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.

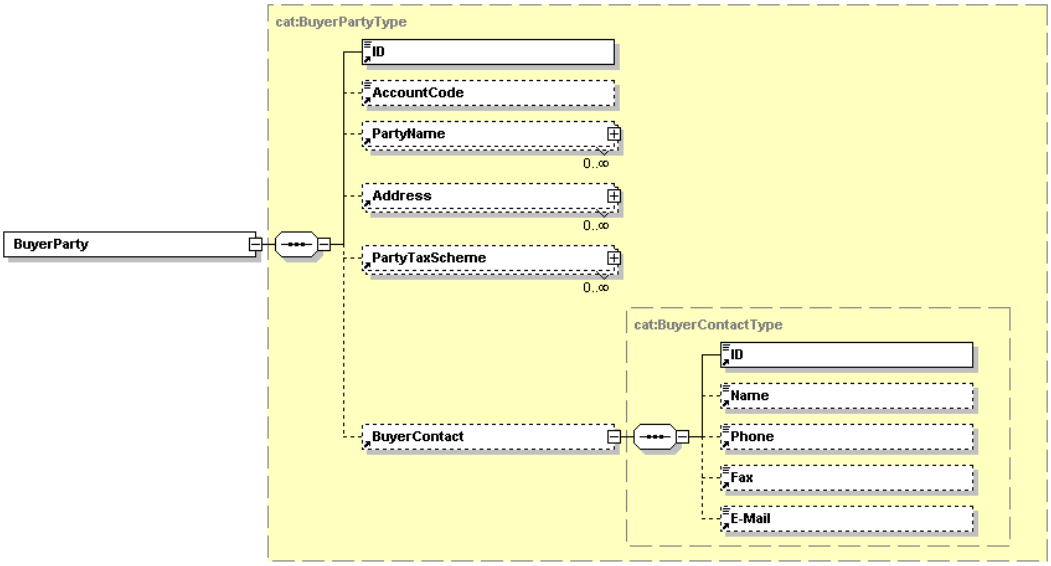
I would like to show these problems in the further chapters in much more detail.

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	Buyer	Party		Party Tax	Party Tax

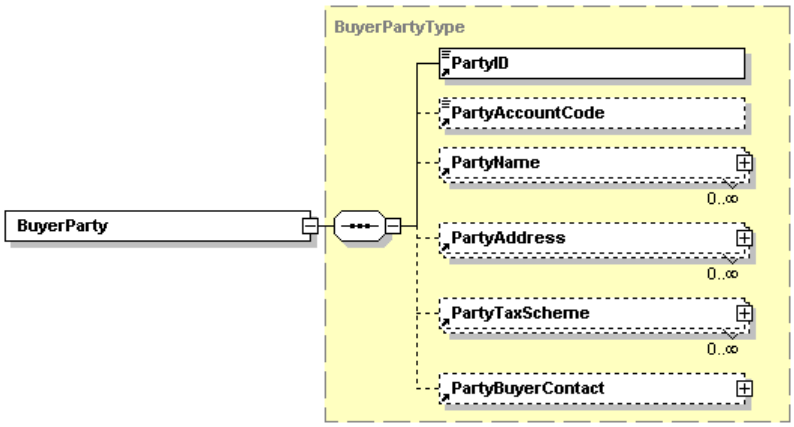
Scheme				Scheme	Scheme
Buyer_ Party. Buyer_ Contact	Buyer	Party	Buyer	Contact	Contact

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

2.1.2 Solution by using global declared elements

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

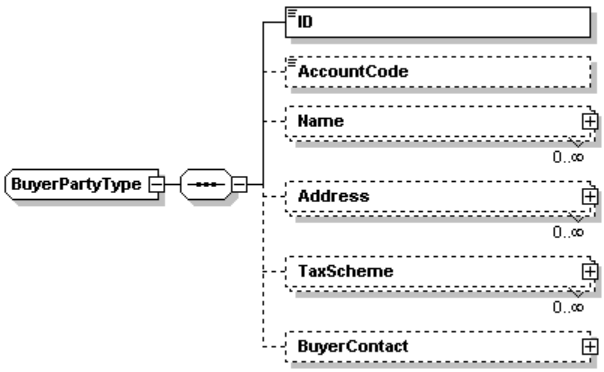
For example:



