Document id	Title	Organisation /Author	Date	Status
TR_LDWG001	EXPRESS-to-OWL conversion routine	LDWG – Pieter Pauwels LDWG – Walter Terkaj LDWG – Jakob Beetz	12 Oct. 2015	FINAL

BuildingSMART Proposed Recommendation: EXPRESS-to-OWL conversion routine

Executive summary:

In this Technical Report, the BuildingSMART Linked Data Working Group (LDWG) documents the EXPRESS-to-OWL conversion routine that it proposes to the standardisation committee of BuildingSMART. This conversion routine is independent of any practical implementation in software. It lists the ways in which the diverse data types in EXPRESS can be uniformly converted into OWL concepts. When implementing this conversion routine, an OWL ontology can be obtained from the IFC schema (tested on IFC2X3_TC1, IFC2X3_Final, IFC4, IFC4_ADD1).

Central to this EXPRESS-to-OWL conversion routine are three criteria:

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.

There are numerous ways to map an EXPRESS schema onto an OWL ontology. This has been investigated in depth and at length during the Technical Session of the 3rd International Workshop on Linked Data in Architecture and Construction (LDAC), held in TU Eindhoven in July 2015. The decisions made there are available in the report at http://ldac-2015.bwk.tue.nl/LDAC_2015_workshopreport.pdf. These decisions are followed in the current proposal. Many of the considerations in this Technical Report are also discussed at length in the article in Automation in Construction by P. Pauwels and W. Terkaj "*EXPRESS to OWL for construction industry: towards a recommendable and usable ifcOWL ontology*".

More information: http://www.buildingsmart-tech.org/future/linked-data

1. Introduction

The EXPRESS-to-OWL conversion presented in this document is relying on the IFC4_ADD1.exp EXPRESS schema. All examples used in this document come from this schema. However, the conversion procedure is also applicable on IFC2X3_Final.exp, IFC2X3_TC1.exp, and IFC4.exp.

Examples are documented using Fragments of EXPRESS declarations and Fragments of OWL declarations in the Turtle syntax (TTL).

2. Schema definition

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.

Each EXPRESS schema is converted into a separate ontology identified by a unique URI. The following URI design is used for these ontologies:

http://www.buildingsmart-tech.org/ifcOWL/[schemaName]

with [schemaName] being one of the four schema names.

Fragment 2 shows in what this results in OWL. A number of Dublin Core (dce) metadata annotations is 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)" ,
                        "Walter Terkaj  (walter.terkaj@itia.cnr.it)" ;
        dce:contributor         "Aleksandra Sojic (aleksandra.sojic@itia.cnr.it)" ,
                                "Jakob Beetz (j.beetz@tue.nl)" ,
                                "Maria Poveda Villalon (mpoveda@fi.upm.es)" ,
                                "Nam Vu Hoang  (nam.vuhoang@gmail.com )";
         rdfs:comment           "Ontology automatically generated from the EXPRESS sche-
                ma IFC4_ADD1' using the 'IFC-to-RDF' converter developed by Pieter Pau-
                wels (pipauwel.pauwels@ugent.be), based on the earlier versions from Jyrki
                Oraskari  (jyrki.oraskari@aalto.fi) and Davy Van Deursen (da-
                vy.vandeursen@ugent.be)" ;
        dce:title               "IFC4_ADD1" ;
        dce:description         "OWL ontology for the IFC conceptual data schema and ex-
                                change file format for Building Information Model (BIM) da-
                                ta" ;
        dce:date                "2015/10/02" ;
        dce:identifier          "IFC4_ADD1" ;
        dce:language            "en" ;
        vann:preferredNamespacePrefix       "ifc" ;
        vann:preferredNamespaceUri          "http://www.buildingsmart-
                                            tech.org/ifcOWL/IFC4_ADD1" ;
        owl:imports             <http://purl.org/voc/express> ;
        cc:license              <http://creativecommons.org/licenses/by/3.0/> ;
```

Fragment 2 - Definition of the ifcOWL ontology.

The ontologies rely on a number of existing ontologies. Fragment 3 shows the prefixes and the URIs of the ontologies that are relied upon. Most notable here are the prefixes:

- list: <http://www.co-ode.org/ontologies/list.owl#>
  This ontology contains the classes and properties that are used in the ifcOWL ontology to represent ordered list structures (e.g. list:hasNext).
- expr: <http://purl.org/voc/express#>
  This ontology contains the classes and properties that are specific to the EXPRESS language and are not actually part of the IFC schema (e.g. expr:hasDouble).

