# Position Paper: Global versus Local

## Proposal 01, 10 February 2003

**Document identifier:**

draft-stuhec-globVloc-01.doc

**Location:**

**Author:**

Gunther Stuhec <gunther.stuhec@sap.com>

**Abstract:**

This position paper outlines the use and definition of facets within the UBL library content.

**Status:**

This is V01 of the identifier position paper intended for consideration by the OASIS UBL Naming and Design Rules subcommittee and other interested parties.

If you are on the ubl-ndrsc@lists.oasis-open.org list for subcommittee members, send comments there. If you are not on that list, subscribe to the ubl-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to ubl-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

# Table of Contents

# 1 Introduction

At October 16, 2002 the UBL NDRSC made the decision that they’re using global declared elements instead of local defined elements in our UBL schemas.

Since I have a reasonable perl-script for generating xml-schema output from the different kinds of excel spreadsheets, I'm testing the different possibilities for the representation of xml-schemas.

Therefore, for me it was very easily possbile to see the advantages and disadvantages of the declaration of global elements or local elements which are based on complex types. Additionally I can see this behaviour by using implementations (SAP or XML native databases) or by developing of interfaces by using diverse computer languages or scripts (JAVA, XSLT etc.).

By this level of knowledge, I have seen that the using of global declared elements do have some disadvantages, which might be k.o. criterias. The main problem of that is the global definition of tag names. This problem involves negatively the design time, the developing of highly reusable interfaces and/or implementations and the processing during the run time.

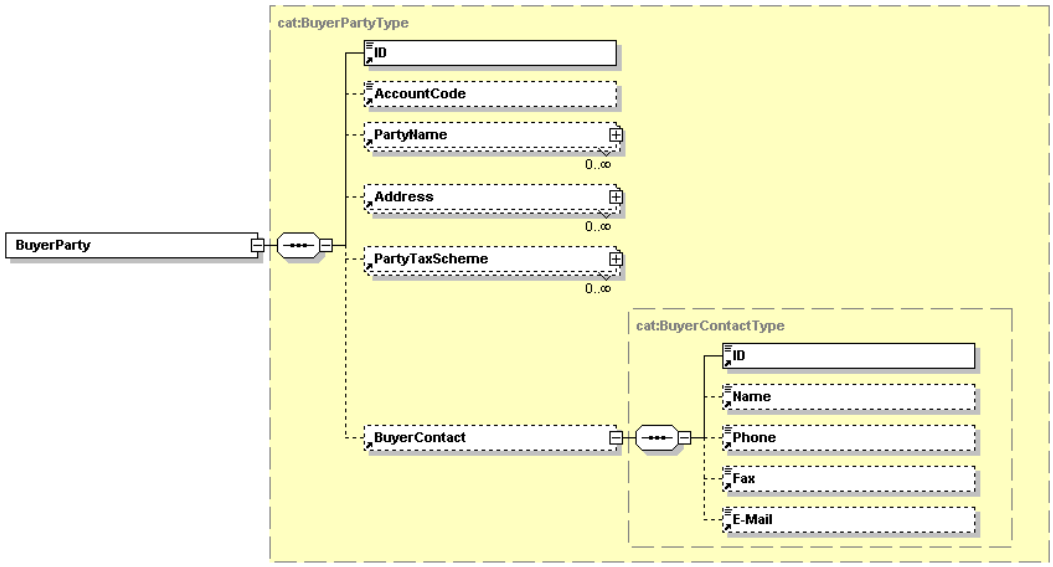
I would like to show these problems in the further chapters in much more detail.

# 2 Real Examples

## 2.1 Inconsistencies of tag-names

### 2.1.1 Problem

We can see some inconsistencies of tag-names in the global element called “BuyerParty”.




The parent element has the tag name “BuyerParty”. Then, we have the child elements ID, AccountCode, *Party*Name, Address, *Party*TaxScheme and BuyerContact. Why do we have sometime the object class term in the tag names and sometimes not? If we look into the spreadsheet, than we see that all child elements have the same object class term.

| BIE Dictionary Entry Name | Object Class Qualifier | Object Class | Property Qualifier | Property Term | Representation Term |
|---|---|---|---|---|---|
| Buyer_ Party. Details | Buyer | Party | | Details | Details |
| Buyer_ Party. Identification | Buyer | Party | | Identification | Identifier |
| Buyer_ Party. Account ID. Code | Buyer | Party | | Account ID | Code |
| Buyer_ Party. Party Name | Buyer | Party | | Party Name | Party Name |
| Buyer_ Party. Address | Buyer | Party | | Address | Address |

| | | | | | | |
|---|---|---|---|---|---|---|
| | Buyer_ Party. Party Tax Scheme | Buyer | Party | | Party Tax Scheme | Party Tax Scheme |
| | Buyer_ Party. Buyer_ Contact | Buyer | Party | Buyer | Contact | Contact |

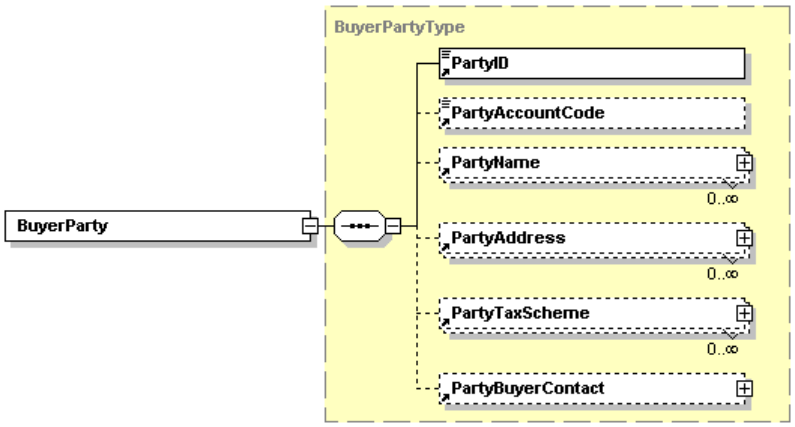
73

74 For an automatic generating system of schemas, it will be very hard to find out, which child-
75 elements must has be a object class qualifier and which of the child-elements not. There does not
76 existing any rule, which is defining the difference between the tag names with object class terms
77 and without object class terms.

## 78 2.1.2 Solution by using global declared elements

79 For an automatic generating system will be easier, if exists some common rules. That means, if
80 we're using global declared elements, must the object class term existing in the tag-names of all
81 child elements, too.

82 For example:

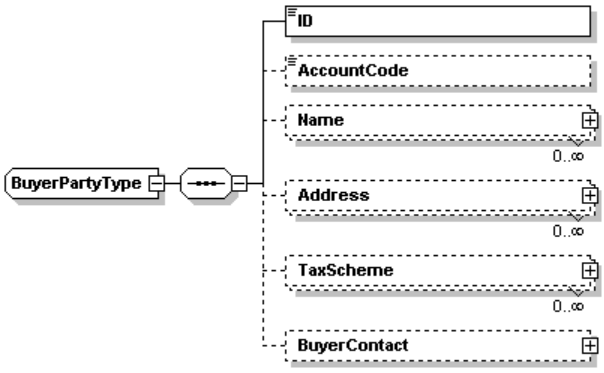



83

84 The disadvantage of that rule is, that we will get always long tag-names with redundancies. That
85 means that the object class term always existing in the parent element and in all child elements,
86 too. We're generating to much and unnecessary information. In particular, if we're generating the
87 tag names with some very long object class term, like "TransportHandlingUnit" or
88 "TransportEquipmentMeasurement".

