# Implementation and Experiments with an IFC-to-Linked Data Converter

Nam Vu Hoang, nam.vuhoang@aalto.fi
*Aalto University, School of Science, Finland*

Seppo Törmä, seppo.torma@aalto.fi
*Aalto University, School of Science, Finland*

**Abstract**
The Linked Data technologies enable the sharing of building data in a manner compliant with the fragmented and variable nature of construction projects. Linked Data allows distributed online publication of structural and semantic data as well as cross-dataset linking and granular access to objects. This paper presents an open-source Java application for converting IFC schemas from EXPRESS into OWL2 ontologies and IFC data from STEP physical file format (SPFF) into RDF graphs aligned with the ontologies. The converter allows users to choose different target OWL2 profiles, various RDF serialization formats, and especially different levels of type information. By using preset settings, it is possible to produce a sequence of multilayered ifcOWL ontologies where each subsequent ontology is a superset of its predecessors and includes all layers of them. Thus, if IFC data is converted from a STEP file into an RDF graph using one of the preset settings, the resulting graph is compatible with any ontology in that sequence. That allows users to choose the most appropriate amount of type information to use even after converting data. Advanced users can also customize conversion parameters and include pre- or post-conversion plugins, for instance, to filter or modify data. The paper includes experimental results on the characteristics of different ontologies and related computational tasks.

**Keywords:** IFC, EXPRESS, SPFF, OWL2, RDF

## 1 Introduction

The volume of building-related data is growing, but the datasets produced over the lifecycle of a building, unfortunately, are fragmented into separate information silos. Their utilization is hampered by the problems to access them and to integrate interrelated data across datasets. The authors in (Beetz 2009a, Beetz et al 2009b, Pauwels et al 2011, Törmä 2013, Törmä 2014) proposed solutions to address these problems by making building data available on the Web using the Web of Data technologies, i.e., Semantic Web and Linked Data. The potential benefits are significant:

- the data about individual building objects could be accessed in a granular fashion;
- the online access brings the same, most recent version of the data available for all parties;
- data management can be distributed according to the structure of any project consortium;
- the semantic description of data enables the automation of workflows and reasoning tasks;
- and possibilities of cross-dataset linking allow the integration of data from multiple datasets based on the needs of use cases.

A crucial enabler of this vision is conversion of building data from existing formats to the Web of Data representations. This research focuses on BIM models represented according to the IFC standard. We present the IFC2LD converter that can translate: (1) the IFC schema (represented in EXPRESS language) to an ontology represented in OWL (Web Ontology Language), and (2) IFC data to RDF (Resource Description Framework), a representation of structural data on the Web. The latter of the conversions is relatively straightforward, whereas the former is more complicated due to the mismatch of representational mechanisms EXPRESS and OWL. Moreover, OWL can support different

kinds of reasoning tasks. However, the use of more expressive representational primitives of OWL, increases the computational complexity of reasoning.

Consequently, the conversion of IFC schema to OWL is designed in a multilayered fashion. In this context, there is a sequence of three ontology layers, where each subsequent contains more type information of the IFC objects than its predecessor. All these ontology layers are compatible with essentially the same IFC-derived RDF data. Depending on the needs of an application, any of the ontology layers can be used with the same data.

In this paper, we describe the nature of the multilayer conversion, the implementation of the IFC2LD converter, the validation of the ontology layers and datasets, and the empirical experiments with the converter. In Section 2, we explain the theoretical basis of the conversion. In Section 3, we introduce our IFC2LD converter. In Section 4, we describe the validation of and experiments with the converter. We end with conclusions and topics of future research in Section 5.

## 2 Theoretical Basis for Generating ifcOWL Ontologies and ifcRDF Datasets

This section shortly describes the fundamental principles of translating: IFC-EXPRESS schemas into ifcOWL ontologies and IFC-SPFF data files into ifcRDF datasets.

### 2.1 Basic Requirements

The resulting ifcOWL ontology should contain *at least important information* about the original IFC schema that is required for particular practical tasks such as querying, linking, validating, and reasoning. It cannot include functions, rules, type constraints, and derived attributes specified in the IFC schema after keywords `FUNCTION`, `RULE`, `WHERE`, and `DERIVE` because of their procedural nature and the complexity of the resulting constraints. At least, the ifcOWL ontology must be compatible with as many reasoners and other Semantic tools as possible.

Naturally, the derived ifcRDF datasets must be compatible with the corresponding ifcOWL ontologies. They should contain *all information* so that they could be converted back into IFC data files. However, so far we have not encountered use cases that would require one party to modify the data produced by others. Therefore, we focus on scenarios where ifcRDF datasets are *read-only*.

All constructs in ifcOWL and ifcRDF, related to EXPRESS, STEP or IFC specifications, belong to separate namespaces with prefixes `expr:`, `step:`, and `ifc:` respectively.

### 2.2 ifcOWL Ontology Layers and Their Contents

Ideally, for each version of the IFC schema there would be just one ifcOWL ontology that contains all necessary information about IFC datatypes. However, there are possibilities and reasons to convert the IFC schema to OWL in several different ways. Firstly, a full conversion of the IFC schema produces a huge ontology, which is difficult to understand or manipulate and may reduce the reasoning performance. Secondly, a simple version of the ifcOWL ontology is useful to support ontology reasoning or inexact reasoning by using tools that are compatible only with OWL 2 EL or OWL 2 QL profiles, such as CB, CEL, or ELK[1]. Thirdly, it is unnecessary to include all constraints to ontologies, as the ifcRDF datasets are read-only and generated by the converter from practically correct IFC data files; therefore, they do not require extra validation or checking against the constraints.

