Position Paper: UBL Specialization Architecture

Author: Bill Burcham (bill_burcham@stercomm.com)

Date: 4-17-2002

Filename: draft-burcham-specialization-arch-01.doc

1 Summary

A decision has been made that XSD will be the language in which the UBL components will be specified. On its own, this decision gives UBL a leg up on the older EDI standards – since the specification is in a formal language (XSD) and validation tools for that language are readily available, conformance checking against the standard is not only possible but probable (convenient).

UBL defines an XML vocabulary for electronic commerce. Based on xCBL, it has roots in the foundational EDI vocabularies like EDIFACT and X12. It is evident to anyone familiar with the use of those vocabularies that they have given rise to thousands of specializations – derived vocabularies, expressed in various degrees of formality, directly or indirectly, on an original standard [EDI-PAIN].

If the difference between a base vocabulary and a specialized vocabulary could be captured in a formal manner many new efficiencies will be possible. For example:

- automated conformance checking against *specialized* vocabularies[1]
- succinct specification of *specialized* vocabularies – in terms of their difference from a base vocabulary
- futuristic, applications such as automatic translation across vocabularies – driven by the inter-vocabulary relationships expressed formally

The purpose of this paper is to develop a position for the architecture that will enable the formal specification of specialized vocabularies. Various alternatives will be explored and reconciled with requirements and extant UBL design decisions. Where appropriate, architectural layers will be delineated so as to arrive at an architecture amenable to phased implementation.

2 Problem Description

Definition 1

Vocabulary. A vocabulary defines a set of terms. In the UBL architecture, a vocabulary consists of one or more XSD complex types defined in one or more XML namespaces.

Definition 2

specialized vocabulary. A specialized vocabulary is a vocabulary that contains at least one specialized type. A specialized vocabulary is defined in terms of a base, or generalized vocabulary.

Certain core tenets are inherited from UBL's basis in ebXML and the Joint Core Components initiative [UBL-CHARTER]:

[1] We believe that conformance checking against a specialized vocabulary is actually more important than conformance checking directly against the (base) UBL vocabulary, since the overwhelming majority of communication will occur using specialized vocabularies (including specializations of specializations).

Axiom 1

specialization happens. Users of a vocabulary create derivative works on purpose and by accident, formally and informally. From [UBL-CHARTER].

Requirement 1

specialization formalism. UBL supports (imposes) a formalism for the specification of specializations with respect to the base vocabulary. It is the structure of such a specification and its method of generation that is a prime subject of this position paper. From [UBL-CHARTER].

Requirement 1 represents a giant leap forward from legacy vocabularies. It is the codification of what in the EDI domain is called "implementation conventions".

Now a rule from NDRSC:

Design Decision 1

UBL is described in XSD. UBL uses XSD to specify constraints on conformant instance documents. It follows that XSD schema validation is the process by which documents are determined to be valid with respect to the UBL specification. From [NDRSC-TBD]

It follows from Design Decision 1 and Axiom 1 that:

Corollary 1

A UBL specialization is described in XSD. Specializations use XSD to specify constraints on conformant instance documents. (i.e. conformant with respect to the specialization). It follows that XSD schema validation is the process by which documents are determined to be valid with respect to a specialization of the UBL specification. From [Design Decision 1].

This is *not* necessarily to say that XSD is the sole mechanism for *relating* a specialization to its base (generalization) rather, that a specialization has an XSD representation.

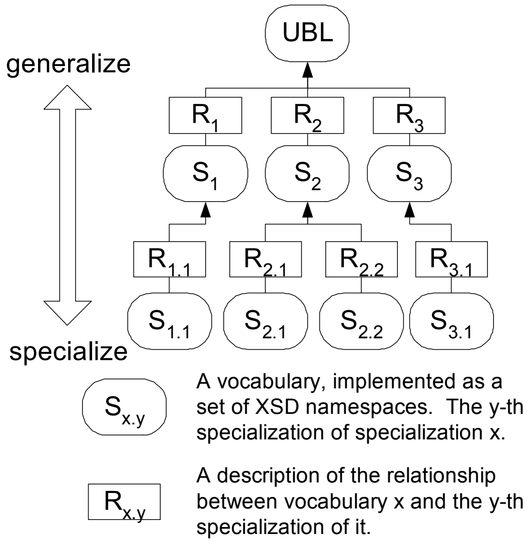
Another corollary to Design Decision 1 is:

Corollary 2

transitive specialization. A specialization may specialize either base UBL or a specialization of base UBL. Any specialization architecture must be recursively applicable. The purpose of this transitivity rule is to explicitly make "in scope" the problem of specializing a specialization. From [Design Decision 1].

