From Model to Markup

XML Representation of Product Data

Joshua **Lubell** <lubell@nist.gov>

Abstract

Business-to-consumer and business-to-business applications based on the Extensible Markup Language (XML) tend to lack a rigorous description of product data, the information generated about a product during its design, manufacture, use, maintenance, and disposal. ISO 10303, also informally known as the Standard for the Exchange of Product model data (STEP), defines a robust and time-tested methodology for describing product data throughout the lifecycle of the product. I discuss some of the issues that arise in designing an XML exchange mechanism for product data and compare different XML implementation methods currently being developed for STEP.

Table of Contents

Note

Commercial equipment and materials are identified in order to describe certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose. Unified Modeling Language, UML, Object Management Group, and other marks are trademarks or registered trademarks of Object Management Group, Inc. in the U.S. and other countries.

1. Product Data and XML

Although various standards groups and consortia are busy developing XML specifications for electronic business applications, these XML vocabularies often focus more on the business processes themselves than on the function and structure of goods being bought or sold. Business-to-consumer and business-to-business applications based on the Extensible Markup Language (XML) [W3C-XML] therefore tend to lack a rigorous description of product

data. This data, which includes the information generated about a product during its design, manufacture, use, maintenance, and disposal, is essential to the success of any enterprise looking to minimize production costs, improve quality, or reduce maintenance expenses. A standard XML representation of product data would go a long way toward enabling truly interoperable Web-based product design, manufacturing, and lifecycle support applications.

ISO 10303, also informally known as the Standard for the Exchange of Product model data (STEP) [ISO10303-1] [Kemmerer], is a family of standards defining a robust and time-tested methodology for describing product data throughout the lifecycle of the product. STEP is widely used in Computer Aided Design (CAD) and Product Data Management (PDM) systems. Major aerospace and automotive companies have proven the value of STEP through production implementations [PDES]. But STEP is not an XML vocabulary. Product data models in STEP are specified in EXPRESS (ISO 10303-11) [ISO10303-11] [Schenk], a modeling language combining ideas from the entity-attribute-relationship family of modeling languages with object modeling concepts.

To take advantage of XML's popularity and flexibility, and to accelerate STEP's adoption and deployment, the ISO group responsible for STEP is developing methods for mapping EXPRESS schemas and data to XML. The rest of this paper discusses some of the issues that arise in designing an XML exchange mechanism for product data. It also compares two implementation approaches: a direct mapping from EXPRESS to XML syntax rules, and an indirect mapping by way of the Unified Modeling Language™ (UML).

2. Encoding STEP Data as XML

As a simple example of how STEP data might be encoded in XML, consider the following EXPRESS definition of a product. This definition declares **product** to be an *entity type* and **name**, **quantity**, **description**, and **frame_of_reference** to be *attributes* (EXPRESS attributes, not XML attributes) of that entity type. A product's **name** and **description** are strings, its **quantity** is an integer, and its **frame_of_reference** is a set of one or more **product_context** EXPRESS entities (I will define **product_context** later). The **description** is optional; all other EXPRESS attributes are required.

```
ENTITY product;
   name : STRING;
   quantity : INTEGER;
   description : OPTIONAL STRING;
   frame_of_reference : SET[1:?] OF product_context;
END_ENTITY;
```

Now consider an XML representation of a particular product. The **product** entity type and its EXPRESS attributes are represented as elements. I use XML elements rather than XML attributes to represent EXPRESS attributes for the following reasons:

- XML elements can contain other markup, making it easier to represent EXPRESS attributes such as **frame_of_reference** whose values are aggregates of EXPRESS entity instances.
- Using XML elements to represent EXPRESS attributes guarantees that names of XML attributes used to represent metadata, such as the <Product> element's **id** attribute, will not conflict with EXPRESS attributes.

The **id** attribute, which uniquely identifies the product, does not correspond to any EXPRESS construct.

```
<Product id="p1">
  <name>Wonder Widget</name>
  <quantity>3</quantity>
  <frame_of_reference>
    ...

  </frame_of_reference>
</Product>
```

The preceding XML representation is an *early binding* in that the tag names correspond directly to their counterparts in the EXPRESS. In a *late binding*, the named components of the XML vocabulary do not directly correspond to EXPRESS names. Instead of defining EXPRESS names as tags, a late binding specifies them in the data either as

XML attribute values or as element content. For example, a late-bound XML representation of the "Wonder Widget" product with EXPRESS names specified as XML attribute values might look like this:

```
<entity id="p1" name="Product">
  <attribute name="name">
    <string_value>Wonder Widget</string_value>
  </attribute>
  <attribute name="quantity">
    <integer_value>3</integer_value>
  </attribute>
  <attribute name="frame_of_reference">
    ...

  </attribute>
</entity>
```

Although late bindings are more verbose than early bindings, a late-bound EXPRESS-to-XML mapping is better suited for XML applications involving multiple EXPRESS schemas. A late binding allows for a single tag set to be used for all EXPRESS models, since the XML vocabulary defined by the tag set corresponds to EXPRESS metadata objects rather than to objects defined in the model. If an early-bound strategy is used for such applications, there must be a distinct XML language for each EXPRESS schema. Early bindings are therefore most useful for XML applications implementing a single EXPRESS schema. Early bindings are less verbose, more human-readable, and simpler to process than late bindings, and they are also better equipped to make use of XML software tools. As a result, STEP implementers tend to prefer early bindings, and I focus exclusively on early bindings for the remainder of this discussion.

3. Representation Issues

The following are some issues that arise when attempting to reformulate EXPRESS models as XML. This enumeration of issues is not exhaustive; there are various other challenges involved in representing EXPRESS-modeled product data as XML. Two additional issues that are particularly thorny are the representation of EXPRESS attributes whose values are aggregates (lists, sets, arrays, or bags) and the handling in XML of multiple inheritance and other complex inheritance relationships in EXPRESS. In order to keep this paper brief, I do not discuss the latter issue. The former issue, discussed at length in [Barkmeyer], makes it a problem to represent EXPRESS attributes as XML attributes. Thus, I map EXPRESS attributes to XML elements in my examples in this discussion.