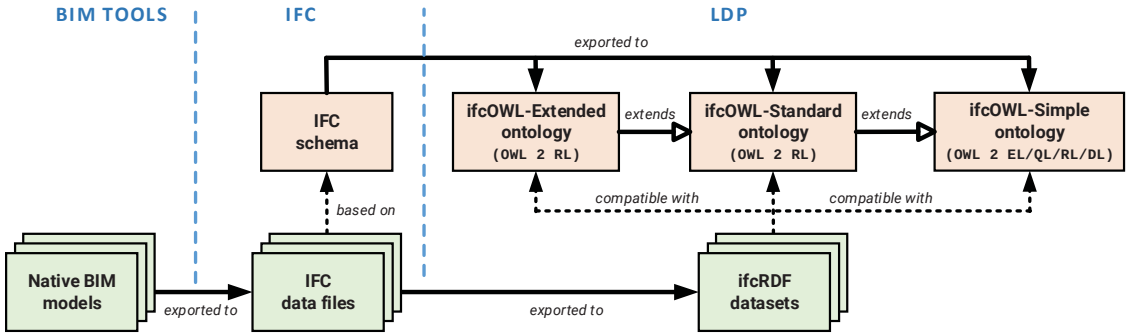


**Figure 1** Producing IFC ontologies and datasets for use in Linked Data Platform

[1] OWL implementations: http://www.w3.org/2001/sw/wiki/OWL/Implementations

Hence, we convert each IFC schema into a flexible multilayer ifcOWL ontology in which ifcRDF datasets are compatible with *any* of the layers (Figure 1). The more comprehensive levels provide increasingly detailed type information about the same ifcRDF data (Table 1). The ifcOWL-Simple[2] layer has been designed to be compatible with any of OWL 2 EL, OWL 2 QL, OWL 2 RL, and OWL 2 DL profiles in order to support a maximum variety of Semantic Web tools. The ifcOWL-Standard layer is compatible with OWL 2 RL and OWL 2 DL, whereas ifcOWL-Extended can be used only with OWL 2 DL (or OWL 2 Full).

**Table 1** Contents of ifcOWL layers

| Layer | ifcOWL-Simple | ifcOWL-Standard | ifcOWL-Extended |
|---|---|---|---|
| Compatible OWL profiles | OWL 2 EL/QL/RL/DL | OWL 2 RL/DL | OWL 2 DL |
| Ontology metadata | + | + | + |
| Basic constructs | + | + | + |
| Type definitions | + | + | + |
| Type inheritance relations | + | + | + |
| Property definitions | + | + | + |
| Property value ranges | | + | + |
| Keys of entity classes | | + | + |
| Property cardinalities | | | + |

## 2.3 Converting IFC Schemas into ifcOWL Ontology Layers

An EXPRESS schema includes six groups of datatypes, namely *simple*, *entity*, *enumeration*, *defined*, *select*, and *aggregation* types. In the following, we call *classes* in ifcOWL that are equivalent to these types by the same names, e.g., *defined class*, *entity class*, *select class*. Each group of classes should have a common superclass, such as `expr:Defined`, `expr:Enum`, `expr:Select`, `expr:Entity`, and `expr:Collection`.

**Simple datatypes** are primitive datatypes built in EXPRESS, divided into two groups: logical and non-logical. Only non-logical types have similar XSD datatypes that are: a) supported by all OWL 2 profiles (Motik et al 2014); b) listed among preferred datatypes for Linked Data Platform (Burleson et al 2014). The non-logical datatypes are `STRING` (similar to `xsd:string`), `BINARY` (`xsd:hexBinary`), `INTEGER` (`xsd:integer`), `REAL`, and `NUMBER` (both – `xsd:decimal`).

Logical datatypes, i.e. `BOOLEAN` and `LOGICAL`, are equivalent to enumeration classes, because `LOGICAL` has three predefined values, i.e., `TRUE`, `FALSE`, `UNKNOWN`; and type `xsd:boolean` is unsupported by OWL 2 EL/QL.

Since IFC mainly uses simple datatypes for declaring as base types of defined datatypes, it makes sense to declare them in the same way as defined datatypes (Listing 1).

**Listing 1** Declaration of simple class `expr:REAL` and defined class `ifc:IfcLengthMeasure` (all layers)

```
expr:REAL a owl:Class ; rdfs:subClassOf expr:Defined .
expr:hasReal a owl:DatatypeProperty , owl:FunctionalDataProperty ;
  rdfs:domain expr:REAL ;  rdfs:range xsd:decimal .
ifc:IfcLengthMeasure a owl:Class ; rdfs:subClassOf expr:REAL .

```

**Enumeration datatypes** represent fixed sets of possible string values. An *enumeration class* in ifcOWL is `owl:Class`, which have some predefined named individuals. In each ifcOWL layer, the declaration of the individuals depends on whether the compatible OWL 2 profiles support axiom `ObjectOneOf` with multiple individuals (Listing 2, a-b). The fact that some individuals, for instance, `ifc:NOTDEFINED`, belong to different enumeration classes should not cause any problems as enumeration values are always assigned to entity attributes with obviously declared types.

