UBL Guidelines for Customization Version 1.0

Public Review Draft 01

30 September 2008

Specification URIs:
This Version:
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0prd01.pdf (Authoritative)
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0prd01.doc
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0prd01.html

Previous Version:
N/A

Latest Version:
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0.pdf
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0.doc
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0.html

Technical Committee:
OASIS Universal Business Language (UBL) TC

Chair(s):
Jon Bosak
Tim McGrath

Editor(s):
Michael Grimley
Mavis Cournane
Tim McGrath
G. Ken Holman
Jon Bosak

Related work:
This specification is related to:

- UBL 1.0 Context Methodology

Abstract:
This document provides practical guidance in creating UBL-conformant and UBL-compatible document schemas.

Status:
This document was last revised or approved by the UBL TC on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical

Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/ubl/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/ubl/ipr.php).

If there is a non-normative errata page for this specification, it is located at http://www.oasis-open.org/committees/ubl/.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the “OASIS IPR Policy”). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an “AS IS” basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS’ procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names “OASIS” and “UBL” are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for guidance.

Table of contents

Table of figures

1 Introduction

The OASIS Universal Business Language Technical Committee (UBL TC) has produced a vocabulary that, for many user communities, can be used "as is." However, the TC also recognizes that some user communities must address use cases whose requirements are not met by the UBL off-the-shelf solution. These Guidelines are intended to aid such users in developing custom solutions based on UBL.

The goal of these UBL customization guidelines is to maintain a common understanding of the meaning of information being exchanged between specific implementations.

The determining factors governing when to customize may be business-driven, technically driven, or both. The decision should driven by real world needs balanced against perceived economic benefits.

1.1 Definition of terms

To assist with the scoping of this document, let us begin with some definitions:

- **Customization**: The alteration of something in order to better fit requirements.
- **UBL customization**: The description of XML instances, or XML-based applications acting on those instances, that are somehow based on or derived from the UBL Standard.
- **Data Type:** Defines the set of valid values that can be used for a particular Basic Business Information Entity. A Data Type is specified as a restriction of an ebXML Core Component Type. In UBL, Data Types are expressed as XML Schema simple and complex types.
- **Information entity:** A piece of data or a group of pieces of data with a unique definition. Following the concepts of the ebXML Core Component Technical Specification (CCTS), an information entity can be a Basic Business Information Entity (BBIE), an Association Business Information Entity (ASBIE), or an Aggregate Business Information Entity (ABIE). In UBL, information entities are expressed as XML information items.
- **Information item:** An XML document's information set consists of a number of information items; the information set for any well-formed XML document will contain at least a document information item and several others.[1]
- **UBL conformant schema:** A schema created by a community of interest that validates customized document constraints without violating UBL standard schema document constraints.
- **UBL standard schema:** A normative conformant UBL schema published by OASIS.
- **UBL conformant instance:** An instance that validates against a UBL standard schema.
- **UBL compatible:** Consistent with the principles behind UBL's models or their development.

[1] See http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem

1.2 Conformance vs. compatibility

Once the need to customize UBL has been determined, designers must decide whether the result will be UBL *conformant* or UBL *compatible*. Although the UBL TC will not be involved in determining whether customizations are conformant, compatible, or otherwise, we supply these definitions as a point of reference for those who might.

1.2.1 UBL conformance

UBL conformance at the instance and schema level means that there are no constraint violations when validating the instance against a UBL standard schema. A *UBL conformant instance* is an instance that validates against a UBL standard schema (and does not violate any of the Additional Document Constraints specified in the UBL standard). A *UBL conformant schema* is a schema that will validate only UBL conformant instances.

The UBL TC publishes the UBL standard schemas as OASIS technical specifications. These provide the base vocabulary that ensures common understanding.

Figure 1 shows the scope of UBL conformance. By definition, all schema-valid instances of a conformant customization are schema-valid instances of UBL as well; however, this is not necessarily true the other way around. Not all schema-valid instances of a UBL document will conform to every customization, because some instances will contain elements that are optional in the standard but are omitted from the customization. Indeed, some customizations will be intended primarily to screen out optional instance data that has been deemed unwanted for a particular set of applications.

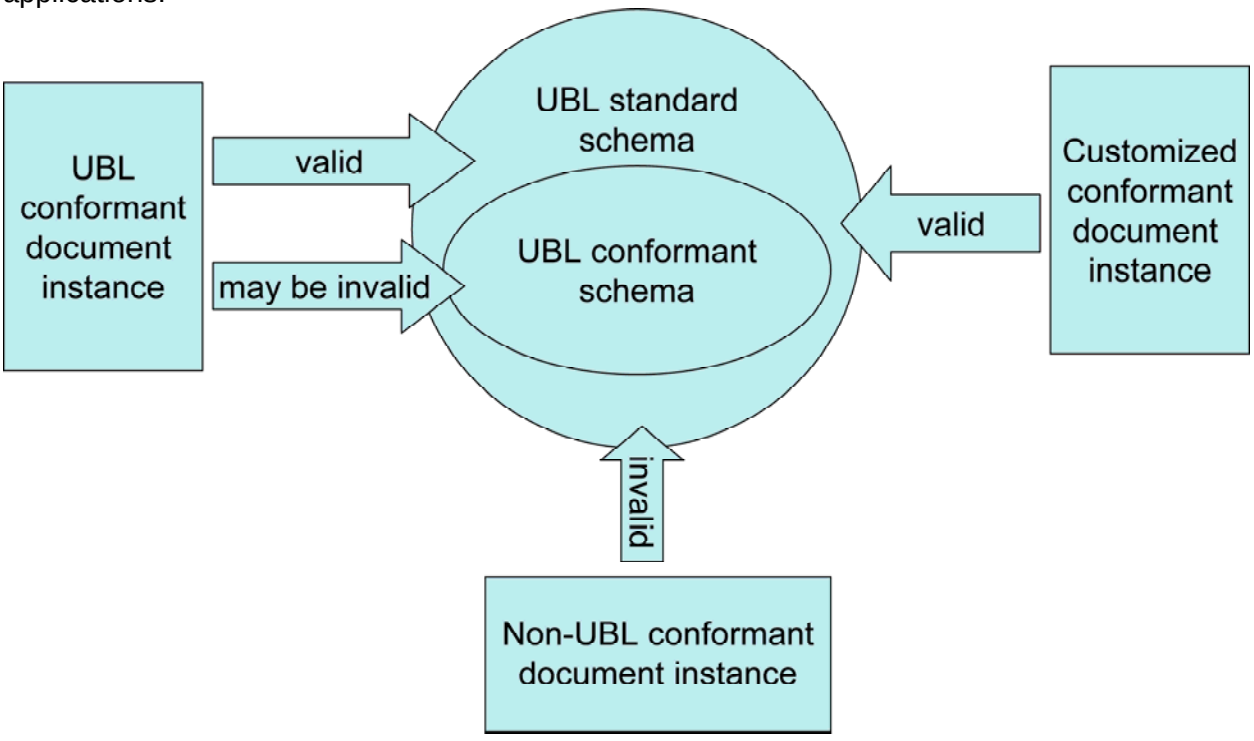

Figure 1. Conformant schemas and document instances

A major advantage of UBL conformance is that it minimizes the need for custom software or modifications to UBL applications designed to process the full UBL Standard — assuming that nonstandard elements have not been added via the UBL extension mechanism (Section 3.1.4).

1.2.2 UBL compatibility

To be UBL compatible means to be consistent with the principles behind UBL's models or their development. These principles are defined in the ebXML Core Component Technical Specification (CCTS) and the UBL Naming and Design Rules (NDR). While conformance and interoperability of these customized documents cannot be guaranteed, we can expect some degree of familiarity through the re-use of common information entities and principles.

Compatibility should be a design objective when creating new document types or extending existing UBL document types.

Figure 2 illustrates the scope of UBL compatibility. Schema-valid instances of a compatible but nonconformant customization are never schema-valid instances of UBL. However, schema-valid instances of UBL may be schema-valid instances of a compatible customization.

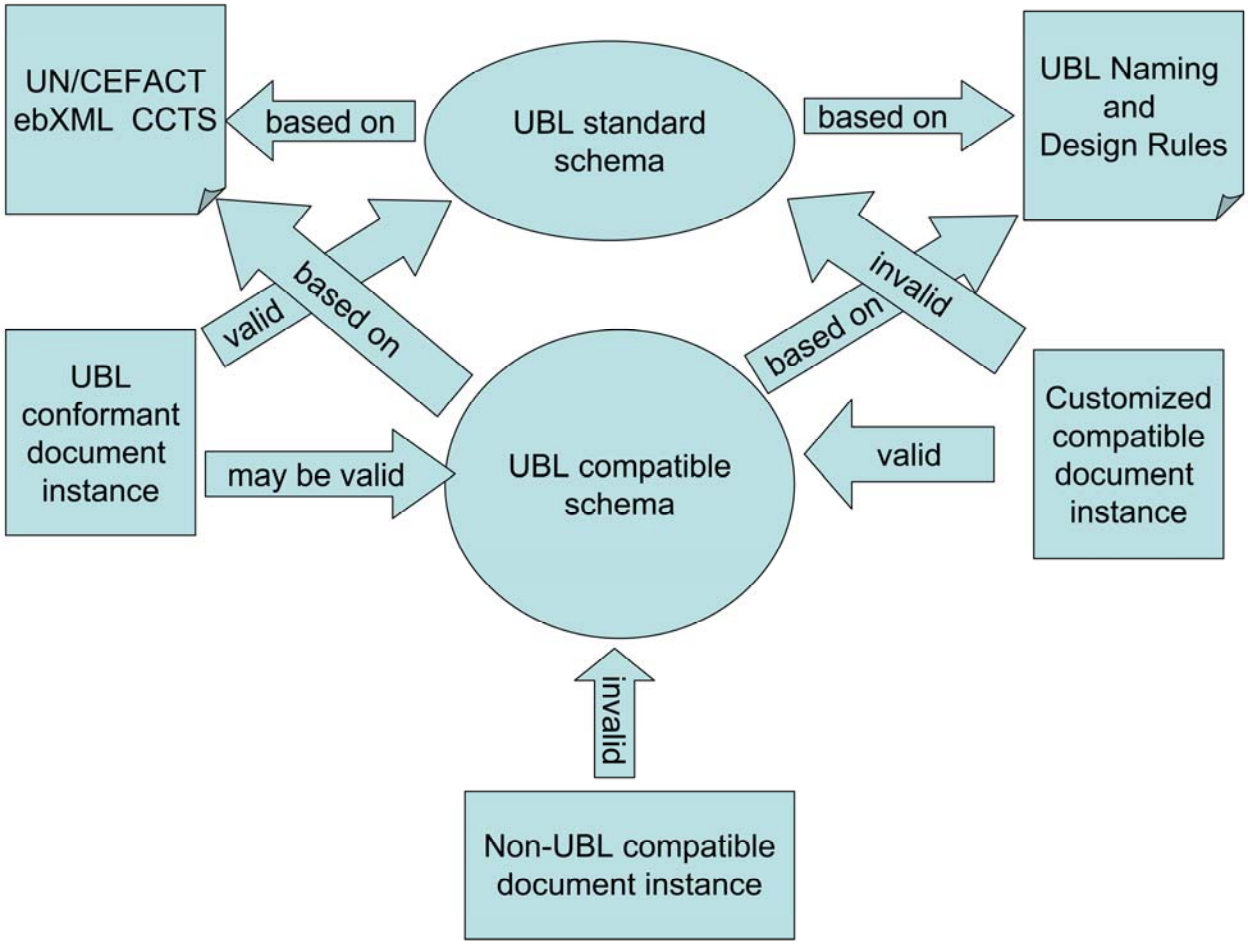

Figure 2. Compatible schemas and document instances

1.2.3 Maintaining common meanings

It is important to recognize that the information entities in UBL should not be repurposed in a customization. That is, customizations must avoid semantic drift in the meaning of UBL entities.