3.1. XML Schema Languages

To specify an XML binding of data modeled in EXPRESS, you must provide an algorithm for mapping EXPRESS constructs to their XML counterparts. The algorithm's input is an EXPRESS schema. Its output is a set of syntax rules for the XML binding's vocabulary. These rules may be specified using any one of the several schema languages available for XML. For example, a specification for **product** using Document Type Definition (DTD) syntax is as follows:

```
<!ELEMENT Product (name, quantity, description?, frame_of_reference)>
<!ATTLIST Product
          id ID #IMPLIED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT frame_of_reference
          ...

>
```

These markup declarations show some of the shortcomings of DTD syntax. Because DTD elements cannot be assigned data types other than #PCDATA (parsed character data), a DTD cannot enforce the constraint that the product's quantity be an integer. Also, DTDs do not permit multiple declarations for the same element. For example,

if our EXPRESS schema declared, in addition to **product**, an entity type called **quantity**, we would be unable to add a second element declaration for **quantity** to our DTD.

A more sensible approach is to specify the binding using an XML schema language without these limitations of DTD syntax, such as the World Wide Web Consortium's (W3C) XML Schema definition language [W3C-XS1][7] or RELAX NG [OASIS-RNG]. These schema languages can define context-sensitive element types and provide a much more extensive collection of built-in data types [W3C-XS2]than DTDs.

It is worth noting that even W3C XML Schema and RELAX NG, with their data typing abilities and support for context-sensitive elements, lack the ability to describe all of the constraints that EXPRESS is capable of representing. For example, these schema languages cannot represent global constraints on a population, such as the requirement that all points on a plane be collinear. Although it is possible to develop an XML language that represents all EXPRESS constraints, an application using that language would have to supply its own logic for interpreting any constraints that could not be validated using a grammar-based schema language or a pattern-based language such as the Schematron assertion language [Jelliffe]. Generic XML software would be of no help. Thus, for all practical purposes, any EXPRESS-to-XML mapping will result in a loss of information. However, the XML representation is able to retain at least *some* of the information present in the EXPRESS schema. For some use cases, the XML schema and data may be all that is needed. Other use cases may require that additional (unmapped) information from the EXPRESS schema be hard-coded into the application responsible for processing the XML-encoded product data.

3.2. Hierarchies

Although XML is hierarchical by nature, EXPRESS product data models lack any intrinsic hierarchical structure. Although EXPRESS provides a rich vocabulary for representing "is-a" relationships between entity types, it lacks a means for explicitly representing part-whole relationships between entity instances. Nothing in an EXPRESS schema identifies an entity instance as part of, or belonging to, a higher-level entity. There are no clues in the EXPRESS schema to suggest which entity types can be mapped to hierarchical constructs in XML. An EXPRESS schema is a network of largely independent entity types connected by relationships modeled using attributes whose values are entity instances.

Accordingly, an EXPRESS data set can be mapped into a sequence of XML elements representing EXPRESS entity instances. Each entity element has an XML ID attribute, providing a unique identifier for the entity instance (as in the point example in the Background section). Assume that EXPRESS attributes are mapped to XML elements rather than XML attributes. Then each element representing an EXPRESS entity type contains elements representing the EXPRESS entity's attributes. In addition, every entity type also maps to a reference element - an XML element type with an XML attribute of type IDREF and empty content. An occurrence of this element represents a reference to the entity instance with the given ID value. In short, the XML data consists of a collection of "flat" elements representing independent entity instances linked together by IDREFs, because no hierarchical structure can be deduced from the EXPRESS model.

For example, consider a **product_context** EXPRESS entity type modeled as follows:

```
ENTITY product_context
  SUBTYPE OF (application_context_element) ;
   discipline_type : STRING;
END_ENTITY;
```

A Wonder Widget with multiple product contexts might look like this in XML:

```
<Product>
  <name>Wonder Widget</name>
  <quantity>3</quantity>
  <frame_of_reference>
    <Product_context-ref xref="pc1"/>
    <Product_context-ref xref="pc2"/>
    <Product_context-ref xref="pc3"/>
  </frame_of_reference>
</Product>
```

```
<Product_context id="pc1">
  <discipline_type>mechanical</discipline_type>
  ...

</Product_context>
<Product_context id="pc2">
  <discipline_type>electrical</discipline_type>
  ...

</Product_context>
<Product_context id="pc3">
  <discipline_type>chemical</discipline_type>
  ...

</Product_context>
```

To make better use of XML's inherent hierarchical structure, you might wish to use a representation method allowing entity instances to be contained within other instances. Such an XML representation method requires some formal annotation of EXPRESS schemas to cue the mapping to useful hierarchical data structures in XML. Some of these annotations might belong to the EXPRESS schema itself, while others might be specific to particular STEP implementations using the schema [Fritz].

An XML representation of a Wonder Widget containing **product_context** entity instances could look something like this (omitting the <Product_context> element content for now):

```
<Product>
  <name>Wonder Widget</name>
  <quantity>3</quantity>
  <frame_of_reference>
    <Product_context id="pc1">
      <discipline_type>mechanical</discipline_type>
      ...

    </Product_context>
    <Product_context id="pc2">
      <discipline_type>electrical</discipline_type>
      ...

    </Product_context>
    <Product_context id="pc3">
      <discipline_type>chemical</discipline_type>
      ...

    </Product_context>
  </frame_of_reference>
</Product>
```

3.3. Inheritance

The EXPRESS specification for **product_context** in the Section 3.2 subsection defines **product_context** to be a subtype of **application_context_element**. Suppose **application_context_element**'s EXPRESS specification is as follows:

```
ENTITY application_context_element
  SUPERTYPE OF (product_context) ;
   name : STRING;
   frame_of_reference : application_context;
END_ENTITY;
```

Thus **product_context** inherits **application_context_element**'s EXPRESS attributes, and any **product_context** instance is also an instance of **application_context_element**. This can be represented using W3C XML Schema

by making a complex type corresponding to **application_context_element** and extending that type to create a complex type corresponding to **product_context**. However, because W3C XML Schema supports only single inheritance of types, type extension cannot be used to represent multiple inheritance or other complex inheritance relationships often used in EXPRESS models.