### 2.1.3 Solution by using local defined elements

We're using the local definition of tag names, instead. Because there is a possibility, that all child elements based on some specific types, but the tag names of these child elements can be shortened by truncation of the object class term. For example:




The equivalent xml schema is:

```
<xsd:complexType name="BuyerPartyType" id="UBL000089">
        <xsd:sequence>
   <xsd:element name="ID" type="cct:IdentifierType" id="UBL000090"/>
   <xsd:element name="AccountCode" type="cct:CodeType" id="UBL000091"
minOccurs="0"/>
   <xsd:element name="Name" type="PartyNameType" id="UBL000092"
minOccurs="0"
       maxOccurs="unbounded"/>
   <xsd:element name="Address" type="AddressType" id="UBL000093"
minOccurs="0"
       maxOccurs="unbounded"/>
   <xsd:element name="TaxScheme" type="TaxSchemeType" id="UBL000094"
minOccurs="0"
       maxOccurs="unbounded"/>
   <xsd:element name="BuyerContact" type="BuyerContactType" id="UBL000095"
minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
```

The advantage is, the child elements can be based always on different types but the tag names itself will be always the same. The tag names can be truncated automatically by one very easy rule:

If the child element representing the same object class, then the object class term must not be shown in the tag element.

This is redundancy free and the complete dictionary entry name can be completed by using for example an XPath navigation path:

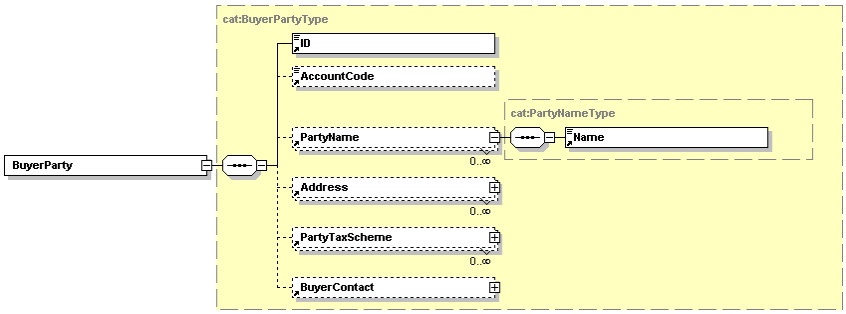
```
/BuyerParty/ID ➔ Buyer_ Party. Identification. Identifier

/BuyerParty/Name ➔ Buyer_ Party. Name. Details

/BuyerParty/TaxScheme ➔ Buyer_ Party. Tax_ Scheme. Details
```

## 2.2 Same sub-element in two or more aggregates with different characteristics

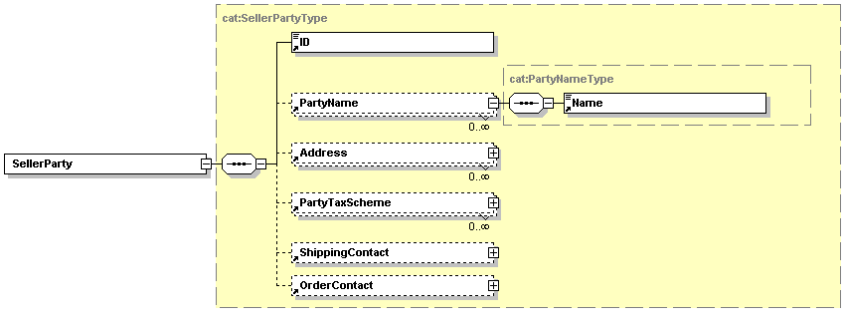
### 2.2.1 Problem

We can have two aggregates, for example BuyerParty and SellerParty and both have some same child elements, like ID or PartyName.

BuyerParty:



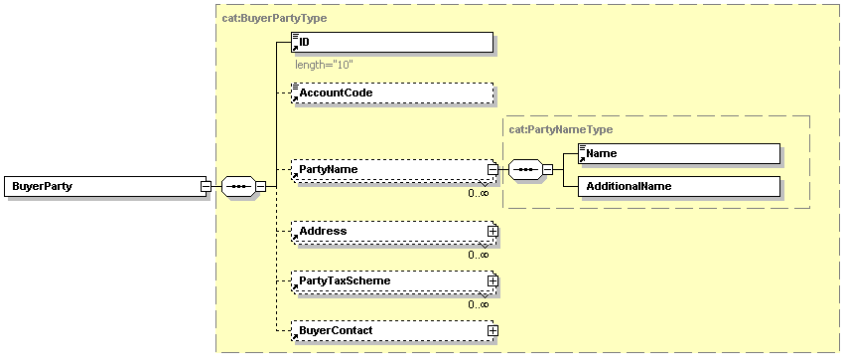

SellerParty:




139

140 But what happens, if we would like to have some specific characteristics for ID or PartyName
141 within the Aggregation “BuyerParty”? For example, the PartyName should have a child-element,
142 like AdditionalName and the ID should be restricted in the maximum length.

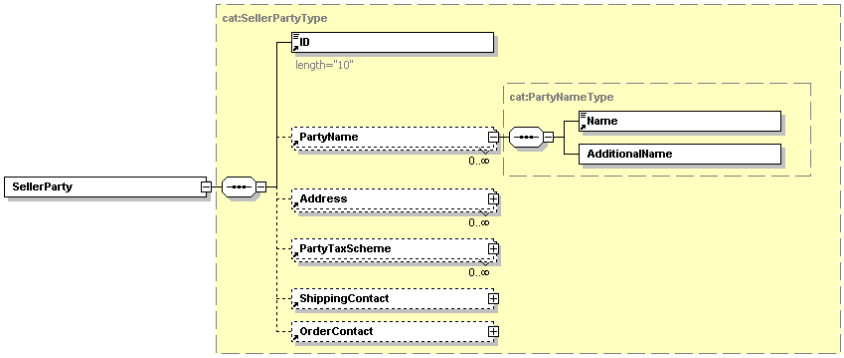
143 Like:




144

## 145 2.2.2 Solution by using global declared elements

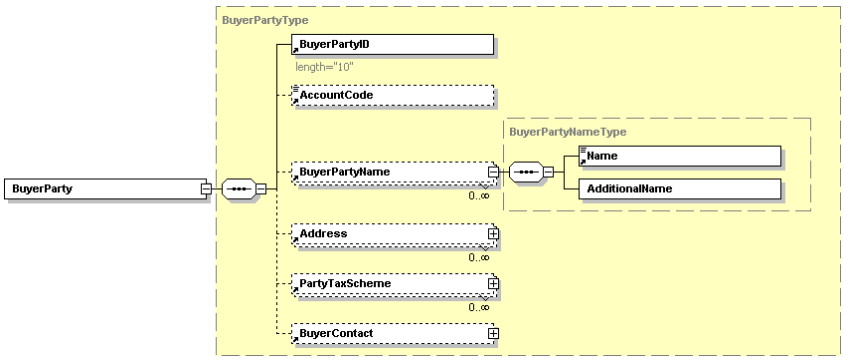
146 If we’re doing the suggested expansion by using the same global declared element, than we
147 would like to see the same expansion in SellerParty, too:




148

149 To avoid this problem, we have to declare some further global elements. But how we will define
150 the tag names itself?

151 Should the tag names of BuyerParty added by the object class terms and object qualifer? Like:




152

153 Why should we do that? And why should SellerParty using the shorter tag names? How can we
154 define a rule for that?

155 I guess, it is very hard to define a rule for this kind of extension, which says, which kind of child
156 elements should have shorter tag names and which kind of tag names should have longer tag
157 names. We would like to run into many conflicts by this.