A change to the definition of a term is contrary to the use of UBL as a tool for conveying common meanings, and it violates semantic conformance to the UBL Standard, even though such violations cannot be caught by schema validation. Contracts between trading partners that

agree to accept UBL documents as legally equivalent to their paper equivalents bind those users to the meanings specified in the published definitions.

1.2.4 Identifying versions, customizations, and profiles

The following information entities at the root of each document will allow instances to identify their precise customization:

- UBLVersionID

  An identifier reserved for UBL version identification. Not actually a customizable value, but necessary to understand which version of UBL is being customized.

- UBLCustomizationID

  An identifier (such as a URI) for a user-defined customization of UBL.

- UBLProfileID

  An identifier (such as a URI) for a user-defined profile of the customization being used. Profiles are further refinements of customizations that enable “families” of customizations to be implemented.

1.2.5 Customization profiles

Customizations of UBL may be refined even further for different scenarios. A profile characterizes the choreography of an interchange. A given document type may have two different sets of constraints in two different profiles of the same customization. For example, an invoice instance used in the choreography of a Basic Procurement profile may not require as many information entities as an invoice instance used in the different choreography of an Advanced Procurement profile.

Thus the three dimensions of the version of a set of UBL document structural constraints are defined by the UBL version (standard), the business process context version (customization), and the choreography version (profile). An instance claiming to satisfy the document constraints for a particular profile in a customization asserts this in the UBLCustomizationID and UBLProfileID entities (see 1.2.4). For example, Stand Alone Invoicing may be a profile for the Northern European Subset customization.

1.3 Overview of customization methodology

The UBL library and document schemas have been developed from conceptual models based on the principles of the ebXML Core Component Technical Specification. These are then expressed in W3C XML Schema (XSD), based upon the UBL Naming and Design Rules. It is these schemas that may be used to both specify and validate UBL conformance.

It is recommended that a similar approach be followed when customizing UBL. Therefore, the following sections discuss conceptual design (Section 2), then the specification of XML documents (section 3), and finally the validation aspects of customization (section 4). These steps are shown graphically in Figure 3 below.

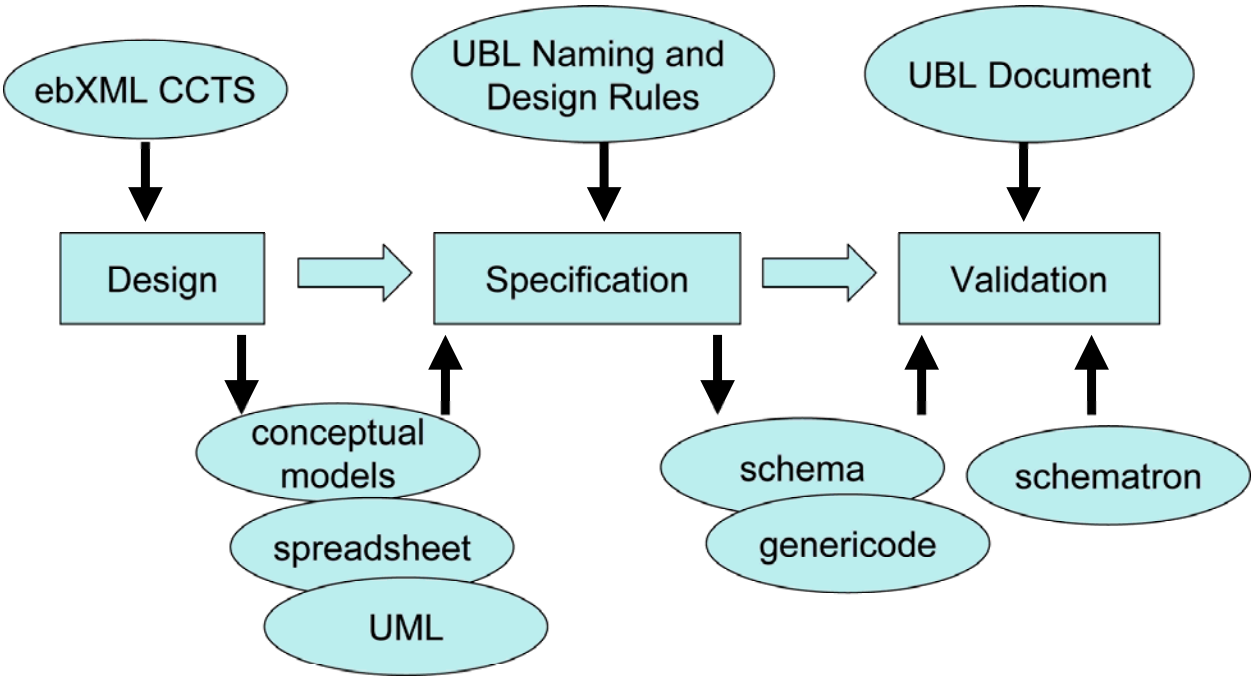

Figure 3. Overview of customization methodology

1.4 Acknowledging OASIS copyright

UBL is provided under the OASIS Royalty Free on Limited Terms policy, and this should be recognized by a customization.

OASIS policies support implementations, subsets, and extensions of OASIS works as long as they acknowledge derivation from OASIS works and do not incorrectly claim compliance with or identity with an OASIS work. If you modify the UBL Invoice schema, for example, you cannot claim that it is still the UBL Invoice schema, but you should acknowledge that the new work was derived from the UBL Invoice schema.

Specifications and models published for use by others that incorporate OASIS work should include the following in an appropriate place, usually near the author's own copyright notice:

Portions copyright (c) OASIS Open 200[8]. All Rights Reserved.

This text can be followed by the OASIS policy URI if the author wishes to provide that reference:

http://www.oasis-open.org/who/intellectualproperty.php

Those who publish such works should take note of the rights available under the OASIS IPR Policy and their limitations, including any notices posted with respect to a specific work. In specific cases there may be parties other than OASIS who, from time to time, post assertions that a license is needed. For IPR notices relating to UBL, see

http://www.oasis-open.org/committees/ubl/ipr.php

OASIS generally welcomes the creation of derivative works, and in appropriate cases, OASIS may assist in publicizing the work in its own channels.

2 Designing for UBL customization

The design of the conceptual models for UBL and its customizations is not affected by the syntactical issues of XML, schema languages, or validation tools. The UBL TC uses spreadsheets and UML for model design, but this is not a requirement.

Designing a customization may involve:

- Adding information entities to meet requirements of a specific business context
- Omitting information entities not needed in a specific context
- Refining the meaning of information entities
- Creating constraints on possible values for information entities (such as code lists)
- Combining (or recombining) and assembling information entities into new aggregations or documents
- Adding business rules

Note that the design models in UBL adhere to CCTS naming conventions. Information entities are referenced by their Dictionary Entry Names, and the terminology used here reflects this.

2.1 Designing for conformance

When designing for UBL conformance (see 1.2.1), the key objective is to create custom models that can be used to specify and validate UBL-conformant instances. A UBL conformant instance is an instance validating against customized document constraints while simultaneously validating against a UBL standard schema.

Consequently, designing for conformance applies primarily to restrictions:

- Subsets of the document model — restricting the number of entities in a document
- Constraints on document content — restricting the possible values an entity can have

In either case, the restriction may be accomplished either by removing optional objects from the UBL model or by checking for their existence in the value validation phase. Minimums can be increased to their maximum, maximums can be decreased to their minimum, and data types can be refined but not extended.

UBL also allows conformant extensions to be made using an extension area provided in the Standard schema (section 3.1.4).

2.1.1 Subsets of the document model

The standard UBL document types have been designed to accommodate a broad range of business process contexts. If all optional elements in a UBL document type were instantiated, the resulting instance would be extremely verbose. For example, if a UBL Order document contained just one instance of all its possible elements, that document would contain approximately 800,000 elements. Most implementations will not need all the information entities defined by the standard document type. The use of subsets allows for the removal from a document model of any optional information entities that are not needed to satisfy business requirements.

It must be noted that subsetting can only be used to remove optional elements or change cardinality in ways that do not reduce the required minimum number of occurrences or extend the permitted maximum number of occurrences of an element. Thus,

- 0..1 can become 1..1 or 0..0 (but not, for example, 1..2)
- 0..n can become 0..1, 1..n, m..n, or 0..0 (where m<n)
- 1..n can become 1..1 or 1..m (where m<n)
- 1..1 cannot be changed

2.1.2 Code list constraints on document content

Using a code list (or an enumerated list) for an information entity is a common customization requirement. Such lists impose instance value constraints. For example, “the Currency Code must be expressed using ISO 4217 codes” is a constraint on the possible values for Currency Code in any document instance claiming to conform to a given schema.

In UBL, there are two levels of constraints for codes:

- Code lists without defined values

  These are not empty lists, they are lists without constraints — in effect, infinite lists of values constrained only by their lexical form.

- Code lists with defined values

  These are explicit lists that constrain possible values for the content.

2.1.3 Other constraints on document content

There are other cases in which the treatment of UBL instances requires customization in order to limit or restrict content values. For example:

- “The Total Value of an Order cannot exceed $100,000.”
- “The length of an Address Line cannot exceed 40 characters.”

Additionally, there are cases relating to the dependencies between values of information entities which also necessitate customization. For example:

- “The Shipping Address must be the same as the Billing Address.”
- “The Start Date must be earlier than the End Date.”

Methods for specifying and validating such constraints are discussed in Sections 3 and 4.

2.1.4 Examples of conformant customizations

The Northern European Subset group (NES), a collaborative effort of state agencies from six European nations, has produced conformant subsets of UBL 2.0 documents by selectively excluding information entities from the UBL library,[2] as shown in Figure 4.

