SECOM Co., Ltd. / VTT Building and Transport
Yoshinobu Adachi
E-Mail: yoshinobu.adachi@vtt.fi

VTT-TEC-ADA-02

Overview of Meta Model Definition in XML for EXPRESS Representation

2001/07/27

1. Introduction

This document describes an overview about meta model definition languages in XML to decide the direction of the intermediate XML schema in the EXPRESS to XML Schema Converter (EXC). The XML schema definition language such as XSD (XML Schema Definition of W3C) is not discussed in this document. The capability of meta model representation is focused in this document.

EXC converts EXPRESS schema to the intermediate XML meta model format first. This XML format should be very natural, equal to the content of EXPRESS and reusable for converting to other schema formats such as SQL or some other meta model format written in XML as secondary meta data format. After creating the intermediate XML meta model format, XSL and XML parser driven applet can be used for conversion to secondary meta model format.

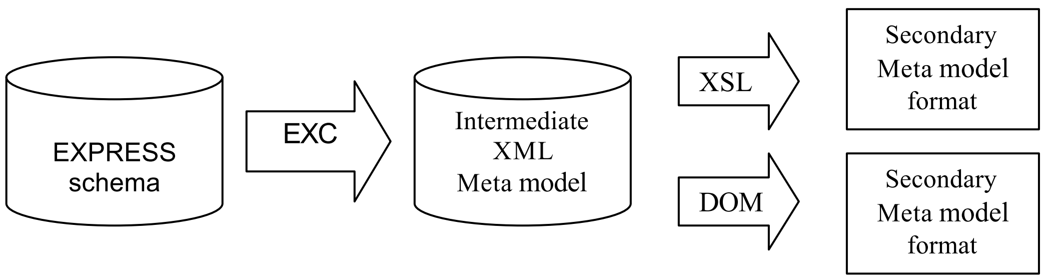
EXPRESS schema
EXC
Intermediate XML Meta model
XSL
DOM
Secondary Meta model format
Secondary Meta model format

Figure 1. Conceptual meta model data flow

Following XML languages are mentioned in this document:

- **STEP Part 28**
- **Open Information Model (OIM)**
- **XML Metadata Interchange (XMI)**
- **IfcXML**
- **BLIS-XML**

Looking into each specification and comparing the feasibility for *the intermediate XML meta data format* and *secondary XML meta data format*.

2. XML Meta Model Languages

The following XML languages are most likely candidates for the intermediate XML format and secondary XML meta data format, from consideration about their potential with schema definition functionality and usable implemented tools.

2.1 ISO STEP Part 28

ISO STEP Part 28 describes XML representation of EXPRESS schema. Part 28 has been developing in TC184/SC4 project number 10303-0028. This project defines an XML encoding method of EXPRESS language and instance data. In this document, the functionality of XML specification for EXPRESS language markup is focused. At the moment, the latest document about STEP Part 28 is shown below:

Product data representation and exchange: Implementation methods: XML representation of EXPRESS schemas and data, ISO TC184/SC4 WG11 N135 [1].

The document describes following three methods about XML specification, one late binding and two early binding approaches. The XML specification is defined by Document Type Definition (DTD) in Part 28.

1. **EXPRESS language markup declarations**
   This is a late binding XML markup declaration set based on the syntax of the EXPRESS schema. In this method, both schema declaration part and data declaration part are included. The schema declaration part is fully compatible with EXPRESS information, therefore this method will be usable for intermediate XML meta data format. Simple examples are shown below:

   - TYPE declaration:

     ```
     TYPE temperature
         = REAL;
     END_TYPE;
     ```

     XML format is:

     ```
     <type_decl>
       <type_id>temperature</type_id>
       <underlying_type>
          <real>
          </real>
       </underlying_type>
     </type_decl>
     ```

   - SELECT TYPE declaration:

     ```
     TYPE flower_colour
        = ENUMERATION OF (red, yellow, white);
     END_TYPE;
     ```

     XML format is:

     ```
     <type_decl>
       <type_id>flower_colour</type_id>
       <underlying_type>
          <enumeration>
             <enumeration_id>red</enumeration_id>
             <enumeration_id>yellow</enumeration_id>
             <enumeration_id>white</enumeration_id>
          </enumeration>
       </underlying_type>
     </type_decl>
     ```