A more general method for representing inheritance in an XML schema — one that addresses the needs of EXPRESS inheritance — is to simulate the inheritance by using grouping constructs. This technique is suitable not only for representing complex inheritance but is also useful for XML schema languages lacking type extension support. For example, using RELAX NG's compact syntax [OASIS-RNG-compact], one can define a representation for **application_context_element** and **product_context** as follows:

```
id.att = attribute id { xs:ID }*
ref.att = attribute xref { xs:IDREF }

Application_context_element.elt =
  (element Application_context_element { id.att,
    Application_context_element.class }) |
  Product_context.elt
Application_context_element.ref =
  Product_context.ref |
  (element Application_context_element-ref { ref.att })
Application_context_element.class =
  (element name { string }) &
  (element frame_of_reference
    { Application_context.elt | Application_context.ref})

Product_context.elt =
  element Product_context { id.att, Product_context.class }
Product_context.ref =
  element Product_context-ref { ref.att }
Product_context.class =
  Application_context_element.class &
  (element discipline_type { string })
```

The preceding approach, which allows for either flat or hierarchical product XML data, supplies three RELAX NG definitions for each EXPRESS entity type (a RELAX NG definition is analogous to a W3C XML Schema <group> element or a DTD parameter entity). These definitions are:

entity-name.class
: Specifies the EXPRESS entity's EXPRESS attributes as an unordered sequence of elements such that they can be used in the entity's element type or in any element type corresponding to a subtype of the entity. For example, since **product_context** inherits **application_context_element**'s EXPRESS attributes, **Product_context.class** combines **Application_context_element.class** with an element representing **product_context**'s **discipline_type** attribute. If an EXPRESS attribute's type is an EXPRESS entity type or an aggregation of entity types, the EXPRESS attribute maps to an element whose content provides a choice between containment and reference. For example, EXPRESS entity **application_context_element**'s **frame_of_reference** attribute maps to an element whose content is a choice between <Application_context_element> and <Application_context_element-ref>.

entity-name.elt
: Specifies an element type corresponding to the EXPRESS entity. If the entity has no subtypes, the element type consists of an ID attribute plus the entity's corresponding ".class" definition. If the entity has one or more subtypes, then its element type consists of an ID attribute plus a choice between the ".class" definitions for the entity and each subtype. For example, since **product_context** is a subtype of **application_context_element**, a **product_context** is also an instance of **application_context_element**. Therefore, I define **Application_context_element.elt** to be a choice between the **application_context_element** element type and the **product_context** element type.

entity-name.ref — Specifies the EXPRESS entity's reference element. If the entity has no subtypes, the reference element is <*entity-name*-ref>. If the entity has subtypes, the reference element can be either <*entity-name*-ref> or the ".ref" definition for any subtype. For example, the reference element for **application_context_element** is a choice between the **application_context_element** reference element and the **product_context** reference element.

The following XML instance of **product_context** is valid with respect to these RELAX NG definitions:

```
<Product_context id="pc1">
  <name>Mechanical 3d Parts</name>
  <discipline_type>mechanical</discipline_type>
  <frame_of_reference>
    <Application_context-ref xref="ac1"/>
  </frame_of_reference>
</Product_context>
```

3.4. Inverse Relationships

It is quite common in product data models for two instances to point to each other. In EXPRESS, an attribute whose value consists of instances, which in turn have attributes pointing to the attribute's entity, is called an *inverse* attribute. For example, in the following EXPRESS specification for **application_context**, the attribute **context_elements** is an inverse for **application_context_element**'s **frame_of_reference** attribute.

```
ENTITY application_context;
   application : STRING;
  INVERSE
    context_elements : SET[1:?] OF application_context_element
                       FOR frame_of_reference;
END_ENTITY;
```

What this means is that:

- An **application_context** instance has a set of one or more context elements.
- These context elements may be either **application_context_element** or **product_context** instances (because **product_context** is a subtype of **application_context_element**).
- The value of each context element's **frame_of_reference** attribute must be the **application_context** instance.

An XML representation of an EXPRESS schema may optionally include an inverse EXPRESS attribute. If the inverse is not included, an XML application can compute its value if necessary. The decision whether to include an inverse is a question of whether the benefit of not having to compute the value every time it is needed is worth increasing the complexity of the XML data. If an EXPRESS attribute and its inverse attribute are both mapped to XML such that the inverse is contained within the EXPRESS attribute, the element corresponding to the inverse attribute *must* allow reference elements as its content. Otherwise, there would be a never-ending nesting of elements. For example, if **application_context**'s **context_elements** attribute were represented in XML by an element with content limited to <Application_context_element> and <Product_context> elements (and not allowing <Application_context_element-ref> or <Product_context-ref>), the resulting XML data would look like this:

```
<Product_context>
  <name>Mechanical 3d Parts</name>
  <frame_of_reference>
    <Application_context>
      <application>configuration controlled 3d design</application>
      <context_elements>
        <Product_context>
          <name>Mechanical 3d Parts</name>
          <frame_of_reference>
            <Application_context>
```

```
            <application>configuration controlled 3d design</application>
            ...
```

4. Implementation Issues

Apart from the question of how to represent product data in XML is the issue of how best to implement the EXPRESS-to-XML mapping. A key requirement here is to provide a means for users of an implementation to specify information needed for the mapping that cannot be inferred from the EXPRESS alone. As I discussed in the Section 3 section, this includes information such as which EXPRESS entities can only exist as a part of another entity and whether inverted EXPRESS attributes should be included in the XML. The ISO group responsible for STEP is considering the following two approaches.

4.1. Direct Mapping from STEP/EXPRESS to XML

A proposed STEP standard, ISO 10303-28 (a.k.a. Part 28) [SC4N194], will specify how to directly map any EXPRESS schema to an XML schema. The generated XML schema defines an XML vocabulary for representing STEP data conforming to the EXPRESS schema. This XML representation of the STEP data can then be processed and validated using XML tools. Input to the Part 28 algorithm is an EXPRESS schema supplemented with user-supplied configuration data. Figure 1 illustrates the Part 28 conversion process.