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

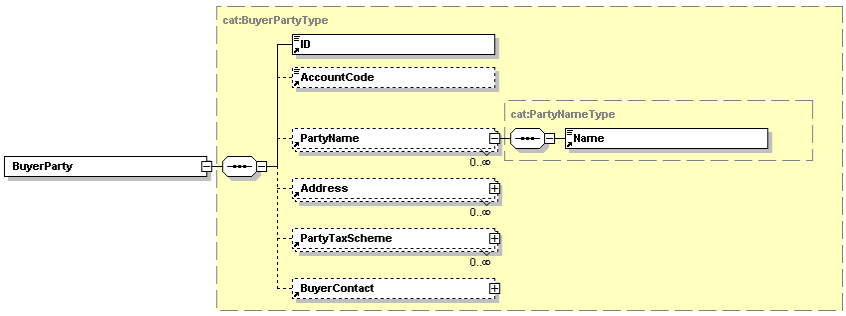
```
/BuyerParty/ID ➔ Buyer_ Party. Identification. Identifier

/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

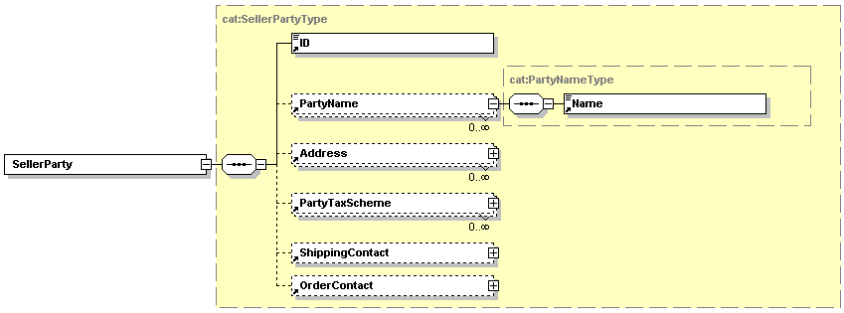
2.2.1 Problem

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

BuyerParty:

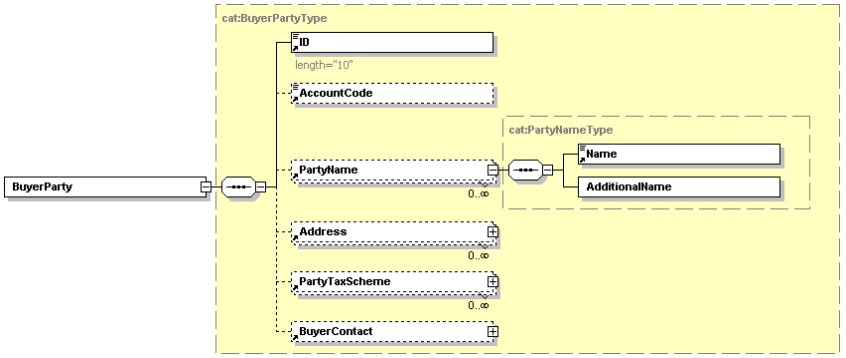


SellerParty:



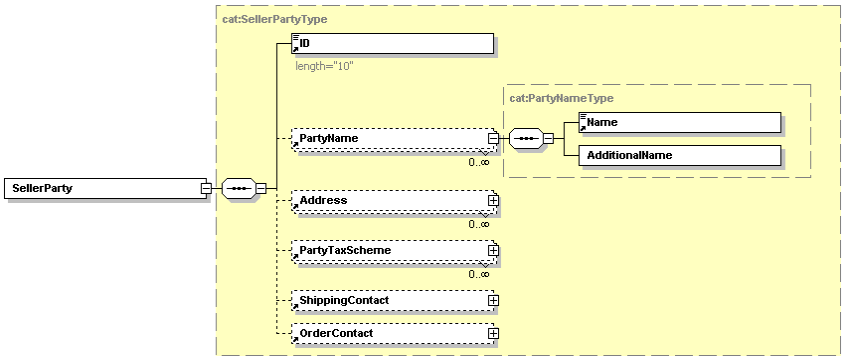
But what happens, if we would like to have some specific characteristics for ID or PartyName within the Aggregation “BuyerParty”? For example, the PartyName should have a child-element, like AdditionalName and the ID should be restricted in the maximum length.

Like:



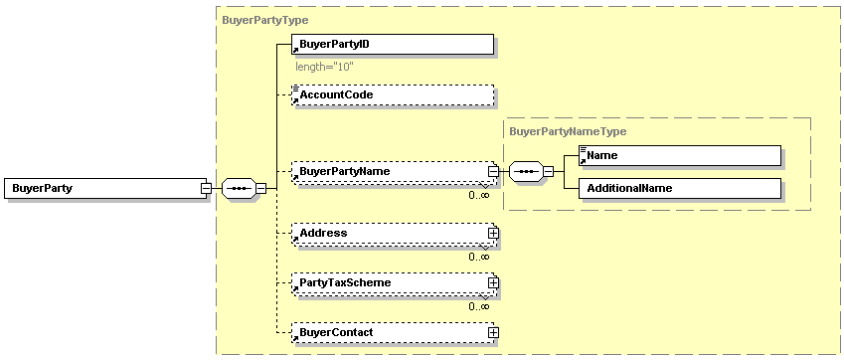
2.2.2 Solution by using global declared elements

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



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

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



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

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

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

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

