UBL 2 Guidelines for Customization, First Edition

Public Review Draft 03

16 October 2009

Specification URIs:
This Version:
http://docs.oasis-open.org/ubl/guidelines/UBL2-Customization1prd03.pdf (Authoritative)
http://docs.oasis-open.org/ubl/guidelines/UBL2-Customization1prd03.doc
http://docs.oasis-open.org/ubl/guidelines/UBL2-Customization1prd03.html

Previous Version:
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0prd02.pdf
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0prd02.doc
http://docs.oasis-open.org/ubl/guidelines/UBL-Customization1.0prd02.html

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

Chairs:
Jon Bosak
Tim McGrath

Editors:
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.

Copyright © OASIS® 2009. All Rights Reserved.

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/.

Copyright © OASIS® 2009. All Rights Reserved.

Notices

Copyright © OASIS® 2008-2009. All Rights Reserved.

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 contents 4
Table of figures 5
1 Introduction 6
1.1 Definition of terms 6
1.2 Informative references 7
1.3 Acknowledging OASIS copyright 7
1.4 Conformance vs. compatibility 8
1.4.1 UBL conformance 8
1.4.2 Code list conformance 9
1.4.3 UBL compatibility 12
1.4.4 Maintaining common meanings 13
1.4.5 Customization profiles 13
1.4.6 Identifying versions, customizations, and profiles 14
1.5 Overview of customization methodology 14
1.6 Calculation models 15
2 Designing for UBL customization 16
2.1 Designing for conformance 16
2.1.1 Subsets of the document model 16
2.1.2 Code list constraints on document content 17
2.1.3 Other constraints on document content 17
2.1.4 Examples of conformant customizations 18
2.2 Designing for compatibility 21
2.2.1 Re-use of UBL 21
2.2.2 Compatible extension of UBL 21
2.2.3 The customization ripple effect 26
3 Specification 29
3.1 Using XML Schema (XSD) 29
3.1.1 Customized schemas 29
3.1.2 New document schemas 30
3.1.3 Subset schemas 31
3.1.4 Using UBLExtension 32
3.2 Using XPath 37
3.3 Using genericode 38
3.4 Using Schematron 38
3.5 Using the UBL library for non-UBL document types 39
3.6 Managing specifications of customizations 39
4 Validation 41
5 Conformance 44

Copyright © OASIS® 2009. All Rights Reserved.

Table of figures

Figure 1. Conformant schemas and document instances 8
Figure 2. Compatible schemas and document instances 13
Figure 3. Overview of UBL development methodology 15
Figure 4. NES subset architecture 18
Figure 5. UBL standard Delivery aggregate 19
Figure 6. NES common Delivery customization 19
Figure 7. NES Invoice Delivery customization 20
Figure 8. Example of conformance with a UBL subset 20
Figure 9. Extending an aggregate information entity 24
Figure 10. An example design for a compatible document type 26
Figure 11. Model of a UBL document type 27
Figure 12. Conformant subsetting (no changes in namespace) 27
Figure 13. Ripple effect — customized aggregate 27
Figure 14. Ripple effect — customized basic information entity 28
Figure 15. An example of a subset schema 31
Figure 16. Overlaying customization schema fragments 32
Figure 17. An example of extending with alien content 33
Figure 18. An example of extending UBL information entities 34
Figure 19. Extension of non-UBL business objects 35
Figure 20. Using a shared ID to connect information in UBLExtension with a line item 36
Figure 21. Replication within UBLExtension 37
Figure 22. Using the UBL library for non-UBL documents 39
Figure 23. The published processing model for UBL 41
Figure 24. A customized processing model supporting forward compatibility 42

Copyright © OASIS® 2009. All Rights Reserved.

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. In UBL, information entities are expressed as XML information items.

Business Information Entity: Following the concepts of the ebXML Core Component Technical Specification (CCTS), a Business Information Entity (BIE) can be a Basic Business Information Entity (BBIE), an Association Business Information Entity (ASBIE), or an Aggregate Business Information Entity (ABIE).

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: To be consistent with UBL information entities and the principles behind UBL's models or their development.

Version: The word "version" used in this document applies to customizations of UBL, dot-releases of UBL, dot-releases of customizations of UBL, and customizations of dot-releases of UBL.

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

Copyright © OASIS® 2009. All Rights Reserved.

XSD: A synonym for W3C XML Schema.

1.2 Informative references

The following documents are referenced in the text.

[CCTS] ISO/TS 15000-5:2005, *Electronic Business Extensible Markup Language (ebXML) -- Part 5: ebXML Core Components Technical Specification, Version 2.01 (ebCCTS).* http://www.iso.org/iso/iso_catalogue /catalogue_tc/catalogue_detail.htm?csnumber=41022

[HISC] OASIS UBL Human Interface Subcommittee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl-hisc

[JAXB] Java Architecture for XML Binding, http://java.sun.com/developer/technicalArticles/WebServices/jaxb/

[JiBX] Binding XML to Java Code, http://jibx.sourceforge.net/

[NDR] OASIS Public Review Draft, *Universal Business Language (UBL) Naming and Design Rules 2.0,* September 2006. http://docs.oasis-open.org/ubl/prd-UBL-NDR-2.0.pdf

[SBS 1.0] OASIS Committee Specification, *Universal Business Language 1.0 Small Business Subset 1.0,* April 2006. 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/xpaths

[XPath File] http://docs.oasis-open.org/ubl/submissions/XPath-files/

[UXO] http://ubl.xml.org/

1.3 Acknowledging OASIS copyright

UBL is provided under the OASIS Royalty Free on Limited Terms policy, and this should be recognized in any customizations.

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[9]. 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

Copyright © OASIS® 2009. All Rights Reserved.

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 through its own channels.

1.4 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 or certifying whether customizations are conformant, compatible, or otherwise, we supply these definitions as a point of reference for those who might.

1.4.1 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.

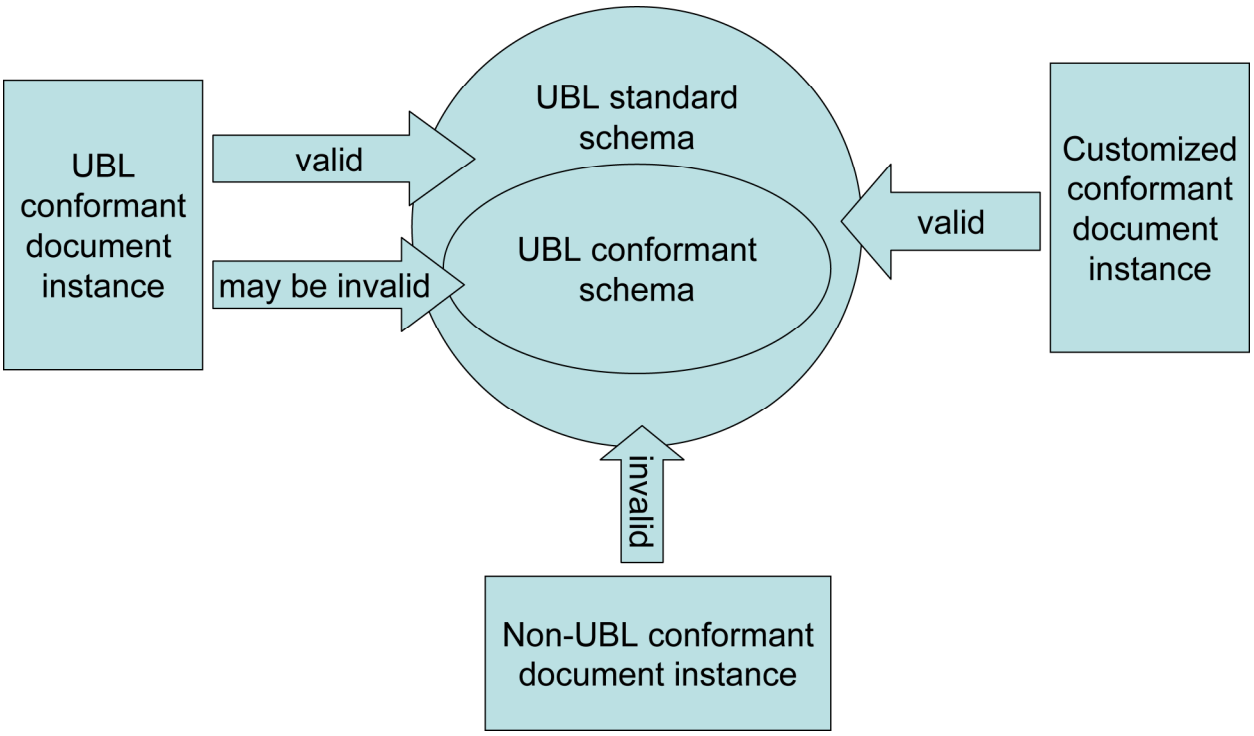


Figure 1. Conformant schemas and document instances

Copyright © OASIS® 2009. All Rights Reserved.

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).

1.4.2 Code list conformance

Conformance (both schema conformance and code list conformance) is important in the context of trading partner agreements. An agreement between two or more trading partners that gives UBL electronic documents the legal force of their paper equivalents should specify both the schemas to be used and the code list values to be accepted, as well a context-dependent assortment of other possible data value constraints beyond the scope of this discussion, such as a list of authorized buyers.[2]

As defined above, UBL conformance is simple: if an instance validates against the published UBL schema, it is UBL conformant, and if it doesn't, it's not.

It should be understood that the values of data items in UBL instances are not included in the concept of UBL conformance. UBL defines the vocabulary, structure, and data typing of conformant UBL instances; it has nothing to say about allowable prices or street addresses or the names of people or companies. UBL is meant to specify the shape and labeling of the data container, not the values that go inside.

UBL conformance is easy to define because many years of industry development invested in the creation of the necessary formalisms (XML, XSD) and software (standard XML/XSD validators) enable us to simply relate UBL conformance to validation against the published UBL schemas. Thus, the mechanism for UBL conformance checking is built right into the definition.

Code list conformance is distinguished from UBL conformance because it is concerned with the values used by trading partners in specific business relationships and cannot be defined in advance by the UBL Technical Committee. However, the TC does provide default versions of the code lists referenced in UBL schemas as a convenience to users.

1.4.2.1 Enforcing code list conformance

Just as with code list conformance itself, it should be understood the UBL 2.0 Standard does not mandate any particular framework for code list checking. The logic needed to check code values appearing in UBL instances can be implemented in many ways.[3] For example, in high-volume transaction processing environments, this checking is more likely to be carried out with custom programming than with the free software tools included in the UBL distribution (see footnote).