So that's an axiom, a requirement, a design decision with two corollaries, and a couple definitions. Hopefully Figure 0 brings this together. It shows how the base and specialized vocabularies (round boxes) are each specified as (sets of) XSD namespaces.

Figure 0


UBL
generalize
specialize
R1
R2
R3
S1
S2
S3
R1.1
R2.1
R2.2
R3.1
S1.1
S2.1
S2.2
S3.1
Sx.y
A vocabulary, implemented as a set of XSD namespaces. The y-th specialization of specialization x.
Rx.y
A description of the relationship between vocabulary x and the y-th specialization of it.

It also shows that there is a formal specification of the relationships (square boxes) between vocabularies. Each of the schemas UBL, S_1 … $S_{3.1}$ may be used to validate instance documents conforming to their respective vocabularies. Some (unspecified) process constructs the S_n from UBL (and more generally, the $S_{x.y}$ from the S_x). That (unspecified) process uses, for instance, to specialize UBL into $S_{1,}$ the formal description $R_{1.}$

Candidate architectures will specify the content and form of the R boxes as well as the process by which the R boxes and generalizations are used to produce specializations.

Now we are set to move deeper into XSD arcana… here is a key tenet from the CMSC:

Design Decision 2

***No namespace aliasing*.** An XSD type is defined within a namespace. Each namespace has a unique name. XSD leaves open the possibility that two schema modules may specify different definitions for a particular namespace. For example:

> in one schema module, "mine.xsd", namespace "foo" contains complex type "Address" having two elements in its content model; in another schema module, "yours.xsd", namespace "foo" contains complex type "Address" having only one element in its content model.

The UBL architecture *prohibits* this sort of namespace "aliasing". Namespaces are immutable. **The UBL specification and conformant specializations never redefine namespaces.** In the example above, schema module "yours.xsd" would have to define a new namespace for its new Address type. From [CMSC-2002-3-29].

Design Decision 2 explicitly prohibits use of the XSD "redefine" element (a variant of import that supports succinct specification of altered, aliased namespaces).

Now we are ready to introduce requirements developed in NDRSC regarding specialization of sequences:

Requirement 2

***sequences*.** UBL supports (in complex type definitions) the notion of a sequence of elements as opposed to only an unordered set. From [NDRSC-TBD].

Requirement 3

mid-sequence element addition. It must be possible for a specialization of a UBL type to add an element between two elements of a sequence. . From [NDRSC-TBD].

In order to understand the next requirement, two definitions must first be understood.

Definition 3

***XSD validation*.** Instances valid with respect to a type containing a required element, must contain the required element**.** This is the straightforward notion of XSD validation: simply that an instance document is either valid or not with respect to the optionality specification of an element's declaration. From [SCHEMA-PRIM].

Definition 4

XSD Type Substitution Principle. A derived type may not take away required elements. XSD derivation enforces a "replaceability" rule: a derived type must be usable in place of its base type. This means that if a specialization is related to its base via XSD derivation then there is no direct way of eliminating in a derived type a required element of a base type. From [SCHEMA-PRIM] and [LISKOV].

And now the requirement:

Requirement 4

remove a required element. It must be possible for a specialization of a UBL type to remove a required element. . We came up with a notion of specializations "taking away required elements". In this case, Definition 3 is maintained, but the specialization (a brand new type) changes the optionality of the element such that document instances may be valid with respect to the specialization and yet the element may be missing. From [NDRSC-TBD].

On its face this UBL notion seems to be incompatible with the XSD type substitution principle (Definition 4). Architecture proposals will either resolve this apparent contradiction, or our assumptions must change.

2.1 Processing Logic Perspective

The whole point of the UBL effort is to bring new efficiencies to electronic commerce applications yet we have so far said nothing about how those applications might benefit from a specialization architecture. In this section we take the perspective of the application processing XML instance documents conforming to the UBL vocabulary or a specialization.

We assume that such applications are represented in XPath [XPATH] or a language derived from XPath (such as XQuery or XSLT). Justify a bit more.

What follows is a list of requirements from the perspective of an application (processing logic) operating on UBL (or specialized) instance documents. These requirements are developed here for the first time and do not reference requirements outside this document.

For purposes of illustration, the following schemas are used to represent base (UBL) and specialized vocabularies. TODO: add description here.

And here are sample instances of the various vocabularies. TODO: add description here.

Requirement 5

Inheritance Selection. Select on an instance of the base, or a specialization, content of an element of a type defined on the base and inherited by the specialization.

XPath: `/ubl:Order/Header/Address/Street/text()`

Expected results: this selects nothing in company-y-doc.xml because companyY:AddressImpl isn't derived from ubl:AddressImpl.

Requirement 6

Extension Selection. Select on an instance of the derived, content of an element of a type defined only in the derived.