**Listing 2a** Declaration of enumeration class `ifc:IfcAssemblyPlaceEnum` (Simple and Standard layers)

```
    ifc:IfcAssemblyPlaceEnum a owl:Class ; rdfs:subClassOf expr:Enum .
    ifc:SITE, ifc:FACTORY, ifc:NOTDEFINED a ifc:IfcAssemblyPlaceEnum .
```

[2] ifcOWL ontology layer files for IFC 2x3 TC1, IFC 4, IFC 4 ADD1 are available at http://drumbeat.cs.hut.fi/owl

**Listing 2b** Declaration of enumeration class `ifc:IfcAssemblyPlaceEnum` (Extended layer)

```
ifc:IfcAssemblyPlaceEnum a owl:Class ; rdfs:subClassOf expr:Enum ;
     owl:oneOf ( ifc:SITE ifc:FACTORY ifc:NOTDEFINED ) .
```

**Select datatypes** represent unions of other datatypes. A *select class* in ifcOWL represents an `owl:Class` with many subclasses. In each ifcOWL layer, the declaration of union-member classes as subclasses depends on whether all compatible OWL profiles support axiom `ObjectUnionOf` and whether there are more than one union-member (Listing 3, a-b).

**Listing 3a** Declaration of select class `ifc:IfcTrimmingSelect` (Simple and Standard layers)

```
ifc:IfcTrimmingSelect a owl:Class ; rdfs:subClassOf expr:Select .
ifc:IfcCartesianPoint, ifc:IfcParameterValue rdfs:subClassOf ifc:IfcTrimmingSelect .
```

**Listing 3b** Declaration of select class `ifc:IfcTrimmingSelect` (Extended layer)

```
ifc:IfcTrimmingSelect a owl:Class ; rdfs:subClassOf expr:Select ;
  owl:unionOf ( ifc:IfcCartesianPoint ifc:IfcParameterValue ) .
```

**Defined (named) datatypes** are similar to simple datatypes, but with value constraints (ignored by the converter) and meaningful *names*, which are important in IFC. Each defined type is a subset (or an equivalent set) of a simple type or another defined type. For instance, defined type `IfcPositiveLengthMeasure` is a subset of defined type `IfcLengthMeasure` that is equal to simple type `REAL`. A *defined class* is an `owl:Class` that has only one property named in format `expr:hasXXX`, where `XXX` is the base simple datatype's name written in Pascal case (e.g., `hasString`, `hasReal`) (Listing 1).

IFC uses type `IfcTimeStamp`, which is equivalent to `INTEGER` and represents Unix time values, because there is no datetime type built in EXPRESS. However, this type is converted into class `ifc:IfcTimeStamp` that has property `xsd:hasDateTime` with type `xsd:dateTime`, supported by all OWL 2 profiles.

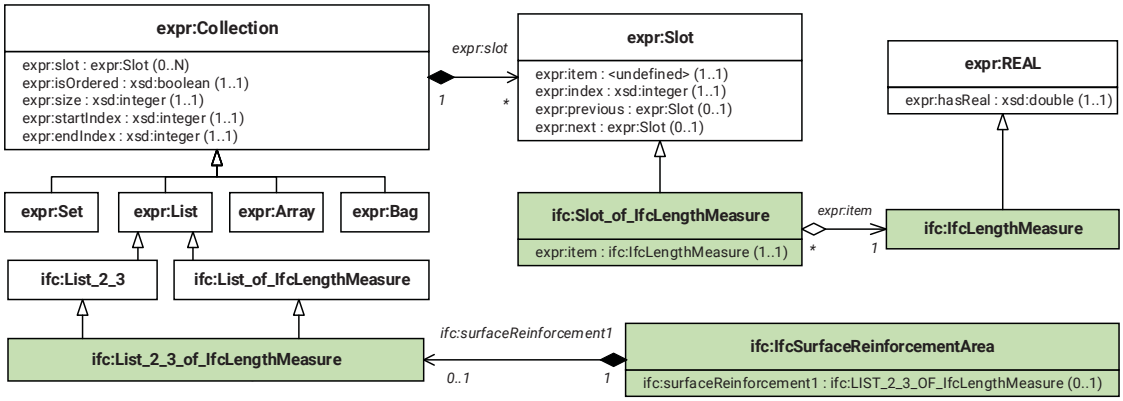


**Figure 2** Class diagram of a fragment of ifcOWL

**Aggregated datatypes** are four different collection types (`ARRAY`, `LIST`, `SET`, and `BAG`) that are specified as ordered or unordered, with start/end indexes or with min/max cardinalities. All collection classes are adapted to the one common design template (Figure 2). An ordinary aggregated class must have two super aggregated classes: one restricts the number of slots and another restricts the slot type (`ifc:List_2_3` and `ifc:Lis_of_IfcLengthMeasure` in Listing 4, a-c). Both are subclasses of either class `expr:Set`, `expr:List`, `expr:Array`, or `expr:Bag`. Their common superclass `expr:Collection` has one multiple property `expr:slot` with type `expr:Slot`. Class `expr:Slot` in its turn wraps the actual item value in property `expr:item` (Listing 4, b).