```
BuyersCatalogueItemIdentificationItemMeasurement (35 Bytes)

SellersHandlingUnitDespatchLineDespatchedQuantity (49 Bytes)

ManufacturersHandlingUnitDespatchLineOrderLineID (48 Bytes)

ManufacturersTransportEquipmentRefrigerationStatusIndicator (59
Bytes)
```

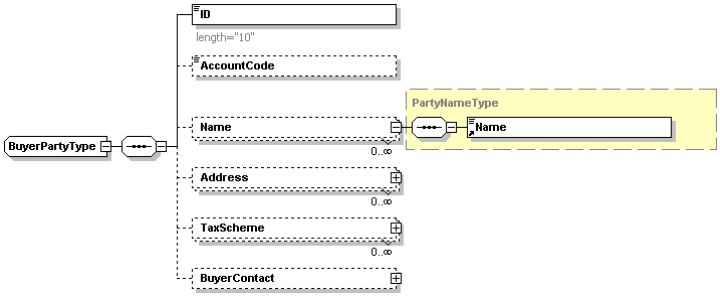
But we've to think about it:

- 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"/>
```

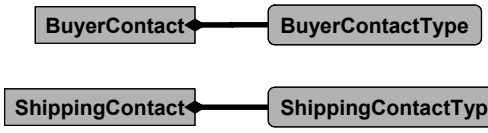
```
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="BuyerPartyNameType" id="UBL000397">
  <xsd:sequence>
    <xsd:element ref="Name" id="UBL000398"/>
    <xsd:element name="AdditionalName"/>
  </xsd:sequence>
</xsd:complexType>
```

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>00000000000000000000000000120321</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:

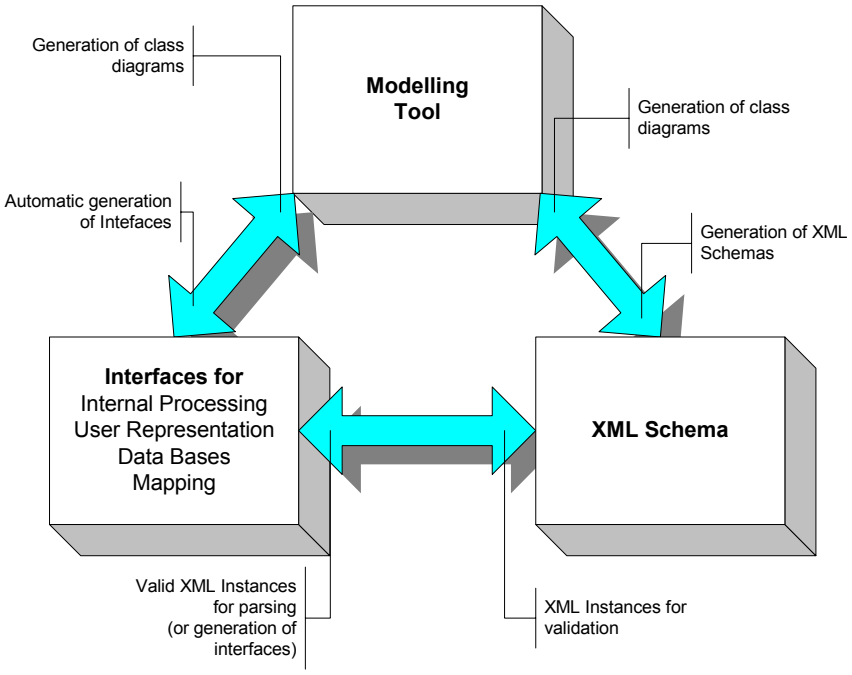
No knowledge in XML schema definition is necessary

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.

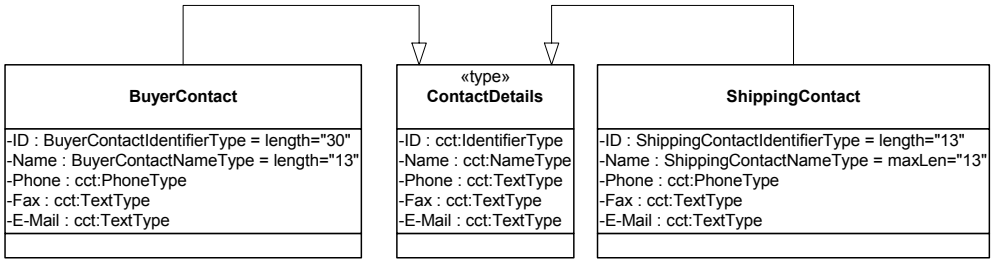
Therefore, it will be very narrow interfaces between modeling, xml and developing of interfaces, in future (see following picture).



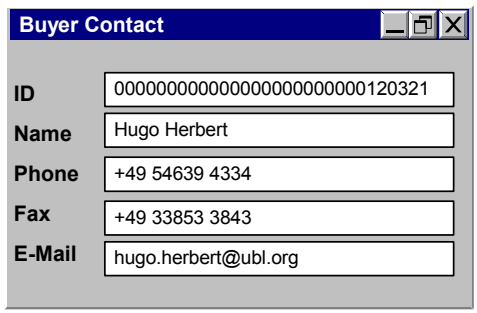
This is only possible, if all names and structure will be always consistent and have always the same meaning. This structures can be used in many times without any big effort.

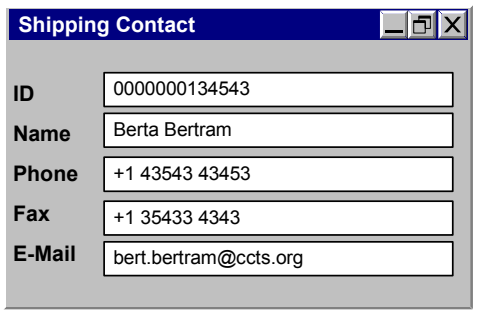
For example:

Object Oriented Modelling by Class Diagrams



Generating user interfaces

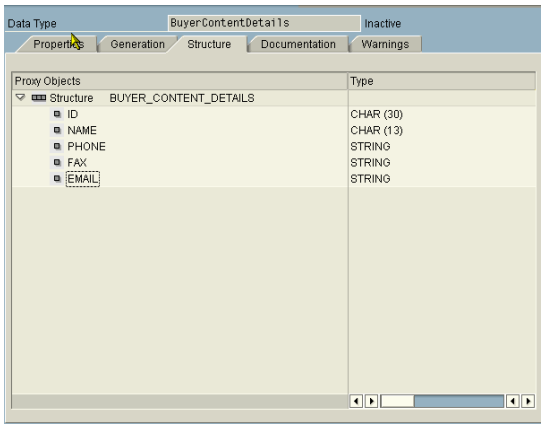


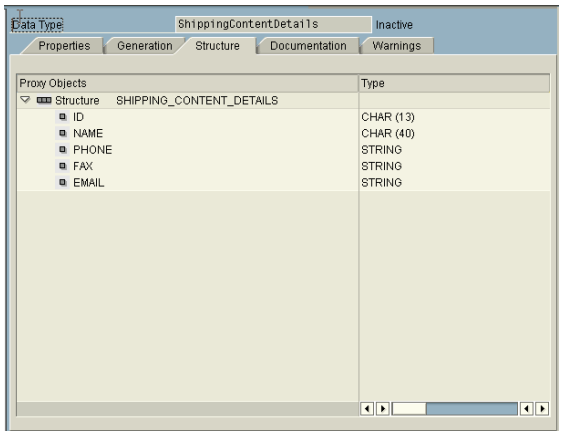


Generating of database tables

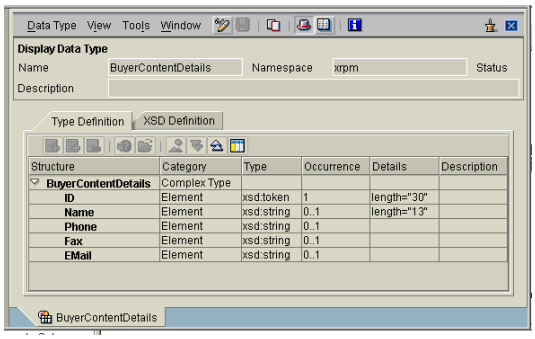
	Buyer	Shipping
ID	00000000000000000000000120321	0000000134543
Name	Hugo Herbert	Berta Bertram
Phone	+49 54639 4334	+1 43543 43453
Fax	+49 33853 3843	+1 35433 4343
E-Mail	hugo.herbert@ubl.org	bert.bertram@ccts.org

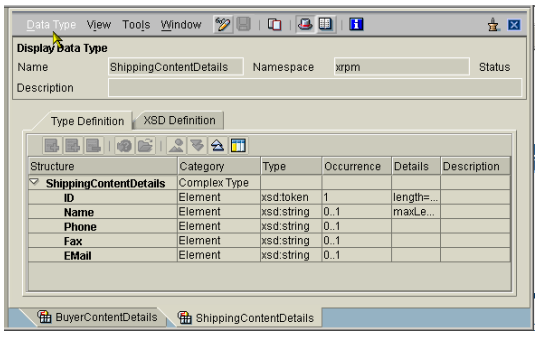
Generating ABAP-Objects for SAP development environment





