

# 3 Schema Building Block Library for XML Interop Platform



**Organisations engaging in XML-based transactions** can now extensively rely on XML Schema [1][2] to define the format of XML documents exchanged in the transaction and to validate these documents. An XML Schema Definition (XSD) defines what documents the sender's organisation and system can generate as well as what documents the receiver's organisation and system must be able to process. It is possible, however, to write strictly or generally equivalent yet different XML Schema Definitions that define and validate the same or similar sets of XML documents. Organisations cannot afford to manage large quantities of redundant XML Schema Definitions authored ad libitum by staff, partners and consultants. Achieving effective reuse and therefore scalability is almost impossible if the definition of the same components take different forms in different XML Schema Definitions. This situation compels design guidelines and a design methodology

XML Interop Platform (XIP) [5][6] is a platform for the interoperation of XML-based trading systems. It offers a central depository of frequently used elementary schemas that allow e-business data types to be reused, therefore enabling swifter development and deployment of e-business trading systems. The schemas in this depository must be necessary - they must at least describe adequately the kind of real life data item. They must be sufficient and 'tight' - they must not allow more data than is in reality admissible as the kind of data item intended. It sounds simple enough, but requires discipline in the kind of schemas being admitted before we could safely use the schemas in the depository with peace of mind that we are describing exactly the kind of data intended.

This article describes how we can go about building such a central depository, how it can be utilised, and why the characteristics inherent in the kinds of schemas admitted enable the schema users to enjoy certain benefits had the schemas not been disciplined as suggested.

**Chin Chee Kai**  
Director, SoftML Pte Ltd  
Deputy Chairman, Information Exchange Technical Committee (IETC)  
Chairman, XML Working Group of IETC

## 1 A Central Depository of Schemas

The XML Interop Platform (XIP) v2 [6] is a trial platform to build up e-trading infrastructures which can interoperate with different standards; iron out standards implementation issues; and refine standards. To this end, building a central depository of schemas describing commonly used business information entities (BIEs) which are reusable by subsequent users would contribute a significant step towards both facilitating end-users implementing their e-trading systems and encouraging quicker and easier use of e-business standards. We assume here that we would be using the XML Schema (XSD) language [1][2] developed by the World-Wide Web Consortium (W3C) to describe data schemas in the central depository.

The idea of a central depository of schemas is not entirely a new idea. If one begins to break down business documents into identifiable data blocks from, say invoices and order forms, one could almost immediately see certain commonly used data blocks that beg loudly for reuse through perhaps a method similar to a shared library. The address structure of recipients and senders, for example, is common across almost all business documents if the address structure could be made sufficiently general. The invoice meta data such as invoice number and date format would not change from invoice to invoice and is certainly reusable. There are many other such block structures that can be 'factored' out and made reusable through the concept of a central depository of schemas. So the concept itself is not entirely revolutionary.

But if the central depository is made up of a mere collection of any sort of contributed schemas, be they from international standards bodies, local standards committees, or from generous organisations or individuals, the 'look and usage' of these schemas are likely to be as different as the number of contributed schemas. To get a feel of the kinds of variety that XSD offers for data structure descriptions, the reader is referred to [8]. We are not talking about the use or lack of a central index to make searching a desired schema faster. The awkwardness of such an implementation of central depository is more like this: Once a user finds a schema that looks somewhat similar to the sort of schema he or she intends to write and would like to reuse this schema, the amount of effort and time needed to read through and test the schema to ascertain its extent of data structure description may be as much as, or if not more than, the time he or she spent describing from scratch the kind of schema that is intended. If that happens, it would defeat the purpose of creating and maintaining such a central depository of reusable schemas in the first place.

So the bottom line guarding the usefulness of such a central depository of schema would be a sort of assurance that a potential depository user may assume about the nature and properties of the schemas from the depository. The level of utility of the central depository of schemas to a user would be how immediate the schemas may be used by a potential user. In the ideal case, a user who needs to include an address data field, for example, would just search for an address schema within the central depository and import it into his or her document schema. He or she would do so with the safe assumption that the address schema so imported would not 'accidentally' allow more than addresses to be validated when the actual data arrives. It would also be a safe assumption that if the actual data is a valid address, then the address schema would necessarily validate the actual data as admissible instead of rejecting it. In short, a useful schema within the central depository would describe the set of admissible XML [3] instances with just the necessary and sufficient boundaries as to carry through the intended business meaning. In other words and taking address schema as an example without loss of generality of the explanation, an address schema should describe only the format of permitted address data; no valid address data should be rejected by the address schema and no non-address data (which may still be valid XML instances) should be admitted as 'address data' by the address schema during validation.



Here, we distinguish between the semantics of data and the lexical form of the data. While the former could not usually be determined up-front at all, much less completely, by XSD, the latter can be fully described by XSD alone. A non-existent but otherwise properly written address in a Delivery Order document, for instance, would not be detected for its invalidity just by looking at the data field in the XML instance unless a database of addresses could be checked against. Likewise a seemingly incorrectly spelled company name, such as 'Trafalgarr Pte Ltd' would not by itself be spotted since it could possibly be a valid company name despite its somewhat funky look. Our focus here is more on using XSD to validate properly written BIEs such as addresses and names. Thus, both the above examples would be validated OK and submitted for a second phase semantics verification before the document's data is processed further.