**Listing 4a** Declaration of aggregated class `ifc:List_2_3_of_IfcLengthMeasure` (all layers)

```
ifc:List_2_3_ OF_IfcLengthMeasure a owl:Class ;
     rdfs:subClassOf ifc:List_2_3 , ifc:List_of_IfcLengthMeasure .
ifc:List_2_3 , ifc:List_of_IfcLengthMeasure a owl:Class ; rdfs:subClassOf expr:List .
ifc:Slot_of_IfcLengthMeasure a owl:Class ; rdfs:subClassOf expr:Slot .
```

**Listing 4b** Additional property constraints of aggregated superclasses (Simple, Extended layers)

```
ifc:List_of_IfcLengthMeasure rdfs:subClassOf [ a owl:Restriction ;
    owl:onProperty expr:slot ; owl:allValuesFrom ifc:Slot_of_IfcLengthMeasure ] .
ifc:Slot_of_IfcLengthMeasure rdfs:subClassOf [ a owl:Restriction ;
    owl:onProperty expr:item ; owl:allValuesFrom ifc:IfcLengthMeasure ] .
```

**Listing 4c** Additional property constraints of aggregated superclasses (Extended layers)

```
ifc:List_2_3 rdfs:subClassOf
  [ a owl:Restriction; owl:onProperty expr:slot ; owl:minCardinality "2"^^xsd:integer ],
  [ a owl:Restriction; owl:onProperty expr:slot ; owl:maxCardinality "3"^^xsd:integer ].
```

**Entity datatypes** are similar to classes in object-oriented languages. All entity attributes are converted into single functional object properties. If an attribute is optional then its minimum cardinality is `0`. If the type of an attribute is a set, a list or an array then additionally aggregated classes, e.g., class `ifc:List_2_3_of_IfcLengthMeasure`, must be created (Figure 2) (Listing 5, a-c).

**Listing 5a** Declaration of entity class `ifc:IfcSurfaceReinforcementArea` (all layers)

```
ifc:IfcSurfaceReinforcementArea a owl:Class ; rdfs:subClassOf expr:Entity .
ifc:shearReinforcement , ifc:surfaceReinforcement1 a owl:ObjectProperty .
```

**Listing 5b** Additional declaration of entity class `ifc:IfcSurfaceReinforcementArea` (Standard, Extended layers)

```
 ifc:IfcSurfaceReinforcementArea
     rdfs:subClassOf [ a owl:Restriction; owl:onProperty ifc:shearReinforcement ;
         owl:allValuesFrom ifc:IfcRatioMeasure ] ;
     rdfs:subClassOf [ a owl:Restriction; owl:onProperty ifc:surfaceReinforcement1;
         owl:allValuesFrom ifc: List_2_3_of_IfcLengthMeasure ] .
 ifc:shearReinforcement , ifc:surfaceReinforcement1 a owl:FunctionalProperty .
```

**Listing 5c** Additional declaration of entity class `ifc:IfcSurfaceReinforcementArea` (Extended layer)

```
ifc:IfcSurfaceReinforcementArea
    rdfs:subClassOf [ a owl:Restriction; owl:onProperty ifc:shearReinforcement ;
        owl:minCardinality "0"^^xsd:integer ] ;
    rdfs:subClassOf [ a owl:Restriction; owl:onProperty ifc:shearReinforcement ;
        owl:maxCardinality "1"^^xsd:integer ] ;
    rdfs:subClassOf [ a owl:Restriction; owl:onProperty ifc:surfaceReinforcement1 ;
        owl:minCardinality "0"^^xsd:integer ] ;
    rdfs:subClassOf [ a owl:Restriction; owl:onProperty ifc:surfaceReinforcement1 ;
        owl:maxCardinality "1"^^xsd:integer ] .
```

Inverse attributes in EXPRESS are different to inverse properties in OWL. In OWL, if property `p1` is `owl:inverseOf` property `p2`, then assertion `p1(x,y)` implies `p2(y,x)`, where `x` and `y` are individuals. Whereas, assertion `p1(x, list(`$y_1$`, `$y_2$`, ... , `$y_n$`))` in EXPRESS implies assertions `p2(`$y_i$`, x)` for each `i` from `1` to `n`. For instance, attribute `IfcTableRow.OfTable` is inverse to attribute `IfcTable.Rows`, the type of which is a `LIST[1..?] OF IfcTableRow`. Therefore, ifcOWL ontologies must declare inverse attributes as normal object properties (Listing 6a, 6b). Moreover, OWL 2 EL and OWL 2 QL do not support axiom `ObjectInverseProperty` at all.

**Listing 6a** Declaration of inverse property `ifc:ofTable` (all layers)

```
ifc:ofTable a owl:ObjectProperty, expr:InverseProperty .
```

**Listing 6b** Declaration of inverse property `ifc:ofTable` (Standard, Extended layers)

```
ifc:IfcTableRow rdfs:subClassOf [ a owl:Restriction;
    owl:onProperty ifc:ofTable ; owl:allValuesFrom ifc:Table ] .
```

## 2.4 Converting IFC Data into ifcRDF Datasets

By default, the IFC-SPFF-to-RDF converter produces ifcRDF datasets *without information loss* that are compatible with *any* ifcOWL layer. However, users often want to choose alternative solutions, which fit their specific requirements better. For example, when the Sematic tools used support OWL 2 RL or OWL 2 DL, there is no need to name every individual, it is also possible to use type `xsd:double` as equivalent to datatype `REAL` (instead of `xsd:decimal` by default). In addition, filtering redundant data, for instance, inverse properties or geometric representations, results in reducing the dataset sizes and improving reasoning performance.