[2] See http://www.nesubl.eu/documents/nes2.4.6dae77a0113497f158680001674.html

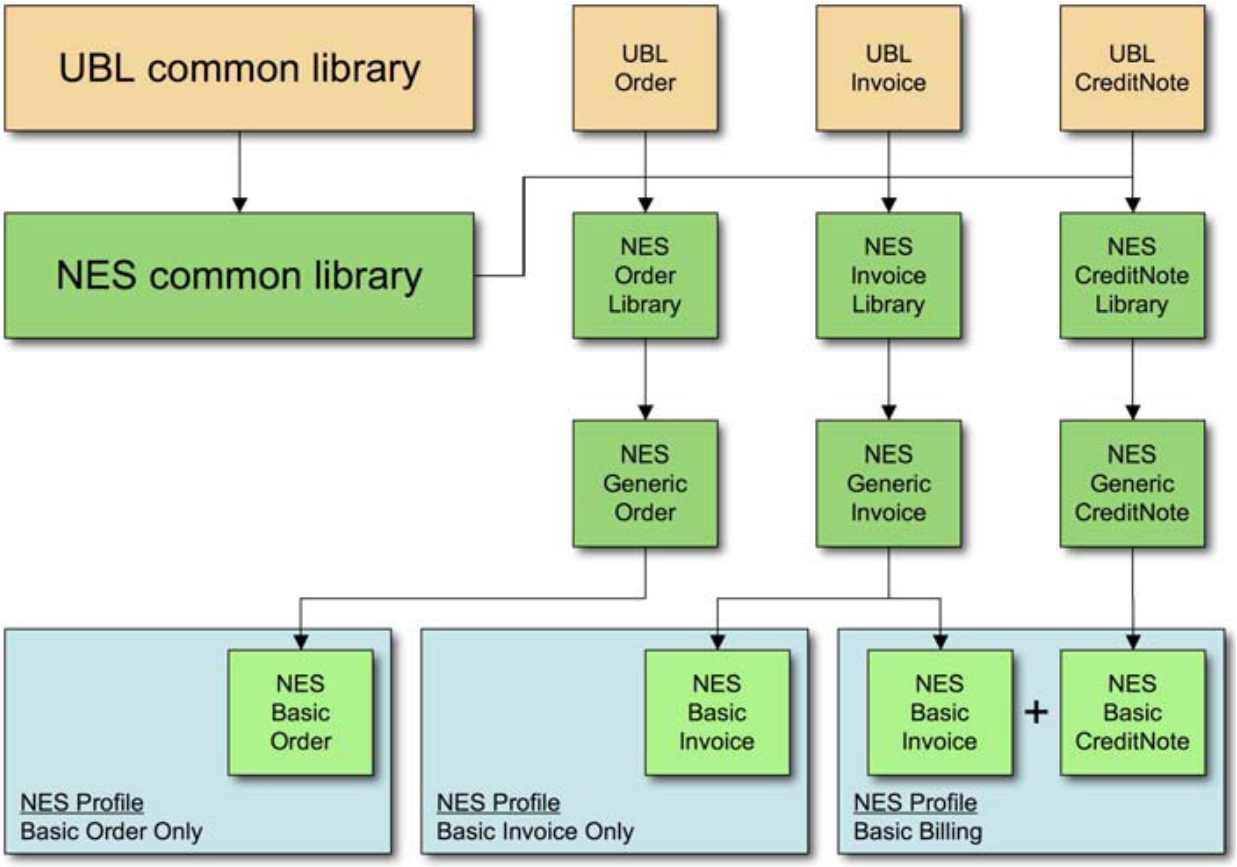

Figure 4. NES subset architecture

The Delivery structure is an example of an aggregate entity that is used across several documents and processes. In UBL, Delivery is defined in the UBL Common Library (see Figure 5) and includes several entities that are not required in the NES processes and documents. (As noted before, entities are referenced at the modeling level by their CCTS Dictionary Entry Names, not by the XML element names generated from these according to the UBL Naming and Design Rules.)

Figure 5. UBL common Delivery aggregate

220 The standard Delivery is restricted to meet NES requirements at the NES Common Library level as shown in Figure 6.

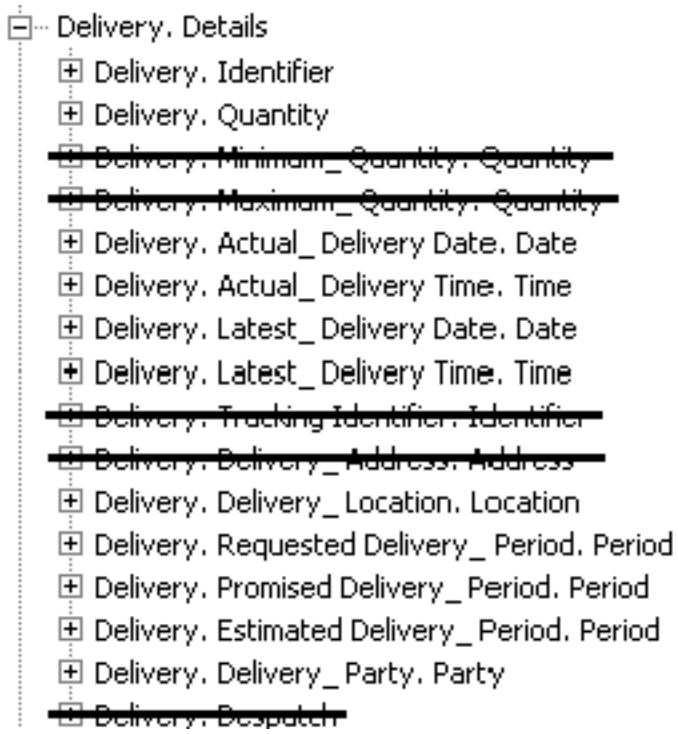

Figure 6. NES common Delivery customization

However, even at the NES Common Library level, the Delivery subset still contains entities that do not make sense in the context of specific documents. For example, the NES project have determined that it not logical to have Minimum_Quantity and Latest_DeliveryDate entities in an Invoice document. Therefore, NES requires one more level of subset customization where only entities relevant to specific document types are present. The NES Invoice Library further restricts Delivery as shown in Figure 7.

Figure 7. NES Invoice Delivery customization

This approach ensures that all NES conformant document instances are UBL conformant as well, as shown in Figure 8.

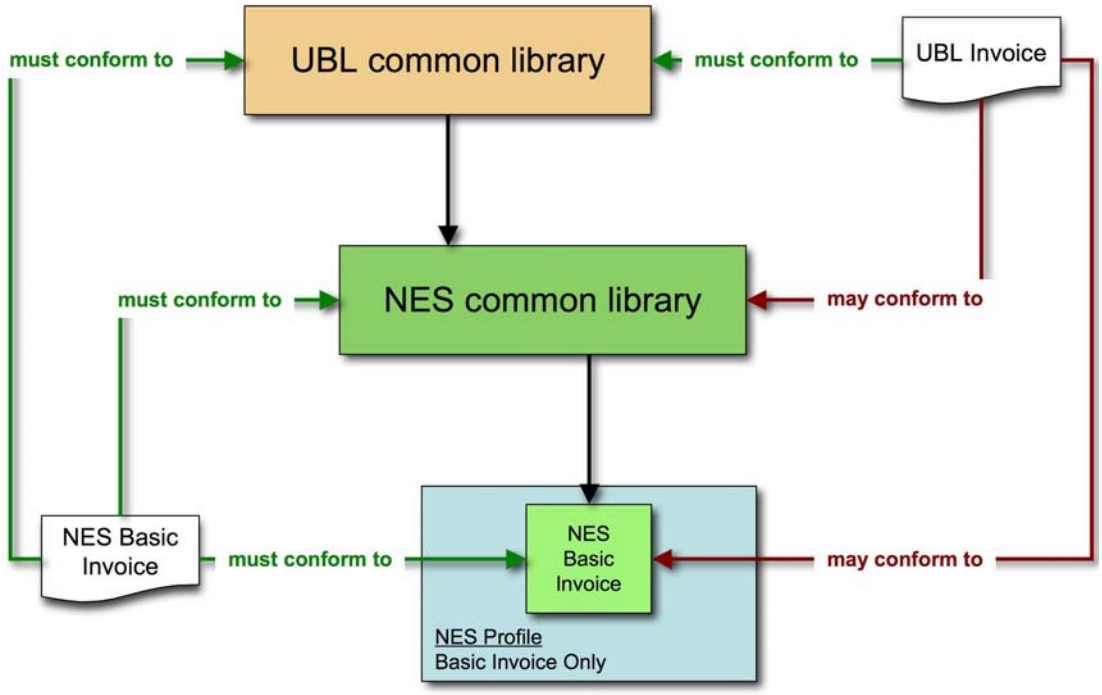

Figure 8. Example of conformance with a UBL subset

2.2 Designing for compatibility

When designing for compatibility (see 1.2.2), the key objective is to re-use as much of the UBL model as possible. Where this is not possible, the guiding principles of UBL should be followed, in particular, its adherence to the UN/CEFACT ebXML Core Component Technical Specification (CCTS).

Compatible extensions (as opposed to conformant extensions) can be made in parts of a schema outside the extension area provided in the Standard version. This may be required where the context of use for a particular entity differs from the context assumed by the UBL model. It also allows validation checks to be built into the compatible schema that cannot be enforced in the extension area of a conformant schema.

2.2.1 Re-use of UBL

Two categories of the UBL library are candidates for re-use in a customization:

– Business Information Entities (BIEs)

  Re-using UBL information entities keeps customization as closely aligned with UBL as possible and prevents an unnecessary proliferation of entities requiring maintenance.

  A key objective should be to re-use existing UBL BIEs at the highest possible level. For example, it is better to re-use the UBL BuyerParty element than to create a competing entity with similar content.

– Data Types

  CCTS defines a set of Core Component Types that should be the basis for all data types.

2.2.2 Compatible extension of UBL

If re-use of existing UBL entities is not feasible, it is possible to customize by extending the UBL library. One may add to the UBL model additional information entities that are needed to satisfy business requirements.

CCTS indicates context of use in several ways, such as by qualifying the Property Term of the entity's Dictionary Entry Name.

Example

Address. Country Subentity Code. Code can be qualified as *Address. Canadian_ Country Subentity Code. Code*, indicating that the context of use for the Subentity code values is Canadian provinces.

If a required aggregate entity has the same structure as a UBL entity, then it should not be a redefinition but a re-use by association.[3] The qualifying terms used to name the new association entity should describe the role it plays.

Example

If an *Address* is required for a *Party*'s local address and this uses the normal address structure, it could be modelled as *Party. Local_ Address*.

If the new aggregate entity does not have the same structure as a UBL entity, then the required entity has a new name, not a qualified name. The new aggregate may associate with the UBL entity being extended.

Example

If an *Address* has additional entities when the address is in Japan, then a new aggregate entity called *Japanese Address* would be created. This is not a qualification, but a new name. Ideally this should contain the original *Address* structure by association plus the new Japanese entities.

