau et al. [24] (1778) and our proposal (1578).

The proposal by Hoang and Törmä [46] is an exception, since most of the early proposals include the domains and ranges, as well as the latter (renaming) option (e.g. Schevers and Drogemuller [38]). Also in Barbau et al. [24], the renaming option is chosen, using the naming convention `[ClassName]_has_[PropertyName]` for all properties. We propose to turn this around, into `[PropertyName]_of_[ClassName]`, so that one immediately sees the property name when needing to instantiate this particular class. Moreover, we propose to add an additional `rdfs:label` annotation property that retains the original name of the property, which is not included in Barbau et al. [24]. Finally, we propose to rename only the properties that are not unique within one and the same schema, allowing us to stay as close as possible to the naming used in the original EXPRESS schema.

5.9. Naming conventions for enumeration individuals

The same renaming issue emerges for the individuals enumerated in an EXPRESS enumeration data type. We proposed to rename only the duplicate individuals

in an ENUMERATION (see Fragment 30). This results in 1627 named individuals, as opposed to 1155 and 1158 for Barbau et al. [24] and Hoang and Törmä [46], who both suggest to *not* rename the individuals that belong to more than one enumeration (see Fragment 32). In fact, these individuals simply belong to multiple OWL classes in the resulting ifcOWL schema.

6. Conclusions

In this article, we have looked into the conversion of EXPRESS schemas into OWL ontologies. We have specifically looked into the conversion of IFC4_ADD1.exp into an ifcOWL ontology. Such conversion procedures have been proposed before. Yet, as of now, the resulting ontologies do not appear to evolve into one referential standard or recommended ifcOWL ontology.

Therefore, we looked into the existing efforts in obtaining an ifcOWL ontology from the EXPRESS schemas of IFC, and we analysed which features would be required in order to obtain a usable and recommendable (industry-wide) ifcOWL ontology. We ended up with the following requirements or criteria for a usable and recommendable ifcOWL ontology:

1. The ifcOWL ontology should remain in OWL2 DL.
2. The ifcOWL ontology should match the original EXPRESS schema as closely as possible.
3. The ifcOWL ontology is to be used primarily to allow the conversion of IFC instance files into equivalent RDF graphs.

Following these criteria, we proposed a conversion procedure that results in an ifcOWL ontology that goes beyond the existing proposals and might indeed evolve into a recommendable version. This might in turn better support application development for construction industry relying on semantic web technologies.

A number of open issues still needs to be addressed by further research efforts:

- *RULE and FUNCTION declarations* are procedural algorithms that cannot be converted into OWL2 DL class expressions. An alternative approach needs to be used for converting these declarations.
- *WHERE rule declarations* are not included in the proposed conversion approach, although they can be converted to some extent. This is the subject of Terkaj and Sojic [23] for the specific case of ifcOWL.
- This is one of the first conversion proposals that properly handles INVERSE attributes in EXPRESS, except for a small number of exceptional cases. In a future version of the IFC schema, such exceptional cases might be avoided, which would likely be beneficial for both the EXPRESS schema and the ifcOWL ontology.
- In our conversion proposal, we have opted to convert LIST data types using additional statements (express:List, express:hasNext), after Drummond et al. [47], as was also proposed by most of previous EXPRESS to OWL conversion proposals. In the future, it might prove to be a better approach to use one of the conversion options outlined in Pauwels et al. [49].
- The current proposal suggests a 'maximal' version of the ifcOWL ontology, i.e. an ifcOWL ontology that uses all available constructs available in OWL2 DL ($\mathcal{SROIQ}(\mathcal{D})$). In a next step, this ontology should be used to generate ontologies that are in the OWL profiles with less expressiveness (DL, EL and QL), in which a lot of the work done by Hoang and Törmä [46] is to be relied upon.
- Through an *ontological analysis of IFC*, the ifcOWL ontology might be greatly simplified and it might become easier to use the IFC information. This is the subject of Borgo et al. [75] and of de Farias et al. [50].
- The EXPRESS to OWL conversion approach was built to be general purpose, even if it was tested mainly for IFC. The approach will *still need to be tested on other EXPRESS schemas* in order to make it really general purpose.
- *Exploitation of ontology-related added values*: CWA validation, OWA reasoning, extension of IFC data model and integration with other ontologies.