```
@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 cc:    <http://creativecommons.org/ns#> .
@prefix expr:  <http://purl.org/voc/express#> .
@prefix vann:  <http://purl.org/vocab/vann/> .
@prefix list:  <http://www.co-ode.org/ontologies/list.owl#> .
```

Fragment 3 - Printout of the ontologies that are used by the proposed ifcOWL ontology.

3. Simple data type declarations

Simple data types (i.e. NUMBER, REAL, INTEGER, LOGICAL, BOOLEAN, STRING, BINARY) are commonly used in an IFC schema. As these data types are inherently part of the EXPRESS language, rather than the IFC schema, these data types are maintained in a separate ontology, namely the EXPRESS ontology prefixed as 'expr:' in Fragment 3. In this EXPRESS ontology, the simple data type concepts (e.g. expr:REAL in Fragment 4) are available as classes with datatype properties (e.g. expr:hasDouble in Fragment 4) pointing to xsd literals (e.g. xsd:double in Fragment 4).

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

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

Fragment 4 - Printout of the RDF graph representation for the simple data type REAL in the ifcOWL schema.

The LOGICAL data type is a special case, as there is no equivalent xsd literal which can hold the values TRUE, FALSE, and UNKNOWN. For this particular datatype, a special expr:LogicalEnum class is constructed that is an enumeration of the individuals expr:TRUE, expr:FALSE, and expr:UNKNOWN (see Fragment 5). The rest of the conversion pattern follows the same approach as for the other simple data types, namely with a class expr:LOGICAL and an object property (instead of a datatype property) expr:hasLogical pointing to one of the three enumeration values.

```
expr:hasLogical  a   owl:FunctionalProperty , owl:ObjectProperty ;
       rdfs:domain  expr:LOGICAL ;
       rdfs:label   "hasLogical" ;
       rdfs:range   expr:LogicalEnum .

expr:LOGICAL  a        owl:Class ;
       rdfs:subClassOf  [ a               owl:Restriction ;
                        owl:allValuesFrom  expr:LogicalEnum ;
                        owl:onProperty     expr:hasLogical
                      ] .

expr:LogicalEnum  a      owl:Class ;
       rdfs:subClassOf  expr:ENUMERATION .

expr:TRUE  a       expr:LogicalEnum , owl:NamedIndividual ;
       rdfs:label  "TRUE" .

expr:FALSE  a      expr:LogicalEnum , owl:NamedIndividual ;
       rdfs:label  "FALSE" .

expr:UNKNOWN  a     expr:LogicalEnum , owl:NamedIndividual ;
       rdfs:label  "UNKNOWN" .
```

Fragment 5 - Printout of the RDF graph representation of the expr:LOGICAL class and its three named individuals.

Also the classes and properties in Fragment 5 are specific to EXPRESS and are thus part of a separate EXPRESS ontology at <http://purl.net/voc/express#>.

4. Defined (named) data type declarations

A number of defined data types is typically available in any EXPRESS schema of IFC. Most declarations are similar to the one given in Fragment 6 (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 6 - Printout of the defined data type declaration IfcAreaDensityMeasure.

These defined data types are converted into classes, which are then declared as sub-classes of the data type to which they are referring (see Fragment 7).

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

Fragment 7 - Printout of the RDF graph representation for the IfcAreaDensityMeasure defined data type.

5. Aggregation data type declarations

Often used data types in the IFC schema are the aggregation data types SET, LIST, and ARRAY. There exist also a BAG data type in EXPRESS, but it is not used in the IFC schema at the moment, so it is also not handled in the conversion routine proposed here.

5.1 SET data type declarations

The SET aggregation data types are unordered aggregations of instances that are supposed to be different from each other. These aggregations are typically declared for attributes of EXPRESS entities (see Fragment 8).

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

Fragment 8 - Printout of a non-OPTIONAL SET data type declaration in RDF, with a lower cardinality restriction of 1 (at least one instance should be present in the SET).

An unordered aggregation 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 (see top of Fragment 9). OWL restrictions for this non-functional property allow to represent object type and cardinality constraints (see bottom of Fragment 9).