**The header section** consists of entities of defined in STEP types `FileDescription`, `FileName`, and `FileSchema`. The converter reads them to generate STEP-specific metadata for the ifcRDF dataset and duplicate some of them as Dublin Core metadata annotations[3] (Listing 7).

**Listing 7** A fragment of the header section of a SPF

```
<http://example.org>  a void:DataSet ;
   dcterms:created "2015-02-19T13:22:15Z"^^xsd:dateTime ;
   dcterms:creator "John Smith"^^xsd:string ;
   step:fileName  :MetaData_FileName .
:MetaData_FileName a step:FileName ;
   step:author "John Smith"^^xsd:string ;
   step:originating_system "Tekla Structures 20.1"^^xsd:string ;
   step:time_stamp "2015-02-19T13:22:15Z"^^xsd:dateTime  .
```

**The data section** consists of entities, each has a local unique number and a list of attributes. The converter must translate such attributes and their inverse attributes (if any) into object properties according to the type structure defined in ifcOWL layers (Listing 8).

All entities of types derived from `IfcRoot` have global identifiers. They are converted into IRI resources with format `GUID_<guid>`; where `guid` is in the hexadecimal string representation (not including character `'$'`). Because OWL 2 EL and OWL 2 QL do not support anonymous individuals, all objects standing for other entities or entity attributes also must have IRIs defined in some format, such as `ENTITY_<entityNumber>`, `ATTR_<entityNumber>_<level1Count>_<level2Count>_...` (Listing 8). Note that these names are unique only inside one dataset and are changeable from version to version, thus using them for building linksets is strongly not recommended.

**Listing 8** A fragment of the data section of a SPF

```
:GUID_1916794F-5605-4499-9AB0-F155DE8D3B6C            # entity as an URI
   a ifc:IfcPropertySet ;                             # entity type
   ifc:globalId [ a ifc:IfcGloballyUniqueId ;         # property with defined type
       expr:hasString "0P5dbFLWL4cPgmyLNUZJji"^^xsd:string ] ;
   ifc:hasProperties  :ATTR_111_2 .                   # property w/ aggregated type
:ATTR_111_2 a ifc:Set_1_N_of_IfcProperty ;
   expr:size "3"^^xsd:integer ;
   expr:itemType ifc:IfcProperty ;
   expr:slot :ATTR_111_2_1 , :ATTR_111_2_2 , :ATTR_111_2_3 .
:ATTR_111_2_1 a ifc:Slot_of_IfcProperty ;
   expr:item :ATTR_111_2_1_1 ;
:ATTR_111_2_1_1 a ifc:IfcPropertySingleValue ;
       ifc:name "initial_GUID"^^xsd:string .
```

Currently, the IFC2LD converter allows users to choose:

- including inverse properties into ifcRDF datasets or ignoring them;
- converting unnamed entities into anonymous individuals (allowed in OWL 2 RL/DL) or naming them with local entity numbers (Listing 8);
- translating `REAL` values into literals with the default type `xsd:decimal` literals (supported by all OWL profiles), or `xsd:double` (OWL 2 RL/DL), or `owl:real` (OWL 2 EL/QL).

## 3 Implementation of The IFC2LD Converter

Our open-source Java application named **IFC2LD** converter[4], distributed freely with the MIT License, provides both application program interface (API) and command-line interface (CLI) for generating ifcOWL ontologies and ifcRDF datasets. In particular, one can use it for:

- loading IFC schema files in EXPRESS format (ISO 10303-11) into Java objects `IfcSchema`;
- loading IFC data files in STEP format (ISO 10303-21) into Java objects `IfcModel`;
- converting `IfcSchema` and `IfcModel` objects into Jena model objects[5];
- exporting Jena model objects to triples stores;
- and exporting Jena model objects to text files in different RDF formats, i.e., RDF/XML, N3, Turtle, NTRIPLES, NQUADS, with or without GZIP compression.

[3] DCMI metadata terms: http://dublincore.org/documents/dcmi-terms/
[4] IFC2LD source code and documentations: https://github.com/Web-of-Building-Data/ifc2ld.git.
[5] Jena model: http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Model.html

For supporting future IFC schemas, we designed `IfcSchema` and `IfcModel` by using the Dynamic Object Model pattern, which allows on-the-fly creating types, objects, and their relationships. It is possible to change many processing and output options through the XML configuration file. In the release packages[6], there are many sample codes of using APIs and sample scripts of using CLIs.

IFC2LD is currently in use in the DRUMBEAT Web-of-Building-Data platform[7]. It converts the uploaded IFC models to RDF for access through a REST interface and SPARQL end-points.

## 4 Experimental Results

The goals of experiments are validating and comparing alternative solutions, analyzing statistical measures of the output ontologies and datasets, and testing those using so called Competency Questions (CQs). In this section, we evaluate the correctness and effectiveness of our proposed solution. We then show that the suggested algorithms are efficient in the practical scenarios.