7. Acknowledgments

The authors would like to thank the reviewers for their great efforts, which helped to improve the article significantly. The first author gratefully acknowledges the financial support provided by the Special Research Fund (BOF) of Ghent University. The research of the second author has been partially funded by MIUR under the Italian flagship project La Fabbrica del Futuro, Subproject 2, research project Product and Process Co-Evolution Management via Modular Pallet configuration, and by the EU 7th FP under the grant agreements

No: 314156, Engineering Apps for advanced Manufacturing Engineering.

References

[1] C. M. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM handbook: a guide to building information modeling for owners, managers, architects, engineers, contractors, and fabricators, John Wiley & Sons, Hoboken, NJ, USA, 2008.

[2] T. Liebich, Y. Adachi, J. Forester, J. Hyvarinen, S. Richter, T. Chipman, M. Weise, J. Wix, Industry Foundation Classes IFC4 Official Release, available online: http://www.buildingsmart-tech.org/ifc/IFC4/final/html/index.htm (Last accessed on 21 August 2015), 2013.

[3] BuildingSMART International, BuildingSMART - International home of openBIM, available online: http://www.buildingsmart.com/ (Last accessed on 21 August 2015), 2014.

[4] International Organization for Standardization, ISO 10303-11: Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual, available online: http://www.iso.org/iso/iso_catalogue/catalogue_tc/-catalogue_detail.htm?csnumber=38047 (Last accessed on 21 August 2015), 2004.

[5] BuildingSMART International, Summary of IFC releases, available online: http://www.buildingsmart-tech.org/specifications/-ifc-releases/ (Last accessed on 21 August 2015), 2014.

[6] BuildingSMART International, PSD for IFC4 Summary, available online: http://www.buildingsmart-tech.org/specifications/pset-releases/psd-for-ifc4/psd-for-ifc4-summary (Last accessed on 21 August 2015), 2015.

[7] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific American 284 (5) (2001) 35–43.

[8] J. F. Sowa, Semantic networks, in: S. C. Shapiro (Ed.), Encyclopedia of Artificial Intelligence, John Wiley & Sons, New York, NY, USA, second edn., 1493–1511, 2092.

[9] G. Schreiber, Y. Raimond, RDF 1.1 Primer - W3C Working Group Note 24 June 2014, W3C Working Group Note, available online: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/ (Last accessed on 21 August 2015), 2014.

[10] F. Baader, W. Nutt, Basic description logics, in: F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, Cambridge, MA, USA, 47–100, 2003.

[11] M. Hennessy, The semantics of programming languages, John Wiley & Sons, Chichester, UK, 1990.

[12] D. Beckett, T. Berners-Lee, Turtle - Terse RDF Triple Language - W3C Team Submission 28 March 2011, W3C Team Submission, available online: http://www.w3.org/TeamSubmission/-turtle/ (Last accessed on 21 August 2015), 2011.

[13] T. Berners-Lee, D. Connolly, Notation 3 (N3): a readable RDF syntax - W3C Team Submission 28 March 2011, W3C Team Submission, available online: http://www.w3.org/TeamSubmission/n3/ (Last accessed on 21 August 2015), 2011.

[14] D. Brickley, R. V. Guha, RDF Schema 1.1 - W3C Recommendation 25 February 2014, W3C Recommendation, available online: http://www.w3.org/TR/rdf-schema/ (Last accessed on 21 August 2015), 2014.

[15] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, S. Rudolph, OWL 2 Web Ontology Language Primer (Second Edition) - W3C Recommendation 11 December 2012, W3C Recommendation, available online: http://www.w3.org/TR/-2012/REC-owl2-primer-20121211/ (Last accessed on 21 August 2015), 2012.

[16] D. L. McGuinness, F. van Harmelen, OWL Web Ontology Language Overview - W3C Recommendation 10 February 2004, W3C Recommendation, available online: http://www.w3.org/-TR/owl-features/ (Last accessed on 21 August 2015), 2004.

[17] W3C OWL Working Group, OWL2 Web Ontology Language Document Overview (Second Edition) - W3C Recommendation 11 December 2012, W3C Recommendation, available online: http://www.w3.org/TR/owl2-overview/ (Last accessed on 21 August 2015), 2012.

[18] W3C OWL Working Group, OWL2 Web Ontology Language Document Overview - W3C Working Draft 27 March 2009, W3C Working Draft, available online: http://www.w3.org/TR/-2009/WD-owl2-overview-20090327/. Last accessed on 21 August2015., 2009.