```
ifc:innerCurves_IfcArbitraryProfileDefWithVoids
        a          owl:ObjectProperty ;
        rdfs:domain  ifc:IfcArbitraryProfileDefWithVoids ;
        rdfs:label   "InnerCurves" ;
        rdfs:range   ifc:IfcCurve .

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

Fragment 9 – Printout of the non-functional InnerCurves object property, and of the owl:Class declaration for the IfcArbitraryProfileDefWithVoids entity data type declaration, including restrictions.

5.2 LIST and ARRAY data type declarations

LIST and ARRAY data type declarations are harder to express in OWL declarations. Both of them represent an ordered collection of elements. Herein, we propose to use the same conversion routine for both data type declarations. An example LIST data type declaration is given in Fragment 10, showing a LIST of at least 3 and at most 4 INTEGER instances.

```
TYPE IfcCompoundPlaneAngleMeasure = LIST [3:4] OF INTEGER;
...
END_TYPE;
```

Fragment 10 – An example LIST data type declaration.

The LIST data type declaration in Fragment 10 is declared as part of a defined data type declaration (see Fragment 6 and 7). Hence, the conversion routine for defined data type declarations is proposed for the IfcCompoundPlaneAngleMeasure data type, subclassing it to the expr:INTEGER_List class (see Fragment 11).

```
ifc:IfcCompoundPlaneAngleMeasure
        rdf:type            owl:Class ;
        rdfs:subClassOf  ifc:IfcDerivedMeasureValue , expr:INTEGER_List ;
```

Fragment 11 – Printout of the RDF graph for the IfcCompoundPlaneAngleMeasure data type.

The expr:INTEGER_List class is part of the EXPRESS ontology, where it is defined as a subclass of the list:OWLList class. This list ontology (list:) provides a number of object properties (e.g. list:hasNext) that are used in the ifcOWL ontology to represent the cardinality restrictions for the LIST data type. In the case of IfcCompoundPlaneAngleMeasure (LIST [3:4]), this results in the OWL restrictions shown in Fragment 12 and 13.

```
ifc:IfcCompoundPlaneAngleMeasure
rdfs:subClassOf  [
        a               owl:Restriction ;
        owl:onProperty      list:hasNext ;
        owl:someValuesFrom  [
                a               owl:Restriction ;
                owl:onProperty      list:hasNext ;
                owl:someValuesFrom  [
                        a               owl:Restriction ;
                        owl:onProperty      list:hasNext ;
                        owl:someValuesFrom  expr:INTEGER_List
                ]
        ]
] ;
```

Fragment 12 – Concatenation of OWL restrictions to indicate that there should be at least three instances in the expr:INTEGER_List.

```
ifc:IfcCompoundPlaneAngleMeasure
rdfs:subClassOf  [
        a               owl:Restriction ;
        owl:allValuesFrom  [
                a               owl:Restriction ;
                owl:allValuesFrom  [
                        a               owl:Restriction ;
                        owl:allValuesFrom  [
                                a                   owl:Restriction ;
                                owl:onClass             expr:INTEGER_EmptyList ;
                                owl:onProperty          list:hasNext ;
                                owl:qualifiedCardinality  "1"^^xsd:nonNegativeInteger
                        ] ;
                        owl:onProperty     list:hasNext
                ] ;
                owl:onProperty     list:hasNext
        ] ;
        owl:onProperty     list:hasNext
] .
```

Fragment 13 – Concatenation of OWL restrictions to indicate that there should be at most four instances in the expr:INTEGER_List.