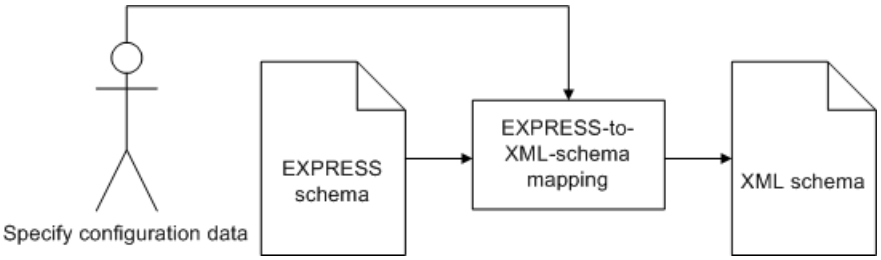

Figure 1. Converting an EXPRESS schema into an XML schema using Part 28

The configuration data, specified using an XML format, supplies information such as:

- Which EXPRESS entity instances should be contained within other entity instances.
- Whether inverted EXPRESS attributes should be included in the XML mapping.
- The mapping between EXPRESS simple types (INTEGER, REAL, BINARY, etc.) and XML Schema simple types.
- Which EXPRESS attributes, if any, should map to XML attributes.
- Which EXPRESS entities or attributes, if any, should be renamed.

An initial version of Part 28 was approved by ISO as a Technical Specification earlier this year. However, this version uses DTD syntax to specify mappings of EXPRESS to XML, resulting in a sub-optimal solution. Furthermore, it specifies three different XML mapping algorithms: a late binding method and two early binding methods. As a result, Part 28 edition 1 is a lengthy and complex document of more than 300 pages. Recognizing the limitations of the first edition, ISO has begun work on the second edition of Part 28 employing W3C XML Schema. The Part 28 edition 2 EXPRESS-to-XML-schema mapping and configuration language are still under development. Progress has been slower than anticipated — both because XML representation of EXPRESS schemas and data is a non-trivial undertaking, and also because of a shortage of software tools capable of processing EXPRESS and people with EXPRESS expertise. Paradoxically, the chief motivator for representing STEP product models in XML — EXPRESS's niche status and weak software tool support relative to that of XML — is also a barrier to achieving this goal.

4.2. Mapping Indirectly by Way of UML

Another proposed STEP standard, ISO 10303-25 (a.k.a. Part 25) [SC4N169], defines a mapping from EXPRESS to static structure (class) diagrams specified using the Unified Modeling Language (UML) [OMG-UML], a graphical language standardized by the Object Management Group™ (OMG) for modeling software systems. UML, unlike EXPRESS, is widely used and enjoys widespread software support. However, like EXPRESS (and unlike XML), UML is first and foremost a modeling language. UML has many, but not all, of the capabilities of EXPRESS. UML also has some capabilities needed relevant to XML schema development that EXPRESS lacks, such as extensibility and the ability to distinguish between association and composition relationships. Although Part 25's primary purpose is to enable EXPRESS schemas to be processed using UML tools, it can also serve as an attractive alternative to Part 28 for generating an XML schema from an EXPRESS schema. This is because UML models are exchanged using the XML Metadata Interchange (XMI) [OMG-XMI] syntax.

XMI is an XML vocabulary that permits ASCII-based exchange of metadata between UML modeling tools. XMI's elements and attributes reflect the UML metamodel rather than any particular UML model. Thus, for an EXPRESS schema represented as a UML class diagram, the XMI representation of the diagram is late-bound with respect to the EXPRESS schema definitions. Although XMI data is extremely verbose and is not intended for humans to read, an XMI representation of a UML model can be transformed into an XML schema that is early-bound with respect to the EXPRESS schema definitions. In fact, the XMI specification includes production rules for obtaining a schema (actually a DTD) from an XMI-encoded UML metamodel.

Building upon the XMI specification's production rules, some individuals in the UML community have developed their own mappings from the UML to W3C XML Schema taking advantage of UML stereotypes, tagged values, and constraints to fine-tune the mapping [Carlson] [Provost]. Stereotypes, tagged values, and constraints are UML facilities that enable modelers to add metadata to their models, serving a similar purpose to the configuration language proposed for Part 28. Carlson's mapping has been implemented in *hyperModel*, a Web-based tool [XMLmodeling] that transforms XMI into both W3C XML Schema and RELAX NG. I used the output from hyperModel as a starting point for developing the RELAX NG definitions in this paper.

Figure 2 illustrates the conversion of an EXPRESS schema into an XML schema using Part 25 plus UML tools. First the EXPRESS schema is converted into XMI. The XMI is then read by a UML modeling tool and, using the UML tool, a user adds metadata as needed to optimize the mapping to an XML schema. Finally, an XMI-to-XML-schema tool converts the annotated UML model into an XML schema.

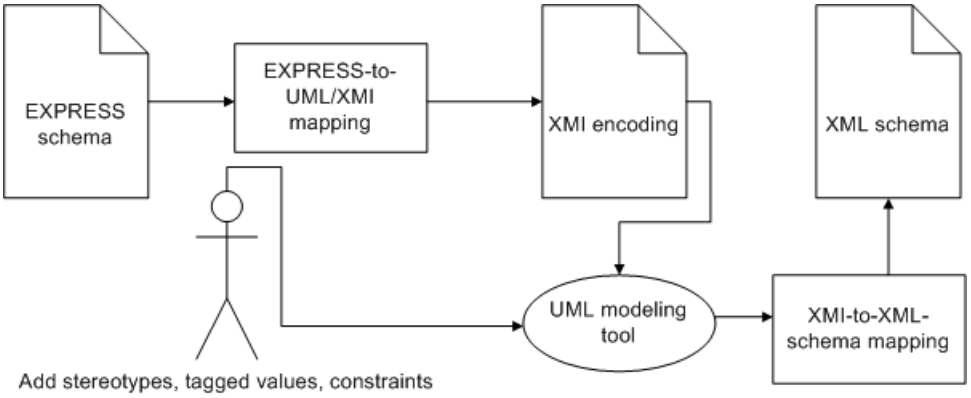

Figure 2. Converting an EXPRESS Schema into an XML Schema using Part 25 and UML Tools

Although Figure 2 is more complicated-looking than Figure 1, EXPRESS-unaware UML tools can perform most of the work. The only part of the process that has to handle EXPRESS is the EXPRESS-to-UML/XMI mapping. This approach is an appealing alternative to the Part 28 edition 2 approach because it takes advantage of existing and/or emerging UML tools, reducing the need to develop EXPRESS-specific tools (and to rely on scarce EXPRESS expertise) to design and implement the mapping.