Storing into a repository for providing and mapping to another dialects





Developing and/or generating interface applications

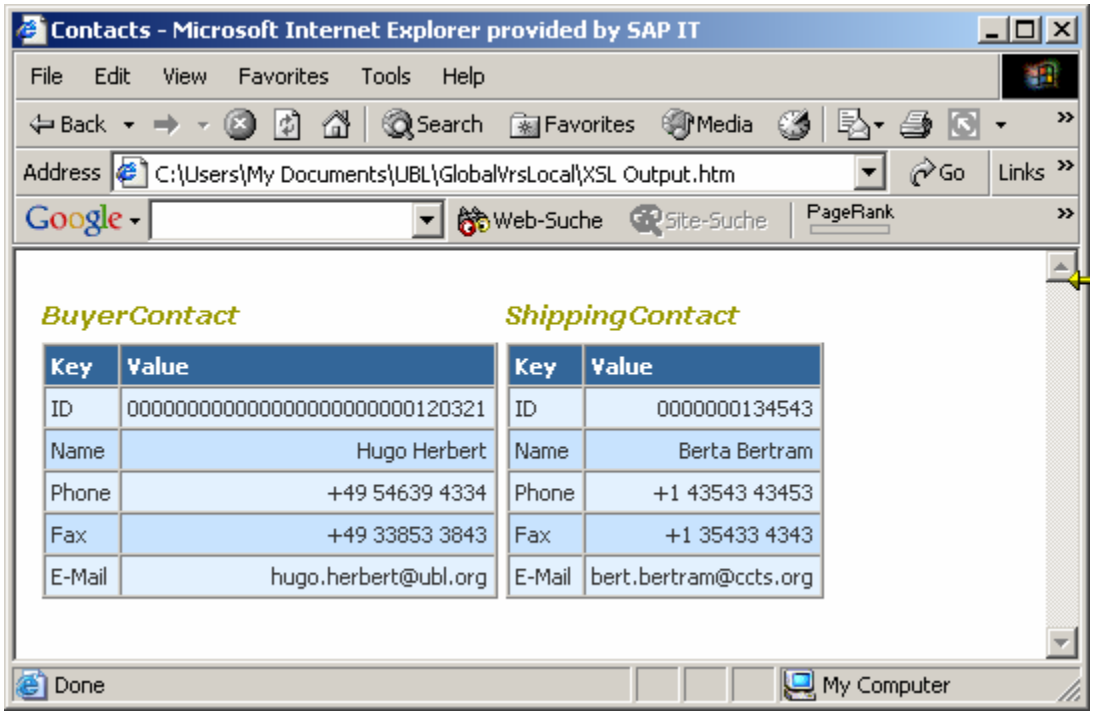
```
use XML::SimpleObject;

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

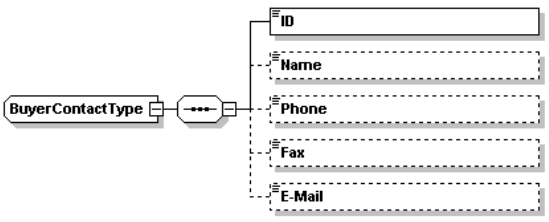
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY nbsp " ">
]>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:output method="html" indent="yes" encoding="UTF-8"/>
<xsl:template match="/">
  <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>
```

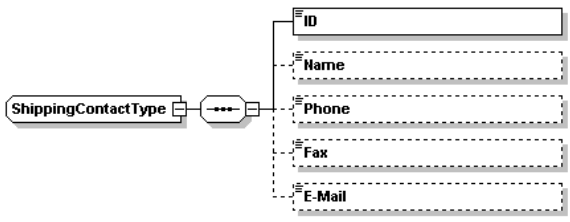
```
                        <xsl:otherwise>
                                <xsl:value-of select="'lightrow'"/>
                        </xsl:otherwise>
                    </xsl:choose>
                </xsl:attribute>
                <td>
                    <xsl:value-of select="name()"/>
                </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





All examples above base on the same schema structure of the ACC “ContactDetails”. It would be always used for “BuyerContactDetails” and “ShippingContactDetails” without additional mappings.

Only some applications (ABAP-Objects and database tables) need the restrictions of the length of the BIEs ID and name. Therefore it is necessary to define some additional complex types “BuyerContactIdentifierType”, “BuyerContactNameType”, “ShippingContactIdentifierType” and “ShippingContactNameType” with this restricitions. Because, this restrictions would be useful for the validation of XML instances and it is necessary for the automatic generation of ABAP Objects or database tables..

4 Recommedation

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

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

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

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.