6. Constructed data type declarations

6.1 Enumeration data type declarations

Enumeration data type declarations are declared as presented in Fragment 14. The list of values given in this declaration limits the values that an instance of this enumeration data type can have.

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

Fragment 14 – 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).

In the proposed EXPRESS-to-OWL conversion routine, Enumeration data types are converted into regular OWL classes. The individuals that are declared in EXPRESS, are also declared in ifcOWL as instances or Named Individuals of the Enumeration class (Fragment 15). As can be seen in Fragment 15, a Named Individual can be an instance of a number of OWL enumeration classes (e.g. ifc:SITE).

```
ifc:IfcAddressTypeEnum
          a                    owl:Class ;
          rdfs:subClassOf          expr:ENUMERATION .

ifc:DISTRIBUTIONPOINT
          a                    ifc:IfcAddressTypeEnum , owl:NamedIndividual ;
          rdfs:label           "DISTRIBUTIONPOINT" .

ifc:SITE
          a                    ifc:IfcAddressTypeEnum , ifc:IfcCrewResourceTypeEnum ,
                    ifc:IfcAssemblyPlaceEnum , owl:NamedIndividual ;
          rdfs:label           "SITE" .
```

Fragment 15 – Printout of the RDF graph for the IfcAddressTypeEnum class, also showing two Named Individuals of this class (ifc:DISTRIBUTIONPOINT and ifc:SITE).

6.2 Select data type declarations

A select data type declaration is identified by the keyword SELECT, as shown in Fragment 16. Any instantiation of such data type should refer to one instance of the listed types or entities.

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

Fragment 16 – Printout of the SELECT data type declaration IfcMetricValueSelect, which refers to a list of allowed data type instantiations (IfcAppliedValue, IfcMeasureWithUnit, IfcReference, IfcTable, IfcTimeSeries, and IfcValue).

Similar to the conversion of Enumeration data types, Select data types are converted into OWL classes. The classes that appear in the list of data types in the Select data type declaration are declared as subclasses of the Select data type. So, for the example in Fragment 17, ifc:IfcValue is declared as a subclass of ifc:IfcMetricValueSelect. Also in this case, multiple superclasses can be available.

```
ifc:IfcMetricValueSelect
        a                   owl:Class ;
        rdfs:subClassOf     expr:SELECT .

ifc:IfcValue
        a                   owl:Class ;
        rdfs:subClassOf     expr:SELECT , ifc:IfcAppliedValueSelect ,
                            ifc:IfcMetricValueSelect .
```

Fragment 17 – Printout of the RDF graph for the IfcMetricValueSelect class and its subclass: IfcValue.

7. Entity (named) data type declarations

The core of the IFC EXPRESS schemas is represented in the Entity data type declarations. Two example Entity data type declarations are given in Fragment 18 and 19 (IfcBSplineCurve and IfcObject). These two example declarations contain most, if not all, attribute declaration types that can be part of an entity data type declaration.

```
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
        :=                                          IfcListToAr-
        ray(ControlPointsList,0,UpperIndexOnControlPoints);
        WHERE
                SameDim : SIZEOF(QUERY(Temp <* ControlPointsList |Temp.Dim <>
                                                    ControlPointsList[1].Dim)) =   0;
END_ENTITY;
```

Fragment 18 – Printout of the IfcBSplineCurve Entity data type declaration.

```
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 19 – Printout of the IfcObject Entity data type declaration.

All entity data types are commonly converted into OWL class declarations, as is also displayed in Fragment 20 for IfcBSplineCurve. For this class, subclass relations are included if they are present in the EXPRESS declaration (SUPERTYPE OF() and SUBTYPE OF()). Also disjointness is taken into account. In the case of IfcBSplineCurve, the IfcBoundedCurve supertype is actually declared as an abstract supertype of IfcBSplineCurve, Ifc-

CompositeCurve, IfcIndexedPolyCurve, IfcPolyline, and IfcTrimmedCurve. Such a declaration implies disjointness, which is included in the OWL representation in Fragment 20.