[19] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, OWL2 Web Ontology Language Reference Profiles (Second Edition) - W3C Recommendation 11 December 2012, W3C Recommendation, available online: http://www.w3.org/TR/-owl2-profiles/ (Last accessed on 21 August 2015), 2012.

[20] R. Brachman, H. Levesque, Knowledge Representation and Reasoning, Elsevier, 2004.

[21] J. Tao, E. Sirin, J. Bao, D. McGuinness, Extending OWL with Integrity Constraints, in: V. Haarslev, D. Toman, G. Weddell (Eds.), International Workshop on Description Logics (DL2010), vol. 573 of *CEUR Workshop Proceedings*, 137–148, 2010.

[22] H. Perez-Urbina, E. Sirin, K. Clark, Validating RDF with OWL Integrity Constraints, available online: http://docs.stardog.com/icv/icv-specification.html. Last accessed on 21 August 2015, 2012.

[23] W. Terkaj, A. Sojic, Ontology-based representation of IFC EXPRESS rules: An enhancement of the ifcOWL ontology, Automation in Construction 57 (2015) 188–201.

[24] R. Barbau, S. Krima, S. Rachuri, A. Narayanan, X. Fiorentini, S. Foufou, R. D. Sriram, OntoSTEP: Enriching product model data using ontologies, Computer-Aided Design 44 (6) (2012) 575–590.

[25] J. Beetz, J. Van Leeuwen, B. de Vries, IfcOWL: a case of transforming EXPRESS schemas into ontologies, Artificial Intelligence for Engineering Design, Analysis and Manufacturing 23 (1) (2009) 89–101.

[26] C. Bizer, T. Heath, T. Berners-Lee, Linked data - the story so far, International Journal on Semantic Web and Information Systems 5 (3) (2009) 1–22.

[27] C. Bizer, A. Jentzsch, R. Cyganiak, State of the LOD cloud, available online: http://lod-cloud.net/state/ (Last accessed on 21 August 2015), 2011.

[28] M. Schmachtenberg, C. Bizer, H. Paulheim, State of the LOD cloud 2014, available online: http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/ (Last accessed on 21 August 2015), 2014.

[29] R. Cyganiak, A. Jentzsch, The linking open data cloud diagram, available online: http://lod-cloud.net/ (Last accessed on 21 August 2015), 2014.

[30] P. Pauwels, D. Van Deursen, R. Verstraeten, J. De Roo, R. De Meyer, R. Van de Walle, J. Van Campenhout, A semantic rule checking environment for building performance checking, Automation in Construction 20 (5) (2011) 506–518.

[31] S. Abdul-Ghafour, P. Ghodous, B. Shariat, E. Perna, A common design-features ontology for product data semantics interoperability, in: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, 443–446, 2007.