To get a sense of how UML can eliminate the need to develop a separate language for configuring an EXPRESS-to-XML mapping, consider the UML diagrams in Figure 3 and Figure 4. Both diagrams represent a

Product_identification EXPRESS schema consisting of the **product**, **product_context**, **application_context_element**, and **application_context** entity types as UML classes. In both diagrams, EXPRESS attributes whose values are simple types are represented as UML attributes, and the subtype relationship between **product_context** and **application_context_element** is represented using a generalization link (the arrow with the triangular head). In Figure 3, a "flat" representation of the EXPRESS schema, the **context_elements** and **frame_of_reference** EXPRESS attributes are represented as associations without any added semantics. In Figure 4, a hierarchical representation of the EXPRESS, the associations include solid diamonds, which indicate that **product_context** is contained within **product** and **application_context** is contained within **application_context_element** and **product_context** (because **product_context** is a subtype).

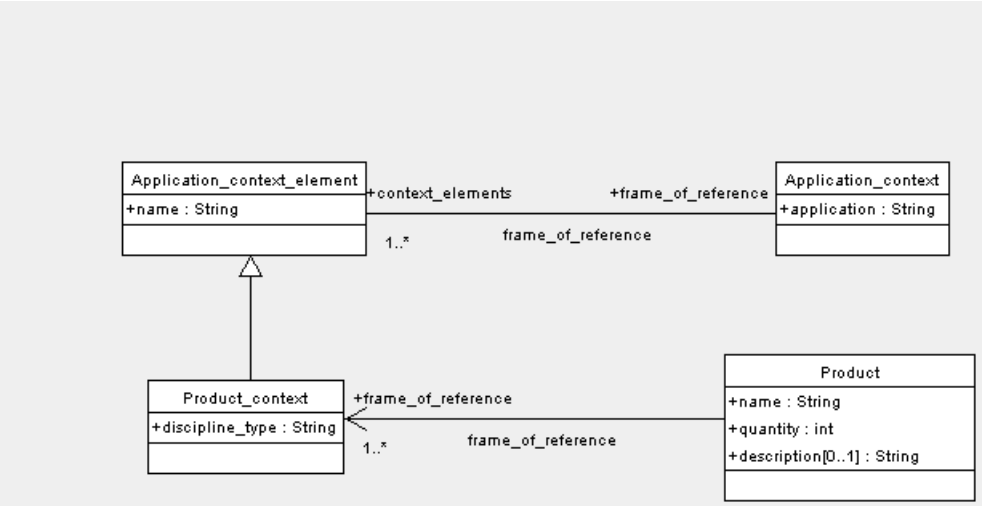

Figure 3. "Flattened" UML Class Diagram for Product_identification EXPRESS Schema

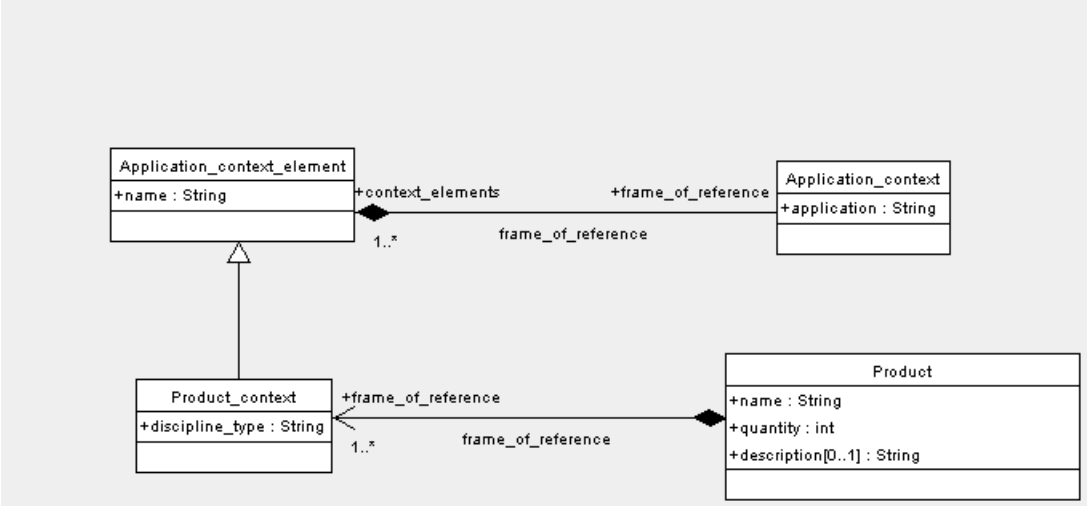

Figure 4. UML Class Diagram for Hierarchical Representation of Product_identification EXPRESS Schema

Like Part 28, Part 25 is a work in progress. An initial Draft Technical Specification was recently defeated by ISO. The ballot comments included concerns about Part 25's scope and conformance criteria being ambiguous [SC4N1356]. None of the ballot comments, however, were critical of the general strategy of mapping EXPRESS to UML.

5. Observations

Developers of mappings from product data models to XML have learned some valuable lessons. The most important lesson is that, although XML makes for an excellent data exchange syntax, XML is *not* well suited for information modeling. After all, if XML were a good modeling language, it would be much easier to develop a robust mapping from EXPRESS to XML. UML, on the other hand, *is* a good modeling language and, therefore, it should be easier to map EXPRESS to UML than to XML. Although mapping EXPRESS to UML does not make the difficulties of representing product models in XML go away, it offloads a significant part of the effort from the small and resource-strapped STEP developer community to the much larger and resource-rich UML developer community.

Another lesson learned is that the choice of XML schema language used in the mapping is not very important, as long as the schema language supports context-sensitive element types and provides a library of simple data types for items such as real numbers, integers, Booleans, enumeration types, and so on. Therefore, both the W3C XML Schema definition language and RELAX NG are good XML schema language choices. The DTD is a poor schema language choice for the reasons discussed earlier in the Section 3.1 subsection. Although W3C XML Schema's type extension capability is useful for preserving some of a product data model's subtype/supertype information, type extension's usefulness is limited to single inheritance relationships. A W3C XML Schema has to simulate multiple inheritance and other kinds of inheritance supported by EXPRESS by using grouping and choice constructs.

A final lesson learned from the ISO STEP community's experience with the Part 25 and Part 28 projects is the importance of "keeping it simple" while at the same time not losing sight of requirements. Developers of Part 28 have had to resist the very human tendency to increase the project's scope and add complexity to the standard. They are currently grappling with the task of making sure the Part 28 configuration language meets the needs of EXPRESS modelers and XML developers without becoming so bloated that it overshadows the EXPRESS-to-XML mapping algorithm. Meanwhile, Part 25's creators are faced with the task of adding clarity and rigor without sacrificing the brevity and simplicity of the initial Draft Technical Specification. With some luck and a lot of hard work, these efforts to make STEP product models accessible to users of XML and UML will be successful, and the result will be a lowering of the barriers preventing widespread industry adoption of STEP.