---

[2] The Model UBL Letter Agreement developed for U.S. users by the American Bar Association can be found at http://www.oasis-open.org/committees/document.php?document_id=24992. International users are referred to the UNECE Electronic Commerce Agreement at http://www.unece.org/cefact/recommendations/rec31/rec31_ecetrd257e.pdf.

[3] In order to demonstrate the concept, and as a convenience for users, UBL 2.0 publishes its default code lists using the OASIS Genericode 1.0 specification and provides a set of software for performing code list checking using a freely available XSLT processor (see Appendix E of the UBL 2.0 Standard). But the XSLT or other technology file that drives this process (defaultCodeList.xsl) can easily be replaced by a custom user-defined XSLT file to expand or limit the set of acceptable code list values and optionally provide further back-end filtering.

Copyright © OASIS® 2009. All Rights Reserved.

1.4.2.2 UBL conformance of UBL-provided code lists

With the exception of four UN/CEFACT code lists in UBL 2.0 noted below, any UBL-provided code list can be modified to suit the agreed-upon requirements of any trading relationship without breaking UBL conformance, because most UBL code list values are schema-constrained by nothing but their data type.

However, this does not change the basic idea of code list conformance; it simply moves conformance determination to a different part of the constraint checking process.

1.4.2.2.1 Code list conformance ownership

A key reason for moving code list value specification out of the UBL schemas is to distinguish the parts of constraint checking belonging to UBL proper from parts that can be modified without deviating from UBL conformance.

The owners of code list conformance criteria are the organizations (typically standards bodies of some kind) responsible for creating and maintaining code lists, and statements about code list conformance must therefore be related to the relevant owner. Thus, for example, an agreement to exchange only instances validating against UBL standard schemas and using only those country codes appearing in the ISO 3166-1 country code list is an agreement to exchange only instances that are *both* UBL conformant *and* ISO 3166-1 conformant.

This doesn't change the basic idea of code list conformance; an instance that conforms to ISO 3166-1 is an instance that uses only ISO 3166-1 country codes. But this is not UBL conformance, it is ISO 3166-1 conformance.

1.4.2.2.2 Simple code list conformance specification

As a convenience to implementers, the UBL distribution package includes a number of code lists that offer a way to simplify trading partner agreements in the default case. Trading partners working from a template such as the Model UBL Letter Agreement who are willing to default to the code lists provided in the UBL 2.0 distribution (for example) can simply agree to conform to “the code lists provided in the OASIS Standard UBL 2.0 distribution.” If deviations are small, they can use some formula such as “the code lists provided in the OASIS Standard UBL 2.0 distribution, with the exception that country codes shall be restricted to US, CA, and MX.” (Note that these examples are not intended as legal advice, but are given to illustrate the concept.)

It must be remembered, however, that the conformance in question is not to UBL, but rather to the code lists included in the UBL distribution, many of which are defined by other organizations. With the exception of the four UN/CEFACT lists in UBL 2.0 discussed below, modifications to the code lists provided in the UBL distribution will have no effect on UBL conformance, though they may well implicate trading partner agreements governing the exchange of UBL instances.

Conformance to the code lists included in the UBL 2.0 distribution will in effect bind to the following:

UBL 2.0 code lists referenced only in the file defaultCodeList.xsl

Defined by external agencies

UNECE Rec 19 Transport Mode Codes
UNECE Rec 20 Unit of Measure Codes
UNECE Rec 21 Packaging Type Codes
UNECE Rec 24 Transportation Status Codes
UNECE 3155 Communication Address Code Qualifiers
UNECE 4461 Payment Means
UNECE 4465 Adjustment Reason Descriptions

Copyright © OASIS® 2009. All Rights Reserved.

UNECE 8053 Equipment Type Code Qualifiers
ISO 3166-1 Country Codes
ISO 4217 Alpha Currency Codes

Defined by UBL

UBL Chip Codes
UBL Document Status Codes
UBL Latitude Direction Codes
UBL Line Status Codes
UBL Longitude Direction Codes
UBL Operator Codes
UBL Substitution Codes

UBL 2.0 code lists imported via the UN/CEFACT UDT schema module

ISO Currency Codes (as UN/CEFACT 54217:2001)
ISO Language Codes (as UN/CEFACT 5639:1988) (not currently used)
IANA MIME Media Type Codes (as UN/CEFACT IANAMIMEMediaType:2003)
UNECE Unit Codes (as UN/CEFACT 66411:2001)

1.4.2.2.3 Complex code list conformance specification

The set of acceptable code values in a code list can sometimes vary depending on where in an instance document they are found. Such code list differentiation can be an easy way to add some very useful business logic. For example, two partners might agree that the list of acceptable country codes for Customer Party addresses is different from the list of acceptable country codes for Supplier Party addresses.

Acceptable code values can also depend conditionally on other instance data values; for example, it might be desired to assert the condition “If the country code is FR, then the currency code must be EUR.”

To meet this need, the OASIS Code List Representation Technical Committee has developed both a standard XML encoding for code lists (genericode) and a powerful and sophisticated methodology for formally representing agreements about the association of code lists and code list subsets with specific portions of XML instances (Context/value association, or CVA).[4]

1.4.2.3 UN/CEFACT schema modules in UBL 2.0

The inherently simple distinction between schema validation and the checking of data values in UBL instances is sometimes obscured by the fact that XSD does provide a mechanism for enumerating lists of the allowable values of data items. With one exception, UBL does not use this mechanism.

The single exception is the enumeration in UBL 2.0 of allowable values for four standard code lists recommended by UN/CEFACT:

- ISO Currency Codes,
- ISO Language Codes,
- IANA MIME Media Type Codes, and
- UNECE Unit Codes.

[4] See the TC web site at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=codelist for details.

Copyright © OASIS® 2009. All Rights Reserved.

These code list values are specified as xsd:enumerations in four schema modules defined by UN/CEFACT and are imported by all UBL 2.0 schemas via the UN/CEFACT Unqualified Data Type schema module (see Section 5.2.4 of the UBL 2.0 Standard and the xsd/common directory of the 2.0 distribution). Because of this exception, a UBL 2.0 instance that contains a value for one of these codes that does not appear in the enumerations imported by the UN/CEFACT UDT module is, technically speaking, not UBL conformant, because it will fail UBL schema validation.

The exception for these four code lists resulted from a policy decision to adopt the UN/CEFACT list schemas and the UN/CEFACT UDT schema module as published by UN/CEFACT. From the architectural standpoint, this approach is contrary to the approach taken for constraint checking of all other UBL instance values. For this reason, the UBL TC has already determined that this exception for UN/CEFACT recommendations will not persist in UBL 2.1. In versions of UBL 2 later than 2.0, all code lists, including the four published by UN/CEFACT, will be provided in the form that is already used in 2.0 for other code lists.

1.4.3 UBL compatibility

To be UBL compatible means to be consistent with UBL information entities and 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 their design 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.

Copyright © OASIS® 2009. All Rights Reserved.

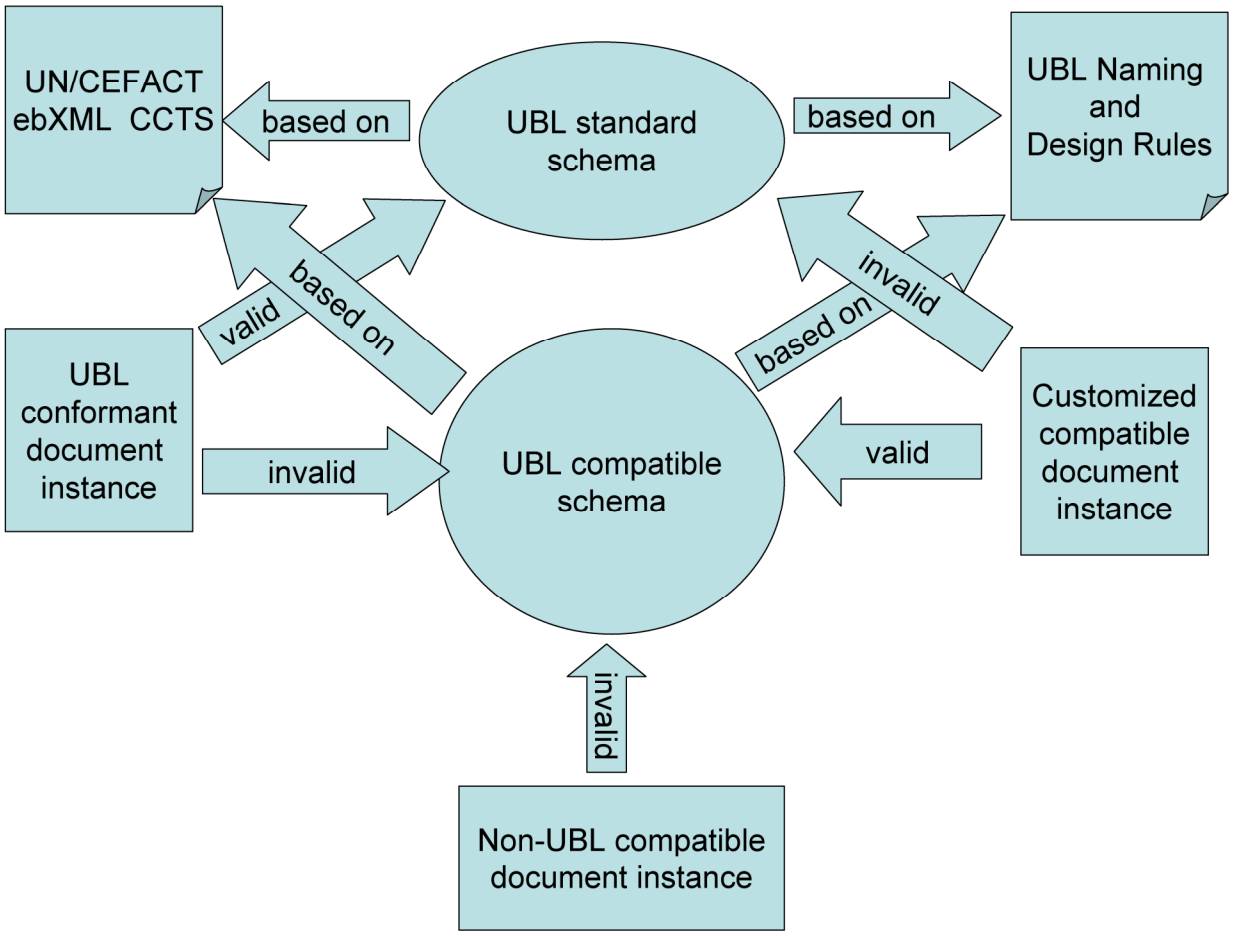


Figure 2. Compatible schemas and document instances

1.4.4 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 information entities.