### 4.1 OWL Profile Validation

In order to check whether the ifcOWL ontology layers align with the OWL 2 profiles as defined in Table 1, we used the OWL API[8]. Each experiment consists of four steps: configuring the initial options (target OWL profile, conversion algorithms, etc.); exporting an ifcOWL ontology layer into a text file; loading the ontology file by OWL API; and checking if it is within the target OWL profile. The last step was done by using method `checkOntology()` provided by classes that implement interface `OWLProfile`, such as `OWL2DLProfile`, `OWL2ELProfile`, `OWL2QLProfile`, and `OWL2RLProfile`.

The experimental results show that all solutions in Section 2 are correct. Besides, they revealed many restricts of OWL 2 profiles, some of which are introduced in (Motik et al 2012), as follows:

- In all profiles, nothing can be declared as `owl:sameAs`, `owl:equivalentClass`, or `rdfs:subClassOf` to XSD datatypes. Therefore, simple and defined datatypes must be replaced by equivalent XSD datatypes or wrap XSD datatypes using boxing techniques.
- OWL 2 EL and OWL 2 QL do not support data range `xsd:boolean`. Thus, both logical simple type, i.e., `BOOLEAN` and `LOGICAL`, must be represented as enumeration classes.
- OWL 2 EL and OWL 2 QL do not support axiom `InverseObjectProperties`. Consequently, ifcOWL ontologies must not include inverse properties.
- OWL 2 EL/QL/RL do not allow axioms `ObjectOneOf` and `DataOneOf` that contain multiple individuals or literals. Hence, enumeration values must be declared by using axiom `ClassAssertion` (Listing 2a, line 2).
- OWL 2 EL/QL/RL do not supported `ObjectUnionOf`. OWL 2 DL do not allow axiom `ObjectUnionOf` with a single union-member. In such cases, union-member classes must be declared by using axiom `SubClassOf` (Listing 3a, line 2).
- OWL 2 EL/QL do not allow anonymous individuals. Thus, the converter must name all individuals (Listing 8).

Table 2 shows the potentials of OWL 2 profiles on supporting some significant for ifcOWL axioms. It implies that the ifcOWL-Standard layer, which requires axioms `ObjectAllValuesFrom` and `HasKey` for declaring property value ranges and entity keys, is compatible with OWL 2 RL or OWL 2 DL. The ifcOWL-Extended layer, which requires axioms `ObjectMinCardinality` and `ObjectMaxCardinality` for declaring cardinalities of lists and sets, is compatible only with OWL 2 DL.

**Table 2** Supporting OWL axioms by different OWL 2 profiles

| Axiom | OWL 2 EL/QL | OWL 2 RL | OWL 2 DL |
|---|---|---|---|
| `ObjectOneOf` | partially (≤ 1 value) | partially (only in subclass expr.) | **yes** |
| `ObjectUnionOf` | no | partially (only in subclass expr.) | **yes** |
| `ObjectAllValues` | no | **yes** | **yes** |
| `HasKey` | no | **yes** | **yes** |
| `ObjectMinCardinality` | no | no | **yes** |
| `ObjectMaxCardinality` | no | partially (only 0 or 1) | **yes** |

[6] IFC2LD release packages: http://drumbeat.cs.hut.fi/ifc2ld
[7] DRUMBEAT Web-of-Building-Data platform: http://drumbeat.cs.hut.fi
[8] The OWL API: http://owlapi.sourceforge.net/

## 4.2 Evaluating Alternative Solutions

Regarding to every aspect, besides our proposed solutions (Section 2), there are alternative ways. Although they are also valid within certain OWL profiles, they make ontologies and datasets more complex, break the data structure consistency, or bring to loss of important information.

### 4.2.1 Converting simple and defined datatypes into equivalent XSD datatypes

As value ranges, simple and defined datatypes are similar to XSD datatypes. However, XSD datatypes are `rdfs:Datatype`. Whereas, defined types must be `owl:Class`; otherwise, they cannot be `rdfs:subClassOf` of select classes (Listing 2a). Moreover, names of defined types are important in IFC. For instance, because a select-type `IfcValue` object can be an `IfcLengthMeasure` value, an `IfcMassMeasure` value, or something else, applications need to know the exact value type for detecting the kind and the corresponding unit of the value.

Although non-logical simple types do not have that sort of restrictions, we decided to convert them into `owl:Class` as well. This approach keeps the data structure consistency, allows all entity attributes to be `owl:ObjectProperty`, and slightly increases dataset sizes because 2% of attributes have non-logical simple types.

### 4.2.2 Using domain and range constraints

The purposes of using domains and ranges are restricting, verifying and inferring types of property subjects and objects. Type inference is useless because of the obvious type declaration of each individual in ifcRDF datasets (Listing 8). There are two ways of type restrictions: using `owl:allValuesFrom` and using combination `rdfs:domain`/`rdsf:range`. The first way restricts values of a property bound to a class, and it allows appearing of one property name in different classes, like in IFC schemas. By contrast, the second way restricts values of *global* properties. The basic facilities of domains and ranges do not provide any direct way to indicate that properties are *local* to a class (Schreiber & Raimond 2014). Using complex union classes as the domain and the range of a property that belongs to some classes can affect to the reasoning performance (Listing 9). Including entity type names into properties names allows avoiding duplications of property names, but makes them too long, especially for use in queries. Defining common superproperties with short names for querying, e.g. `ifc:basisSurface` in Listing 10, makes the ontology more complicated.