   - ENTITY declaration:

```
ENTITY pond
   ABSTRACT SUPERTYPE OF
      (ONEOF (fish_pond, amphibian_pond) ANDOR ornamental_pond);
   maintained_by: OPTIONAL water_treatment_system;
   plants: SET[1:?] OF aquatic_plant;
   water_volume: positive_integer;
   water_ph: ph;
END_ENTITY;

ENTITY ornamental_pond
   SUBTYPE OF (pond);
   ornaments: SET [1:?] OF pond_ornament;
DERIVE
   SELF¥pond.water_volume: positive_integer :=
   water_volume_per_ornament * SIZEOF(ornaments);
END_ENTITY;
```

The corresponding XML markup would be:

```
<entity_decl>
  <entity_id>pond</entity_id>
  <abstract_supertype>
    <supertype_and_or>
      <supertype_one_of>
          <entity_ref>fish_pond</entity_ref>
        <entity_ref>amphibian_pond</entity_ref>
      </supertype_one_of>
      <entity_ref>ornamental_pond</entity_ref>
    </supertype_and_or>
  </abstract_supertype>
  <explicit_attr_block>
    <explicit_attr>
      <attribute_id>maintained_by</attribute_id><optional/>
      <base_type>
        <entity_ref>water_treatment_system</entity_ref>
      </base_type>
    </explicit_attr>
    <explicit_attr>
      <attribute_id>plants</attribute_id>
      <base_type>
        <set_type>
          <bound_spec>
            <lower_bound>
              <integer_literal>1</integer_literal>
            </lower_bound>
            <upper_bound>
               <unset/>
            </upper_bound>
          </bound_spec>
          <base_type>
              <entity_ref>aquatic_plant</entity_ref>
          </base_type>
        </set_type>
      </base_type>
    </explicit_attr>
    <explicit_attr>
      <attribute_id>water_volume</attribute_id>
      <base_type>
        <type_ref>positive_integer</type_ref>
      </base_type>
    </explicit_attr>
    <explicit_attr>
      <attribute_id>water_ph</attribute_id>
      <base_type>
        <type_ref>ph</type_ref>
```

```
          </base_type>
        </explicit_attr>
      </explicit_attr_block>
    </entity_decl>

    <entity_decl>
        <entity_id>ornamental_pond</entity_id>
        <subtype_of>
          <entity_ref>pond</entity_ref>
        </subtype_of>
        <explicit_attr_block>
          <explicit_attr>
            <attribute_id>ornaments</attribute_id>
            <base_type>
              <set_type>
                <bound_spec>
                  <lower_bound>
                    <integer_literal>1</integer_literal>
                  </lower_bound>
                  <upper_bound><unset/>
                  </upper_bound>
                </bound_spec>
                <base_type>
                  <type_ref>pond_ornament</type_ref>
                </base_type>
              </set_type>
            </base_type>
          </explicit_attr>
        </explicit_attr_block>
        <derive_clause>
          <derived_attr>
            <qualified_attribute>
              <entity_ref>pond</entity_ref>
              <attribute_ref>water_volume</attribute_ref>
            </qualified_attribute>
            <base_type>
              <type_ref>positive_integer</type_ref>
            </base_type>
            <term><multiply/>
              <constant_ref>water_volume_per_ornament</constant_ref>
              <function_call><sizeof/>
                <attribute_ref>ornaments</attribute_ref>
              </function_call>
            </term>
          </derived_attr>
        </derive_clause>
      </entity_decl>
```

The DTD for EXPRESS language markup declarations are written in the document [1].

2. EXPRESS-typed Early Binding (ETEB)

ETEB is designed for EXPRESS driven instance data in XML, therefore ETEB is not suitable for describing the EXPRESS schema information. In this method, ETBT XML data is defined by a specific DTD that is converted from the corresponding EXPRESS schema. This means that ETEB has no fixed DTD and it is difficult to get schema information form DTD format (DTD is not XML instance). ETEB is intended to represent XML instance data, therefore this method is not good for EXC. A feature of ETEB is the inheritance mechanism that sub type class elements appear under the super class element.