For example, 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 should bind those users to the meanings specified in the published definitions.

1.4.5 Customization profiles

Customizations of UBL may apply to a set of business processes within a given context of use. Within each specific business process, a profile characterizes the choreography of the interchanges.

Defining different profiles means that a given document type may have different sets of constraints in each profile within the same customization family. For example, an Invoice instance used for a profile that involves only an Order and Invoice being exchanged may not require as many information entities as an Invoice instance used in a profile for a complete supply chain.

Copyright © OASIS® 2009. All Rights Reserved.

Thus the three dimensions of the set of UBL document structural constraints are defined by the UBL version (standard), the implementation context (customization), and the business process (profile). For example, Stand Alone Invoicing is a *profile* of the Northern European Subset *customization* of the UBL 2.0 *standard*.

1.4.6 Identifying versions, customizations, and profiles

A document instance claiming to satisfy the constraints for a particular profile in a customization asserts this using the following information entities at the root of each document:

- UBLVersionID

  An identifier reserved for UBL version identification. Not actually a modifiable value, but required 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.

For example, Stand Alone Invoicing, which is a profile of the Northern European Subset customization of the UBL 2.0 standard, is identified as:

- UBLVersionID: 2.0
- UBLCustomizationID: NES
- UBLProfileID: urn:www:nesubl:eu:profiles:profile1:ver1.0

1.5 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 models are then expressed in W3C XML Schema (XSD), based on the UBL Naming and Design Rules. It is these schemas that are used to both specify and validate UBL conformance. The steps involved in UBL development are shown in Figure 3 below.

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).

Copyright © OASIS® 2009. All Rights Reserved.

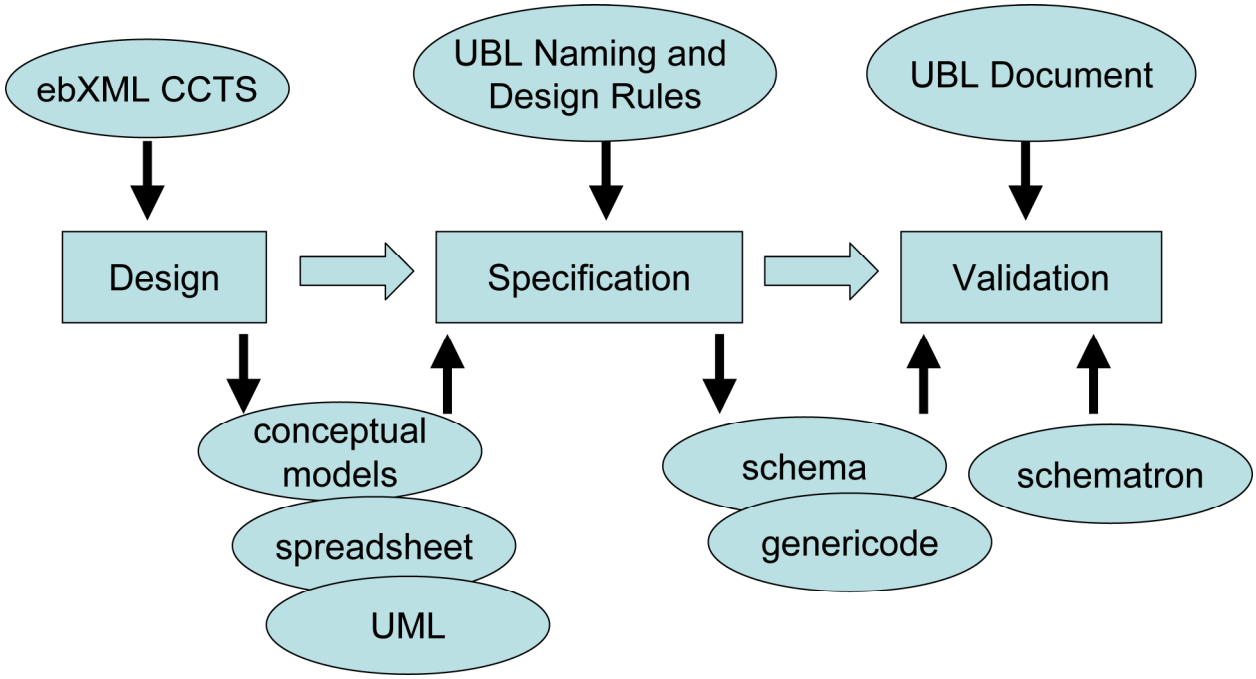


Figure 3. Overview of UBL development methodology

1.6 Calculation models

The UBL Technical Committee does not prescribe a calculation model that governs how values in instances are calculated (for example, the inclusion of allowances in a line extension amount). Any actual implementation of UBL should document its calculation model via a prose description or a formal description using, for example, the methodology being developed by the OASIS Test Assertion Guidelines Technical Committee.

Copyright © OASIS® 2009. All Rights Reserved.

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 optional 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 1.3.1), 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:[5]

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

2.1.1 Subsets of the document model

The standard UBL document types have been designed to accommodate a broad range of contexts. As a result, 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 information entities, that document would contain approximately 800,000 elements and attributes. 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 the specific business requirements of an implementation.

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

Copyright © OASIS® 2009. All Rights Reserved.

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..m, 1..n, m..n, or 0..0 (where m<n)

1..n can become 1..1, m..n, or 1..m (where m<n)

1..1 cannot be changed

2.1.2 Code list constraints on document content

Constraining the values for an information entity to a fixed set (such as with a code list) is a common customization requirement. For example, “the Currency Code must be expressed using ISO 4217 codes” is a constraint on the possible values for Currency Code in a document instance.

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. These are expressed as Unqualified Code Data types.

- Code lists with defined values

  These are explicit lists that constrain possible values for the content. These are expressed as Qualified Code Data types. They are specializations of the Unqualified Code Data type.

2.1.3 Other constraints on document content

There are other cases in which the treatment of UBL instances may require 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.”

Co-occurrence constraints apply when the values of one or more information entities are affected by the values of one or more other information entities in the document content. The basis can be the presence or absence of content, or particular values of content. For example,

“For each Party, one or both of Party ID and Party Name must be present, but not neither.”

“The Shipping Address must be the same as the Billing Address.”

“The Start Date must be earlier than the End Date.”

A value calculation is another form of constraint. For example:

“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.”

Methods for specifying and validating such business rule constraints are discussed in section 3.4.

Copyright © OASIS® 2009. All Rights Reserved.

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,[6] as shown in Figure 4.

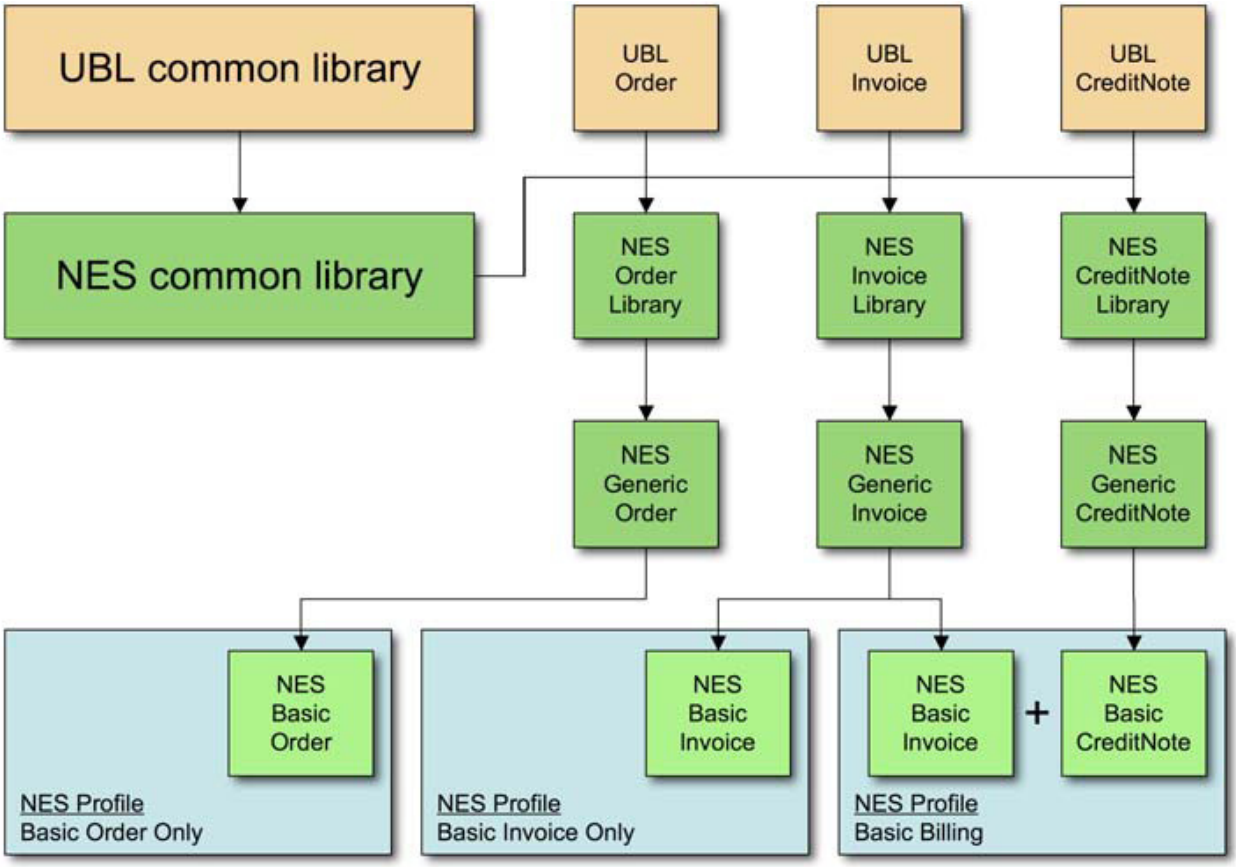


Figure 4. NES subset architecture

In the NES customization, the Delivery aggregate is an example of an information entity that is used across several documents and processes. The UBL standard Delivery is defined in the UBL Common Library (see Figure 5).[7]

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

[7] Note that information entities are referenced at the modeling level by their CCTS Dictionary Entry Names, not by their XML element names.

Copyright © OASIS® 2009. All Rights Reserved.