158 If we're using the global declared elements, it is useful, that all tag names are fully qualifed by
159 always the complete dictionary entry name. This is only the one possibility to avoid the conflicts,
160 as described above in an automatic way.

161 By this way, we will get very long tag-names, like:

162

```
BuyersCatalogueItemIdentificationItemMeasurement (35 Bytes)

SellersHandlingUnitDespatchLineDespatchedQuantity (49 Bytes)

ManufacturersHandlingUnitDespatchLineOrderLineID (48 Bytes)

ManufacturersTransportEquipmentRefrigerationStatusIndicator (59
Bytes)
```

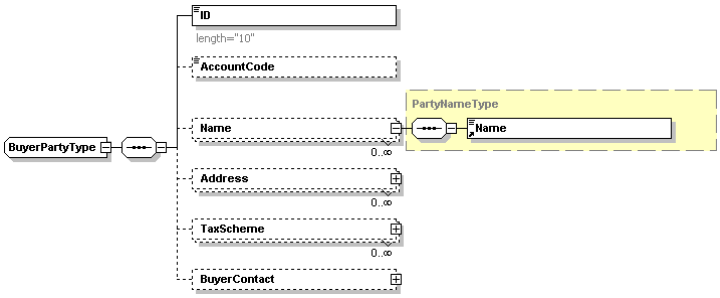
But we've to think about it:

- Many of the applications (databases, interfaces, erp-systems, user-interfaces) can not handle directly with tag names, which are longer as 30 bytes. A mapping (additional processing step) into shorter tag names is necessary.
- Many business documents in the real life have over 10.000 positions. Long tag names would decrease the speed of using, processing and transferring, tremendously.
- Very long tag names usually are not human readable any more. A mapping into much more understandable tag names is necessary.

### 2.2.3 Solution by using local defined elements

All local defined child elements can have tag names, which always based on the dictionary entry name and shortened by the same truncation rules. Each child element can be base on different types. These types can be the common CCTs or the common CCs. If this type The specific characteristics like AdditionalName or length="10" can be defined in this specific types. The types itself can be distinguished by fully qualified names, which can be the same as the dictionary entry name of each BIE.

Example:



The xml schema for this type is:

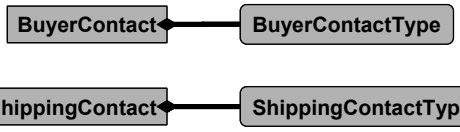
```
<xsd:complexType name="BuyerPartyType" id="UBL000089">
  <xsd:sequence>
    <xsd:element name="ID" type="cct:IdentifierType" id="UBL000090">
      <xsd:annotation>
        <xsd:documentation>length="10"</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="AccountCode" type="cct:CodeType" id="UBL000091"
minOccurs="0"/>
    <xsd:element name="Name" type="PartyNameType" id="UBL000092"
minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="Address" type="AddressType" id="UBL000093"
minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="TaxScheme" type="TaxSchemeType" id="UBL000094"
minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="BuyerContact" type="BuyerContactType"
id="UBL000095" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BuyerPartyID">
  <xsd:simpleContent>
    <xsd:restriction base="cct:IdentifierType">
      <xsd:length value="10"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="BuyerPartyNameType" id="UBL000397">
  <xsd:sequence>
    <xsd:element ref="Name" id="UBL000398"/>
    <xsd:element name="AdditionalName"/>
  </xsd:sequence>
</xsd:complexType>
```

By this solution can be all element tag names in the shortest possible way. All elements can be based on different types. Therefore, the tag names do always have a common understanding and no confusion. All tag names might be short enough for further using in interfaces, databases, user interfaces etc. without mapping into shorter names.

## 2.3 Synchronization of Types

### 2.3.1 Problem

All global declared elements of aggregates based currently on a type with the same name. For two or more different same aggregates with distinguished names exist two ore more equivalent types. Like “BuyerContact” and “ShippingContact”, both aggregates based on the specific types “BuyerContactType” and “ShippingContactType”. But both types have exactly the same structure.




Schema of BuyerContact and BuyerContactType:

```
<xsd:element name="BuyerContact" type="BuyerContactType"/>
<xsd:complexType name="BuyerContactType" id="UBL000078">
  <xsd:sequence>
    <xsd:element ref="ID"/>
    <xsd:element ref="Name" minOccurs="0"/>
    <xsd:element ref="Phone" minOccurs="0"/>
    <xsd:element ref="Fax" minOccurs="0"/>
    <xsd:element ref="E-Mail" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

Schema of ShippingContact and ShippingContactType:
<xsd:element name="ShippingContact" type="ShippingContactType"/>
<xsd:complexType name="ShippingContactType" id="UBL000595">
  <xsd:sequence>
    <xsd:element ref="ID"/>
    <xsd:element ref="Name" minOccurs="0"/>
    <xsd:element ref="Phone" minOccurs="0"/>
    <xsd:element ref="Fax" minOccurs="0"/>
    <xsd:element ref="E-Mail" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

### 2.3.2 Solution by using global declared elements

If we would like to make all elements unique in an automatic manner (see chapter 2.1 and 2.2), all global declared elements must be based on specific types. But all child elements within these types must have the object class term in the element tag name. There is no other possibility to differentiate each child element which has some specific characteristics (facets of leaf-elements or substructure of aggregates) in a unique and automatic way.

You will see this in the following example. Some characteristics of the same BBIEs within the two aggregates “BuyerContact” and “SellerContact” are different. Therefore it is necessary to declare a global element for every BBIE (BuyerContactID, BuyerContactName, ShippingContactID and ShippingContactName) which have some different characteristics. And why should we do that for BBIEs with different characteristics and not for the BBIEs which have the same characteristics? This would become some inconsistencies and would be not handable by parsers for defining interfaces very efficiently.

Example:

Schema of BuyerContact and BuyerContactType:

```
<xsd:element name="BuyerContact" type="BuyerContactType"/>
<xsd:complexType name="BuyerContactType" id="UBL000078">
  <xsd:sequence>
    <xsd:element ref="BuyerContactID"/>
    <xsd:element ref="BuyerContactName" minOccurs="0"/>
    <xsd:element ref="BuyerContactPhone" minOccurs="0"/>
    <xsd:element ref="BuyerContactFax" minOccurs="0"/>
    <xsd:element ref="BuyerContactE-Mail" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Schema of ShippingContact and ShippingContactType:

```
<xsd:element name="ShippingContact" type="ShippingContactType"/>
<xsd:complexType name="ShippingContactType" id="UBL000595">
  <xsd:sequence>
    <xsd:element ref="ShippingContactID"/>
    <xsd:element ref="ShippingContactIName" minOccurs="0"/>
    <xsd:element ref="ShippingContactIPhone" minOccurs="0"/>
    <xsd:element ref="ShippingContactIFax" minOccurs="0"/>
    <xsd:element ref="ShippingContactIE-Mail" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Schema of globale declared elements and the belonged types:

```
<xsd:element name="TimezoneOffsetMeasure" type="cct:TextType"/>
<xsd:element name="ShippingContactID"/>
<xsd:complexType name="ShippingContactIDType">
  <xsd:simpleContent>
    <xsd:restriction base="cct:IdentifierType">
      <xsd:length value="13"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="BuyerContactID"/>
<xsd:complexType name="BuyerContactIDType">
```

```
  <xsd:simpleContent>
    <xsd:restriction base="cct:IdentifierType">
      <xsd:length value="30"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="ShippingContactName"/>
<xsd:complexType name="ShippingContactNameType">
  <xsd:simpleContent>
    <xsd:restriction base="cct:NameType">
      <xsd:maxLength value="40"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="BuyerContactName"/>
<xsd:complexType name="BuyerContactNameType">
  <xsd:simpleContent>
    <xsd:restriction base="cct:NameType">
      <xsd:maxLength value="50"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```

### 2.3.3 Solution by using local defined elements

Better is this, if we solve this problem by using local defined elements. Because all element names ar readable enough, short as possible and truncated automatically by some fixed rules. The most important thing is that all elements within the aggregation with the same object class term do have the same tag names. This helps for a common understanding and makes the implementation and parsing of aggregates more automatizeable. All elements refer to the specific types. The types can either be a very generic CC/CCT or can be a BIE with some specific (restricted) characteristics.