Changing or specializing an entity's definition changes the entity (see 1.2.3). Therefore, a new object must be defined.

Example

In UBL, *Communication. Channel. Text* is defined as "The method of communication expressed as text." If an entity is required to define the Skype name as a specific communication channel, then a new entity (perhaps called *Communication. Skype Name. Text*) should be defined.

2.2.2.1 New basic entities

A customization may require new basic entities. These should be based on an existing UBL or CCTS data type (or a refinement thereof). Where the new basic entity is included in an aggregate entity it will result in a new aggregate entity being defined as well. (see 2.2.3)

Example

A *Japanese Address* may have an additional entity called *Prefecture. Text*. This new basic entity would use the standard *Text* data type.

When establishing a new basic entity, it is necessary to associate it with a data type. This also defines the Representation Term part of the entity's Dictionary Entry Name. There are standard

[3] The use of qualified Dictionary Entry Names is not visible in the UBL name (element name), but it does affect the XML type used for the definition.

data types available as part of CCTS. These are known as unqualified data types. There are also some UBL defined data types, known as qualified data types.

2.2.2.1.1 Refined data types

In cases where the required entity's representation does not fit an existing data type, a new qualified data type may be required. New qualified data types can be based on either UBL qualified data types or CCTS unqualified data types.

In UBL, only Code types are qualified, but this does not preclude customizers creating their own qualified data types from other CCTS unqualified data types.

2.2.2.1.2 Refined code types

A basic entity represented by Code type in UBL may be refined with a set of known values. UBL itself provides two sets of definitions for Code types:

- Without defined values (the CCTS unqualified code data type)

  For example, *Country Subentity_ Code* (in *Address*) is assigned the *Code* type.

- With defined values (the code data type is qualified by UBL)

  For example, *Identification_ Code* (in *Country*) is assigned the *Country Identification_ Code* type.

Example

In UBL, *Currency_ Code. Type*, which qualifies a CCTS unqualified data type, is a restriction on the *Code. Type*. A customization for *European Currency_ Code. Type*, may qualify the UBL qualified data type and further restrict the *Currency_ Code Data Type* to specific European currency code values.

Assigning a qualified code list to a basic entity that was previously unqualified restricts the infinite list into a finite list, so this restriction on possible content values defines a subset. Therefore, assigning a qualified code list to a basic entity that was previously unqualified is a *conformant* restriction.

Assigning a new qualified code type to a basic entity already having assigned values will only be a conformant customization if the new qualified code list values are a subset of original qualified code type.

Note that UBL does not arbitrarily create sets of code list values. Where possible, standard international code sets from ISO, UN/ECE, or other standards development agencies should be used.

2.2.2.2 New associations

Aggregate entities are included in a document model by associating them with a parent aggregate. This association is defined as an association entity.

If the required aggregation has the same structure as an existing aggregate, a new association should be created with the existing aggregate (as in 2.2.2). This new association represents a new use of the aggregate and so qualifying terms can be used to describe the new role.

Example

In UBL, *Address* is re-used in contexts such as *Postal_ Address*, *Delivery_ Address* and *Pickup_ Address*. They all share the same structure as *Address* with the terms "Postal," "Delivery," and "Pickup" providing the qualification.

2.2.2.3 New aggregates

A new aggregate should be created if the required aggregation does not exist in UBL or is an extension of an existing aggregate, making it no longer conformant. This aggregate should have a qualified name to avoid possible name collision with future UBL standard aggregates.

A new aggregate may also include the aggregate being extended, as a child by association (as in 2.2.2 above).

Example

UBL itself follows these principles. In UBL, *Customer Party* is a new aggregate that has a different structure than *Party*. The *Party* structure is re-used by association in *Customer Party*. In addition, *Customer Party* also contains additional entities. The name *Customer Party* is not a qualification of the name *Party*, but an extension to the UBL *Party* to create a new aggregate. Figure 9 shows the UBL Customer Party aggregate.

CustomerParty	Customer Party. Details
CustomerAssignedAccountID	Customer Party. Customer Assigned_ Account Identifier.
SupplierAssignedAccountID	Customer Party. Supplier Assigned_ Account Identifier.
AdditionalAccountID	Customer Party. Additional_ Account Identifier. Identifier
Party	Customer Party. Party

Figure 9. Extending an ABIE

When creating new aggregates, the principles of UBL should be followed. This means that aggregations are formed as collections of information entities that share functional dependencies. That is, the only entities that belong in an aggregation are basic entities (or associations to other aggregates) whose values may change when the aggregate itself changes.

Examples

1. The description of an item depends on what that item is. If the item changes, then the description changes. This means the description is functionally dependent on the item, and in this case, the entity *Description* should be aggregated into the aggregate *Item*.

2. If the price of a cup of coffee is based on whether it is to take out, drink at the table, or drink at the bar, then the price is functionally dependent on the location. In this case, the entity *Price* should be aggregated into an aggregate called *Price Location*.

In addition, new aggregates should attempt to re-use patterns of UBL structures where possible.

Example

A customization may require a *Purchaser* aggregate instead of the UBL *Buyer Party*. For compatibility, at a minimum, the UBL *Buyer Party* should be the basis for designing the *Purchaser* aggregate. The advantage of re-using UBL constructs is that there is some degree of traceability back to the original UBL model.

2.2.2.4 New document types

Where existing UBL document types do not meet requirements, it is necessary to create a new document model. The key steps in new document assembly are:

1. Select/create the root aggregate for the document type
2. Assemble the required information entities from the UBL library (and/or customized extensions), applying cardinality constraints.
3. Continue this process recursively through all required associations.

Figure 10 demonstrates the structure of a new document type known as Notification based on the UBL Receipt Advice document type.

First, a new aggregate called *Notification* is created. Two associations to the UBL *Party* are used, one qualified as *Carrier_ Party* and the other as *Consignor_ Party*. The association to the UBL *Shipment* is the only other association for a *Notification*. Following down the pathway of associations from *Shipment*, only *Goods Item*, *Consignment*, *Delivery* and *Transport Handling Unit* are used. Each of these, in turn, uses only the required associations. Therefore, the Notification document type is a compatible customization of the UBL Receipt Advice document.

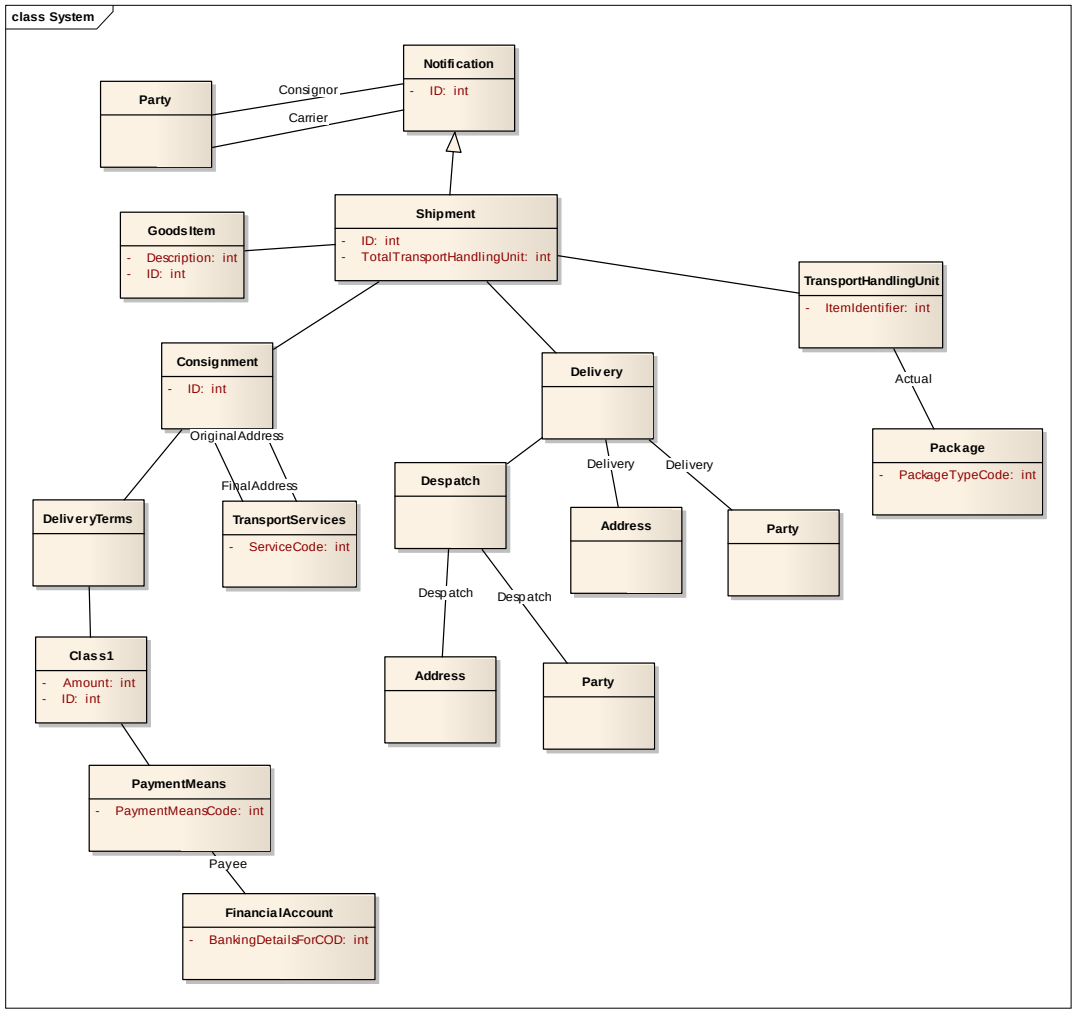

Figure 10. An example design for a compatible document type

2.2.3 The customization ripple effect

The creation of a new information entity or data type affects all entities and data types in its ancestral path. This could be regarded as a ripple effect. Every UBL construct has a distinct, unique identity; any change made within it changes the identity of the whole construct and everything above it in the document tree.

Example

A UBL *Address* is always the same structure. If any entity is added to, or required entity is removed from, the UBL *Address*, it can no longer be identified as the UBL *Address.* It

is recommended that it be given a qualified name to reflect that it is not a standard UBL *Address*. For example, *Customized_Address*.

This change of identity bubbles or ripples upward through any parent of *Customized_ Address*.

This rule guarantees that UBL-consuming code is never “surprised” by an unexpected difference hiding inside an incoming data structure wrongly identified as standard UBL. This difference must at a minimum be indicated by a change in XML namespace.

2.2.3.1 Customized aggregates using subsetting

Consider the following model of a UBL document type, which will be used to illustrate the ripple effect. Every construct is in the ubl: namespace.