6. Appendix: Complete PDM Example

This section shows:

- An XML schema representing a simple but complete EXPRESS schema consisting of the four entity types specified previously in the Section 2 and Section 3 sections.
- Flat and hierarchical XML data valid with respect to the schema.

The XML schema is presented using RELAX NG compact syntax, RELAX NG XML syntax, and as a W3C XML Schema. The W3C XML Schema uses substitution groups and type derivation by extension to model **product_context** as a subtype of **application_context_element**. The RELAX NG XML syntax was generated from the RELAX NG compact syntax using Trang [Clark], a software tool for translating RELAX NG compact syntax into other schema formats.

6.1. EXPRESS Schema

```
SCHEMA Product_identification;

ENTITY application_context;
  application : STRING;
 INVERSE
  context_elements : SET[1:?] OF application_context_element FOR frame_of_reference;
END_ENTITY;

ENTITY application_context_element
  SUPERTYPE OF (product_context) ;
   name : STRING;
   frame_of_reference : application_context;
END_ENTITY;
```

```
ENTITY product;
   name : STRING;
   quantity : INTEGER;
   description : OPTIONAL STRING;
   frame_of_reference : SET[1:?] OF product_context;
END_ENTITY;

ENTITY product_context
  SUBTYPE OF (application_context_element) ;
   discipline_type : STRING;
END_ENTITY;

END_SCHEMA;
```

6.2. RELAX NG Schema (Compact Syntax)

```
datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"

# _____________________________________________________
# MODEL: Product_identification
# _____________________________________________________
start =
  element schema-instance { (Application_context.elt |
    Application_context_element.elt | Product.elt |
    Product_context.elt)+ }

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# CLASS: Application_context
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Application_context.elt =
  element Application_context { id.att, Application_context.class }
Application_context.ref =
  element Application_context-ref { ref.att }
Application_context.class =
  (element application { string }) &
  (element context_elements { Application_context_element.ref+ })

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# CLASS: Application_context_element
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Application_context_element.elt =
  (element Application_context_element { id.att,
    Application_context_element.class }) |
  Product_context.elt
Application_context_element.ref =
  Product_context.ref |
  (element Application_context_element-ref { ref.att })
Application_context_element.class =
  (element name { string }) &
  (element frame_of_reference
    { Application_context.elt | Application_context.ref})

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# CLASS: Product
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Product.elt = element Product { id.att, Product.class }
Product.ref =
  element Product-ref { ref.att }
Product.class =
  (element name { string }) &
  (element quantity { xs:integer }) &
  (element description { string }?) &
```

```
  (element frame_of_reference
    { Product_context.elt+ | Product_context.ref+})

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# CLASS: Product_context
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Product_context.elt =
  element Product_context { id.att, Product_context.class }
Product_context.ref =
  element Product_context-ref { ref.att }
Product_context.class =
  Application_context_element.class &
  (element discipline_type { string })

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# global attributes
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
id.att = attribute id { xs:ID }*
ref.att = attribute xref { xs:IDREF }
```

6.3. RELAX NG Schema (XML Syntax)

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
            datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <!--
    _____________________________________________________
    MODEL: Product_identification
    _____________________________________________________
  -->
  <start>
    <element name="schema-instance">
      <oneOrMore>
        <choice>
          <ref name="Application_context.elt"/>
          <ref name="Application_context_element.elt"/>
          <ref name="Product.elt"/>
          <ref name="Product_context.elt"/>
        </choice>
      </oneOrMore>
    </element>
  </start>
  <!--
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    CLASS: Application_context
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  -->
  <define name="Application_context.elt">
    <element name="Application_context">
      <ref name="id.att"/>
      <ref name="Application_context.class"/>
    </element>
  </define>
  <define name="Application_context.ref">
    <element name="Application_context-ref">
      <ref name="ref.att"/>
    </element>
  </define>
  <define name="Application_context.class">
    <interleave>
      <element name="application">
        <data type="string" datatypeLibrary=""/>
      </element>
```

```
      <element name="context_elements">
        <oneOrMore>
          <ref name="Application_context_element.ref"/>
        </oneOrMore>
      </element>
    </interleave>
  </define>
  <!--
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    CLASS: Application_context_element
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  -->
  <define name="Application_context_element.elt">
    <choice>
      <element name="Application_context_element">
        <ref name="id.att"/>
        <ref name="Application_context_element.class"/>
      </element>
      <ref name="Product_context.elt"/>
    </choice>
  </define>
  <define name="Application_context_element.ref">
    <choice>
      <ref name="Product_context.ref"/>
      <element name="Application_context_element-ref">
        <ref name="ref.att"/>
      </element>
    </choice>
  </define>
  <define name="Application_context_element.class">
    <interleave>
      <element name="name">
        <data type="string" datatypeLibrary=""/>
      </element>
      <element name="frame_of_reference">
        <choice>
          <ref name="Application_context.elt"/>
          <ref name="Application_context.ref"/>
        </choice>
      </element>
    </interleave>
  </define>
  <!--
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    CLASS: Product
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  -->
  <define name="Product.elt">
    <element name="Product">
      <ref name="id.att"/>
      <ref name="Product.class"/>
    </element>
  </define>
  <define name="Product.ref">
    <element name="Product-ref">
      <ref name="ref.att"/>
    </element>
  </define>
  <define name="Product.class">
    <interleave>
      <element name="name">
        <data type="string" datatypeLibrary=""/>
      </element>
```