**Listing 9** Using union classes for domains and ranges

```
ifc:basisSurface a owl:ObjectProperty ;
  rdfs:domain [ a owl:Class ; owl:unionOf
     (ifc:IfcCurveBoundedPlane ifc:IfcPointOnSurface ifc:IfcRectangularTrimmedSurface )];
  rdfs:range [ a owl:Class ;  owl:unionOf ( ifc:IfcPlane ifc:IfcSurface ) ] .
```

**Listing 10** Using long-name properties as subproperties of a common short-name property

```
ifc:basisSurface a owl:ObjectProperty .
ifc:basisSurface_of_IfcCurveBoundedPlane rdfs:subPropertyOf ifc:basisSurface .
   rdfs:domain ifc:IfcCurveBoundedPlane ; rdfs:range ifc:IfcPlane .
ifc:basisSurface_of_IfcPointOnSurface rdfs:subPropertyOf ifc:basisSurface .
   rdfs:domain ifc:IfcPointOnSurface ; rdfs:range ifc:IfcSurface .
ifc:basisSurface_of_IfcRectangularTrimmedSurface rdfs:subPropertyOf ifc:basisSurface .
   rdfs:domain ifc:IfcRectangularTrimmedSurface ; rdfs:range ifc:IfcSurface .
```

Therefore, ifcOWL ontologies should contain `owl:allValuesFrom` restrictions, but not domains and ranges, except the domains and ranges of simple datatypes, which are necessary for inferring the base simple type of defined types. For instance, knowing that an `ifc:IfcLengthMeasure` object has property `expr:hasReal`, we can say that `ifc:IfcLengthMeasure` is a subclass of `expr:REAL`.

### 4.2.3 Converting aggregated types into RDF containers or RDF collections

Formally, aggregated types LIST and SET are similar to RDF containers `rdf:Bag` and `rdf:Seq`. However, the RDF containers have no syntactic support in Turtle. Moreover, the W3C RDF Working Group has marked `rdf:Bag` as "archaic" and has suggested doing the same with `rdf:Seq`[9] (before closing of the Group).

[9] The RDF Working Group issue about RDF containers: http://www.w3.org/2011/rdf-wg/track/issues/24

The commonly-used RDF collection `rdf:List` is singly linked list with no properties indicating the list sizes and items indexes. Thus, declaring the list cardinalities, item types and start/end indexes (in case of ARRAY) is very difficult.

Our proposed solution is based on the Ordered List Ontology[10]. By adding properties `expr:startIndex`, `expr:endIndex`, and `expr:isOrdered`, we adapted the ontology with types ARRAY and SET, in order to keep the same data structure for all aggregated types (Figure 2). Besides, we added additional collection classes, such as `ifc:Lis_of_IfcLengthMeasure` for restricting list item types, and `ifc:List_2_3` for constraining list cardinalities (Listing 4, a-c).

### 4.3 Querying ifcOWL ontologies and ifcRDF datasets

Common ways of expressing ontological requirements include Competency Questions (CQs) and descriptions of reasoning tasks (Blomqvist et al 2012). This subsection includes some sample CQs, each is related to a SPARQL query, successfully tested with ifcOWL ontologies in TopBraid Composer Maestro Edition application.

**Listing 11** Getting values of enumeration class `ifc:IfcActionTypeEnum` (all layers)

```
SELECT ?value WHERE { ?value rdf:type ifc:IfcActionTypeEnum }
```

**Listing 12** Getting direct union members of select class `ifc:IfcValue` (all layers)

```
SELECT ?class WHERE { ?class rdfs:subClassOf ifc:IfcValue }
```

**Listing 13** Getting type hierarchy of all classes derived from class `ifc:IfcRoot` (all layers)

```
SELECT ?class1 ?class2
 WHERE { ?class1 rdfs:subClassOf ?class2 . ?class2 rdfs:subClassOf* ifc:IfcRoot }
```

**Listing 14** Getting types of all properties of entity class `ifc:IfcProject` (Standard, Extended layers)

```
SELECT ?entityType ?property ?propertyType WHERE {
   ifc:IfcProject rdfs:subClassOf* ?entityType .
   ?entityType rdfs:subClassOf* expr:Entity ;
      rdfs:subClassOf [ owl:onProperty ?property ; owl:allValuesFrom ?propertyType ] }
```

**Listing 15** Getting item type and cardinalities of attribute `ifc:surfaceReinforcement1` (Extended layer)

```
SELECT ?listType ?itemType ?min ?max WHERE {
   ifc:IfcSurfaceReinforcementArea rdfs:subClassOf
      [ owl:onProperty ifc:surfaceReinforcement1; owl:allValuesFrom ?listType ] .
   ?listType rdfs:subClassOf+ [ owl:onProperty expr:slot; owl:allValuesFrom ?slotType ];
             rdfs:subClassOf+ [ owl:onProperty expr:slot; owl:minCardinality ?min ] ;
             rdfs:subClassOf+ [ owl:onProperty expr:slot; owl:maxCardinality ?max ] } .
   ?slotType rdfs:subClassOf+ [ owl:onProperty expr:item; owl:allValuesFrom ?itemType ].
```

### 4.4 Statistical Analysis

The size of the ifcOWL-Simple layer is about 2 times less than ifcOWL-Standard and 4 times less than ifcOWL-Extended (Table 3) because the latter two contain more information about properties of entity classes. The number of predefined global individuals in ifcOWL-Extended is reduced thanks to property `owl:oneOf` supported in OWL 2 DL and used for declaring enumeration values.