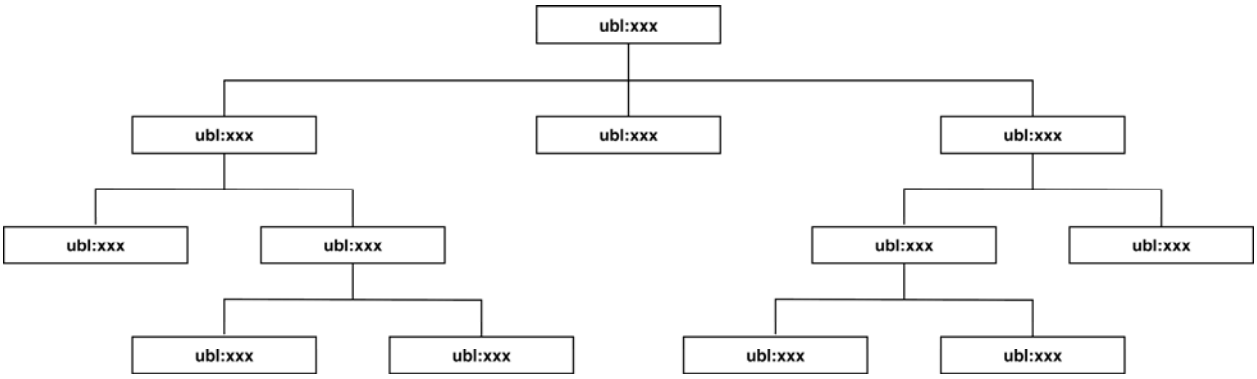

Figure 11. Model of a UBL document type

When a customization is a proper subset of a UBL document type, only *optional* objects are removed (Figure 12). There is no ripple effect; everything keeps the ubl: namespace.

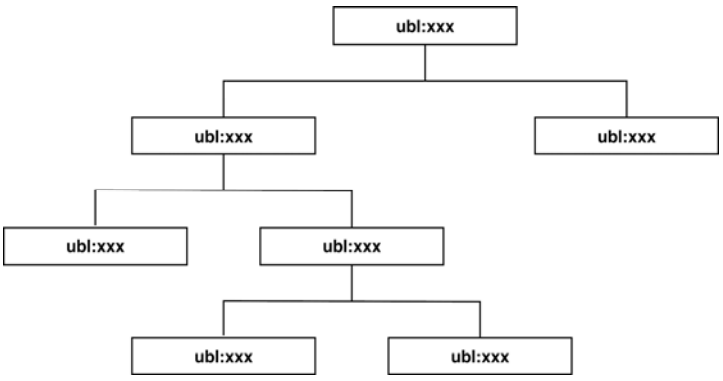

Figure 12. Conformant subsetting (no changes in namespace)

2.2.4 Custom aggregates using UBL entities

When a new aggregate is added to a customized document type, all of its ancestors must also be modified to reflect the new information entity. In the example shown in Figure 13 below, a custom aggregate (“my_ xx1”) is created using standard UBL information entities. It is given a qualified name to indicate that it is not a UBL information entity. Its parent (“new_ xx2”) must then be customized to allow this custom aggregate (“my_ xx1”) in its content model. Accordingly, the document root (“compatible_ xx3”) must also be a customization.

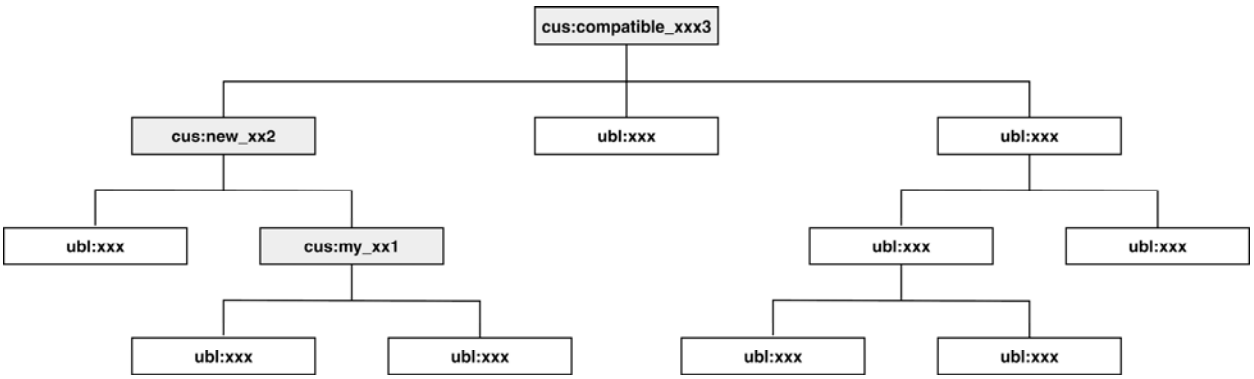

Figure 13. Ripple effect — customized aggregate

2.2.4.1 Custom aggregate using custom entities

When a new information entity is added to a customization, all of its ancestors must also be modified to reflect the new information entity. In the example in Figure 14 below, a customized aggregate (“my_ xx2”) is created by adding a custom basic information entity (“xx1”). Its parent (“new_ xx3”) must then be customized to allow this custom basic entity (“xx1”) in its content model. Accordingly, the document level aggregate (“compatible_ xx4”) must also be a customization.

Note that the new basic information entity may not have a qualified name. Its customization context is given by its parent.

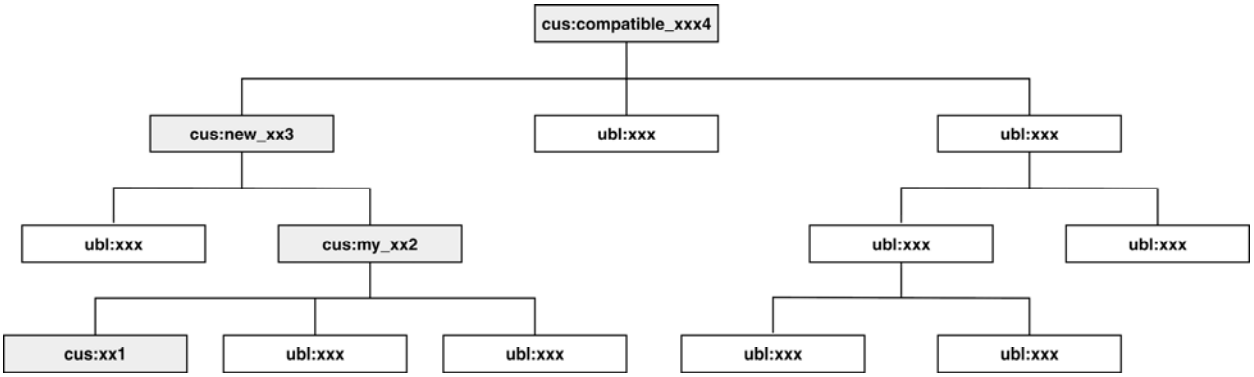

Figure 14. Ripple effect — customized Basic Information Entity

To sum up:

- Customizing a data type creates a new basic information entity
- Customizing a basic information entity creates a new aggregate information entity
- Customizing an aggregate information entity means creating a new aggregate information entity and new associations that refer to it
- Customizing an association creates a new aggregate
- Any new aggregate means a new document model

Any nonconformant customization means a new document model.

3 Specification

A specification is used to describe to communities and developers of document interfaces the set of valid instances of an XML document type. In UBL, these specifications form the basis of the profiles of UBL used in specific business process contexts.

The same customized document instance can be specified using different syntaxes and methods. Several of these are described in this section. UBL does not mandate the use of any given syntax or method for specifying customizations (or profiles) because this choice does not affect the conformance (or compliance) of the document instances to the UBL standard.

3.1 Using XML Schema

The UBL TC uses XSD, the standard XML schema language produced by the World Wide Web Consortium (W3C), to specify its document formats. There are formal Naming and Design Rules [NDR] for the use of XML Schema to specify UBL documents.[4] The UBL Naming and Design Rules should be used where possible when specifying the model using XSD.

Therefore it is appealing to use XML Schema for specifying customized document formats as well. However, there are several ways in which this can be achieved.

3.1.1 Customized schemas

Schema customization formed the basis of the UBL 1.0 Context Methodology. Feedback from those attempting to apply this methodology has led the UBL TC to be more catholic in its approach to customization in UBL 2.0, though the approach recommended in the 1.0 Context Methodology remains valid for customizing by making changes directly to the standard schemas in certain circumstances.

Two scenarios in particular lend themselves to XSD derivations performed on existing types:

- An existing UBL type fits the requirements for the application with modifications supported by XSD derivation. These modifications can include extension (adding new information to an existing type) and/or refinement (restricting the set of information allowed to a subset of that permitted by the existing type).
- No existing UBL type is found that can be used as the basis for the new type. Nevertheless, the base library of core components that underlies UBL can be used to build up the new type so as to ensure that interoperability is at least possible on the core component level.

However, XSD derivation does not support certain customization requirements:

- Unable to declare derivatives of the extension point

  It is not possible to express in an XSD extension or restriction of the published UBL schemas that a given extension element is allowed to be a child of the extension point. Consider the two possibilities based on the published UBL schemas defining the extension element with an xsd:any constraint of ##any to allow any element of any namespace to be a child of the element:

[4] Note that logical constructs known in CCTS as Business Information Entities (BIEs) and Data Types are both implemented in XSD as “types”, and the terminology used in this section reflects this.

Extension

In the case of extension, a deriving schema attempts to add the definition of the customization extension element to the children of the UBL extension point (it is unclear how this is done because of derivation rules in XSD). A validating processor is obliged to first satisfy the base schema expression for the extension element before attempting to satisfy the extension constructs. But the processor will have already consumed all of the particles with the ##any of the base schema before hitting the end of the extension children; thus, when it attempts to validate the presence of the extension element, there are no particles left to be the extension element.

Restriction

Similarly, in the case of restriction, a deriving schema attempts to restrict the definition of the UBL extension point to be elements of any namespace, followed by the customization extension element, followed by elements of any namespace. Again the use of ##any directs the validating processor to consume all children of the extension point, and only when done will it then try to find an extension element which is not there.

- Unable to elide optional elements through derivation

  Should a customization definition wish to elide an optional element and make it totally unavailable, there is no way an XSD schema can restrict an existing content model to indicate that an optional element already declared in the base model is not included in the restricted model.

- Unable to express different enumeration restrictions based on context

  All elements in UBL are global; thus, those with enumerated data types necessarily have global scope across an entire instance. There is no way an XSD schema can restrict an existing content model to indicate that a contextual use of a data type has a different subset of enumerated values than in another contextual use.

- Unable to express co-occurrence constraints

  There is no way to express in an XSD schema a constraint on the existence of, or the contents of, information entities based on the existence of, or the contents of, other information entities.

- Unable to maintain modeling conventions using XSD extension

  In UBL all aggregate entities are modeled with all basic entities listed first as children, followed by all associate entities listed next as children. XSD extension allows additional constructs to be added only after all of the base constructs. Should a revision to a UBL aggregate entity need a new child basic entity, this basic entity cannot be placed before child associate entities when using XSD extension.