[32] A. Yurchyshyna, C. F. Zucker, N. Le Thanh, C. Lima, A. Zarli,

Towards an ontology-based approach for conformance checking modelling in construction, in: Proceedings of the 24th CIB W78 Conference, 195–202, 2007.
[33] A. Yurchyshyna, A. Zarli, An ontology-based approach for formalisation and semantic organisation of conformance requirements in construction, Automation in Construction 18 (8) (2009) 1084–1098.
[34] B. Kadar, W. Terkaj, M. Sacco, Semantic Virtual Factory supporting interoperable modelling and evaluation of production systems, CIRP Annals - Manufacturing Technology 62 (1) (2013) 443–446, ISSN 0007-8506.
[35] W. Terkaj, T. Tolio, M. Urgo, A virtual factory approach for in situ simulation to support production and maintenance planning, CIRP Annals - Manufacturing Technology 64 (1) (2015) 451–454.
[36] S. Törmä, Semantic Linking of Building Information Models, in: Proceedings of the Seventh IEEE International Conference on Semantic Computing, IEEE Computer Society, 412–419, 2013.
[37] S. Törmä, Web of building data–integrating IFC with the Web of Data, in: A. Mahdavi, B. Martens, R. Scherer (Eds.), eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014, CRC Press, 141–147, 2014.
[38] H. Schevers, R. Drogemuller, Converting the Industry Foundation Classes to the Web Ontology Language, in: Proceedings of the First International Conference on Semantics, Knowledge and Grid, IEEE Computer Society, Washington, DC, 556–560, 2005.
[39] L. Zhang, R. R. Issa, Development of IFC-based construction industry ontology for information retrieval from IFC Models, in: Proceedings of the 2011 eg-ice Workshop, University of Twente, The Netherlands, July, 6–8, 2011.
[40] C. Agostinho, M. Dutra, R. Jardim-Goncalves, P. Ghodous, A. Steiger-Garcao, EXPRESS to OWL morphism: making possible to enrich ISO10303 Modules, in: G. Loureiro, R. Curran (Eds.), Complex Systems Concurrent Engineering, Springer London, 391–402, 2007.
[41] S. Krima, R. Barbau, X. Fiorentini, R. Sudarsan, R. Sriram, OntoSTEP: OWL-DL ontology for STEP, National Institue of Standards and Technology, NISTIR 7561.
[42] H. Knublauch, R. W. Fergerson, N. F. Noy, M. A. Musen, The Protégé OWL plugin: An open development environment for semantic web applications, in: The Semantic Web–ISWC 2004, Springer, 229–243, 2004.
[43] P. Pauwels, D. Van Deursen, IFC/RDF: adaptation, aggregation and enrichment, in: Report of the First International Workshop on Linked Data in Architecture and Construction, Ghent, Belgium, 2–5, 2012.
[44] W. Terkaj, G. Pedrielli, M. Sacco, Virtual Factory Data Model, in: Workshop on Ontology and Semantic Web for Manufacturing OSEMA 2012, CEUR Workshop Proceedings, vol. 886, 29–43, 2012.
[45] N. V. Hoang, IFC-to-Linked Data Conversion: Multilayering Approach, in: the Third International Workshop on Linked Data in Architecture and Construction, Eindhoven, Netherlands, 2015.
[46] N. V. Hoang, S. Törmä, Opening BIM to the Web IFC-to-RDF Conversion Software, available online: http://rym.fi/results/opening-bim-to-the-web-ifc-to-rdf-conversion-software/ (Last accessed on 21 August 2015), 2014.
[47] N. Drummond, A. Rector, R. Stevens, G. Moulton, M. Horridge, H. Wang, J. Sedenberg, Putting OWL in Order: Patterns for sequences in OWL, in: OWL Experiences and Directions (OWLEd 2006), 2006.
[48] M. Dean, G. Schreiber, OWL Web Ontology Language Reference - W3C Recommendation 10 February 2004, W3C Recommendation, available online: http://www.w3.org/TR/owl-ref/ (Last accessed on 21 August 2015), 2004.
[49] P. Pauwels, W. Terkaj, T. Krijnen, J. Beetz, Coping with lists in the ifcOWL ontology, in: Proceedings of the 22nd EG-ICE International Workshop, Eindhoven, Netherlands, 113–122, 2015.
[50] T. de Farias, A. Roxin, C. Nicolle, IfcWoD, Semantically Adapting IFC Model Relations into OWL Properties, in: the Third International Workshop on Linked Data in Architecture and Construction, Eindhoven, Netherlands, 2015.
[51] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al., SWRL: A semantic web rule language combining OWL and RuleML, W3C Member submission 21 (2004) 79.
[52] T. Berners-Lee, Notation 3 Logic: An RDF language for the semantic web, available online: http://www.w3.org/DesignIssues/Notation3.html (Last accessed on 21 August 2015), 2005.
[53] Apache, Apache Jena, available online: https://jena.apache.org/ (Last accessed on 21 August 2015), 2015.
[54] D. Beckett, Redland RDF Libraries, available online: http://librdf.org/ (Last accessed on 21 August 2015), 2015.
[55] W. Terkaj, P. Pauwels, OWL ontology file for the IFC4_ADD1.exp EXPRESS schema, available online: http://linkedbuildingdata.net/resources/20150824_IFC4_ADD1.owl (Last accessed on 21 August 2015), 2015.
[56] P. Pauwels, Implementation of IFC-to-RDF conversion by Ghent University (Multimedia Lab - SmartLab) and Aalto University, available online: https://github.com/mmlab/IFC-to-RDF-converter/tree/BS (Last accessed on 21 August 2015), 2015.
[57] S. Zhang, F. Boukamp, J. Teizer, Ontology-based semantic modeling of construction safety knowledge: Towards automated safety planning for job hazard analysis (JHA), Automation in Construction 52 (2015) 29–41.
[58] G. Costa, L. Madrazo, Connecting building component catalogues with BIM models using semantic technologies: an application for precast concrete components, Automation in Construction 57 (2015) 239248.
[59] H. Wicaksono, P. Dobreva, P. Häfner, S. Rogalski, Ontology development towards expressive and reasoning-enabled building information model for an intelligent energy management system, in: Proceedings of the 5th International Conference Knowledge Engineering and Ontology Development, 38–47, 2013.
[60] M. Kadolsky, K. Baumgärtel, R. Scherer, An Ontology Framework for Rule-based Inspection of eeBIM-systems, Procedia Engineering 85 (2014) 293301.
[61] K. Baumgärtel, M. Kadolsky, R. Scherer, An ontology framework for improving building energy performance by utilizing energy saving regulations, in: A. Mahdavi, B. Martens, R. Scherer (Eds.), ECPPM2014: eWork and eBusiness in Architecture, Engineering and Construction, CRC Press, 519526, 2014.
[62] E. Corry, P. Pauwels, S. Hu, M. Keane, J. O'Donnell, A performance assessment ontology for the environmental and energy management of buildings, Automation in Construction 57 (2015) 249259.
[63] S. Cursi, D. Simeone, I. Toldo, A Semantic Web Approach for Built Heritage Representation, in: G. Celani, D. Sperling, J. Franco (Eds.), Computer-Aided Architectural Design: The Next City - New Technologies and the Future of the Built Environment (CAADFutures 2015), vol. 527 of *Communications in Computer and Information Science*, Springer, 383–401, 2015.
[64] D. Di Mascio, P. Pauwels, R. De Meyer, Improving the knowledge and management of the historical built environment with BIM and ontologies: the case study of the Book Tower, in: Pro-