**Table 3** Statistical information of ifcOWL layers generated from IFC 2x3 TC1

| Parameter | ifcOWL-Simple | ifcOWL-Standard | ifcOWL-Extended |
|---|---|---|---|
| **File size** | ~250 Kb | **~610 Kb** | ~930 Kb |
| **Triples** | ~6150 | **~13200** | ~22300 |
| **Defined classes** | ~1400 | **~2900** | ~4200 |
| **Defined properties** | ~1060 | **~2000** | ~2000 |
| **Global individuals** | ~1315 | **~1315** | 3 |

The size of generated ifcRDF files are significantly bigger than the size of original IFC files. The Turtle format is more human-readable, but requires more resource on writing and reading than machine-readable formats, such as N-Triples, N-Quads. We recommend using N-TRIPLES with GZIP compression for work with most of RDF stores.

[10] The Ordered List Ontology: http://purl.org/ontology/olo/core#

**Table 4** Sizes of ifcRDF datasets serialized in different file formats

| File format | Sample model 1 | Sample model 2 |
|---|---|---|
| **SPFF (source)** | 10 Kb | 29 Mb |
| **Turtle** | 75 Kb | 150 Mb |
| **N-Triples** | 152 Kb | 250 Mb |
| **N-Triples + gzip** | 15 Kb | 25 Mb |

## 5 Summary and Future Works

In this paper, we presented the theoretical basis for the conversion, introduced our IFC2LD converter, and then evaluated the correctness and effectiveness of its outputs in practical scenarios.

The conversion of the IFC schema to OWL2 is carried out in a layered manner. There are three layers – ifcOWL-Simple, ifcOWL-Standard, and ifcOWL-Extended – each is compatible with IFC data converted to ifcRDF. The layers provide increasingly detailed type information about the objects and properties in ifcRDF. The ifcOWL-Simple layer enables the use of advanced reasoning tools. The ifcOWL-Extended translates as many constraints of the IFC Schema as possible into OWL. This is important in use cases that require modifications to ifcRDF models and the further data validation.

IFC2LD is a Java application with Web interface for converting IFC schemas into OWL2 ontologies and IFC data into RDF graphs aligned with the ontologies.

We tested and evaluated the conversions in four major steps: (1) OWL profile validation to filter wrong solutions; (2) comparing alternative conversion approaches; (3) running test queries on ontologies and datasets; and (4) analyzing statistic parameters. The experimental result showed that the proposed solution give the best results from different aspects.

There are areas where the development of converter must be continues further, such as:

- running test of reasoning performance on huge ifcRDF datasets with different ontologies;
- producing recommendations on use of ifcOWL ontology layers;
- producing ifcRDF versions of model views based on the Model View Definitions (MVD);
- supporting link generation across datasets;
- implementing the reversion conversion from ifcRDF datasets into IFC data files.

## Acknowledgements

This work has been carried out at Aalto University in research projects DRUM (RYM/PRE 2010-2014) and DRUMBEAT (2014-2017), funded by Tekes, Aalto University, and the participating companies.

## References

Beetz, J. (2009a). *Facilitating distributed collaboration in the AEC/FM sector using semantic web technologies*, Ph.D. dissertation, PhD Thesis, Eindhoven University of Technology. ISBN 978-90-6814-618-9, 2009.

Beetz, J., Van Leeuwen, J. & De Vries, B. (2009b). IfcOWL: A case of transforming Express schemas into ontologies, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 23, no. 01, pp. 89–101, 2009.

Blomqvist, E., Sepour, A.S. & Presutti, V. (2012). Ontology Testing – Methodology and Tool. In *EKAW 2012*, LNAI 7603, pp. 216-226, 2012.

Burleson, C., Gutierrez, M.E. & Mihindukulasooriya, N. (2014). Linked Data Platform Best Practices and Guidelines. *W3C Working Group Note 25 Aug 2014.* http://www.w3.org/TR/ldp-bp/

Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A. & Lutz, C. (2012). OWL 2 Web Ontology Language: Profiles (2nd ed.). *W3C Recommendation 11 Dec 2012.* http://www.w3.org/TR/owl2-profiles/

Pauwels, P., De Meyer, R., & Van Campenhout, J. (2011). Interoperability for the design and construction industry through semantic web technology. In *Semantic Multimedia* (pp. 143-158). Springer Berlin Heidelberg.

Schreiber, G. & Raimond, Y. (2014) RDF 1.1 Primer. *W3C Working Group Note 25 Feb 2014.* http://www.w3.org/TR/rdf11-primer/

Törmä, S. (2013). Semantic linking of building information models. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on* (pp. 412-419). IEEE.

Törmä, S. (2014). Web of Building Data – Integrating IFC with the Web of Data. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014*, 141.