XPath: `/ubl:Order/Header/Address/POBox/text()`

Expected results: this works on company-y-doc.xml: /Order/Header/Address/companyY:POBox/text()

Requirement 7

Polymorphic Selection. Select content based on it's base class, that is, write an XPath that will select all Addresses in any document regardless of their "actual" type.

This one does what you'd expect on either a vanilla UBL doc or a doc containing instances of specialized types -- it selects the whole Address structure (even if it's specialized):

XPath: `/ubl:Order/Header/Address/*`

Requirement 8

Tunneling Reuse Selection. Select content of an element of a type defined in the base, even though that element is content for an element of a type defined in the derived.

This one selects CountryCode/Code/text() even though the CountryCode is held in an instance of a type _derived_ from the UBL type. Call this "Tunneling Reuse":

XPath: `/ubl:Order/Header/Address/CountryCode/Code/text()`

Expected Results: tunneling works for companyY as well.

TODO: give a diagram of tunneling reuse selection.

Requirement 9

Global Polymorphic Selection. Select all Addresses (regardless of their actual (sub)type). When executed on a vanilla UBL doc it selects AddressImpl's. When executed on a doc containing instances of companyX:AddressImpl, it selects those:

XPath: `//Address`

?: For some reason, this one does *not* select the companyX:AddressImpl:

`//companyX:AddressImpl`

AHA! it's because companyX:AddressImpl is not an *tag name* -- it's a type name. So that's not a valid XPath!

Requirement 10

Global Extension Selection. Select the specialized POBox ("Extension Selection" with //):

XPath: `//POBox`

Requirement 11

Selection on Type. Find all elements of type companyX:AddressImpl. This works in XPath 1.0:

//*[@xsi:type='companyX:AddressImpl']

This works too:

```
//*[local-name()='POBox']
```

TBD: add “selection on type or subtype” XPath 2.0 thingie.

2.2 Evaluation Criteria

Candidate architectures will be evaluated based on the extent to which they are consistent with the requirements, definitions, axioms and corollaries in problem description. In addition, these criteria will be used:

Criterion 1

Phased Delivery. UBL will be delivered in two phases [UBL-CHARTER]. Delivery of a mechanism for constructing specialized vocabularies from generalized ones is not scheduled until phase 2. Candidate architectures that offer phased implementation are preferable to those that don’t.

Each candidate architecture will be evaluated according to the criteria in this table:

Criterion	reference	weight	Facet1	Facet2
Specialization Formalism	Requirement 1			
UBL Described in XSD	Design Decision 1			
Specializations Described in XSD	Corollary 1			
Transitive Specialization	Corollary 2			
No Namespace Aliasing	Design Decision 2			
mid-sequence element addition	Requirement 3			
Remove a required element	Requirement 4			
Phased Delivery	Criterion 1			
Inheritance Selection	Requirement 5			
Extension Selection	Requirement 6			
Polymorphic Selection	Requirement 7			
Tunneling Reuse Selection	Requirement 8			

Global Polymorphic Selection	Requirement 9			
Global Extension Selection	Requirement 10			
Selection on Type	Requirement 11			

3 Options

Options introduction.

3.1 Option 1: Interface-Based XSD (a.k.a. Paella)

We have demonstrated the solution to R1. *We need to add a use-case for R2*.

The Paella proposal seeks to satisfy requirements 1 and 2 without running afoul of Design Decisions 0.1-0.3. It does this by breaking what would be monolithic (UBL) types into two pieces: interface and implementation. The two pieces together comprise the complete realization of the UBL "core component".

3.2 Option 2: bar

4 Recommendation

5 References

CMSC-2002-3-29	*Minutes of 29-MAR-02 Call,* UBL CMSC, http://lists.oasis-open.org/archives/ubl-cmsc/200203/msg00009.html
EDI-PAIN	*Implementation Conventions, or Why EDI Is Such a Pain*, Mike Rawlins, Rawlins EC Consulting, http://www.metronet.com/~rawlins/x12imp.html
LISKOV	*The Liskov Substitution Principle,* http://www.objectmentor.com/resources/articles/lsp.pdf, http://www.amazon.com/exec/obidos/ASIN/0201657686/qid=1018999063/sr=8-1/ref=sr_8_67_1/002-3294692-6949657
NDRSC-TBD	Placeholder for NDRSC decisions. TBD.
SCHEMA-PRIM	*XML Schema Part 0: Primer,* http://www.w3.org/TR/xmlschema-0/
UBL-CHARTER	*UBL TC Charter,* http://oasis-open.org/committees/ubl/charter/ubl.htm
XPATH	*XML Path Language (XPath),* version 1.0, W3C Recommendation, November 16, 1999, http://www.w3.org/TR/1999/REC-xpath-19991116

TBD: add appendixes for style sheets, schemas and instance documents.