For instance, an EXPRESS definition and converted DTD are shown below:

```
ENTITY vehicle;
    vin : STRING;
END_ENTITY;
```

```
ENTITY car SUBTYPE OF (vehicle);
   make  : STRING;
    car_model : STRING;
END_ENTITY;

ENTITY truck SUBTYPE OF (vehicle);
    capacity : INTEGER;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Vehicle (Vehicle.vin, Vehicle-subtypes?)>
<!ATTLIST Vehicle
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "vehicle"
    late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Car (Car.make, Car.car_model)>
<!ATTLIST Car
    id ID #IMPLIED
    express_entity_name NMTOKEN #FIXED "car"
    late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Truck (Truck.capacity)>
<!ATTLIST Truck
    id ID #IMPLIED
     express_entity_name NMTOKEN #FIXED "truck"
     late-bound-element NMTOKEN #FIXED "partial_entity_instance">

 <!ELEMENT Vehicle-subtypes ((Car | Truck)+)>
```

3. Object Serialization Early Binding (OSEB)

OSEB is designed for EXPRESS driven instance data in XML, and OSEB is an early binding method, so that this is also not suitable for EXC use. The attributes that are defined in super class appear in the leaf class element. For instance, an EXPRESS definition and converted DTD are shown below:

```
ENTITY point;
x : REAL;
y : REAL;
END_ENTITY;

ENTITY cartesian_point
SUBTYPE OF (point);
z : REAL;
END_ENTITY;
```

As DTD declarations:

```
<!ELEMENT Point EMPTY>
<!ATTLIST Point
  x-id  ID  #REQUIRED
  X  CDATA  #REQUIRED
  Y  CDATA  #REQUIRED>

<!ELEMENT Cartesian_point EMPTY>
<!ATTLIST Cartesian_point
  x-id  ID  #REQUIRED
  X  CDATA  #REQUIRED
  Y  CDATA  #REQUIRED
  Z  CDATA  #REQUIRED>
```

2.2 Open Information Model (OIM)

OIM(Open Information Model) is developed by MDC(Meta Data Coalition) for describing meta data information. MDC OIM is a vendor-neutral and technology independent specification. Microsoft is one of big name in MDC and uses OIM in their products, i.e. SQL Server, as meta model interchange format.
OIM uses DTD for XML specification. OIM covers several areas about meta model exchanging such as:

- Analysis and Design Model
  - Unified Modeling Language
  - UML Extensions
  - Common Data Types
  - Generic Elements
- Object and Component Description Model
  - Component Descriptions
- Database and Data Warehousing
  - Database Schema Elements
  - Data Transformation Elements
  - OLAP Schema Elements
  - Record Oriented Legacy Databases
- Knowledge management Model
  - Schematic Definition Elements

UML part and Database and Data Warehousing part seem to be usable for EXC. If EXC uses the UML part, there is a need to resolve the difference between EXPRESS and UML. For UML encoding, the XMI (XML Metadata Interchange) that was developed by OMG seems to be better than OIM. The database schema elements part (DBM) seems to be close to database table definition rather than neutral meta data definition. OIM DBM includes following information:

- Schema definition
- Table definition
- Column (attribute) type
- Data type
- Other database schema definition (nullable, sort key, etc)

