UBL Library Content Methodology

The purpose of this document is two-fold:

1. To explain how we got to where we are with the UBL vocabulary, we felt it necessary to provide a background to the rationale and framework in which we conducted our work.
2. This document will eventually become a guide for those wishing to maintain or extend the UBL Library.

An initial analysis has been completed for the Order structure used in xCBL3.0. From this base we will seek to extend and refine our model to accommodate other business documents and constructs from other e-business vocabularies.

We have deliberately chosen to adapt an existing XML vocabulary (xCBL 3.0), not because we wish to promote the xCBL view, we simply decided that it was better than starting with a blank page. (Lisa put your xCBL overview here)

We have attempted to identify all the Basic Information Entities[1] involved in this document following the guidelines of the ebXML Core Component Technical Specification (CCTS)[2]. Much of our terminology is taken from this specification. We intend our library and its document definitions to be compliant with the ebXML Core Component Technical Specification.

These BIEs were captured as a logical model in both XML Schema (XSD) and spreadsheet[3] (Excel) form. The XML Schema logical model was then encoded as an XML Schema following the UBL Naming and Design Rules. This schema was then used to assemble the components into a UBL Order in XML Schema (XSD) form. However, in theory, the model could also be used to generate other XML schema languages (e.g. DTD, RELAX, etc.).

Our current library now contains the structure for an Order document. It also contains a core library of structural components (known as re-usable types) and their content components (Basic BIEs). Both of these reference the ebXML Core Component Types for the data types of their Basic BIEs.

Process Steps

1. We analyzed the xCBL constructs to identify the Basic Information Entities.

   The practices we adopted were:

   a. Remove redundant nesting, i.e. one element contains only one element which contains only one element.

      i. "Listof" structures are a form of redundant nesting in that only the object itself needs be defined, the "Listof" is described by using the cardinality of 0..n or 1..n.

      ii. Nesting may be necessary to allow for the correct cardinality. That is two objects may need to be repeated in pairs. We need to define a container(Aggregate) to contain the pair and this aggregate has a cardinality of 0..n.

   b. Implementation features such as "CodedOther" elements do not form part of the BIE library.

   c. Elements with names ending in "Number" or "Name" or "ID" may not actually be numeric, actual names or true identifiers. We were guided by the logical use of the object, not its name.

---

[1] A Basic Information Entity is defined by the CCTS as "A piece of business data or a group of pieces of business data with a unique business semantic definition. A *Business Information Entity* can be either a *Basic Business Information Entity* (BBIE) or an *Aggregate Business Information Entity* (ABIE)."

[2] Available from http://www.ebtwg.org/news/040402.html.

[3] Because of limitations with current XML Schema design tools we found it easier to manipulate the model using a two-dimensional matrix, i.e. a spreadsheet. We then automated the construction of the XML Schema from this matrix.

d. Check to see this BIE is not already defined as a Re-usable Type. This may mean defined in another context. For example BuyerDetails and SellerDetails are instances of Party in two different contexts.

   If it is possible to associate the BIE with an existing Re-usable Type, then skip the next step.

e. Define a new Re-usable Type.

   Every Aggregate BIE is an instance of a Re-usable Type. The Re-usable Type defines the structure of the Aggregate BIE.

2. Gave the BIE a name

   a. Each BIE has both a UBL Name and a BIE Dictionary Entry Name. Rules for deriving these names are given in Step 6.

   b. Validate the use of name components

      A proper analysis of name components should allow us to say…

      A [Representation Term] represents the [Property Qualifier, Property Term] of the Object Class.

      e.g.

      'an Identifier represents the Identifier of the Party', or

      'a Contact represents the Shipping Contact of the Party', or

      'a Code represents the Identification of a Language'

3. Checked the occurrences rules, based initially on xCBL 3.0, but with UBL interpretation.

4. Identified new BIEs

   There were cases where we found missing pieces of information. We then created a new BIE.

5. Identified candidate Core Components

   A UBL BIE without context is a Core Component as defined by the CCTS. In the cases where we have not identified an existing Core Component, these become Candidate Core Components. We intend to submit these candidates to the appropriate UN/CEFACT group in the near future.

6. Establish the UBL Library

   Our UBL Library metamodel contains:

   UBL UID

      This is a 9-character string, unique across the entire UBL library, starting with the characters "UBL", followed by a six-digit number.

   xCBL Name

      The name of the element as given in xCBL 3.0.

   UBL Name

      For Basic BIEs, these are constructed from 'property qualifier' + 'property term'+ 'representation term'.

      For Aggregate BIEs, these are constructed from 'property qualifier' + 'property term'+ 're-usable type'

      There are also some refinements:

- Property Term is removed if it is the same as the Representation Term.
- "Identifier" as a representation term is abbreviated to "Id", except where it is the only component used.
- "Text" as a representation term is not used in the name.
- Embedded spaces in components are removed.

This name is used for the XML tag name (ref: Naming and Design Rules).

BIE Dictionary Entry Name

These are constructed from 'object class' + 'property qualifier' + 'property term'+ 'representation term' with a ". " separator. Property Term is removed if it is the same as the Representation Term.