Example, the declaration of BuyerContactType and SellerContactType:

```
<xsd:complexType name="BuyerContactType" id="UBL000078">
  <xsd:sequence>
    <xsd:element name="ID" type="cat:BuyerContactIDType" id="UBL000079"/>
    <xsd:element name="Name" type="cat:BuyerContactNameType"
      id="UBL000080" minOccurs="0"/>
    <xsd:element name="Phone" type="cct:TextType" id="UBL000081"
      minOccurs="0"/>
    <xsd:element name="Fax" type="cct:TextType" id="UBL000082"
      minOccurs="0"/>
    <xsd:element name="E-Mail" type="cct:TextType" id="UBL000083"
      minOccurs="0"/>
  </xsd:sequence>
```

```
</xsd:complexType>

<xsd:complexType name="ShippingContactType" id="UBL000595">
  <xsd:sequence>
    <xsd:element name="ID" type="cat:ShippingContactIDType"
      id="UBL000596"/>
    <xsd:element name="Name" type="cat:ShippingContactNameType"
      id="UBL000597" minOccurs="0"/>
    <xsd:element name="Phone" type="cct:TextType" id="UBL000598"
     minOccurs="0"/>
    <xsd:element name="Fax" type="cct:TextType" id="UBL000599"
     minOccurs="0"/>
    <xsd:element name="E-Mail" type="cct:TextType" id="UBL000600"
     minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

The BBIEs (ID and Name) based on the types which have some restricted characteristics:

```
<xsd:complexType name="BuyerContactIDType">
  <xsd:simpleContent>
    <xsd:restriction base="cct:IdentifierType">
      <xsd:length value="30"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="BuyerContactNameType" id="UBL000397">
  <xsd:simpleContent>
    <xsd:restriction base="cct:NameType">
      <xsd:length value="13"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```

```
<xsd:complexType name="ShippingContactIDType">
  <xsd:simpleContent>
    <xsd:restriction base="cct:IdentifierType">
      <xsd:length value="13"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="ShippingContactNameType">
  <xsd:simpleContent>
    <xsd:restriction base="cct:NameType">
      <xsd:maxLength value="40"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```

All another BBIEs (child elements) based on the standard CCT, because for these BBIEs is no restriction necessary.

## 2.4 Reusability in Interfaces and Implementations

### 2.4.1 Problem

One of the biggest benefits of XML is the development of very efficient interfaces and applications with a high reusability. But this must be based on very efficient XML schemas as well as XML instances. Otherwise, you will have the same effort as without XML.

The most of interfaces (for databases, userinterfaces, to applications etc.) using the tag names of XML structures for defining variables or database tables, normally. It should be the possibility that we can reuse all BIEs and CCs for the different development requirements, too. And this will be possible, if we have always a common understanding or processing of all BIEs without any additional mappings or control procedures. This helps us, to develop applications in a very fast and cheap way.

If exists inconsistencies in tag names especially, you will loose these advantages in developing, rapidedly.

### 2.4.2 Solution by using global declared elements

Global delared elements do have always inconsistencies in tag names. Because all tag names itself must be unique and if you have the same BIE with two different characteristics, you have to define two different elements with different tag names. By this, you must query in the program every tag name itself and this makes the programming very inefficient.

For example, you have the following instance with global declared elements:

```
< BusinessDocument>
< BuyerContact>
  < BuyerContactID>00000000000000000000000120321</cat:BuyerContactID>
  < BuyerContactName>Hugo Herbert</cat:BuyerContactName>
  < BuyerContactPhone>+49 54639 4334</cat:BuyerContactPhone>
  < BuyerContactFax>+49 33853 3843</cat:BuyerContactFax>
  < BuyerContactE-Mail>hugo.herbert@ubl.org</cat:BuyerContactE-Mail>
</BuyerContact>
<ShippingContact>
  <ShippingContactID>0000000134543</cat:ShippingContactID>
  <ShippingContactName>Berta Bertram</cat:ShippingContactName>
  <ShippingContactPhone>+1 43543 43453</cat:ShippingContactPhone>
  <ShippingContactFax>+1 35433 4343</cat:ShippingContactFax>
  <ShippingContactE-Mail>bert.bertram@ccts.org</cat:ShippingContactE-Mail>
</ShippingContact>
```

```
</BusinessDocument>
```

And you would like to express the information in the following format:

```
Buyer:
ID: 00000000000000000000000120321
Name: Hugo Herbert
Phone: +49 54639 4334
Fax: +49 33853 3843
E-Mail: hugo.herbert@ubl.org

Shipper:
ID: 0000000134543
Name: Berta Bertram
Phone: +1 43543 43453
Fax: +1 35433 4343
E-Mail: bert.bertram@ccts.org
```

There is a very inefficient procedure necessary, because you have to parse every tag name separately and you have to generate another output tag information, because the global declared elements are too long and not understandable in a common way.

The following perl script shows the problematic in more detail:

```
use XML::SimpleObject;

my $parser = new XML::Parser (ErrorContext => 2, Style => "Tree");
my $xmlobj = new XML::SimpleObject ($parser->parse($XML));

print "Buyer: \n";
process_buyer_contact ($xmlobj->child("BusinessDocument")-
>children("BuyerContact"))
print "Shipper: \n";
process_shipping_contact ($xmlobj->child("BusinessDocument")-
>children("ShippingContact"))

process_buyer_contact {
       my $contact;
       printf( "ID: $s\n", $element->child("BuyerContactID")-
>value );
       printf( "Name: $s\n", $element->child("BuyerContactName")-
>value );
       printf( "Phone: $s\n", $element-
>child("BuyerContactPhone")->value );
       printf( "Fax: $s\n", $element->child("BuyerContactFax")-
>value );
```

```
      printf( "E-Mail: $s\n", $element->child("BuyerContactE-
Mail")->value );
}

process_shipping_contact {
      my $contact;
      printf( "ID: $s\n", $element->child("ShippingContactID")-
>value );
      printf( "Name: $s\n", $element-
>child("ShippingContactName")->value );
      printf( "Phone: $s\n", $element-
>child("ShippingContactPhone")->value );
      printf( "Fax: $s\n", $element->child("ShippingContactFax")-
>value );
      printf( "E-Mail: $s\n", $element->child("ShippingContactE-
Mail")->value );
}
```

### 2.4.3 Solution by using local defined elements

If the tags of child elements within same aggregates (based on same ACCs) have always the same names, it will be much easier and efficient to develop interfaces for processing the instances.

The following instance based on local defined elements. The aggregates BuyerContact and SellerContact based on the ACC (ContactDetailsType). By this, all equivalent child elements have the same tag names.