Microsoft SQL Server 2000 has an important functionality that can import OIM data within its Meta Data Services. OIM of database schema elements seems to be suitable for the secondary XML meta model format rather then intermediate XML meta model format.
Following XML data shows a sample encoding of OIM DBM:

```
<?xml versio ="1.0" ?>
  <oim:Tra sfer xml s:oim= "http://www.mdcinfo.com/oim/oim.dtd"
                xml s:dbm= "http://www.mdcinfo.com/oim/dbm.dtd" >
    <dbm:Catalog id= "_1"  ame= "sales"  comme ts= "Sample catalog" >
      <dbm:CatalogSchemas>
        <dbm:Schema id= "_2"  ame= "dbo" >
        <dbm:SchemaTables>
          <dbm:Table id= "_3"  ame= "Customer" >
            <dbm:Colum SetColum s>
              <dbm:Colum  id= "_6"  ame= "CustomerID"  IsNullable= "0"  />
              <dbm:Colum  id= "_7"  ame= "Name"  IsNullable= "0"  />
              <dbm:Colum  id= "_8"  ame= "Address"  IsNullable= "1"  />
              <dbm:Colum  id= "_9 "  ame= "Phone"  IsNullable= "1"  />
            </dbm:Colum SetColum s>
          </dbm:Table>
          <dbm:Table id= "_4"  ame= "Order"  stimatedRows= "10000" >
            <dbm:Colum SetColum s>
              <dbm:Colum  id= "_10"  ame= "CustomerID"  IsNullable= "0"  />
              <dbm:Colum  id= "_11"  ame= "OrderID"  IsNullable= "0"  />
              <dbm:Colum  id= "_12"  ame= "Date"  IsNullable= "1"  />
            </dbm:Colum SetColum s>
          </dbm:Table>
          <dbm:Table id= "_5"  ame= "OrderItem"  stimatedRows= "100000" >
            <dbm:Colum SetColum s>
```

```
                <dbm:Colum  id= "_13"   ame= "CustomerID"  IsNullable= "0"  />
                <dbm:Colum  id= "_14"   ame= "OrderID"  IsNullable= "0"  />
                <dbm:Colum  id= "_15"   ame= "LineNo"  IsNullable= "0"  />
                <dbm:Colum  id= "_16"   ame= "Description"  IsNullable= "1"  />
                <dbm:Colum  id= "_17"   ame= "Quantity"  IsNullable= "0"  />
                <dbm:Colum  id= "_18"   ame= "UnitPrice"  IsNullable= "0"  />
              </dbm:Colum SetColum s>
              <dbm:TableU iqueKeys>
                <dbm:U iqueKey id= "_19"   ame= "PK_OrderItem"  IsPrimary= "1" >
                  <dbm:KeyColum s>
                    <dbm:Colum  href= "#_14"  />
                    <dbm:Colum  href= "#_15"  />
                  </dbm:KeyColum s>
                </dbm:U iqueKey>
              </dbm:TableU iqueKeys>
            </dbm:Table>
          </dbm:SchemaTables>
        </dbm:Schema>
        </dbm:CatalogSchemas>
      </dbm:Catalog>
    </oim:Tra sfer>
```

2.3 XML Metadata Interchange (XMI)

The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG UML 1.1) and metadata repositories (OMG MOF, Meta Object Facility based) in distributed heterogeneous environments. UNISYS, IBM, Oracle, Rational Software and Fujitsu mainly contributed to develop XMI specification and XMI has already implemented in their products for exchanging UML models.

XMI has a very similar capability with STEP Part28 EXPRESS representation in XML. Basically, the Class element equals to Entity declaration element of STEP Part28. XMI mainly includes following meta data:

- Class type
- Class inheritance
- Data type (includes Corba data type)
- Attribute type
- Other UML information (association, containment, etc)

XMI specification is written by DTD.
Following XML data shows a very conceptual XMI sample that describes a fragment of a UML model:

```
<Model xmi.id="i00000001">
       <name>model1</name>
       <ownedElement>
              <Class xmi.id="i00000002">
                     <name>class1</name>
                     <feature>
                            <Attribute xmi.id="i00000003">
                                   <name>attribute1</name>
                                   <type><integer/></type>
                            </Attribute>
                     </feature>
              </Class>
       </ownedElement>
</Model>
```

2.4 ifcXML

ifcXML is defined by IAI Model Support Group lead Thomas Liebich. ifcXML defines a XML encoding method of IFC model data and also can use for meta data of IFC model. Therefore ifcXML is also considered to have a very similar feature of STEP Part 28. ifcXML is written by XML Schema Definit ion (XSD) that is defined by W3C. ifcXML mainly includes following information:

- Entity type
- Attribute type
- Data type (basically within XSD)
- Enumeration
- Select type
- Entity inheritance
- Entity reference type

The following ENTITY definitions would be mapped as:

```
ENTITY IfcAddress
    ABSTRACT SUPERTYPE OF (ONEOF(IfcPostalAddress, IfcTelecomAddress));
      Purpose           : OPTIONAL STRING;
      Description       : OPTIONAL STRING;
END_ENTITY

ENTITY IfcPostalAddress
  SUBTYPE OF(IfcAddress);
      AddressLines     : OPTIONAL LIST [1:?] OF STRING;
      Town           : OPTIONAL STRING;
      PostalCode       : OPTIONAL STRING;
END_ENTITY;

ENTITY IfcTelecomAddress
  SUBTYPE OF(IfcAddress);
      TelephoneNumbers        : OPTIONAL LIST [1:?] OF STRING;
      FacsimileNumbers        : OPTIONAL LIST [1:?] OF STRING;
END_ENTITY;
```