Object Class

Object Class is a 'logically related group of properties', i.e. a collection that makes business sense. We also refer to these things as Re-usable Types, but they are also known as Classes (to the OO and UML world) or Entities (to database designers).

For components of a Re-usable Type, the value of the Object Class should always be the name of the Re-usable Type. This may be subject to 'de-normalizing' for design reasons (eg. The OrderHeader and OrderSummary components belong to Order Object Class. OrderHeader and OrderSummary are structural containers only.)

Property Qualifier

Property Qualifier is only used for Aggregate BIEs. It is the 'context' of the relationship with another Re-usable Type. That is, it is the role this object plays within its association with the 'parent' type.

This does not apply to Basic BIEs. If it appears that a Basic BIE needs a property qualifier then it is either:

(a) an inadequate property name,

(b) two distinct Basic BIEs, or

(c) a candidate group of Basic BIEs that may be another Aggregate (ie Re-usable Type).

Property Term

A Property Term identifies the specific item within its Object Class. This represents the distinguishing characteristic or property of the dominant area of interest and shall occur naturally in the definition. It may also be known as an attribute (to database designers). The combination of Object Class, Property Qualifier (if appropriate) and its Property Term, should give the basic semantic meaning of the item.

Representation Term

Defines the structure of valid values for BIEs.

Basic BIEs use one of the Core Component Representation Terms as defined by CCTS.

Aggregate BIEs use the Representation Term of "Details" as defined by CCTS.

NB. The property of 'unique identification' may be provided by any of the three Representation Terms.

- 'Identifier' - when the set of values is informally defined, defined by an unofficial source or for private use. This Representation allows us to associate an agency that will ensure uniqueness within their own code sets. Only one occurrence of the word "Identifier" need be in the UBL Name or the BIE Dictionary Entry Name.

- 'Code' - when the set of values is formally defined by an officially recognized agency (e.g. ISO). This would add a "Identifier. Code" to the UBL Name and the BIE Dictionary Entry Name.
- 'Name' - when the set of values is an informal text string. This is relatively rare situation, but would add a "Identifier. Name" to the UBL Name and the BIE Dictionary Entry Name.

Type

For Basic BIEs this should be an appropriate Core Component Type as defined in CCTS.

For Aggregates BIEs this should be the name of the associated UBL re-useable type. This type will be defined elsewhere in the model.

You may want to envisage this as being a forward pointer to other structures.

Occurrence

Based on our interpretation of the xCBL schema we defined the optionality (whether a component is mandatory or not) and its cardinality (how many times it may appear in a given instance). We denoted this using an occurrence notation, where
0..1, means it may occur once only or not at all.
1..1, means it must occur once only
0..n, means it may occur many times or not at all.
1..n, means it must occur once and may occur many times.

Basic/Aggregate

Either Basic (does not contain further BIEs) or Aggregate (does contain further BIEs).

All Basic BIEs are instances of Core Component Types.

All Aggregate BIEs are instances of re-usable Types.

UBL Definition

Originally derived from the xCBL definition (or Core Component catalogue definition if available) and supplemented by the UBL team.

Code Lists/Standards

In those cases where the BIE is a "Code. Type", and its value is to be described in an external standard, then the particular code-list or standard should be identified here.

There is currently a position paper within UBL on how these code sets will be applied.

Analyst Notes

This is a list of comments, queries and notes made as the work is done.

Core Component UID

This is the UID of the related core component, in those cases where a direct correlation exists. This information is based on the current Core Component Catalog and is still immature.

Context Business Process

Context Region (Geopolitical)

Context Official Contraints

Context Product

Context Industry

Context Role

The handling of Contexts is the vehicle by which UBL will extend its BIEs to allow for customization and extension.

At this stage we have assumed a general context for all our BIEs as being the Business Process of "Procurement".

These will be refined and defined in the next release. (ref: Context Methodology and Context Drivers team within UBL)

Editor's Notes

Where editorial comments need to be applied to an entry's definition.

7. Assemble the Order Document

Document assembly involves establishing the structural components required for an Order document. These are encoded as an XML Schema (XSD).

We chose to view the document definitions as context-driven extensions to our core library of Re-usable Types. This meant the same metadata structures would apply to both models.

Because our metamodel is an XML Schema, the resultant document structures are already suitable as a valid XML Schema.

We determined that the Order required three substructures, "OrderHeader", "LineItem" and "OrderSummary". Of these, "LineItem", was considered a Re-usable Type. That is, it may be re-used in other contexts (e.g. Ship Notice, Invoice, etc.). So in the Order document we re-used the LineItem from our core library in the context of "Order".

All this means that to assemble an Order we only need to define the overarching three part structure and then the substructures specific to Order documents, that is, OrderHeader and Order Summary.

This explains why the Order schema document itself is relatively small and the majority of component definitions reside in the core library of Re-usable Types.

Assembling other document types should require undertaking similar high-level structural definitions and then referencing Re-usable Types from our core library using the required context to achieve customization.

8. Ongoing Maintenance

The UBL Library can expand both in its core library (of Re-usable Types) and in context-specific document definitions.

Each requires similar refinements and improvements in both the content definitions and in the methodology outlined here.