```
<BusinessDocument>
<BuyerContact>
  <ID>00000000000000000000000120321</ID>
  <Name>Hugo Herbert </Name>
  <Phone>+49 54639 4334</Phone>
  <Fax>+49 33853 3843</Fax>
  <E-Mail>hugo.herbert@ubl.org</E-Mail>
</BuyerContact>
<ShippingContact>
  <ID>0000000134543</ID>
  <Name>Berta Bertram</Name>
  <Phone>+1 43543 43453</Phone>
  <Fax>+1 35433 4343</Fax>
  <E-Mail>bert.bertram@ccts.org</E-Mail>
</ShippingContact>
</BusinessDocument>
```

Therefore, you can develop some reusable subroutines (function) for processing the structure for the ACC “ContactDetails” and you can use this functions every time if you have to process some

ABIEs which are based on “ContactDetails”. Further, you have a higher reusability of the tag names. You can use this tag names for the understandable representation, without any mapping.

The following perl script shows the reusability of same structures and common tag names:

```
use XML::SimpleObject;

my $parser = new XML::Parser (ErrorContext => 2, Style => "Tree");
my $xmlobj = new XML::SimpleObject ($parser->parse($XML));

print "Buyer: \n";
process_contact ($xmlobj->child("BusinessDocument")-
>children("BuyerContact"))
print "Shipper: \n";
process_contact ($xmlobj->child("BusinessDocument")-
>children("ShippingContact"))


process_contact {
       my $contact;
       foreach my $element ( $contact->child ) {
              printf( "%s: $s\n", $element, $element->value );
       }
}
```

For processing “BuyerContact” and “ShippingContact” is only one function necessary. And you can use tag names directly for showing in the output. Therefore, you will get the same output, but much more efficiently.

```
Buyer:
ID: 00000000000000000000000120321
Name: Hugo Herbert
Phone: +49 54639 4334
Fax: +49 33853 3843
E-Mail: hugo.herbert@ubl.org

Shipper:
ID: 0000000134543
Name: Berta Bertram
Phone: +1 43543 43453
Fax: +1 35433 4343
E-Mail: bert.bertram@ccts.org
```

# 3 Reusability

XML offers us the possibility to have a reusability in two different ways:

Structure and Elements

Programming and Interfaces

A business document language will be accepted worldwide, if we as developer of this language recognize both ways of reusability. Therefore it is a must for UBL to consider both areas. Otherwise, UBL will be ignored on the one hand side from the designers of business documents and on the other hand side from the developers of interfaces and applications. And this can not be happen for a standard, which will be the only one business language over the world at one time.

## 3.1 Reusability of Structures and Elements

A structure and elements should be so often used as possible. Global declared element offers for this reason some advantages more. All elements based on a fixed tag name and on a fixed structure. Therefore, you can refer to these elements only. There is no wrong definition and no wrong interpretation. But this will be only effective, if you would like to define business documents.

The problem of global declared elements is that all elements are declared in the same hierarchy. This leads to inconsistencies in defining of the tag names. Especially, if you have some child elements which based on same BCCs or ACCs but it has different characteristics or sub-structures. This inconsistencies influence the modeling and programming, seriously.

Therefore it will be better, if the name of same child-elements and in different aggregates always the same. And this is only reachable by using local defined elements. The tag names of these elements will be consistent, too, if the tag names always be based on the dictionary entry name and if these names always be shortened in the same manner (UBL tag name truncation rule).

## 3.2 Programming and Interfaces

The definition of business documents will be mostly done by modeling-tools (like UML class diagrams) in future. Because these modeling tools considers the following parts:

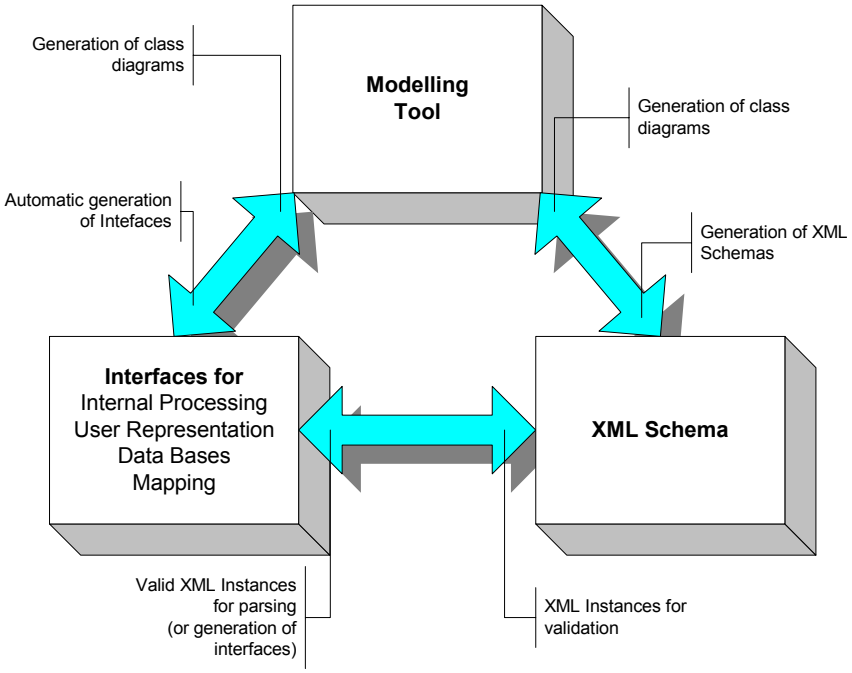
No knowledge in XML schema definition is necessary

Automatic generation of XML schemas

Automatic generation of different types of interfaces.

Especially the smallest companies do not have any knowledge about complex XML schemas. Therefore a couple of software vendors developing on graphical modeling and business document interaction tools, which give the small companies the great possibility to participate on e-Collaboration. The users of business documents will be not confronted with XML itself, in future. This will be the only one internal physical format.

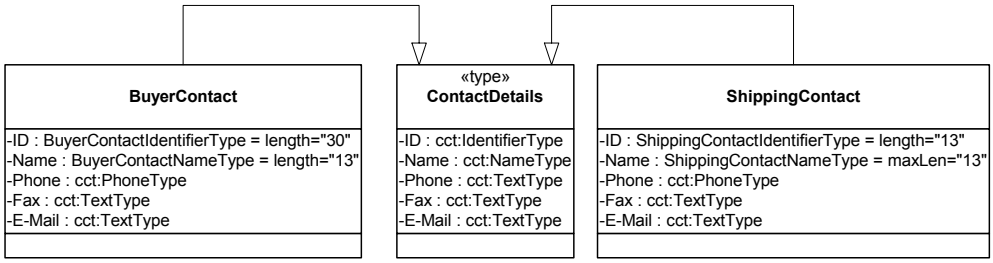
Therefore, it will be very narrow interfaces between modeling, xml and developing of interfaces, in future (see following picture).



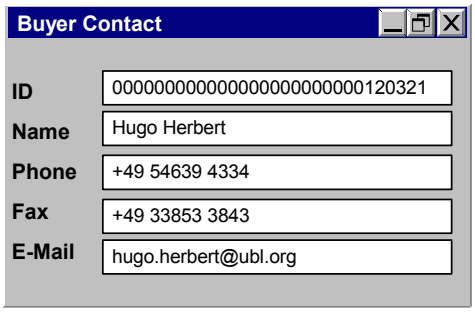

This is only possible, if all names and structure will be always consistent and have always the same meaning. This structures can be used in many times without any big effort.

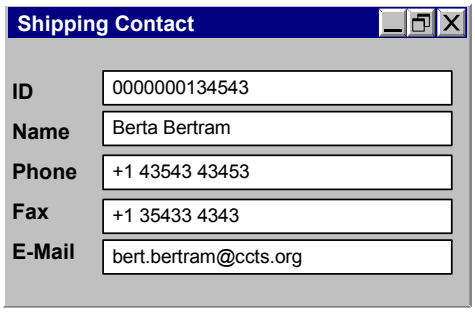
For example:

## Object Oriented Modelling by Class Diagrams




## Generating user interfaces

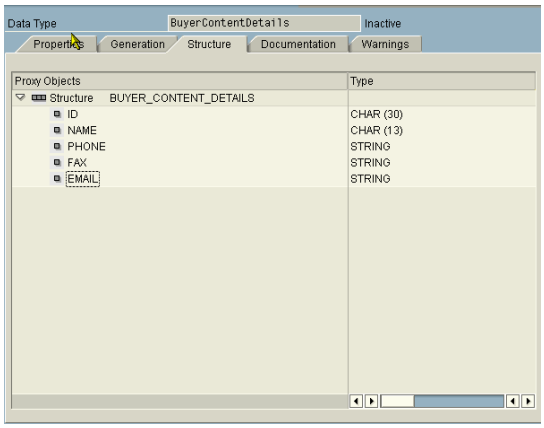


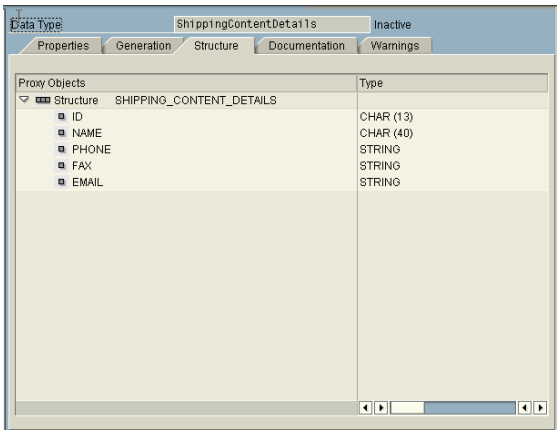




## Generating of database tables

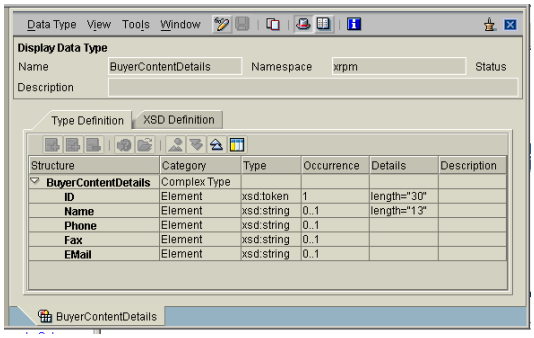
| | Buyer | Shipping |
|---|---|---|
| ID | 00000000000000000000000120321 | 0000000134543 |
| Name | Hugo Herbert | Berta Bertram |
| Phone | +49 54639 4334 | +1 43543 43453 |
| Fax | +49 33853 3843 | +1 35433 4343 |
| E-Mail | hugo.herbert@ubl.org | bert.bertram@ccts.org |

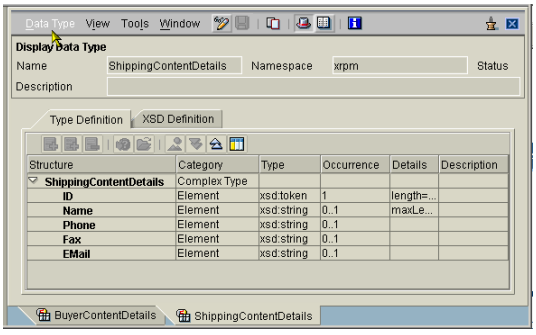
## Generating ABAP-Objects for SAP development environment




**Storing into a repository for providing and mapping to another dialects**




**Developing and/or generating interface applications**