- Delivery. Details
  - Delivery. Identifier
  - Delivery. Quantity
  - Delivery. Minimum_ Quantity. Quantity
  - Delivery. Maximum_ Quantity. Quantity
  - Delivery. Actual_ Delivery Date. Date
  - Delivery. Actual_ Delivery Time. Time
  - Delivery. Latest_ Delivery Date. Date
  - Delivery. Latest_ Delivery Time. Time
  - Delivery. Tracking Identifier. Identifier
  - Delivery. Delivery_ Address. Address
  - Delivery. Delivery_ Location. Location
  - Delivery. Requested Delivery_ Period. Period
  - Delivery. Promised Delivery_ Period. Period
  - Delivery. Estimated Delivery_ Period. Period
  - Delivery. Delivery_ Party. Party
  - Delivery. Despatch

Figure 5. UBL standard Delivery aggregate

This includes several information entities that are not required in the NES processes and documents, so the standard Delivery is restricted at the NES Common Library level as shown in Figure 6.

- Delivery. Details
  - Delivery. Identifier
  - Delivery. Quantity
  - ~~Delivery. Minimum_ Quantity. Quantity~~
  - ~~Delivery. Maximum_ Quantity. Quantity~~
  - Delivery. Actual_ Delivery Date. Date
  - Delivery. Actual_ Delivery Time. Time
  - Delivery. Latest_ Delivery Date. Date
  - Delivery. Latest_ Delivery Time. Time
  - ~~Delivery. Tracking Identifier. Identifier~~
  - ~~Delivery. Delivery_ Address. Address~~
  - Delivery. Delivery_ Location. Location
  - Delivery. Requested Delivery_ Period. Period
  - Delivery. Promised Delivery_ Period. Period
  - Delivery. Estimated Delivery_ Period. Period
  - Delivery. Delivery_ Party. Party
  - ~~Delivery. Despatch~~

Figure 6. NES common Delivery customization

However, even at the NES Common Library level, the Delivery subset still contains information 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_Delivery Date information entities in an Invoice document. Therefore, NES requires one more level of subset

Copyright © OASIS® 2009. All Rights Reserved.

customization where only information entities relevant to specific document types are present. The NES Invoice Library further restricts Delivery as shown in Figure 7.

- Delivery. Details
  - Delivery. Identifier
  - Delivery. Quantity
  - ~~Delivery. Minimum_ Quantity. Quantity~~
  - ~~Delivery. Maximum_ Quantity. Quantity~~
  - Delivery. Actual_ Delivery Date. Date
  - Delivery. Actual_ Delivery Time. Time
  - ~~Delivery. Latest_ Delivery Date. Date~~
  - ~~Delivery. Latest_ Delivery Time. Time~~
  - ~~Delivery. Tracking Identifier. Identifier~~
  - ~~Delivery. Delivery_ Address. Address~~
  - Delivery. Delivery_ Location. Location
  - ~~Delivery. Requested Delivery_ Period. Period~~
  - Delivery. Promised Delivery_ Period. Period
  - Delivery. Estimated Delivery_ Period. Period
  - Delivery. Delivery_ Party. Party
  - ~~Delivery. Despatch~~

Figure 7. NES Invoice Delivery customization

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

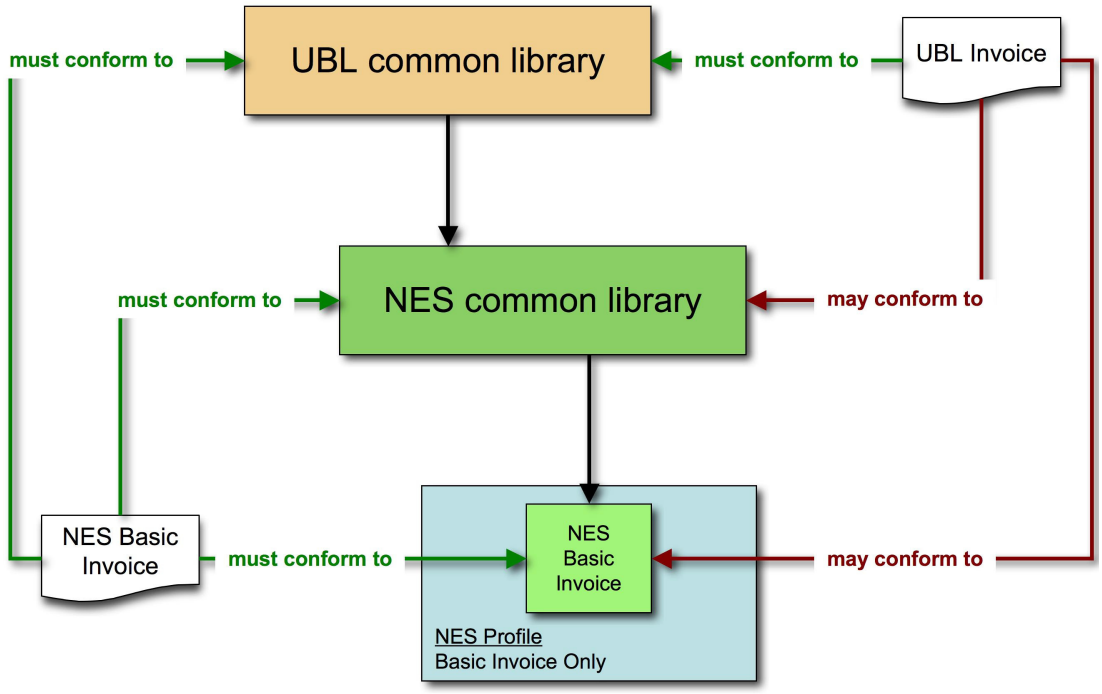


Figure 8. Example of conformance with a UBL subset

Copyright © OASIS® 2009. All Rights Reserved.

2.2 Designing for compatibility

When designing for compatibility, 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 application of the UN/CEFACT ebXML Core Component Technical Specification (CCTS).

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 information 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 Party aggregate than to create a competing information 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 information entities is not feasible, customizers may need to add to the UBL model additional information entities to satisfy business requirements. In these situations it is possible to extend the UBL library in a compatible manner.

Extension means to add to, or associate with, existing entities additional information that may be required for a particular context of use. That is, an extension creates a superset of the original information entity. It is recommended that such an extension include the original information entity as an association from the information entity that extends it. For example, UBL Customer Party is an extension of Party because it contains additional information required if the party is a Customer. Structurally, Customer Party has an association to Party, making Customer Party a superset of Party.

Compatible extensions can be implemented in parts of a schema outside the extension area provided in the Standard version. This allows validation checks to be built into the compatible schema that cannot be enforced in the extension area of a conformant schema.

2.2.2.1 Using qualified names

UBL supports the CCTS principle of qualifying the Property Term of an information entity's Dictionary Entry Name to indicate specialized re-use of an information entity.

The use of qualified Dictionary Entry Names is not apparent in the UBL name (XML element name) because the underscore character is omitted. However, it does affect the XML type used, because only the unqualified name is used to identify the XML type for the definition.

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. The XML element name would be CanadianCountrySubentityCode. However, the XML type would be defined as CountrySubentityCodeType.

Copyright © OASIS® 2009. All Rights Reserved.

2.2.2.2 Re-using aggregate information entities

The principle applied is that if a required aggregate information entity has the same structure as a standard UBL information entity, then it should not be a redefinition but a re-use by association. The qualifying terms used to name the new association information entity then 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 defined as *Party. Local_ Address*.

If the new aggregate information entity does not have the same structure as a standard UBL information entity, then the required information entity has a new name, not a qualified name. If possible, the new aggregate may then be associated with the UBL information entity being extended.

Example

If an *Address* has additional information entities when the address is in Japan, then a new aggregate information 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 information entities.

2.2.2.3 New basic information entities

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

When establishing a new basic information entity, it is necessary to associate it with a data type. This is determined by the Representation Term part of the information entity's Dictionary Entry Name.

Example

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

Changing or specializing an information entity's definition changes the information entity (see 1.3.4). Therefore, a new basic information entity must be defined.

Example

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

In UBL, Representation Terms are implemented either as standard CCTS data types (known as unqualified data types) or as UBL defined data types (qualified data types).

2.2.2.3.1 Qualified data types

In cases where the required information 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.

Copyright © OASIS® 2009. All Rights Reserved.

2.2.2.3.2 Qualified code data types

A basic information 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). There are no constraints on the values used for instances of this information entity.

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

- With defined values (the code data type is qualified by UBL). All values for instances of this information entity must exist in the given code list.

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

Code list customization can be applied in many situations:

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 new *CarbonEmissionRating* information entity may use a formal coding system.

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.

Assigning a qualified code list to a basic information 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 information entity that was previously unqualified is a *conformant* restriction. Whereas assigning a new qualified code type to a basic information 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.

Example of customized code data type

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* could further qualify the UBL qualified data type and further restrict the *Currency_ Code Data Type* to specific European currency code values.

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

Copyright © OASIS® 2009. All Rights Reserved.

2.2.2.4 New associations

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

By re-using the unqualified aggregate (*Address*), the same XML type (AddressType) will be used for implementation of all these information entities.

2.2.2.5 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. When creating new aggregates, there are some general principles to follow:

1. 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 information 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 aggregate information entity

2. Aggregations should comprise collections of information entities that share functional dependency. That is, the only information entities that belong in an aggregation are basic information entities or associations to other aggregates whose values may change when the aggregate itself changes.

Examples

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 information entity *Description* would be aggregated into the aggregate *Item*.

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

Copyright © OASIS® 2009. All Rights Reserved.

information entity *Price* would be aggregated into an aggregate perhaps called *Coffee Location*.

3. 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.

2.2.2.6 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 assembling new document type structures 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. For all required associations from these information entities assemble the required information entities (and/or customized extensions), applying cardinality constraints.
4. Continue step 3 recursively through all required associations.

As an example, Figure 10 demonstrates the structure of a new document type known as Notification (actually 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.

Copyright © OASIS® 2009. All Rights Reserved.