3.1.2 New document schemas

XSD schemas are used in UBL to express normative document constraints. It is possible to express the same document constraints in other schema languages such as RELAX NG or even by using Turing-complete assertion languages such as Schematron. Since UBL uses XSD for its standard schemas, however, it is assumed in the following that new schemas based on UBL will use XSD simply to save labor. If XSD is chosen, new compatible document types should adhere to the UBL Naming and Design Rules, and if other formalisms are chosen, the UBL NDR conventions should be followed where possible.

Several tools exist for generating new UBL NDR conformant document schemas from logical models. Some of these are listed at the UBL online community website, ubl.xml.org (http://ubl.xml.org/products).

3.1.3 Subset schemas

Where the requirements are for a pure subset (as noted in 2.1.1 and illustrated in Figure 12), it is possible to prune a UBL document schema to create a new, smaller schema defining only the subset required.

Because UBL relies on a common library of re-usable types, this approach does not support the restriction of selective types based on context. That is, an Address when used in one part of the subset schema cannot have a different restriction from an Address in another part of the document.

One approach for producing subset schemas is to work with the UBL schemas as input and use the XML comment construct to elide all of the information entities not used by the customization. A human reader of the schema specifications can see all of the UBL standardized constructs, easily distinguishing those that are in the customization and those that are not.

Another approach for producing subset schemas is to work at an abstract model level and to synthesize the schema fragments from scratch from the subset model. This is the approach taken by the NES project group[5] (see 2.1.4). Figure 15 shows the schema fragment that specifies the NES Invoice Delivery customization shown in Figure 7.

```
<xsd:complexType name="DeliveryType">
  <xsd:sequence>
    <xsd:element ref="cbc:ID" minOccurs="0"/>
    <xsd:element ref="cbc:Quantity" minOccurs="0"/>
    <xsd:element ref="cbc:ActualDeliveryDate" minOccurs="0"/>
    <xsd:element ref="cbc:ActualDeliveryTime" minOccurs="0"/>
    <xsd:element ref="DeliveryLocation" minOccurs="0"/>
    <xsd:element ref="PromisedDeliveryPeriod" minOccurs="0"/>
    <xsd:element ref="EstimatedDeliveryPeriod" minOccurs="0"/>
    <xsd:element ref="DeliveryParty" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Figure 15. An example of a subset schema

3.1.4 Using UBLExtension

The one exception to the general rule that only subsets are conformant is the UBLExtension element. If new entities are added to an existing document type exclusively in the extension area, instances validating against the extended schema are still UBL conformant. But in these cases, schema validation cannot ensure the structural integrity of the new entities.

The UBLExtension element found at the beginning of all UBL documents allows communities of interest to specify additional information entities. Conformance is not affected by the content of the UBLExtension, as it may contain any type of information entity (by using <xsd:any> in its declaration). If new entities are added to a UBL document type only in the extension area, any

[5] See http://www.nesubl.eu/documents/nesvalidationtools.4.6f60681109102909b80002641.html

instances validating against the extended schema are still UBL conformant (but may not be UBL compatible).

The UBLExtension element is not one of UBL's logical document models. It is a structural device that allows arbitrary extensions to a UBL document type without affecting UBL conformance. As such, it is an artefact of document specification, not document design.

Having only one location for extensions manages the expectations of applications for locating added non-standard constructs. Note that extended entities are not allowed anywhere else in a UBL document type outside of the UBLExtension element, otherwise validation against standard UBL schemas will report errors of unexpected content.

Injudicious use of UBLExtension will obviously have damaging consequences for understanding the meaning of information in the documents. UBLExtension should never be used for information that may properly be conveyed in standard UBL types elsewhere in the document. Metadata available on each UBLExtension should be used to identify the nature and source of the extension.

There are two situations where UBLExtension may be considered appropriate:

1. Where the requirement is to incorporate alien content in a standard UBL document type that cannot be contained as an Attachment.

   In the following example, UBLExtension is used to specify legacy EDIFACT information entities (defined here as LegacyExtension) that must be included in the document instance for message routing purposes.

```
<ext:UBLExtensions>
  <ext:UBLExtension>
    <cbc:ID>WMP1</cbc:ID>
    <cbc:Name>WMData</cbc:Name>
    <ext:ExtensionAgencyID>EAI1</ext:ExtensionAgencyID>
    <ext:ExtensionAgencyName>EAN1</ext:ExtensionAgencyName>
    <ext:ExtensionAgencyURI>EAU1</ext:ExtensionAgencyURI>
    <ext:ExtensionURI>urn:wmdata.dk:example</ext:ExtensionURI>
    <ext:ExtensionReasonCode>OPT</ext:ExtensionReasonCode>
    <ext:ExtensionReason>wmdata legacy invoice material
          </ext:ExtensionReason>
    <ext:ExtensionContent>
      <wmp:LegacyExtension xmlns:wmp="urn:urn:wmdata.dk:example">
              ... legacy invoice stuff ...
       </wmp:LegacyExtension>
    </ext:ExtensionContent>
  </ext:UBLExtension>
</ext:UBLExtensions>
```

Figure 16. An example of extending with alien content

2. Where a customizing organization wishes to extend information entities in a standard UBL document type and still have their documents be validated by the standard UBL schema.

   In the example in Figure 17, the UBL *Address* has been extended to include a *Postoffice* information entity. This new structure is known as *AlternativePostalExtendedAddress*.