```
use XML::SimpleObject;

my $parser = new XML::Parser (ErrorContext => 2, Style => "Tree");
my $xmlobj = new XML::SimpleObject ($parser->parse($XML));

print "Buyer: \n";
process_contact ($xmlobj->child("BusinessDocument")-
>children("BuyerContact"))
print "Shipper: \n";
```

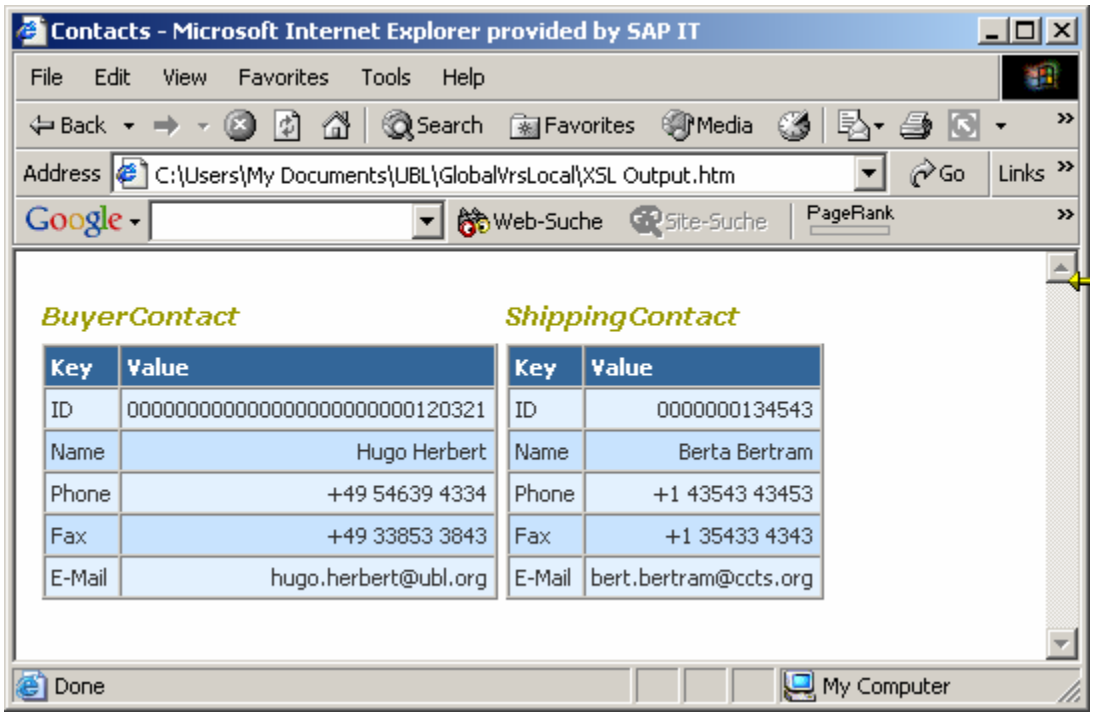
```
process_contact ($xmlobj->child("BusinessDocument")-
>children("ShippingContact"))


process_contact {
       my $contact;
       foreach my $element ( $contact->child ) {
              printf( "%s: $s\n", $element, $element->value );
       }
}
```

Developing and/or generating XSLT-Scripts

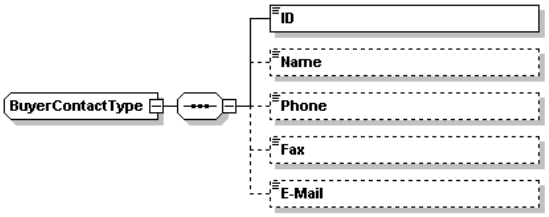
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY nbsp " ">
]>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:output method="html" indent="yes" encoding="UTF-8"/>
<xsl:template match="/">
  <html>
    <head>
      <title>Contacts</title>
      <link rel="stylesheet" type="text/css"
href="0020068250000000584722001E.css"></link>
    </head>
    <body>
      <table>
        <tr>
          <xsl:for-each select="BusinessDocument">
               <xsl:apply-templates select="./*"/>
          </xsl:for-each>
        </tr>
      </table>
    </body>
  </html>
</xsl:template>
<xsl:template match="*">
  <td>
    <h2>
      <xsl:value-of select="name()"/>
    </h2>
    <table border="1" cellspacing="0" cellpadding="3">
      <tr>
        <th scope="col">Key</th>
        <th scope="col">Value</th>
      </tr>
      <xsl:for-each select="./*">
        <tr>
          <xsl:attribute name="class">
               <xsl:choose>
```