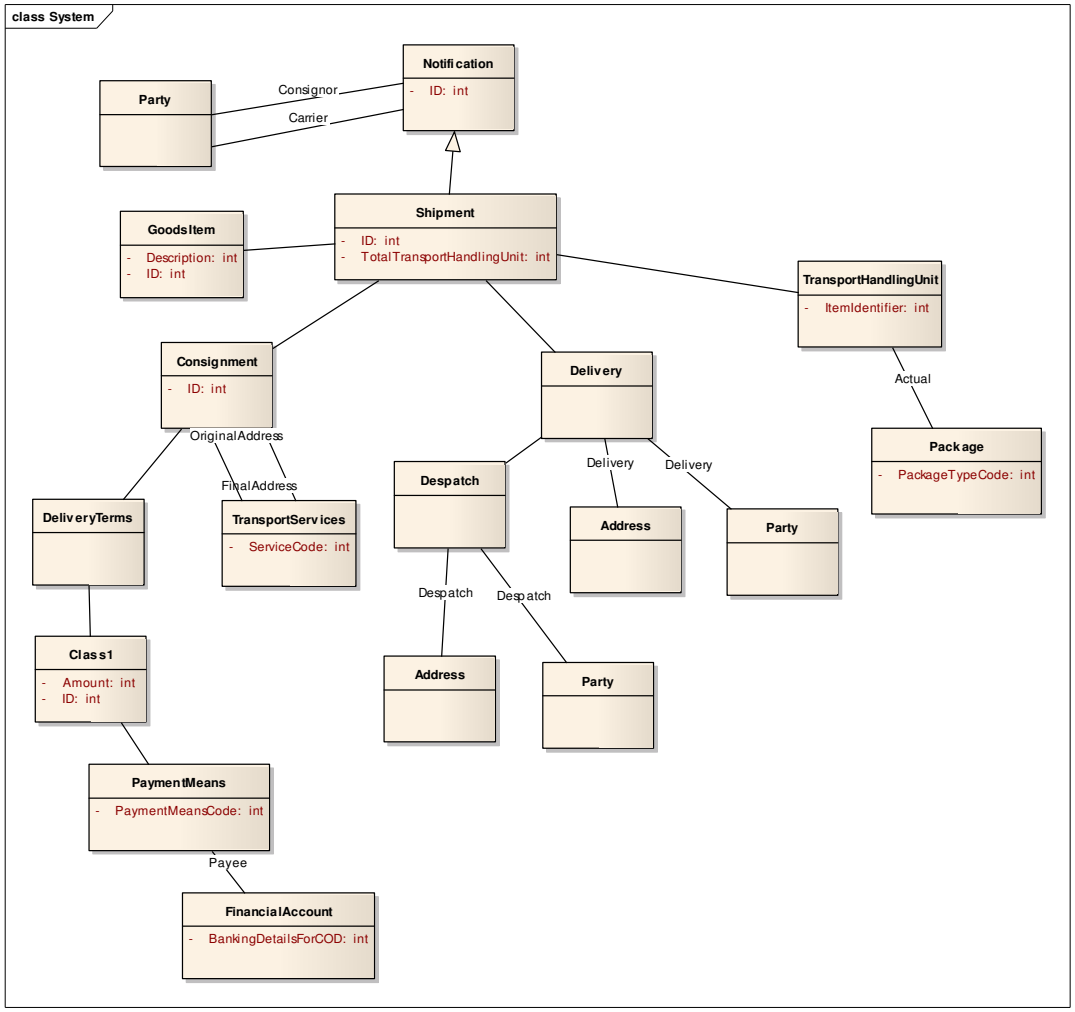


Figure 10. An example design for a compatible document type

2.2.3 The customization ripple effect

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

Example

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

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.

Copyright © OASIS® 2009. All Rights Reserved.

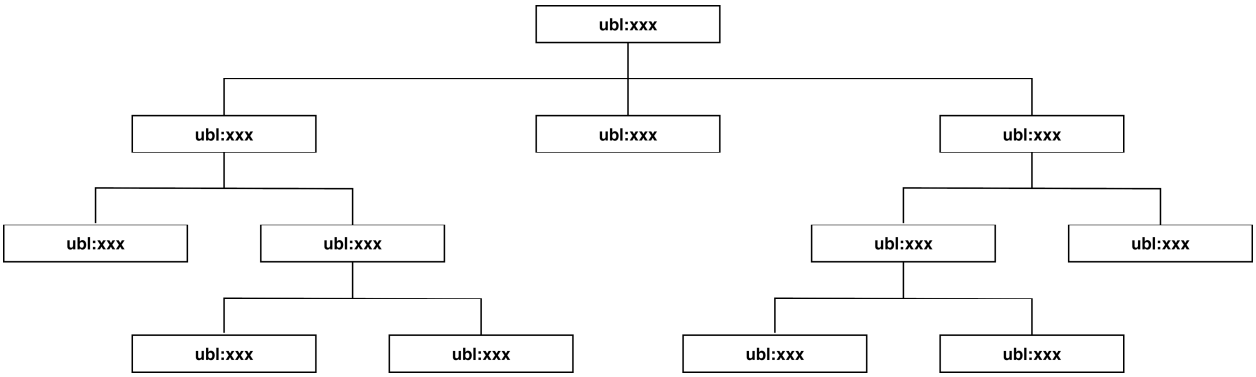


Figure 11. Model of a UBL document type

2.2.3.1 Customized aggregates using subsetting

620 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.

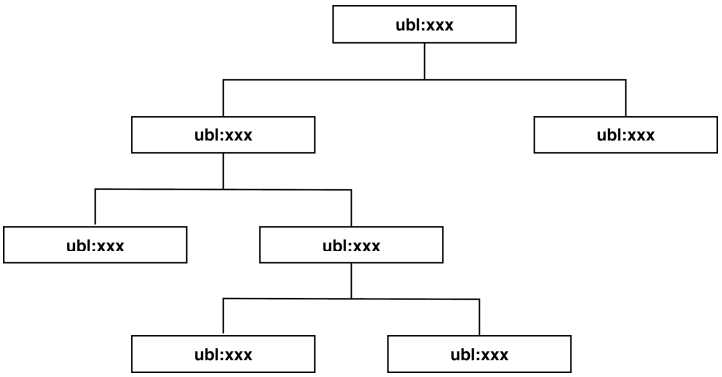
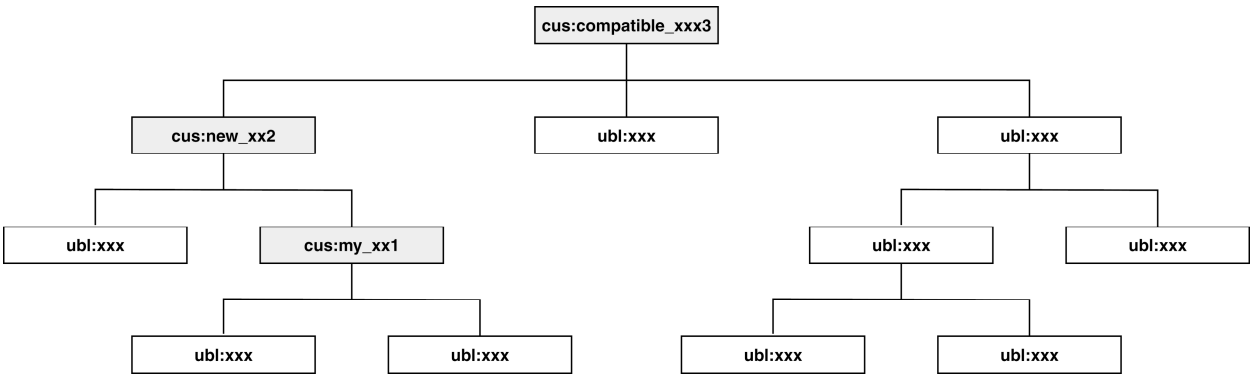


Figure 12. Conformant subsetting (no changes in namespace)

2.2.3.2 Custom aggregates using UBL information entities

625 When a new aggregate is added to a customized document type, all of its parents must also be modified to reflect the new information entity. In the example shown in Figure 13 below, a custom aggregate (“myxx1”) is created using standard UBL information entities. Its parent (“newxx2”) must then be customized to allow this custom aggregate (“myxx1”) in its content model. Accordingly, the document root (“compatiblexx3”) must also be a customization.



630

Figure 13. Ripple effect — customized aggregate

Copyright © OASIS® 2009. All Rights Reserved.

2.2.3.3 Custom aggregate using custom information 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 information entity (“xx1”) in its content model. Accordingly, the document level aggregate (“compatible_xx4”) must also be a customization.

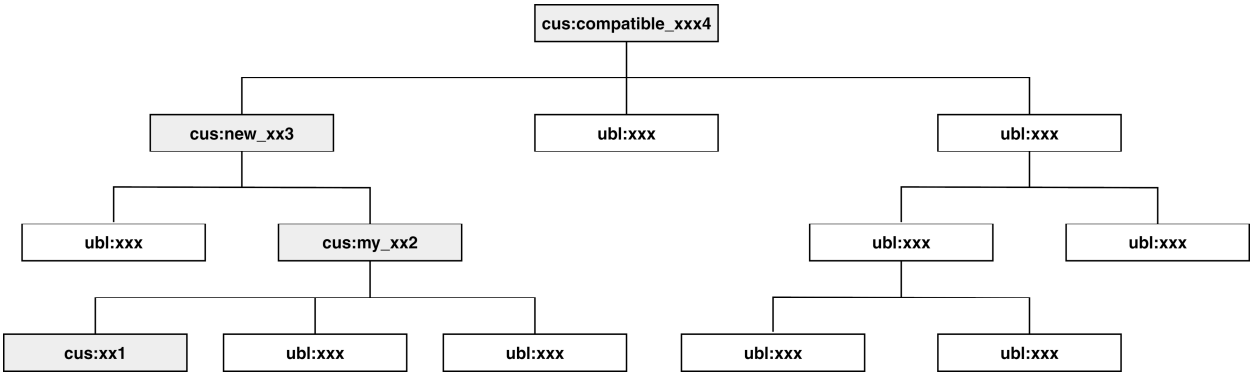


Figure 14. Ripple effect — customized basic information entity

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.*

Copyright © OASIS® 2009. All Rights Reserved.

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. These specifications form the basis of the customizations and profiles of UBL used in specific 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 (XSD)

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.[8] The UBL Naming and Design Rules should be used 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.

At least 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

[8] 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.

Copyright © OASIS® 2009. All Rights Reserved.

element with an xsd:any constraint of ##any to allow any element of any namespace to be a child of the element:

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 directly 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. Instead, the restricted model must re-list the entire collection of elements with the exception of the one that is to be removed.

- 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 information entities are modeled with all basic information entities listed first as children, followed by all associate information entities listed next as children. XSD extension allows additional constructs to be added only after all of the base constructs. Should a customization to a UBL aggregate information entity need a new child basic information entity, this basic information entity cannot be placed before child associate information 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 imperative or declarative assertion languages such as Schematron. Since UBL uses XSD for its standard schemas, however, it is assumed in the following that new schemas

Copyright © OASIS® 2009. All Rights Reserved.