```
<Order>
  <ext:UBLExtension>
    <ext:ExtensionContent>
      <eac:AlternativePostalExtendedAddress>
        <ebc:Postoffice>South Bridgtow</ebc:Postoffice>
        <eac:Address>
          <cbc:Postbox>2234</cbc:Postbox>
          <cbc:PostalZone>ZZ99 0AA</cbc:PostalZone>
          <cac:Country>
            <cbc:IdentificationCode>GB</cbc:IdentificationCode>
          </cac:Country>
        </eac:Address>
      </eac:AlternativePostalExtendedAddress>
    </ext:ExtensionContent>
  </ext:UBLExtension>
  ... standard UBL Order document ...
</Order>
```

Figure 17. An example of extending UBL information entities

3.1.4.1 Referencing information in UBLExtension

There are some challenges when using UBLExtension to specify optional extensions to aggregates that may have many occurrences — that is, when the extended entity has a minimum cardinality of zero and the aggregate being extended has a maximum cardinality of many.

The problem arises when instances contain items of a certain type that may or may not be extended by information in the extension area. For example, suppose we require extension to the UBL aggregate, *Item*, to allow a *CarbonEmissionRating,* and not all *Items* have a rating. Then in a given instance of a document, some *Items* may be extended to include their *CarbonEmissionRating* and others may not. The challenge is how to specify in the UBLExtension area which *CarbonEmissionRating* belongs to which *Item* in the main body of the document.

This problem can be generalized as the need to specify the precise context (or position in the document tree) pointed to by each element in the UBLExtension. There are two approaches to solving this.

1. Use a reference identifier.

   Many constructs in UBL, for example line items and parties, are already modeled to have identifiers. Reusing these identifiers in extension content provides a natural association between content found under the extension point and content found in the standardized constructs, resulting in a virtual extended record. In Figure 18 the UBL LineItem/ID is used to establish which line item the LineItem/custInfo applies to.

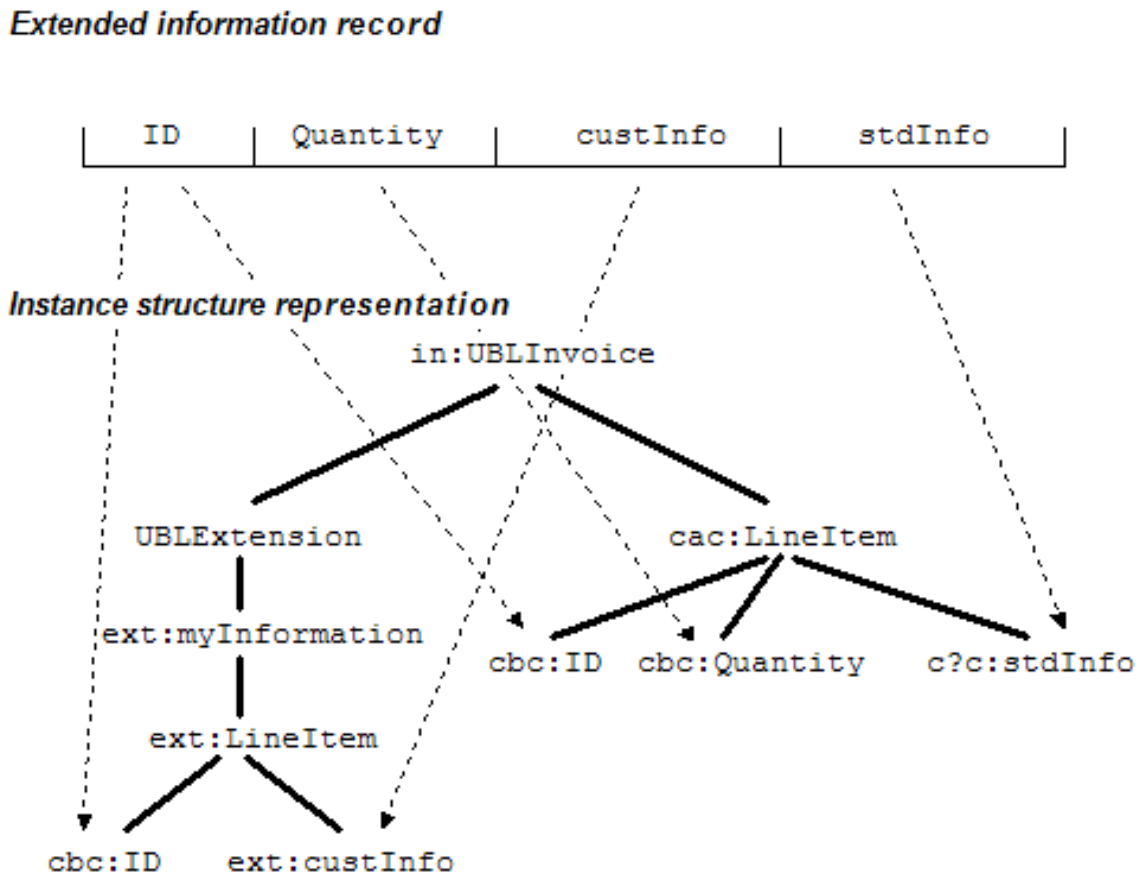

Figure 18. Using a shared ID to connect information in UBLExtension with a line item

Some UBL aggregates have no identifiers, however, and in such cases a surrogate unique identifier must be used to link the entity in the extension with the relevant entity in the document body.

2. Replicate the entire aggregate in the UBLExtension.

   UBLExtension can contain a copy of the associated information from the body of the document instance so that the extensions are found in their context.

   In Figure 19, the entire *LineItem* (with the additional entity) is repeated in the UBLExtension, and an appropriately configured application finds all the extended records in one place.

   Replication may require increasingly larger portions of the document to be included in the UBLExtension to unambiguously identify the context of an extended entity. Taking this to its extreme, it is possible to specify the entire body of the extended document in the UBLExtension. This means that each document contains two versions — one (in the body) without any extensions and the other (in the UBLExtension) as the required document including extensions. The body of the document then contains the UBL conformant information and the UBLExtension contains the actual document content required for the business process.

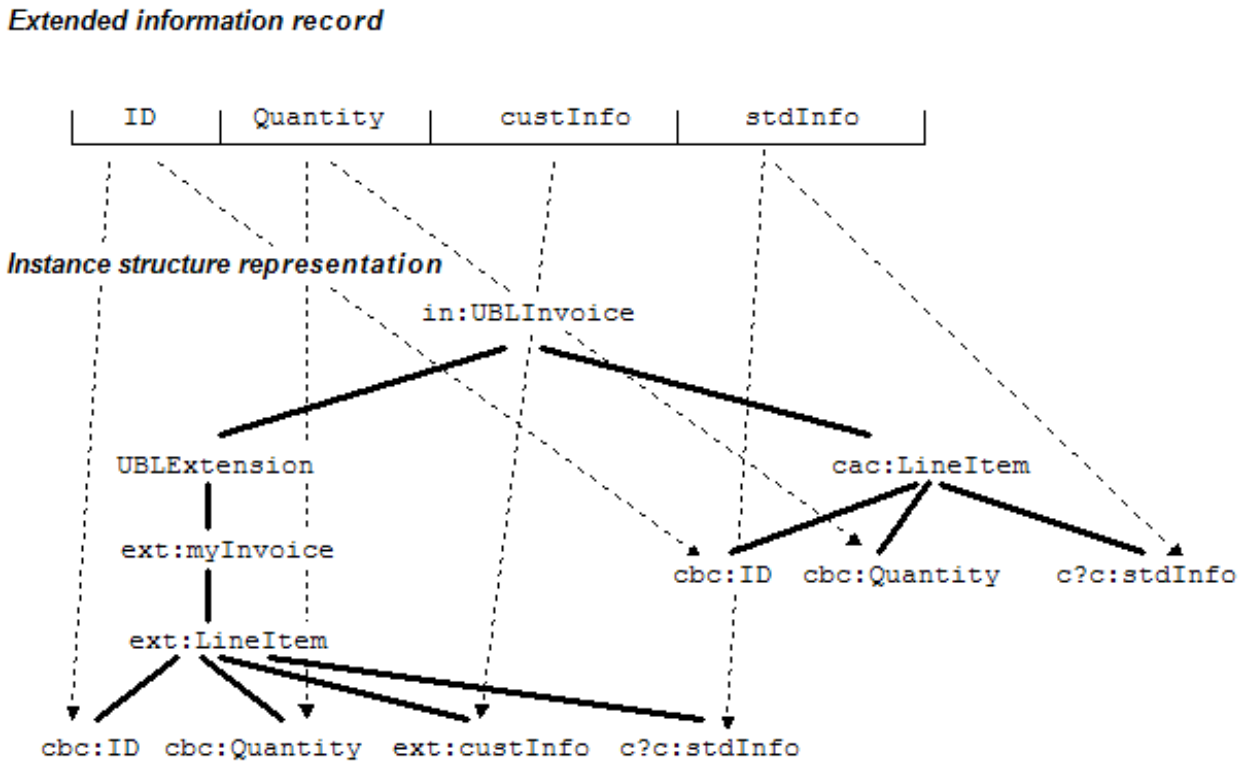

Figure 19. Replication within UBLExtension

3.2 Using XPath

XPath syntax may also be used to specify a customization. The XPath recommendation [XPath 1.0] defines a model for the information found in XML instances. The specification describes well-formed instances (which may or may not be valid). It focuses on the information found in the instance and not the syntax used in the instance to express the information.

Because XPath specifies the absolute document structure in its entirety, it is possible to restrict selective types based on context. For example, an Address when used in one part of the schema may have a different customization than in any other context.

The UBL Human Interface Subcommittee [HISC] project has created an XML vocabulary for enumerating information entities in a catalogue of available XPath addresses from the document element to all entities allowed by a given document model described by a schema or to all entities found in a particular XML instance. The normative instance of an XPath file for a given document model is an XML instance of the XPath file vocabulary [XPath File]. This instance can be machine-processed by any XML-aware application and can also be used to create human-readable reports and diagnostic materials.

The UBL NDRs make it straightforward to create XPath files from the published XSD expressions,[6] and XPath files for UBL schemas are publicly available [UBL-XPath]. These XPath files express in a programmatically processed form all of the possible combinations of XML hierarchy for the information entities described by each UBL document type. The size of the resulting files makes this technique best suited to restrictions or subsets of UBL document

[6] Note that XPath files need not be generated from XSD schemas or XML instances. The UBL logical models can also be used as a source for creating XPath files.

types. The UBL Small Business Subset version 1.0 [SBS1.0] is an example of how a subset may be specified using XPaths.

3.3 Using genericode

UBL uses the genericode XML format to specify values (and associated metadata) for code lists. The latest code lists used in UBL are found in the subdirectories of

http://docs.oasis-open.org/ubl/os-UBL-2.0-update/cl/gc/

The cefact directory under cl/gc has the code lists associated with the supplemental components of the CCTS unqualified data types. These components are found in all UBL basic information entities whose types are derived from the related unqualified data types.

The default directory under cl/gc has the TC-defined code lists associated with UBL basic information entities whose types are derived from the CCTS CodeType. The values in this list constrain the supplied “second pass value validation” example found in the sample validation directory:

http://docs.oasis-open.org/ubl/os-UBL-2.0-update/val/

The special-purpose directory under cl/gc has a selection of code lists that users may find useful in their deployment of UBL but that are not included in the supplied value validation example.

The genericode OASIS standard is recommended as the syntax for specifying constrained sets of possible values in customizations of UBL as well.

Code list customization can be applied in many cases:

- Extensions of new types: Where a new type has been added in a customization that requires a code (or other form of value constraint). For example, the *CarbonEmissionRating* (above) may use a formal coding system. Instances with these values are not UBL conformant.
- Extensions of existing types: Where a new value for an existing type has been added in a customization as a code (or other form of value constraint). For example, a customization may need an as-yet-not-standardized new code for *PaymentMeansCode*. Instances with these values are not UBL conformant.
- Restrictions of specified code lists: Where an existing type has an existing list of applicable codes and a customization needs to restrict the use of codes to a subset. For example, restricting *PaymentMeansCode* to only cash and credit card and no other means of payment. Instances with these values are UBL conformant.
- Restrictions of unspecified code lists: Where an existing type without an existing list of applicable codes has a customized code (or other form of value constraint) applied to it. For example, restricting *CountrySubentityCode* to the US state codes in a profile for the United States. Instances with these values are UBL conformant.
- Identifiers: Where a basic information entity uses a type derived from the CCTS IdentifierType. Instances with these values are UBL conformant.
- Values for any other basic information entity: Where a basic information entity uses any type and the customization wishes to constrain the value to one of a controlled set of values. Instances with these values are UBL conformant.

Note that genericode only provides a way of specifying the values of a code list; it does not provide for specifying the contexts in which the values are used. An example of a specification providing contextual use of values from genericode files is Context/Value Association (CVA), which was used by the UBL TC in the creation of the artefacts in the sample validation directory.

The latest versions of both the genericode OASIS standard and the CVA work-in-progress can be found linked from:

http://www.oasis-open.org/committees/codelist

While genericode provides for the specification of list-level metadata (about the list of codes as a whole) and value-level metadata (about each coded value found in the list), the CVA file provides for the specification of instance-level metadata (about the list-level metadata associated with a particular coded value used in an instance).

When a customization creates any kind of code list in genericode, it has the obligation to ascribe unique list-level metadata to that list, even if that list is a subset of another list with its own list-level metadata. Every list must be uniquely identified. Where necessary, the CVA file provides for masquerading in an instance the use of a value from a customized list as if it were a value from an original list.

Note that genericode and CVA files may have a role in many processes other than instance validation, such as in constraining data entry.

3.4 Using Schematron

There are some business rules a customization or two trading partners may follow that constrain the values used in UBL documents. Such value constraints cannot be specified easily using schema validation semantics. A useful syntax for the formal assertion of these value constraints is Schematron (ISO/IEC 19757-3).