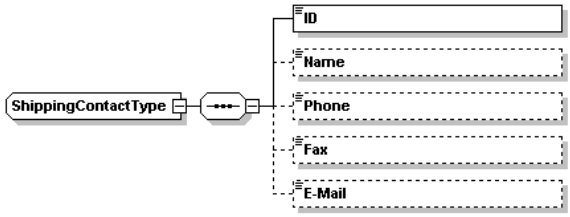
```
                        <xsl:when test="position() mod 2 = 0">
                                <xsl:value-of select="'darkrow'"/>
                        </xsl:when>
                        <xsl:otherwise>
                                <xsl:value-of select="'lightrow'"/>
                        </xsl:otherwise>
                    </xsl:choose>
                </xsl:attribute>
                <td>
                    <xsl:value-of select="name()"/>
                </td>
                <td align="right">
                    <xsl:value-of select="."/>
                </td>
            </tr>
        </xsl:for-each>
    </table>
  </td>
</xsl:template>
</xsl:stylesheet>
```

**Generating HTML-Output**




**Defining complex Types within XML Schemas**




612

613 All examples above base on the same schema structure of the ACC “ContactDetails”. It would be
614 always used for “BuyerContactDetails” and “ShippingContactDetails” without additional mappings.
615 Only some applications (ABAP-Objects and database tables) need the restrictions of the length of
616 the BIEs ID and name. Therefore it is necessary to define some additional complex types
617 “BuyerContactIdentifierType”, “BuyerContactNameType”, “ShippingContactIdentifierType” and

618 “ShippingContactNameType” with this restricitions. Because, this restrictions would be useful for
619 the validation of XML instances and it is necessary for the automatic generation of ABAP Objects
620 or database tables..

# 621 4 Recommedation

622 A consistency of tag names of the same or similar aggregations is necessary to enables a
623 reusability of BIEs in applications, programs and interfaces, too. The consistency is not
624 reachable, if we’re using global declared elements and we would like to have very short tag
625 names itself. Many elements would get completely different tag names itself, although if they
626 would be the same BBIE or ASBIE of different ABIEs, which based on the same ACC, but in
627 different contexts. In particular is a consistency not reachable, if we have hunderts of elements in
628 one namespace and on the same hierarchy.

629 If the consistency and uniformity of tag names is not possible, the efficient reusability in
630 developing of programs/interfaces and automatic generating would be decreasing enormously.

631 Therefore, would I highly recommended that we’re using local defined elements instead of global
632 declared elements. Because this elements can be truncated always in the same manner and you
633 have in all ABIEs which are based on one ACC the same short, human and technical readable
634 tag names.

635

# Appendix A. Bibliography

# Appendix B. Notes

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's procedures with respect to rights in OASIS specifications can be found at 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 specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.

# UBL-NDRSC MESSAGE

[Date Prev] | [Thread Prev] | [Thread Next] | [Date Next] -- [Date Index] | [Thread Index] | [Elist Home]

---

*Subject*: **[ubl-ndrsc] MINUTES: Face to Face 5 Feb 2003**

- *From*: **Lisa-Aeon <lseaburg@aeon-llc.com>**
- *To*: UBL-NDR <ubl-ndrsc@lists.oasis-open.org>
- *Date*: Wed, 05 Feb 2003 21:44:31 -0600

---

Here are the minutes from today's very lively discussion of Local vs. Global. Please read through and be ready to start again tomorrow! Only kidding!

1. Roll call ** didn't get roll call complete **

Bill Burcham - phone in
Mavis Cournane
Mark Crawford
Fabrice Desré
Arofan Gregory - phone in
Michael Grimley - phone in
Eduardo Gutentag
Lisa Seaburg
Gunther Stuhec
Paul Thorpe
Anne Hendry - phone in
Danny Vint (observer, ACCORD)
Dave Carlson (observer)

2. Current position summary

Mavis: issues discussion procedure:
- list issues
- speak by invitation

Fabrice summarizes the case so far. In Burlington we agreed on Global. The main reasons behind Global was element reuse. Qualified elements were also easier to customize. The "Garden of Eden" define a CT and for each CT at least one global element and these global elements are then reused in the BIE.

For each complex type at least one global element.
Mavis: In 0p70 release what did they do?
Mixture - Gunther's algorithm is to first generate for each complex type an element and then wherever complex type is used generate element whose name is derived from the property type name.

Arofan: given that intent was reuse, maybe that's ok.
Bill: you can always make a new type
Eduardo: this is a tangent; should come back to global vs. local.
Mavis: Did we have to bodge our rules for 0p70 to get something usable? Is there anything in 0p70 that we might have to revisit this.

Gunther presents slides (this was sent to list). Over 1700 BIEs. Strongly derived from ebXML CCs. Example of 2 diff BIEs expressing same ABIE. EG. global declared element 'ID'. But then might need restrictions: "Dunns", etc. So need to declare new global declared element. Then must change all aggregate types which are using that global declared element. e.g.. buyer address and seller address aggregate types; and manufacturer address and delivery address - also types. 4 BIEs, 4 separate types. All reference element 'ID' (ref='id').
manufacturer using a specific address (e.g.. Dunns), not this one. What happens then? Have to change all these. Impacts implementation, schemas, any interfaces or applications that are based on or using this aggregation. A better way?

Fragment processing is uncomplicated etc.

B: we are defining the global element as an outcome of where the CT is used in a content model, if some of the ABIEs are never referenced we should never generate the element.

Bill: if I want to do a specialization of a ubl BIE type like id and define a new abie type that will use that bie type id. How do I get a ubl document to carry that new type - can a local element that refs a global element declaration carry a specialization of that global? [Ed note: see last few paras for Bill's test of this - it does work, is possible]
A: for each instance of use where the name is new we generate a new element. In the 0p70 release, they did a mixture.
G: algorithm is to first generate for each CT an element and wherever the CT is used generate a Global element whose name is derived from the property and we end up with spurious generated global element.

G: we made the decision based on short examples and we did not use all the CCTS rules and definitions. This works fine where you are defining a few elements only.
G: We (SAP) looked at the Boeing, EAN.UCC and we have over 1700 different unharmonized BIES and to harmonize them makes it very difficult using Global elements

All BIES based on the CCTS and we need to look at how we can use it for our definitions of our schemas. We have to use the DictionaryElementNames and how do we get short tag names.

One or more global elements are derived from an aggregate TYPE. A bunch of aggregate types have some elements and these refer to global declared elements. the global element will be reusable in the aggregate types. What happens if the same element expresses two different BIEs and they have two different characteristics. For example ID, EANID or DUNSID and we have to define a new global element with a specific name and then new complex types. This new global element will get a completely new name and this new global element impacts some already defined aggregate types. Because I need some restrictions for EANID I have a new ID tag and I have to change all aggregate types which are using that global declared elements.