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 [UXO].

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[9] (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

Figure 16 shows a way to organize schema modifications in creating a subset. The customized schema fragments on the left are overlaid onto a copy of the xsd/ or xsdrt/ subdirectory from the UBL distribution package, replacing the corresponding document schema, aggregates schema, and basics schema. This creates a customization suite of schema fragments representing instances with only those constructs allowed by the customization and not simply all elements allowed by UBL. Only those original schema fragments that correspond to the changed fragments are replaced, thus preserving all of the schema fragment linkages for those fragments that remain unchanged.

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

Copyright © OASIS® 2009. All Rights Reserved.

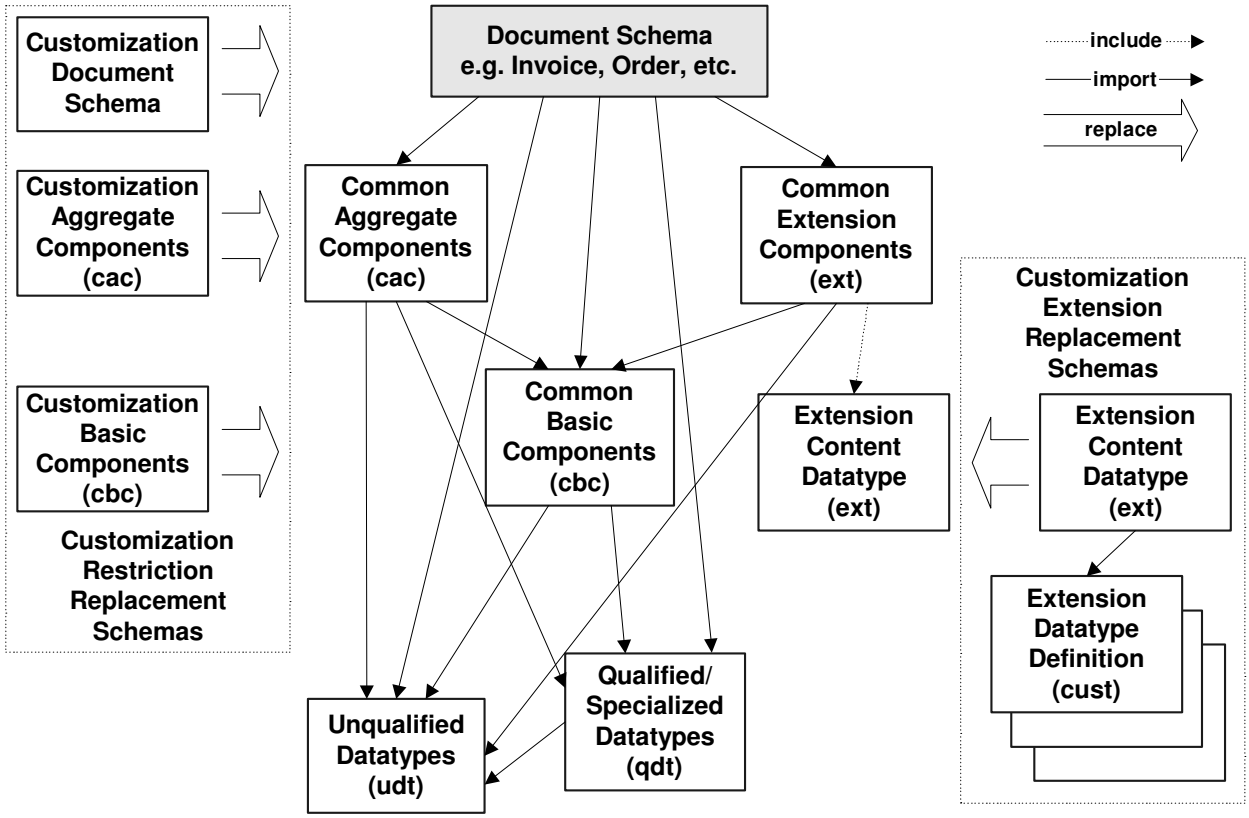


Figure 16. Overlaying customization schema fragments (Crane Softwrights Ltd. Used by permission.)

3.1.4 Using UBLExtension

The one exception to the general rule that only subsets are conformant is the UBLExtension element. If new information 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 information entities.

The UBLExtension element found at the beginning of all UBL documents allows communities of interest to specify additional information entities as part of a UBL standard document. Conformance is not affected by the content of the UBLExtension, as it may contain any type of information entity (because it uses <xsd:any> in its declaration). If new information entities are added to a UBL document type only in the extension area, any instances validating against the extended schema are still UBL conformant (but may not be UBL compatible).

The Extension Content Datatype module is shown in Figure 16, as that fragment that is replaced with the customization's specification of the UBL extension point. This eliminates the need to touch the module expressing the standardized extension metadata in Common Extension Components.

The UBLExtension element is not one of UBL's information entities. 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 information 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.

Copyright © OASIS® 2009. All Rights Reserved.

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.

   Example

   In Figure 17, UBLExtensions is used to specify legacy EDIFACT information entities (defined here as myext:ExtensionContent) that must be included in the document instance for message routing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2"
       xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2"
       xmlns:ext="urn:oasis:names:specification:ubl:schema:xsd:CommonExtensionComponents-2"
       xmlns:mycac="urn:x-mycompany:aggregates"
       xmlns:mycbc="urn:x-mycompany:basics"
       xmlns:myext="urn:x-mycompany:extension"
       xmlns="urn:oasis:names:specification:ubl:schema:xsd:Order-2">
  <ext:UBLExtensions>
    <ext:UBLExtension>
      <cbc:ID>Addr1</cbc:ID>
      <cbc:Name>Alternative address</cbc:Name>
      <ext:ExtensionAgencyID>MC</ext:ExtensionAgencyID>
      <ext:ExtensionAgencyName>My company</ext:ExtensionAgencyName>
      <ext:ExtensionAgencyURI>urn:my-company</ext:ExtensionAgencyURI>
      <ext:ExtensionReasonCode>Optional</ext:ExtensionReasonCode>
      <ext:ExtensionReason>To provide a legacy alternative address</ext:ExtensionReason>
      <ext:ExtensionContent>
        <myext:ExtensionContent>
          ... legacy invoice stuff ...
        </myext:ExtensionContent>
      </ext:ExtensionContent>
    </ext:UBLExtension>
  </ext:UBLExtensions>
  ... remaining standard instance content ...
```

Figure 17. 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.

   Example

   In Figure 18, the UBL *Address* has been extended to include a *Postoffice* information entity. This new structure is known as *AlternativePostalExtendedAddress*.

Copyright © OASIS® 2009. All Rights Reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2"
       xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2"
       xmlns:ext="urn:oasis:names:specification:ubl:schema:xsd:CommonExtensionComponents-2"
       xmlns:mycac="urn:x-mycompany:aggregates"
       xmlns:mycbc="urn:x-mycompany:basics"
       xmlns:myext="urn:x-mycompany:extension"
       xmlns="urn:oasis:names:specification:ubl:schema:xsd:Order-2">
  <ext:UBLExtensions>
    <ext:UBLExtension>
      ... extension meta data ...
      <ext:ExtensionContent>
        <myext:ExtensionContent>
          <mycac:AlternativePostalExtendedAddress>
            <mycbc:PostOffice>South Bridgtow</mycbc:PostOffice>
            <mycac:ShortAddress>
              <cbc:Postbox>2234</cbc:Postbox>
              <cbc:PostalZone>ZZ99 0AA</cbc:PostalZone>
              <cac:Country>
                <cbc:IdentificationCode>GB</cbc:IdentificationCode>
              </cac:Country>
            </mycac:ShortAddress>
          </mycac:AlternativePostalExtendedAddress>
        </myext:ExtensionContent>
      </ext:ExtensionContent>
    </ext:UBLExtension>
  </ext:UBLExtensions>
 ... remaining standard instance content ...
```

Figure 18. An example of extending UBL information entities

Complex extensions are best organized in modules that correspond to components of the standard UBL document structure. Figure 19 shows a set of extensions implemented as a set of four fragments: the specification of a redefined UBL extension point (in the UBL extension namespace); the specification of the apex element of the extension (in another namespace); the specification of ABIE constructs (in yet another namespace); and the specification of extension BBIE constructs (in still another namespace). The suggested apex fragment is analogous to the document schema; it has no corollary in standard UBL and could easily be abandoned if the extension business objects migrated to a later version of the UBL common library.

Copyright © OASIS® 2009. All Rights Reserved.

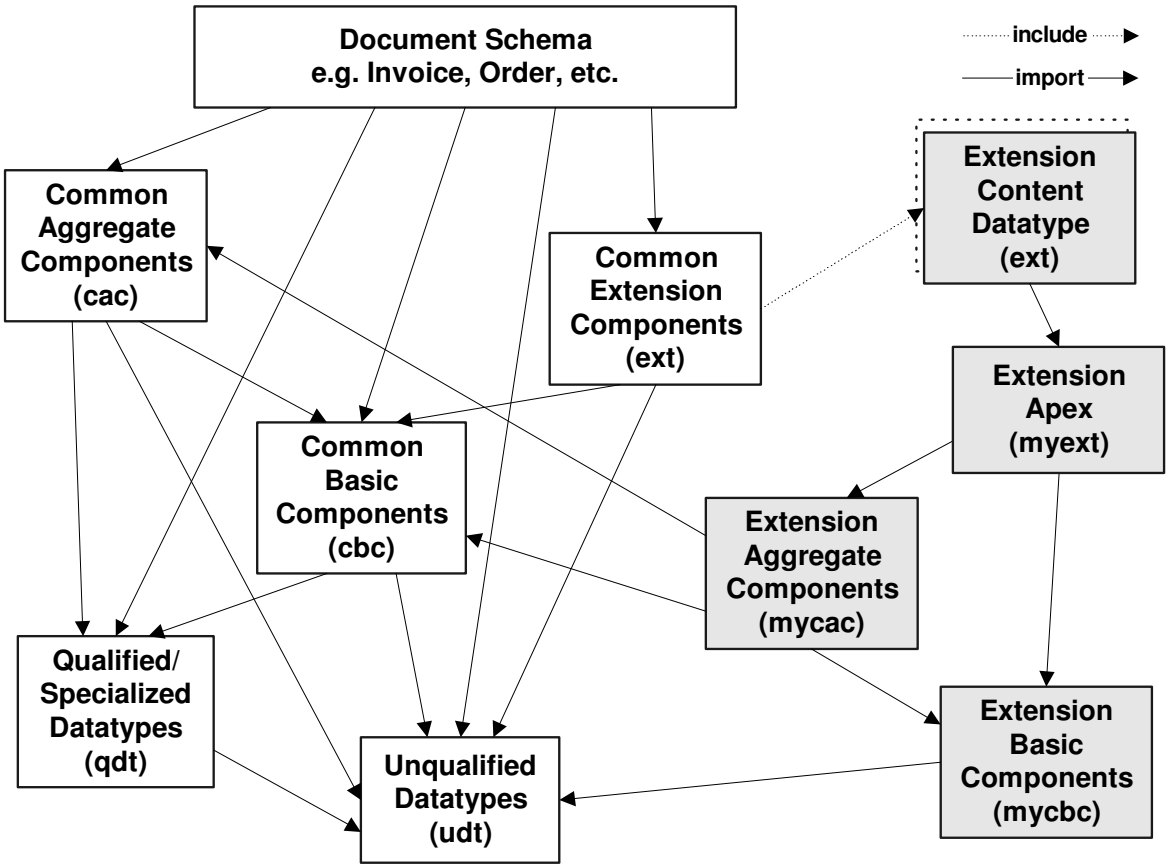


Figure 19. Extension of non-UBL business objects (Crane Softwrights Ltd. Used by permission.)