```
      <element name="quantity">
        <data type="integer"/>
      </element>
      <optional>
        <element name="description">
          <data type="string" datatypeLibrary=""/>
        </element>
      </optional>
      <element name="frame_of_reference">
        <choice>
          <oneOrMore>
            <ref name="Product_context.elt"/>
          </oneOrMore>
          <oneOrMore>
            <ref name="Product_context.ref"/>
          </oneOrMore>
        </choice>
      </element>
    </interleave>
  </define>
  <!--
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    CLASS: Product_context
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  -->
  <define name="Product_context.elt">
    <element name="Product_context">
      <ref name="id.att"/>
      <ref name="Product_context.class"/>
    </element>
  </define>
  <define name="Product_context.ref">
    <element name="Product_context-ref">
      <ref name="ref.att"/>
    </element>
  </define>
  <define name="Product_context.class">
    <interleave>
      <ref name="Application_context_element.class"/>
      <element name="discipline_type">
        <data type="string" datatypeLibrary=""/>
      </element>
    </interleave>
  </define>
  <!--
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    global attributes
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  -->
  <define name="id.att">
    <zeroOrMore>
      <attribute name="id">
        <data type="ID"/>
      </attribute>
    </zeroOrMore>
  </define>
  <define name="ref.att">
    <attribute name="xref">
      <data type="IDREF"/>
    </attribute>
  </define>
</grammar>
```

6.4. W3C XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" version="1.0">
  <!--
    ____________________________________________________
    MODEL: Product_identification
    ____________________________________________________
  -->
  <xs:element name="schema-instance">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="Application_context"/>
        <xs:element ref="Application_context_element"/>
        <xs:element ref="Product"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <!--
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    CLASS: Application_context
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  -->
  <xs:element name="Application_context"
       type="Application_context.class"/>
  <xs:complexType name="Application_context.class">
    <xs:sequence>
      <xs:element name="application" type="xs:string"/>
      <xs:element name="context_elements">
        <xs:complexType>
          <xs:choice>
            <xs:element maxOccurs="unbounded"
   ref="Application_context_element"/>
            <xs:element maxOccurs="unbounded"
   ref="Application_context_element-ref"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attributeGroup ref="id.att"/>
  </xs:complexType>
  <xs:element name="Application_context-ref"
       type="Application_context.ref"/>
  <xs:complexType name="Application_context.ref">
    <xs:attributeGroup ref="ref.att"/>
  </xs:complexType>
  <!--
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    CLASS: Application_context_element
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  -->
  <xs:element name="Application_context_element"
       type="Application_context_element.class"/>
  <xs:complexType name="Application_context_element.class">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="frame_of_reference">
        <xs:complexType>
          <xs:choice>
            <xs:element ref="Application_context"/>
            <xs:element ref="Application_context-ref"/>
          </xs:choice>
```

```
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attributeGroup ref="id.att"/>
</xs:complexType>
<xs:element name="Application_context_element-ref"
     type="Application_context_element.ref"/>
<xs:complexType name="Application_context_element.ref">
  <xs:attributeGroup ref="ref.att"/>
</xs:complexType>
<!--
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  CLASS: Product
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-->
<xs:element name="Product" type="Product.class"/>
<xs:complexType name="Product.class">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="quantity" type="xs:integer"/>
    <xs:element name="description" type="xs:string" minOccurs="0"/>
    <xs:element name="frame_of_reference">
      <xs:complexType>
        <xs:choice>
          <xs:element maxOccurs="unbounded" ref="Product_context"/>
          <xs:element maxOccurs="unbounded" ref="Product_context-ref"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attributeGroup ref="id.att"/>
</xs:complexType>
<xs:element name="Product-ref" type="Product.ref"/>
<xs:complexType name="Product.ref">
  <xs:attributeGroup ref="ref.att"/>
</xs:complexType>
<!--
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  CLASS: Product_context
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-->
<xs:element name="Product_context" type="Product_context.class"
     substitutionGroup="Application_context_element"/>
<xs:complexType name="Product_context.class">
  <xs:complexContent>
    <xs:extension base="Application_context_element.class">
      <xs:sequence>
        <xs:element name="discipline_type" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Product_context-ref" type="Product_context.ref"
     substitutionGroup="Application_context_element-ref"/>
<xs:complexType name="Product_context.ref">
  <xs:complexContent>
    <xs:extension base="Application_context_element.ref"/>
  </xs:complexContent>
</xs:complexType>
<!--
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  global attributes
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
  -->
  <xs:attributeGroup name="id.att">
    <xs:attribute name="id" type="xs:ID"/>
  </xs:attributeGroup>
  <xs:attributeGroup name="ref.att">
    <xs:attribute name="xref" use="required" type="xs:IDREF"/>
  </xs:attributeGroup>
</xs:schema>
```

6.5. Flat XML Data

```
<?xml version="1.0" encoding="UTF-8"?>

<schema-instance>

  <Application_context id="ac1">
    <application>configuration controlled 3d design</application>
    <context_elements>
      <Product_context-ref xref="pc1"/>
      <Product_context-ref xref="pc2"/>
    </context_elements>
  </Application_context>

  <Application_context id="ac2">
    <application>lifecycle support</application>
    <context_elements>
      <Product_context-ref xref="pc3"/>
    </context_elements>
  </Application_context>

  <Product_context id="pc1">
    <name>Mechanical 3d Parts</name>
    <frame_of_reference>
      <Application_context-ref xref="ac1"/>
    </frame_of_reference>
    <discipline_type>mechanical</discipline_type>
  </Product_context>

  <Product_context id="pc2">
    <name>Electrical Components</name>
    <frame_of_reference>
      <Application_context-ref xref="ac1"/>
    </frame_of_reference>
    <discipline_type>electrical</discipline_type>
  </Product_context>

  <Product_context id="pc3">
    <name>Chemically Reactive Items</name>
    <frame_of_reference>
      <Application_context-ref xref="ac2"/>
    </frame_of_reference>
    <discipline_type>chemical</discipline_type>
  </Product_context>

  <Product>
    <name>Wonder Widget</name>
    <quantity>3</quantity>
    <frame_of_reference>
      <Product_context-ref xref="pc1"/>
      <Product_context-ref xref="pc2"/>
      <Product_context-ref xref="pc3"/>
    </frame_of_reference>
  </Product>
```

```
</schema-instance>
```