```
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 20 – Printout of the RDF graph representation for the IfcBSplineCurve class.

7.1 Regular attributes

For each entity data type, a number of regular attributes is typically declared. An example of such a regular attribute declaration is given in Fragment 21. In this case, an attribute Degree is declared that points from an IfcBSplineCurve instance to an IfcInteger instance.

```
ENTITY IfcBSplineCurve
        ...
        Degree : IfcInteger;
        ...
END_ENTITY;
```

Fragment 21 – Printout of a regular attribute declaration in EXPRESS.

Such regular attributes are commonly converted into OWL object properties (see Fragment 22). As the Degree attribute is not pointing to a SET data type declaration, there is only 1 Degree attribute allowed, which is taken into account by declaring the object property as a functional object property.

In the proposed EXPRESS-to-OWL conversion routine, each object property is declared with exactly one data type in its range and exactly one data type in its domain (see Fragment 22), mainly because attributes in EXPRESS always point to at most one EXPRESS data type, and they are defined only within the scope of the entity data type declaration in which they appear (1 to 1 relationship). To allow these 1 to 1 relationships with ranges and domains, all object property URIs are named following the naming convention "propertyName_IfcClassName" (see Fragment 22).

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

Fragment 22 – Printout of the OWL object property declaration for a regular EXPRESS attribute.

In addition to the object property declarations as in Fragment 22, an object type restriction and a cardinality restriction are also added to the OWL class declaration, as is displayed in Fragment 23. The cardinality restriction indicates that there should always be exactly one instantiation of this property, as the EXPRESS attribute is not OPTIONAL and also does not point to a SET aggregation data type.

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

Fragment 23 – Printout of the restrictions that are added to the IfcBSplineCurve class to handle object type and cardinality restrictions.

7.2 Aggregation attributes

A number of attributes points to aggregation data types, in the case of IFC limited to LIST, SET, and ARRAY data types. Fragments 8 to 13 already presented the way in which these aggregation data types are handled. This is repeated in Fragment 24 to 26.

```
ENTITY IfcBSplineCurve
        ...
        ControlPointsList : LIST [2:?] OF IfcCartesianPoint;
        ...
END_ENTITY;
```

Fragment 24 – Printout of the EXPRESS declaration of an attribute pointing to a LIST aggregation data type.

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

Fragment 25 – Printout of the OWL object property declaration for an EXPRESS attribute that points to an aggregation data type.

```
ifc:IfcBSplineCurve
        rdfs:subClassOf   [
                a                   owl:Restriction ;
                owl:allValuesFrom       ifc:IfcCartesianPoint_List ;
                owl:onProperty          ifc:controlPointsList_IfcBSplineCurve
        ] ;
        rdfs:subClassOf   [
                a                       owl:Restriction ;
                owl:onClass             ifc:IfcCartesianPoint_List ;
                owl:onProperty          ifc:controlPointsList_IfcBSplineCurve ;
                owl:qualifiedCardinality        "1"^^xsd:nonNegativeInteger
        ] ;
        rdfs:subClassOf   [
                a               owl:Restriction ;
                owl:allValuesFrom  [
                        a               owl:Restriction ;
                        owl:onProperty      list:hasNext ;
                        owl:someValuesFrom  [
                                a               owl:Restriction ;
                                owl:onProperty      list:hasNext ;
                                owl:someValuesFrom  ifc:IfcCartesianPoint_List
                        ]
                 ] ;
        owl:onProperty     ifc:controlPointsList_IfcBSplineCurve                    ] .
```

Fragment 26 – Printout of the restrictions that are added to the IfcBSplineCurve class to handle object type and cardinality restrictions for the object property pointing to a LIST class.