ceedings of the 13th International Conference on Construction Applications of Virtual Reality, 427–436, 2013.

[65] P. Pauwels, R. Bod, D. Di Mascio, R. De Meyer, Integrating building information modelling and semantic web technologies for the management of built heritage information, in: Proceedings of the Digital Heritage International Congress (DigitalHeritage), 481–488, 2013.

[66] E. Karan, J. Irizarry, J. Haymaker, BIM and GIS Integration and Interoperability Based on Semantic Web Technology, Journal of Computing in Civil Engineering doi:10.1061/(ASCE)CP.1943-5487.0000519.

[67] F. Radulovic, M. Poveda-Villalón, D. Vila-Suero, A. G.-P. Víctor Rodríguez-Doncel, Raúl García-Castro, Guidelines for Linked Data generation and publication: An example in building energy consumption, Automation in Construction 57 (2015) 178–187.

[68] P. Pauwels, IFC repository, available online: http://smartlab1.elis.ugent.be:8889/IFC-repo/. Last accessed on 21 August 2015, 2015.

[69] W. Zhao, J. Liu, OWL/SWRL representation methodology for EXPRESS-driven product information model: Part I. Implementation methodology, Computers in Industry 59 (2008) 580589.

[70] J. Beetz, J. van Leeuwen, B. de Vries, An ontology web language notation of the industry foundation classes, in: Proceedings of the 22nd CIB W78 Conference on Information Technology in Construction, 193–198, 2005.

[71] R. Sudarsan, R. Barbau, S. Krima, OntoSTEP Plugin, available online: http://www.nist.gov/el/msid/ontostep.cfm (Last accessed on 21 August 2015), 2010.

[72] S. A. Abdallah, B. Ferris, The Ordered List Ontology 0.72 - Namespace Document 23 July 2010, available online: http://smiy.sourceforge.net/olo/spec/orderedlist-ontology.html (Last accessed on 21 August 2015), 2010.

[73] R. Kontchakov, M. Zakharyaschev, An Introduction to Description Logics and Query Rewriting, in: Reasoning Web - Reasoning on the Web in the Big Data Era, Springer, 195–244, 2014.

[74] R. Kontchakov, M. Zakharyaschev, An Introduction to Description Logics and Query Rewriting, available online: http://rw2014.di.uoa.gr/sites/default/files/slides/An_Introduction_to_Description_Logics.pdf (Last accessed on 21 August 2015), 2014.

[75] S. Borgo, E. M. Sanfilippo, A. Sojic, W. Terkaj, Ontological Analysis and Engineering Standards: an initial study of IFC, in: V. Ebrahimipour (Ed.), Ontology Modeling in Physical Asset Integrity Management, Springer, 17–43, 2015.

P. Pauwels, W. Terkaj (2016) EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology.
Automation in Construction, Volume 63, Pages 100-133, ISSN 0926-5805.
https://doi.org/10.1016/j.autcon.2015.12.003
(https://www.sciencedirect.com/science/article/pii/S0926580515002435)