A co-occurrence constraint constrains one or more components of document content based on one or more other components of document content. The basis can be the presence or absence of content, or particular values of content. For example, one could assert that for each itemized information entity that is based on the UBL party, one or both of cac:PartyIdentification/cbc:ID and cac:PartyName/cbc:Name must be present, but not neither.

A community or trading partner constraint may be a value calculation not expressible as one of a set of codes, such as the applicability of a particular tax based on the value of an item. For example, one could assert that associated tax information entities are mandatory when the item's value exceeds a specified amount, while they must be absent when the item's value does not exceed a specified amount.

Using Schematron, a customization can specify all such assertions in a declarative fashion independent of how the assertions are actually implemented as running code in a validation process.

Note there are implementations of CVA files that incorporate business rules expressed as Schematron assertions when aggregating all value constraints applicable to XML documents.

3.5 Managing specifications of customizations

It is possible to create a metamodel that describes the various aspects of customization. This may then be used to create and manage document specifications based on customizations of UBL, including any customized BIEs, business rules, and value constraints. This approach has been used by the Danish OIOUBL project[7] to create and maintain their documentation.

A useful source of customization specifications is the UBL community website, ubl.xml.org.

[7] See http://www.oioubl.info/classes/en/

4 Validation

The UBL committee has published a processing model for a UBL system receiving an XML UBL document, as illustrated Figure 20.

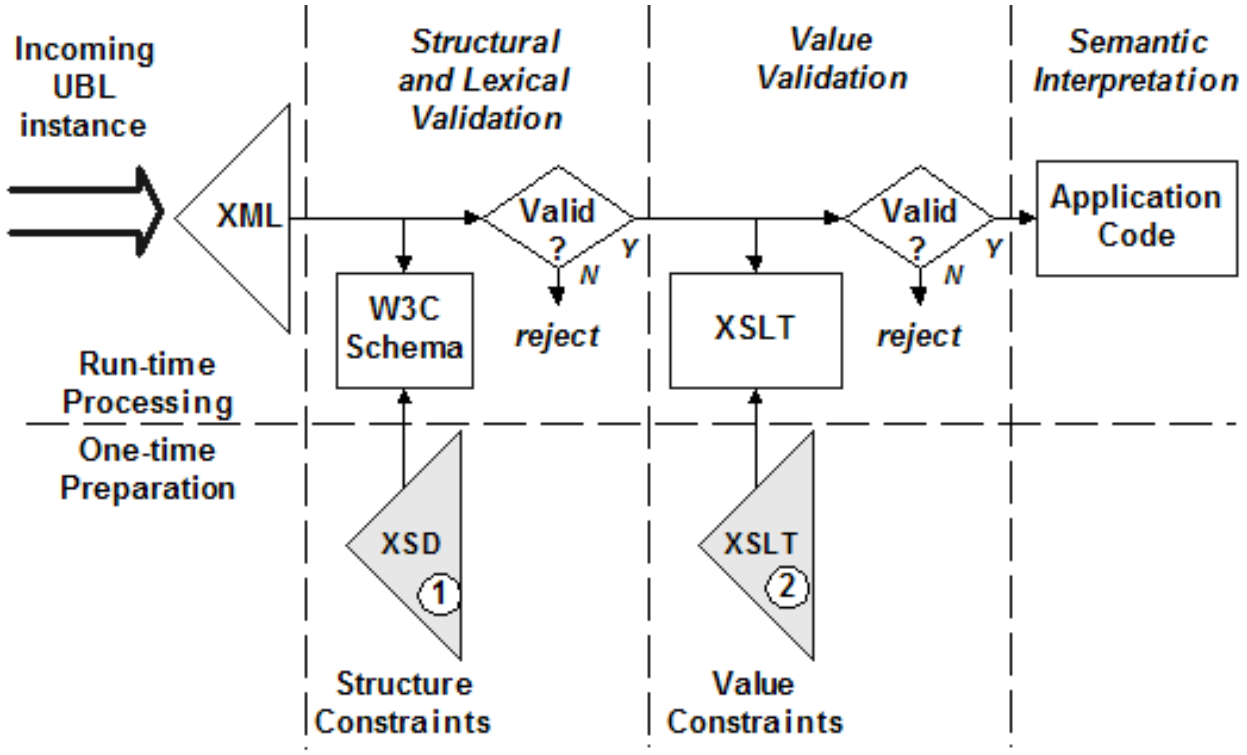

Figure 20. The published processing model for UBL

In this model, two distinct steps are engaged to determine the validity of an instance for processing by a receiving application. The structural and lexical constraints are expressed in the W3C Schema XSD file. The value constraints are expressed in an XSLT file. Standardized versions of each of these two files are included in the UBL 2.0 specification package. Only when an instance has successfully passed structural validation does it make sense to check value validation. At either stage of validation, a failure indicates that the message is to be rejected, either because the document structure or value constraints have been violated.

If the application requires schema validity for the loading of data structures, this is assured by the first step. Checking the value constraints in the second step relieves the application from having to know which constraints apply, and processing can focus on whatever values have been allowed to pass. Thus the application can be quite generic in nature by supporting all possible values. The application does not have to change if the constraints on values change in different business contexts.

A receiving application is assumed to have been programmed to be aware of only the constructs of a particular customization of UBL. It will therefore be deployed with the schemas for that UBL customization and will typically perform validation of received documents in advance of acting on the semantics represented by the information structured and identified in the XML. The customized application receiving an instance conforming to a complete UBL schema, to a different customization, or to a later version of the same customization may find

either unrecognized constructs or recognized constructs in unexpected places. For example, a customization version 2.5 application would not recognize foreign constructs or constructs introduced by the schema for the customization's version 2.7.

The published processing model for like-versioned UBL systems does not support a version 2.5 application receiving foreign content or a version 2.7 instance with unexpected content.

Figure 21 illustrates a processing model augmenting the processing model described in the UBL 2.0 specification.

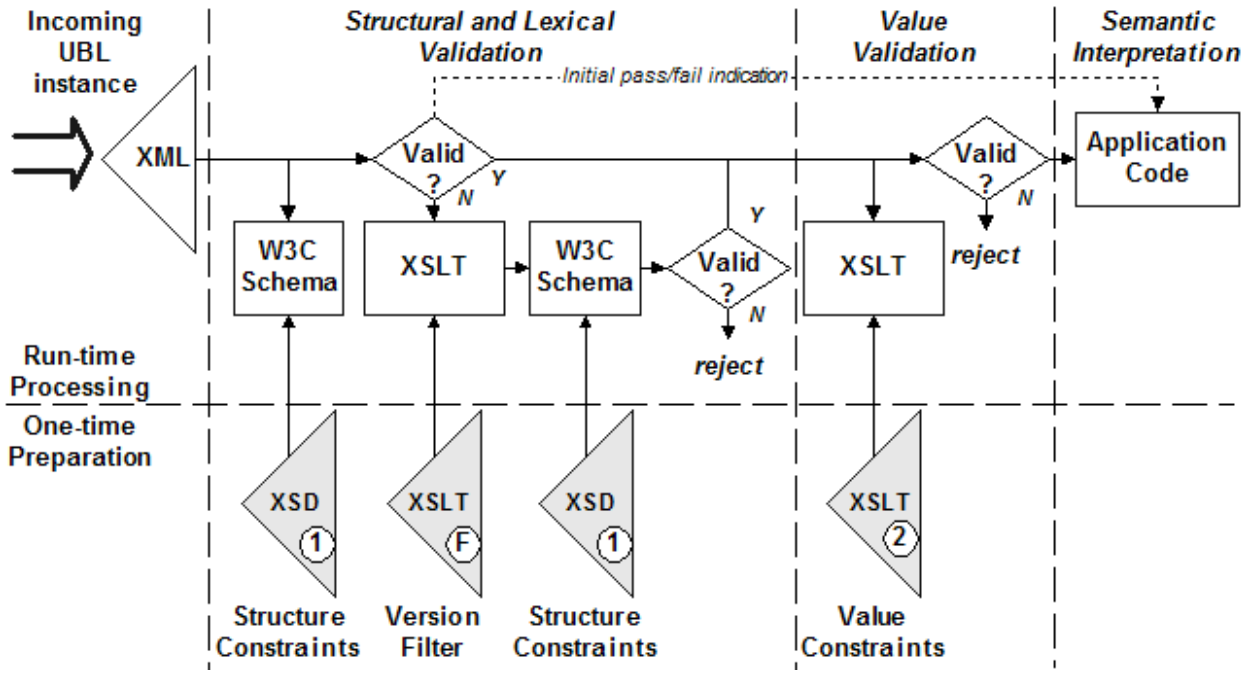

Figure 21. A customized processing model supporting forward compatibility

The word "version" here applies equally well to customizations of UBL, versions of UBL, versions of customizations of UBL, and customizations of versions of UBL.

This alternative processing model for the receiving system uses only that version of UBL schema supported by the receiving system and does not involve any inspection of the XML instance in advance of validation. In this model, an initial schema validation failure indication is recognized to possibly have been triggered by an instance using features added in a schema later than the version supported by the system. After such a failure, an instance pruning process takes away unknown constructs from the instance being validated. The resulting pruned instance can then be checked for schema validity. If successful, the pruned instance is passed to the second stage value validation.

As with the standardized model, passing value validation grants delivery of the instance to the application. In this model, however, a second piece of information accompanies the instance being passed to the application. The application can already assume that value constraints in the document are satisfied. An "initial pass/fail" indication tells the application that the instance it is working with satisfies the structure constraints in either an unmodified ("initial pass") or a modified ("initial fail") state.

An unmodified instance can be acceptable for business processing regardless of the stated version number found in the UBLVersionID element or the string found in the UBLCustomizationID element if all of the business objects found in the instance conform to the

constraints of the application, unused additions in a later version notwithstanding. The application can use out-of-band decision making, including these elements as input, to accept or reject a modified instance for the purposes of doing business.

In both cases, if the instance is delivered to an application, such an application relying on schema validity for inspecting instance content can successfully extract any information in the instance.

Considering the example above, a UBL 2.7 instance without constructs unrecognized by the UBL 2.5 schema would validate using the receiving application's schemas. The instance would be passed to the UBL 2.5-aware application untouched and with an "initial pass" indication. In this case, that the instance is marked 2.7 is irrelevant. A UBL 2.7 instance with unrecognized constructs would fail to validate with the UBL 2.5 schema and would be passed to the application after being pruned to the UBL 2.5 subset and with an "initial fail" indication. In this case, that the instance is marked 2.7 is relevant to the application and to the user deciding how to proceed.

4.1 The version high water mark

Consider the example where a sending system supporting features up to UBL 2.7 generates an instance wherein the highest version of UBL represented by any construct used therein was defined in UBL 2.3. No additions defined by UBL 2.4, 2.5, 2.6, or 2.7 are used within the instance. Though the system supports the creation of a UBL 2.7 instance, the "high water mark" of the structure is only 2.3.

The sending application has to decide to indicate in UBLVersionID the value "2.3" or the value "2.7".

A receiving system supporting only UBL 2.2 would accept the instance after the second check of schema validity (Figure 21). The first check of validity would have triggered the instance pruning through the 2.2 filter and the resulting instance would then validate as 2.2. The application would inspect the instance with the knowledge that the instance failed the initial validation. If it found a version of 2.2 or lower, the application could conclude that the instance was improperly structured and only the pruning process cleaned the instance up. In this example, seeing a version higher than 2.2, the application wouldn't know whether the instance was improperly structured or whether the failure was only the presence of additional content. Nevertheless, the application can use an out-of-band decision to continue with the transaction or reject it. This might include human inspection or authorization.

A receiving system supporting UBL 2.3 would accept the instance structure and the application would be able to inspect the content. There would be no need to inspect the asserted UBL version because the "initial pass/fail indication" cites the successful validation against UBL 2.3 structures. This happens regardless of whether the UBLVersionID states "2.3" or "2.7".

Likewise, a receiving system supporting UBL 2.7 would accept the instance without needing to inspect the UBLVersionID.

Thus, there is no obligation for a sending system to ascertain the high water mark of constructs used in an instance. Indeed, it may be a burden to quality assurance and testing in application development to test that an application meets the high water mark requirement. By always populating UBLVersionID with the highest version of UBL supported by the sending application, this statement will always be true. An instance of UBL 2.3 is, in fact, an instance of UBL 2.7, so it is safe to say "2.7" in the instance.

5 Conformance

This document is intended for guidance in using UBL and therefore contains no conformance requirements.

Appendix A References

[CCTS]	ISO 15000-5 ebXML Core Components Technical Specification
[HISC]	OASIS UBL Human Interface Subcommittee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl-hisc
[NDR]	UBL 2.0 Naming and Design Rules, http://docs.oasis-open.org/ubl/prd-UBL-NDR-2.0.pdf
[SBS 1.0]	UBL Small Business Subset 1.0, http://docs.oasis-open.org/ubl/cs-UBL-1.0-SBS-1.0/
[UBL-XPath]	http://www.oasis-open.org/committees/download.php/16336/UBL-2.0-SBS-20060120-XPath.zip
[XPath1.0]	W3C XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath
[XPath File]	http://docs.oasis-open.org/ubl/submissions/XPath-files/