6.6. Hierarchical XML Data

```
<?xml version="1.0" encoding="UTF-8"?>

<schema-instance>
  <Product>
    <name>Wonder Widget</name>
    <quantity>3</quantity>
    <frame_of_reference>

      <Product_context id="pc1">
        <name>Mechanical 3d Parts</name>
        <frame_of_reference>

          <Application_context>
            <application>configuration controlled 3d design</application>
            <context_elements>
              <Product_context-ref xref="pc1"/>
              <Product_context-ref xref="pc2"/>
            </context_elements>
          </Application_context>

        </frame_of_reference>
        <discipline_type>mechanical</discipline_type>
      </Product_context>

      <Product_context id="pc2">
        <name>Electrical Components</name>
        <frame_of_reference>

          <Application_context>
            <application>configuration controlled 3d design</application>
            <context_elements>
              <Product_context-ref xref="pc1"/>
              <Product_context-ref xref="pc2"/>
            </context_elements>
          </Application_context>

        </frame_of_reference>
        <discipline_type>electrical</discipline_type>
      </Product_context>

      <Product_context id="pc3">
        <name>Chemically Reactive Items</name>
        <frame_of_reference>

          <Application_context>
            <application>lifecycle support</application>
            <context_elements>
              <Product_context-ref xref="pc3"/>
            </context_elements>
          </Application_context>

        </frame_of_reference>
        <discipline_type>chemical</discipline_type>
      </Product_context>

    </frame_of_reference>
  </Product>
</schema-instance>
```

Acknowledgements

I wish to thank my colleagues at NIST, and especially Peter Denno, for reviewing earlier drafts of this paper. I am also grateful to NIST's Advanced Technology Program for funding this work.

Bibliography

[Barkmeyer] Barkmeyer, E.A., and Lubell, J., *XML Representation of EXPRESS Models and Data*, proceedings of *XML Technologies and Software Engineering Workshop*, Toronto. Available on-line at *http://www.mel.nist.gov/msidlibrary/doc/xse2001.pdf*.

[Carlson] Carlson, D., *Modeling XML Applications with UML*, Addison-Wesley, 2001 (ISBN 0201709155).

[Clark] Clark, J., Trang: Translator for RELAX NG Schemas, *http://thaiopensource.com/relaxng/trang.html*.

[Fritz] Fritz, J., and Hardwick, M., *ST-XML Manual* (includes CEB Ed. 2 Specification), STEP Tools, Inc., April 30, 2002. Available on-line at *http://www.steptools.com/projects/xml*.

[ISO10303-1] ISO 10303-1:1994, *Industrial automation systems and integration-Product data representation and exchange-Part 1: Description methods: Overview and fundamental principles*.

[ISO10303-11] ISO 10303-11:1994, *Industrial automation systems and integration-Product data representation and exchange-Part 11: Description methods: The EXPRESS language reference manual*.

[Jelliffe] Jelliffe, R., *The Schematron Assertion Language*, version 1.5. Available on-line at *http://www.ascc.net/xml/schematron/*.

[Kemmerer] Kemmerer, S. (ed.), *STEP: The Grand Experience*, National Institute of Standards and Technology, NIST SP 939, July 1999. Available on-line at *http://www.mel.nist.gov/msidlibrary/doc/stepbook.pdf* (this file is a 4 megabyte download).

[OASIS-RNG] Organization for the Advancement of Structured Information Systems, *RELAX NG Specification*, Committee Specification 3 December 2001. Available on-line at *http://www.oasis-open.org/committees/relax-ng/spec.html*.

[OASIS-RNG-compact] Organization for the Advancement of Structured Information Systems, *RELAX NG DTD Compact Syntax*, Working Draft 7 June 2002. Available on-line at *http://www.oasis-open.org/committees/relax-ng/compact-20020607.html*.

[OASIS-RNG-DTD-compat] Organization for the Advancement of Structured Information Systems, *RELAX NG DTD Compatibility*, Committee Specification 3 December 2001. Available on-line at *http://www.oasis-open.org/committees/relax-ng/compatiblity.html*.

[OMG-UML] Object Management Group, *OMG Unified Modeling Language Specification*, Version 1.4, September 2001. Available on-line at *http://www.omg.org/technology/documents/formal/uml.htm*.

[OMG-XMI] Object Management Group, *OMG XML Metadata Interchange (XMI) Specification*, Version 1.2, January 2002. Available on-line at *http://www.omg.org/technology/documents/formal/xmi.htm*.

[PDES] "Success Stories", PDES, Inc. website, *http://pdesinc.aticorp.org/*.

[Provost] Provost, W., *UML for W3C XML Schema Design*, XML.com, August 7, 2002. Available on-line at *http://www.xml.com/pub/a/2002/08/07/wxs_uml.html*.

[SC4N169] ISO TC184/SC4/WG11 N169, *ISO/CD TS 10303-28, Product data representation and exchange: Implementation methods: EXPRESS to XMI binding*, Draft Technical Specification, 2002-02-14.

[SC4N194] ISO TC184/SC4/WG11 N194, *ISO/TS 10303-28, Product data representation and exchange: Implementation methods: XML Schema governed representation of EXPRESS schema governed data*, working draft, 2002-09-09.

[SC4N1356] SC4 N1356, *ISO-DTS 10303-25, STEP Part 25, Implementation method: EXPRESS to OMG XMI binding, ballot results*, 2002-08-30. Available on-line at *http://www.tc184-sc4.org/SC4_Open/SC4_and_Working_Groups/SC4_N-DOCS*.

[Schenk] Schenk, D. A., and Wilson, P. R., *Information Modeling: The EXPRESS Way*, Oxford University Press, New York, NY, 1994 (ISBN 0-19-508714-3).

[W3C-XML] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 6 October 2000. Available on-line at *http://www.w3.org/TR/REC-xml*.

[W3C-XS1] World Wide Web Consortium, *XML Schema: Part 1: Structures*, W3C Recommendation, 2 May 2001. Available on-line at *http://www.w3.org/TR/xmlschema-1*.

[W3C-XS2] World Wide Web Consortium, *XML Schema: Part 2: Datatypes*, W3C Recommendation, 2 May 2001. Available on-line at *http://www.w3.org/TR/xmlschema-2*.

[XMLmodeling] Carlson, D., XMLmodeling.com, *http://xmlmodeling.com*.

Biography

Joshua **Lubell**
National Institute of Standards and Technology
Manufacturing Systems Integration Division
Gaithersburg
United States of America
lubell@nist.gov

Josh Lubell works at US National Institute of Standards and Technology (NIST) where he applies markup technology toward solving data exchange problems between manufacturing applications. He is a contributor to various standards efforts and speaks regularly at XML-related conferences. His pre-NIST experience includes artificial intelligence systems design and prototyping as well as software development for the building materials industry. He has an M.S. in computer science from the University of Maryland at College Park and a B.S. in mathematics from Binghamton University. He lives in Maryland about midway between Washington DC and Baltimore.