3.1.4.1 Referencing information in UBLExtension

There are some complexities when using UBLExtension to specify optional extensions to aggregates that may have many occurrences. For example, suppose we require extension to the UBL aggregate, *Item*, to allow a *CarbonEmissionRating,* and not all *Items* have a rating.

The problem arises when instances contain items of a certain type that may or may not be extended by information in the extension area. That is, when the extended information entity has a minimum cardinality of zero and the aggregate being extended has a maximum cardinality of many.

Using the previous example, 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) of each element in the UBLExtension. There are at least two approaches to solving this.

1. Use a reference identifier.

   Many constructs in UBL, for example Line Items and Parties, may use 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 20 the UBL LineItem/ID is used to establish which line item the LineItem/custInfo applies to.

Copyright © OASIS® 2009. All Rights Reserved.

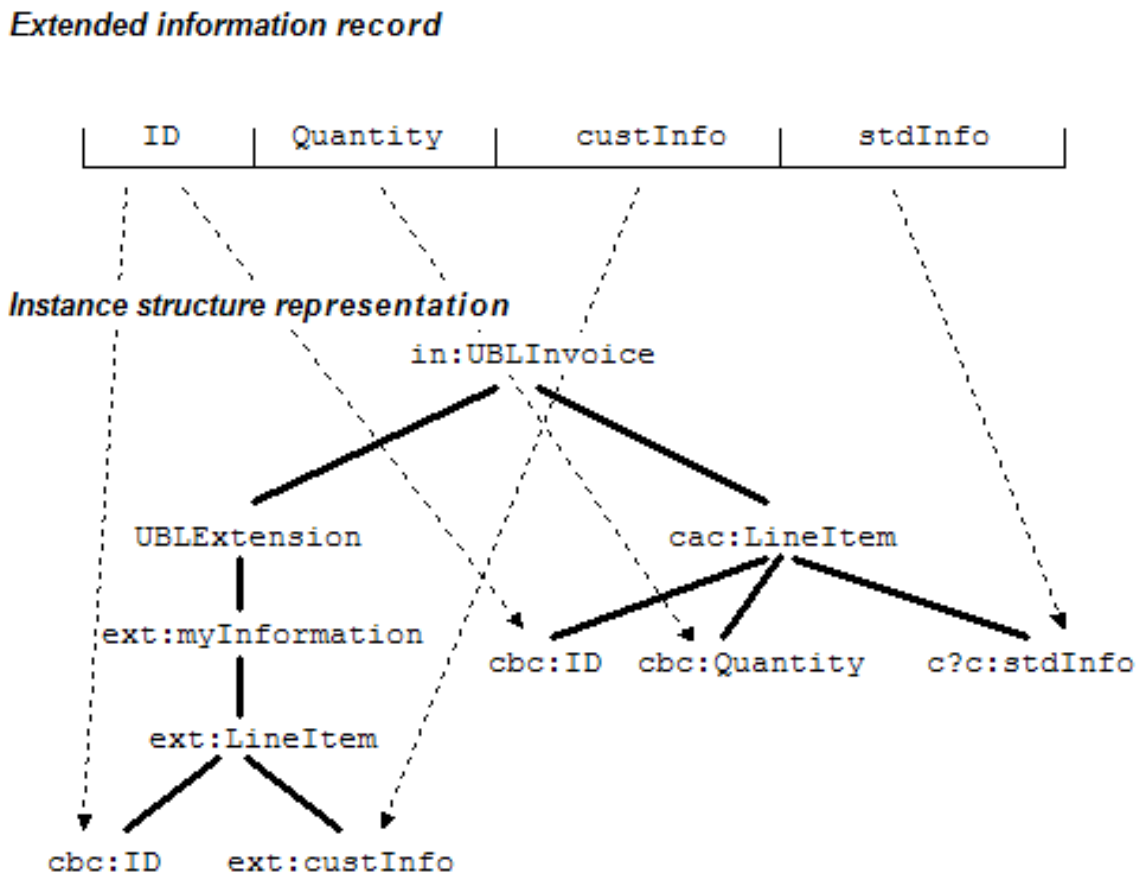


Figure 20. Using a shared ID to connect information in UBLExtension with a line item

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

2. Replicate the entire aggregate in the UBLExtension.

   Using this approach, the 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 21, the entire *LineItem* (with the additional information 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 information entity. Taking this to its extreme may mean specifying the entire body of the extended document in the UBLExtension. This means that each document instance then contains two sets of content — one (in the body) without any extensions and the other (in the UBLExtension) as the required document including extensions. As a result, the body of the document then contains the UBL conformant information (for validation) and the UBLExtension contains the actual document content required for the business process.

Copyright © OASIS® 2009. All Rights Reserved.

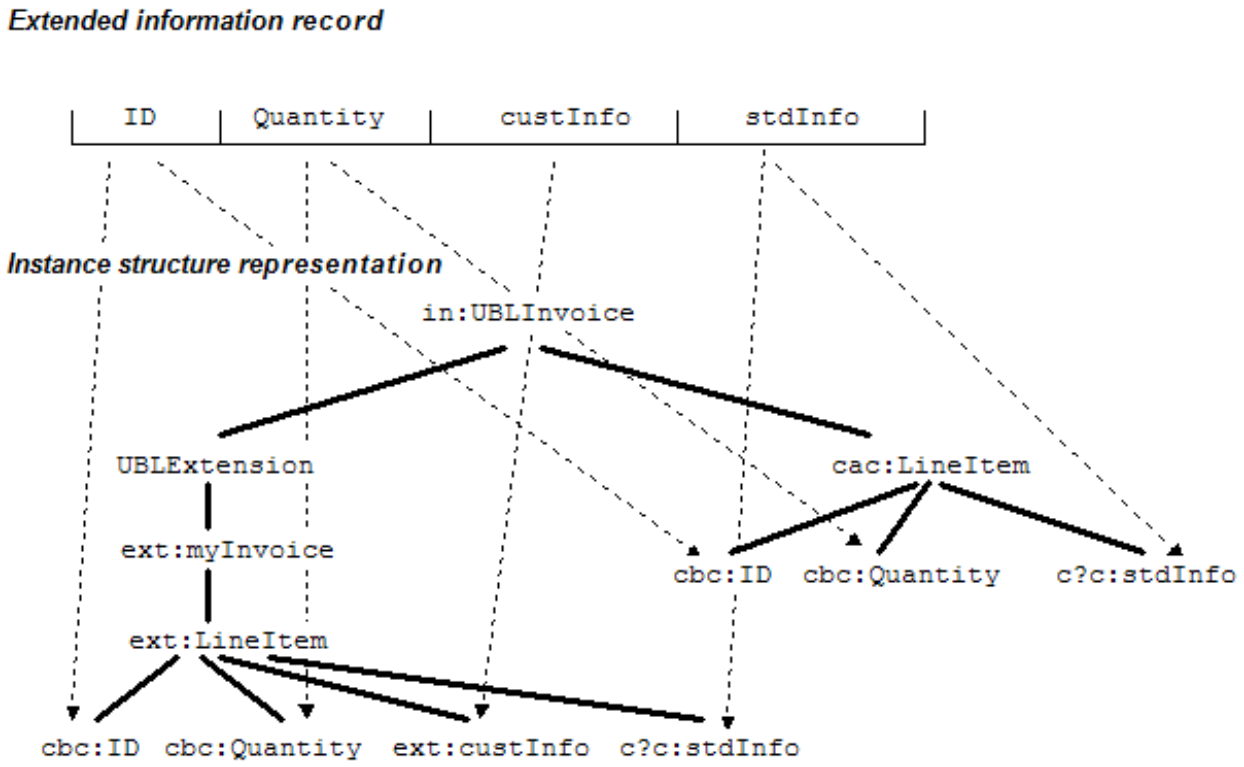


Figure 21. Replication within UBLExtension

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 another part.

The UBL Human Interface Subcommittee [HISC] project has created an XML vocabulary for enumerating information entities in a set of available XPath addresses from the document element to all information entities allowed by a given document model described by a schema or to all information 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,[10] 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

[10] 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.

Copyright © OASIS® 2009. All Rights Reserved.

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 OASIS standard genericode XML format to specify values (and associated metadata) for code lists. These 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 contains 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 CCTS unqualified data type.

The default directory under cl/gc has the standard code lists associated with UBL basic information entities whose types are qualified from the CCTS CodeType.[11]

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

The genericode standard is recommended as the syntax for specifying customized sets of possible values as well.

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.

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).

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

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 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 have uses other than instance validation, such as in constraining data entry.

3.4 Using Schematron

There are many business rules a customization may require that constrain the values used in the documents. Some of these constraints cannot be specified easily using schema validation semantics. A useful syntax for the formal assertion of these type of value constraints is Schematron (ISO/IEC 19757-3).

[11] 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/

Copyright © OASIS® 2009. All Rights Reserved.

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

3.5 Using the UBL library for non-UBL document types

Even when a completely new document type must be defined, it can prove advantageous to use as much of the UBL library as possible. Figure 22 shows an approach to specifying the schema fragments defining a non-UBL document using both UBL and non-UBL business objects. This is similar to the structure suggested in Figure 19 for extending documents from the UBL schema set. Note the aggregate and component extension fragments corresponding to the like UBL fragments. The suggested apex fragment has no corollary in UBL and could be abandoned if the extension business objects migrate to a future version of the UBL common library with the document element receiving an "official" UBL namespace as a new UBL document.