The XML schema definition should be defined as:

```
<xsd:element name="Address" type="AddressType" abstract="true" />
</xsd:element>

<xsd:complexType name="AddressType" abstract="true">
  <xsd:sequence>
      <xsd:element name="Purpose" type=" xsd:string " minOccurs="0" />
      <xsd:element name="Description" type="xsd:string" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="xID" type="xsd:ID" use="required"/>
</xsd:complexType>

<xsd:element name="PostalAddress" type="PostalAddressType" substitutionGroup="Address">
</xsd:element>

<xsd:complexType name="PostalAddressType">
  <xsd:complexContent>
     <xsd:extension base="AddressType">
       <xsd:sequence>
          <xsd:element name="AddressLines" type="xsd:string" minOccurs="0"
maxOccurs="unbounded" />
        <xsd:element name="Town" type="xsd:string" minOccurs="0" />
          <xsd:element name="PostalCode" type="xsd:string" minOccurs="0" />
       </xsd:sequence>
     </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="TelecomAddress" type="TelecomAddressType"
substitutionGroup="Address">
</xsd:element>

<xsd:complexType name="TelecomAddressType">
   <xsd:complexContent>
     <xsd:extension base="AddressType">
       <xsd:sequence>
          <xsd:element name="TelephoneNumbers" type="xsd:string" minOccurs="0"
maxOccurs="unbounded" />
          <xsd:element name="FacsimileNumbers" type="xsd:string" minOccurs="0"
maxOccurs="unbounded" />
```

```
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

2.5 BLIS-XML

BLIS-XML methodology has been developed by BLIS project that is an IFC implementation project. The BLIS-XML for IFC R2.0 is an official XML schema for IFC R2.0. BLIS-XML can be used for XML encoding of IFC model instance data like Part 21, and also has a limited capability of the dictionary for IFC data structure. In another word, BLIS-XML schema can be used for creating database schema. IFCsvr ActiveX Component uses BLIS-XML schema as model dictionary for exporting IFC model data by BLIS-XML format.

XML Data Reduced (XDR) that is mainly developed by Microsoft is used for BLIS-XML for IFC R2.0 schema. MSXML, this is a DOM and SAX parser component developed by Microsoft, uses XDR data for validation of XML instance data.

BLIS-XML methodology is similar to STEP Part28 OSEB. The class inheritance information in EXPRESS is not mapped into BLIS-XML. The attribute definitions that are defined in super classes is mapped the attributes in the leaf class element, not in the child elements. The attributes that are SELECT Type and aggregation of simple type are mapped into child element.

BLIS-XML mainly contains following information:

- Entity type
- Attribute type
- Data type (basically within XDR), EXPRESS types do not keep in XDR.
- Enumeration
- Select type

Here is an example of ENTITY definition and its BLIS-XML encoding format:

```
ENTITY IfcNamedUnit
  ABSTRACT SUPERTYPE OF (ONEOF(
    IfcContextDependentUnit
   ,IfcConversionBasedUnit
   ,IfcSiUnit));
    Dimensions : IfcDimensionalExponents;
    UnitType   : IfcUnitEnum;
  WHERE
    WR1: IfcCorrectDimensions (SELF.UnitType, SELF.Dimensions);
END_ENTITY;

ENTITY IfcSiUnit
  SUBTYPE OF (IfcNamedUnit);
    Prefix : OPTIONAL IfcSiPrefix;
    Name   : IfcSiUnitName;
  DERIVE
    SELF¥IfcNamedUnit.Dimensions : IfcDimensionalExponents
              := IfcDimensionsForSiUnit (SELF.Name);
END_ENTITY;
```

In BLIS-XML schema, IfcSiUnit element definition is:

```
<AttributeType name="XMLID" dt:type="id" />
<AttributeType name="PrefixTitles" dt:type="string" />
…
<ElementType name="IfcSiUnit" content="empty">
  <AttributeType name="Name" dt:type="enumeration" dt:values="METRE SQUARE_METRE
CUBIC_METRE GRAM SECOND AMPERE KELVIN MOLE CANDELA RADIAN STERADIAN HERTZ NEWTON PASCAL
JOULE WATT COULOMB VOLT FARAD OHM SIEMENS WEBER TESLA HENRY DEGREE_CELSIUS LUMEN LUX BECQUEREL
GRAY SIEVERT" required="yes" />
  <attribute type="Name" />
  <attribute type="XMLID" required="yes" />
  <attribute type="UnitType" required="yes" />
  <attribute type="Prefix" required="yes" />
</ElementType>
```

An IFC instance data of BLIS-XML is shown below:

```
<IfcSiUnit XMLID="i821" UnitType="LengthUnit" Prefix="MILLI" Name="METRE" />
```

3. Conclusion

To determine the intermediate XML meta model format, considering about following conditions:

- Easiness of EXPRESS parser development
- Compatibility with EXPRESS schema information
- Easiness of conversion for other meta model format

Table 1 shows a comparison of data category importance between the intermediate XML meta model format and secondary meta model format, additionally supported data categories in each XML format.

Table 1. Data category ranks between intermediate and secondary XML meta data format

Data Categories in EXPRESS	Intermediate XML meta format	Secondary XML meta format	Has definition
Entity	A	A	1,2,3,4,5
Attribute	A	A	1,2,3,4,5
Data type	A	A	1,2,3,4,5
Select type	A	A	1,4,5
Reference type	A	B	1,2,3,4
Enumeration	A	A	1,3,4,5
Aggregate	A	B	1,3,4,5
Inverse	B	D	1
Derive	B	D	1
Inheritance	A	B	1,3,4
Rule	B	D	1
Function	C	D	1
Schema reference	A	C	1

Legend: rank of importance
- A: Necessity
- B: Desirable
- C: Not so important
- D: Not necessary

XML formats:

1.	Part28
2.	OIM
3.	XMI
4.	ifcXML
5.	BLIS-XML

The late binding EXPRESS markup declaration in STEP Part 28 seems to be most likely for the intermediate XML meta model format by following reasons:

1. Easiness of development for EXPRESS parser. The element structure is very similar to EXPRESS syntax.
2. Compatibility with EXPRESS schema information. It contains full information of EXPRESS schema.
3. The late binding approach has a single markup declaration. This shall make development of converter easy, because the markup declaration can be used for any EXPRESS schema.

Here are some comments about other XML formats:

1. XMI seems to be another candidate of the intermediate XML meta model format, because XMI has a strong functionality to describe the meta model framework. Additionally, XMI is based on UML, therefore it is very neutral and many meta model design tools support XMI. XMI should be a target of secondary XML meta model format and develop the converter that transforms the intermediate XML data to XMI.
2. OIM's database element part (namespace is dbm) is suitable for secondary XML meta data format. SQL Server 2000 can import and export the database schema by OIM:dbm.
3. ifcXML has enough schema information for making database schema. IfcXML is suitable for secondary.
4. BLIS has minimum schema information for making database schema. This is suitable for secondary.

4. References

1) Product data representation and exchange: Implementation methods: XML representation of EXPRESS schemas and data, ISO TC184/SC4/WG11 N140 Date: 2000-10-06
ISO/PDTS 10303-28
URL: http://xml.coverpages.org/stepExpressXML.html

2) Open Information Model XML Encoding, Meta Data Coalition,
Version 1.0 Review Draft 2, December 1999
URL: http://xml.coverpages.org/mdc-oim.html

3) XML Metadata Interchange (XMI), Proposal to the OMG OA&DTF RFP 3:
OMG Document ad/98-10-05, October 20, 1998
URL: http://www.omg.org/xml/
http://www-4.ibm.com/software/ad/library/standards/xmi.html

4) XML schema language binding of EXPRESS for ifcXML,
IAI-MSG Thomas Liebich, 2001-03-08
URL: http://cig.bre.co.uk/iai_uk/IFCXML.htm

5) BLIS-XML Methodology for Transforming EXPRESS data Models to XDR, BLIS Project Companies, November 2000
URL: http://cic.vtt.fi/projects/blis/BLIS_XML/index.htm