B: is your concern while CCTs might be derived from one another there is no way to do that with elements.
G: Manufacturer and Seller use specific IDs like EANID of type EANID.type with new characteristics and restrictions, you have to change all your refs into EANID. This impacts our interfaces and our applications i.e. everything that uses those aggregations.

G: all tag names must be unique we have 1700 BIEs we need unique tagnames. The ebXML CCTS dictionary entry names are unique and these are too long for our tag names.

A: It is not clear what a global element is it a BIE or a Core component.

G: Modifications of BIEs lead to change of type and tag names. Changing tag names requires additional effort. Our defined truncation rules cannot be used for global element tag names. Truncated tag names can have different semantic meanings.

B: any discussion predicated on global meaning we have to use long tag names is specious. Op70 has short tag names.

List of issues with Gunther's presentation
-What does it mean to "change" and the use of namespaces
-Impact on Instances with namespaces, long tag names and semantics
-Benefits of instance rules and impact on understanding and processing
-Complexity of schema, ebXML compliance, organization of namespaces in a library
-LCSC process, the need to always generate unique names
-Detailed exception handling

What does it mean to "change" and the use of namespaces
E: The departure point is the UBL library and schemas and those are changed by someone just by adding types and elements at will, with no modification of metadata in the schema itself. It all becomes just a modification of the original schema.
E: You can't just do that. For UBL compliance you have to add a new namespace in which you add a Type and an element but it is in that namespace. It is irrelevant if this is local or global.
B: If I want to do a specialization of some UBL BBIE type like ID and I want to define a new ABIE . How do I get UBL document to carry that new ABIE. If we took schema as it stands and try to do a specialization. Gunther's presentation makes me think there is no way with 0p70 to come in and do a specialization and have it carried in a UBL doc.
A: Import an existing type and extend it with my restriction. Everything using it in the schema must be declared in the new namespaces and be part of the extension. You have created a new doc type that can be processed excluding that one modified bit.
E: If we had local elements, would that be as extensive a modification?
A: There is no diff. Once you change a type you have to change everything between it and the doctype.
B: Under local element schema it is not necessary to do this much modification. You can specialize it at the point of where the specialization wants to be used. You use xsi:type.
A: This dynamic remains true regardless if you use local or global
B: With global elements you won't have the option to use XSI:type at the point of use.
A: XSI type is in the document
G: "change" at the design time of the library. How do you generate new tag names.
If we have 3 new global elements you have to put the object class on to the tag name itself the name is very long.
E: Gunther was talking about design of libraries and not customization.
E: In customization you would use different namespaces.

Impact on namespaces:
A: It is said we have 1700 unique elements, but that is untrue, because they are under different namespaces. You assume that if you have several IDs, they are part of the reusabletypes, their specialization belongs to the new namespace of the document.
A: Example, product ID, restriction of the core ID, in that case disambiguating them with different names, is not true. Don't buy that there are 1700 different elements.
G: The BIEs are collected from different organizations and can be in different namespaces. When we harmonize those into one library what happens.
E: You have to change names. There is no way to avoid huge clashes and issues if you try to harmonize
G: Enormous consequences, some names will be very long,others very short. Not very elegant.
B: Local names means you don't have 1700 things to resolve. You have to resolve the types and not every element of every type

M: Does not mean you don't have to resolve the BIEs.
G: How can we use the same types of aggregation in the same way. Long tag names not very helpful. Interfaces are not readily shareable.
A: What is a type and a BIE. Is a BIE a semantic model to pin down one and only one definition.
M: CCTS calls for everyone of those to be a unique BIE and therefore unique complex type.
G: If you have a library with 1000 different BIES, then same number of complex types and global elements. How do you handle those global tag names efficiently in many different interfaces. For each dictionary entry name and defining the tag name.
A: Last release proved it was unnecessary to do that.
E: At some time using the local element method. Doubts were raised about it . We talked about global and we weighted this. Very unscientific but Eduardo was very conservative. However, resistant to going back to it. The main motive for global was the issue of reusability. At this point we can boil it down to reusability. Gunther is objecting that it does not make reusability easier.
A: Understanding of large vocabulary, it is very easy if within each namespace each name means one set of semantics and one structure. We should use slightly longer tagnames where there are nuances of meaning.
G: We are defining alot of extra rules to handle this issue then.
A: CBL used one namespace and some fairly long tag names. You are trading difficulty in creating the library against the difficulty of using it.
B: IF people agree local would be nice if all our tools understood types and we were all using Xpath 2.0, then a short cut would be to make it incumbent upon the local party to show why this does not matter or demonstrate this can be handled with existing tools.
M: Mark does not agree. It violates a Universal Business Language in XML expressions.

B: what decision would we make if xpath and XSLT 2.0 were in place?
E: decisions shouldn't be based on technology that doesn't exist. but as soon as xpath 2 and XSLT 2 are ready they will explode in the market place. how painful will it be in 2 years to switch decision?
A: also other things like versioning which will benefit, like with CBL, people stuck with a thing that worked best until they could switch - wasn't big deal.
G: Yes, would not be big deal.
G: The automatic generation of these global elements, the developer is not interested in the tag names himself, he is interested in the dictionary entry name. The fixed rules to output these, are more important.
A: I don't think having to fix clashes is as big a problem as the trade off of the locally defined names. Example: When you have 4 things with different names but they are the same things. At that point I should be able to go look at them in the schema and resolve the clashes then.
G: At that point you have to write a tool to go look at this a second time, I am saying the BIE's are unique within the library locally, so I do not need any further tools to do any further work.
A: It doesn't complicate all things, it does not impact users at all. If we have to write more tools, I don't think that is so bad.
E: Either way tools are not relevant., I want to go back to the issue of reusability. I also have an issue with customization. about a half hour ago, I heard it said that local makes customization easier. If we go local it impacts many more issues than just reusability.

B: We keep saying that a name is a sequence of names divided by slashes. the principle is tree structuring our namespace. XPath makes each name globally unique. If we can agree that local has alot of good attributes...

G: If you are using local defined elements you still have the unique names. Sometimes if you use global defined elements, using XPaths, the dictionary entry names...

A: We said that each construct would be uniquely named.
M: I need to clarify something. There is no requirement to follow the tripartite naming. There is no rule saying the name has to be tripartite in the CCTs.
G: ebXML CC in the future will be the standard. This is the preferred
E; Lets try to resolve this.
M: Do we need to discuss how to reverse the last decision made.
E: Those that are meeting today and this week, should come up with a proposal.

A: What is necessary to reverse the decision. Based on the criteria of usability and reusability. What does it look like if you make these changes.
M: Until I see instances documents, I will remain in the other camp. I want to know what is wrong with the current approach and I want to see the new proposed approach.
B: We need to keep processing logic as well (stylesheets) in this debate.

E: I want to see the issue of context modification included.
M: This needs to become a position paper.
A: This F2F should record the issues.

We have decided it is now time for the parties who would like to reverse the decision to concretely demonstrate the problem encountered with concrete examples that would effect whatever we have at this point.

Meeting adjourned: 18:35 GMT

---


**Attachment:** Lisa Seaburg.vcf
*Description:* text/vcard

---

- **Follow-Ups**:
  - **[ubl-ndrsc] updates to modnamever**
    - *From:* Dave Carlson <dcarlson@ontogenics.com>

---

[Date Prev] | [Thread Prev] | [Thread Next] | [Date Next] -- [Date Index] | [Thread Index] | [Elist Home]

Search: this month Match: all Sort by: score
Words: Search | Help

---