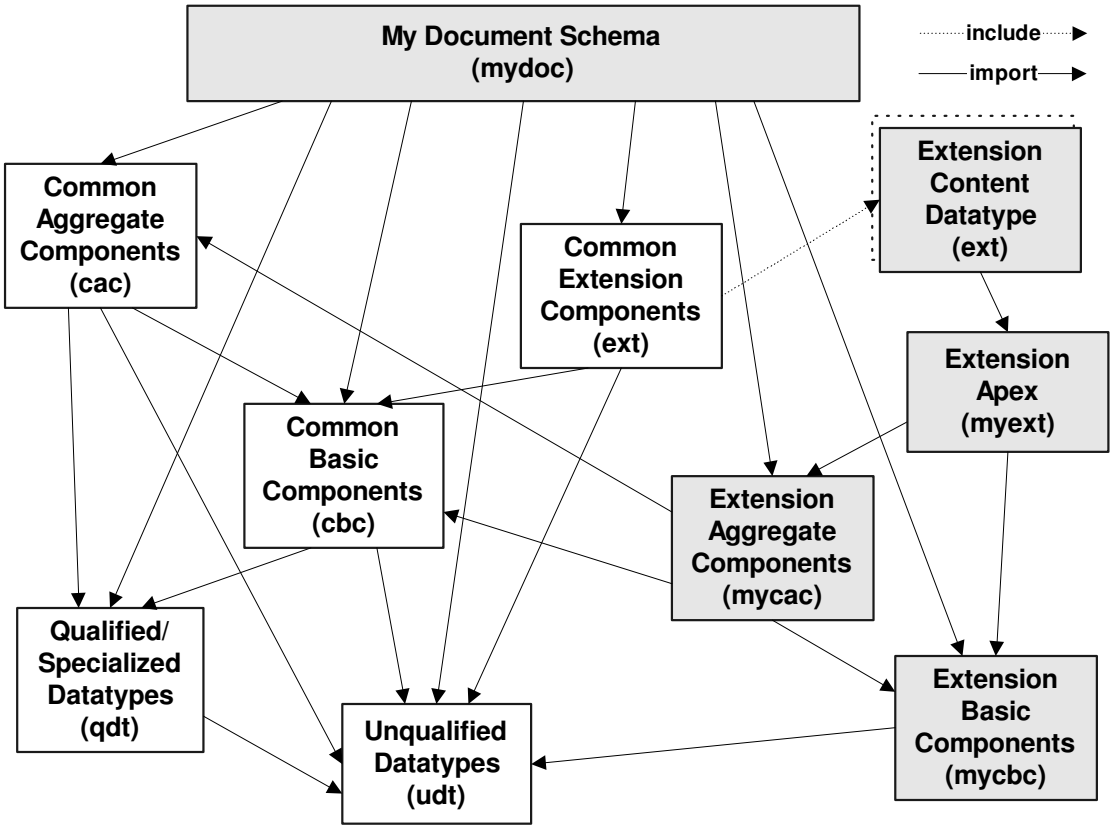


Figure 22. Using the UBL library for non-UBL documents (Crane Softwrights Ltd. Used by permission.)

3.6 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

Copyright © OASIS® 2009. All Rights Reserved.

UBL, including any customized BIEs, business rules, and value constraints. This approach has been used by the Danish OIOUBL project[12] to create and maintain their documentation.

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

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

Copyright © OASIS® 2009. All Rights Reserved.

4 Validation

The UBL committee has published a processing model for a UBL system receiving an XML UBL document, as illustrated in Figure 23.

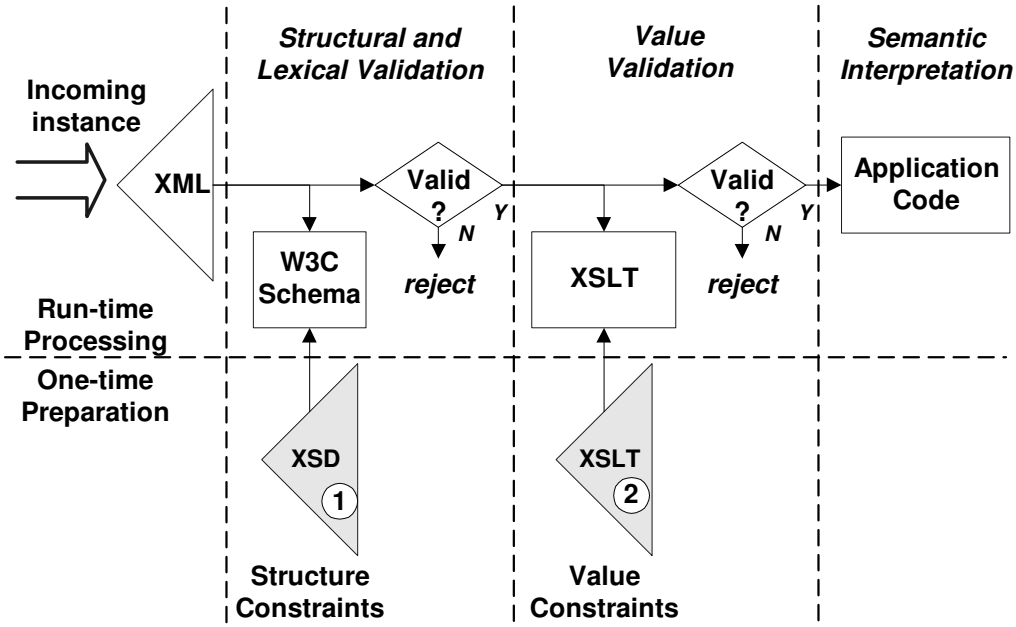


Figure 23. The published processing model for UBL

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.[13] 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 is invalid or because 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

[13] The standard UBL XSD files and a default suite of code list value checks compiled in an XSLT stylesheet are included in the UBL 2.0 specification package.

Copyright © OASIS® 2009. All Rights Reserved.

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

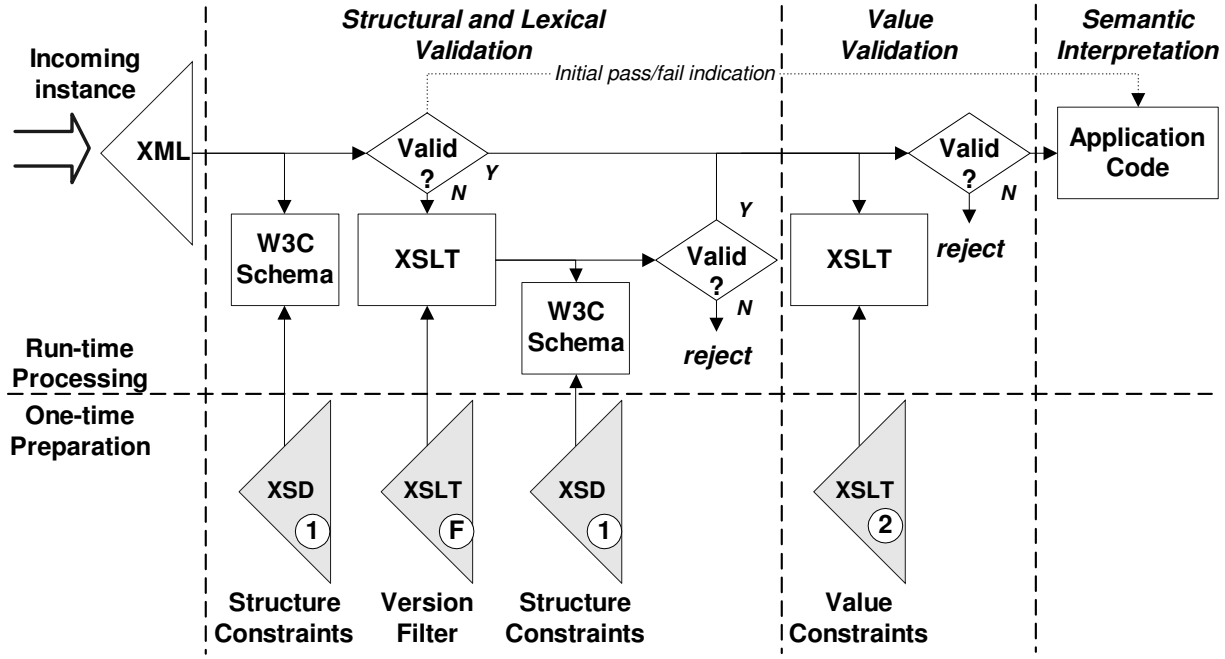


Figure 24. A customized processing model supporting forward compatibility

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 version 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, notwithstanding the presence of additions from a version other than the one the processing application is set up to handle. The application can use out-of-band

Copyright © OASIS® 2009. All Rights Reserved.

decision making based on these unrecognized elements to accept or reject a modified instance for the purposes of doing business.

Whether modified or unmodified, information contained in instances emerging from this process can successfully be extracted by the kind of application that relies on schema validation to inspect instance content. Without some mechanism like the one shown in Figure 24, some instances will be blocked at validation that might, upon further inspection, be judged acceptable because the receiving system simply doesn't need the extra information. JAXB [JAXB] and JiBX [JiBX] are two examples of programming language interfaces to XML in which the programmer validates the incoming instance in order to properly load Java classes. If the interface rejects the instance due to a failure to validate, then the content cannot even be inspected in order for business rules to be applied to business-level rejections. The method shown in Figure 24 guarantees that the application can at least build an input data structure before deciding whether to accept the instance for further processing or, if not, to decide what kind of message to send back to the originating system.

Consider an instance labeled UBL 2.7 coming into a system that validates using a UBL 2.5 schema. First, suppose that the instance happens not to use any elements defined later than 2.5; it validates against the 2.5 schema and is passed to the 2.5-aware application untouched and with an "initial pass" indication. In this case, the 2.7 label is probably irrelevant except as possibly interesting incidental information. Now suppose that the instance does contain unrecognized constructs from schema versions later than 2.5. This instance fails to validate against the 2.5 schema, but when the unrecognized elements are removed, it becomes, in effect, a 2.5 subset instance that gets passed on to the 2.5-aware application with an "initial fail" indication. In this second case, the 2.7 label probably is relevant to the application and to the user in deciding how to proceed.

Copyright © OASIS® 2009. All Rights Reserved.

5 Conformance

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

1020

Copyright © OASIS® 2009. All Rights Reserved.