7.3 Optional attributes

The EXPRESS language provides the option to declare OPTIONAL attributes, as shown in Fragment 27. These attribute declarations are converted as if they were regular attributes (see Fragment 28), the only difference being that a maxQualifiedCardinality restriction is used (bottom of Fragment 29) instead of an exact qualifiedCardinality restriction (see Fragment 23).

```
ENTITY IfcObject
        …
        ObjectType : OPTIONAL IfcLabel;
        …
END_ENTITY;
```

Fragment 27 – Printout of a regular attribute declaration in EXPRESS.

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

Fragment 28 – Printout of the OWL object property declaration for an OPTIONAL EXPRESS attribute.

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

Fragment 29 – Printout of the restrictions that are added to the IfcObject class to handle object type and cardinality restrictions for the OPTIONAL object property.

7.4 Inverse attributes

The EXPRESS language allows to declare INVERSE attributes as well. 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 30, for example, the attribute going in the inverse direction can be found in Fragment 31, namely the RelatedObjects attribute of the IfcRelDefinesByObject entity data type.

```
ENTITY IfcObject
        …
        INVERSE IsDeclaredBy : SET [0:1] OF IfcRelDefinesByObject FOR RelatedObjects;
        …
END_ENTITY;
```

Fragment 30 – Printout of the INVERSE attribute IsDeclaredBy for the IfcObject entity (named) data type declaration.

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

Fragment 31 – Printout of the entity (named) data type declaration IfcRelDefinesByObject.

The INVERSE attribute is converted using the regular conversion procedure, only adding an owl:inverseOf declaration.

```
ifc:isDeclaredBy_IfcObject
        rdfs:domain             ifc:IfcObject ;
        rdfs:range              ifc:IfcRelDefinesByObject ;
        owl:inverseOf  ifc:relatedObjects_IfcRelDefinesByObject ;
        rdf:type                        owl:FunctionalProperty, owl:ObjectProperty .
```

Fragment 32 – Printout of the OWL object property declaration for an INVERSE EXPRESS attribute.

In two particular cases, the owl:inverseOf relation should not be added in order to keep the ontology consistent:

1. when an attribute has two or more INVERSE attributes, otherwise those inverse attributes would be considered equivalent by any OWL inference engine.
2. when a regular attribute or its INVERSE has a LIST or ARRAY as its range, otherwise there would be a domain / range mismatch (because of the particular conversion routine for ordered lists) and thus an inconsistent ontology.

7.5 Derive and where attributes

An example DERIVE attribute declaration is given in Fragment 33; an example WHERE attribute is given in Fragment 34. DERIVE and WHERE attributes are not handled in the current EXPRESS-to-OWL conversion routine.

```
ENTITY IfcBSplineCurve
        ABSTRACT SUPERTYPE OF (ONEOF(IfcBSplineCurveWithKnots))
        SUBTYPE OF (IfcBoundedCurve);
        …
        DERIVE
                UpperIndexOnControlPoints : IfcInteger := (SIZEOF(ControlPointsList) - 1);
                ControlPoints : ARRAY [0:UpperIndexOnControlPoints] OF IfcCartesianPoint
                := IfcListToArray(ControlPointsList,0,UpperIndexOnControlPoints);
        …
END_ENTITY;
```

Fragment 33 – Printout of a DERIVE attribute in EXPRESS.

```
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;
```

Fragment 34 – Printout of a WHERE attribute in EXPRESS.

8. FUNCTION declarations

An example FUNCTION declaration is given in Fig. 35. The conversion of FUNCTION declarations is out of scope for the current proposed EXPRESS-to-OWL conversion routine.

```
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 35 – Printout of a FUNCTION declaration in EXPRESS.

9. RULE declarations

An example RULE declaration is given in Fig. 36. The conversion of RULE declarations is out of scope for the current proposed EXPRESS-to-OWL conversion routine.

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

Fragment 36 – Printout of a RULE declaration in EXPRESS.