Why would having a necessary and sufficient boundary on each schema within the central depository be important to the user? Other than the obvious 'doing it right' argument - that a BIE schema should only describe what it says it describes (for example, an address schema describes only what an address should look like) - there is also a XSD-technical aspect to where the boundary lies. To see why this should be important to the user of the central depository's schemas, we see that the easiest way for a user to utilise the central depository's schemas is to just `<xsd:import>`<sup>1</sup> whichever schema which appears proper to the user. But within a schema, there are `<xsd:element>`s and `<xsd:type>`s which can be defined, named, and subsequently referenced. The `<xsd:element>`s and `<xsd:type>`s may also be global or non-global, depending on how a schema designer writes them. To add to the variety, user could, if unadvised, reuse either the `<xsd:import>`ed `<xsd:element>`s or `<xsd:type>`s. This would seem, on first sight, easy to conclude that the necessary and sufficient boundary be bound upon both `<xsd:element>`s and `<xsd:type>`s. But this conclusion ignores the fact that a top-level XML instance data field can only be validated against a defined global `<xsd:element>`. The same top-level XML instance cannot be validated against a non-global `<xsd:element>`, nor against purely `<xsd:type>`s. More importantly, the same top-level XML instance may be validated against any globally named `<xsd:element>`s. The globalness and non-globalness of `<xsd:element>`s and `<xsd:type>`s within a schema have thus given rise to a few distinct style of schema designs.

We look at two very distinct and somewhat mutually exclusive ways of schema designs, the Venetian Blind style and the Garden of Eden style, in the next section where illustrations with examples would make the distinction clearer. There are other styles, such as Russian Doll and Salami Slice, which differ in their use of named and unnamed, as well as global or local, elements and types. The interested reader is referred to [4] for further insights.

However, we suggest here, without arguments for now, that in order to obtain the usage safety mentioned earlier, the schemas in the central depository should all be Venetian Blind styled without any definition of global `<xsd:element>`s. As for the global `<xsd:element>` which is necessary for data validation, it will be at the point when the user reuses schemas from the central depository to conjure his or her own document types where the user's document type is defined as the only global `<xsd:element>`. Traversing through the user's new document schema as well as all the `<xsd:import>`ed schemas from the central depository, we could find only that lone global `<xsd:element>` which the user writes as the only global `<xsd:element>` throughout. It is in this manner of user's schema construction that we can be assured that the necessary and sufficient boundary for the user's document instances is described.

<sup>1</sup> We use the common notation of prefixing XSD keywords with 'xsd': so that they are not confused with other references or examples.

## 2 Venetian Blind vs Garden of Eden

The XML Schema (XSD 1.0) is an XML-based language specially designed by the World-Wide Web Consortium (W3C) for users to describe desired data sets. The desired data sets constitute a set space consisting of valid XML instances with the necessary lexical structure and data that match the description of the schema. For instance, a user who intends to permit only all XML instances that begin with the body element <Phone> which contains only two sub-elements, namely country phone code <CountryCode> and the phone number <TelephoneNumber> may expect a sample data instance to look like:

```
<Phone xmlns="My Test">
  <CountryCode>65</CountryCode>
  <TelephoneNumber>62115352</TelephoneNumber>
</Phone>
```

Figure 1: XML data instance showing a compound piece of information called <Phone> made up of two more basic pieces of information called <CountryCode> and <TelephoneNumber>

where we have chosen to use the namespace 'MyTest' for illustrative purposes. So in order to compactly describe all the sample instances that may have this structure, we could devise a schema as follows:

```
<xsd:schema version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" targetNamespace="MyTest" xmlns="MyTest">
  <xsd:element name="Phone" type="PhoneType" />
  <xsd:complexType name="PhoneType">
    <xsd:sequence>
      <xsd:element name="CountryCode" type="CountryCodeType" />
      <xsd:element name="TelephoneNumber" type="TelephoneNumberType" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="CountryCodeType">
    <xsd:restriction base="xsd:nonNegativeInteger">
      <xsd:totalDigits value="4" />
      <xsd:fractionDigits value="0" />
      <xsd:minInclusive value="1" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="TelephoneNumberType">
    <xsd:restriction base="xsd:nonNegativeInteger">
      <xsd:totalDigits value="16" />
      <xsd:fractionDigits value="0" />
      <xsd:minInclusive value="1" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Figure 2: A schema written in the 'Venetian Blind' style to describe all data instances that 'look like' the data instance shown in Figure 1



Here, we see that the document element <PhoneType> is being defined as a global element by placing it at the immediate child under <xsd:schema>. It is also highlighted in blue to show its global scope. This is, however, not the case for other elements such as <CountryCode> and <TelephoneNumber> which are being declared within the complex type <PhoneType>. This gives them a local scope, where any element not defined as an immediate child of <xsd:schema> is to be given a local scope.

On the other hand, all the types, including <PhoneType>, <CountryCodeType>, and <TelephoneNumberType>, are declared global, and are marked by yellow backgrounds to highlight their global scope.

This style of schema writing, where only the main document element, if applicable, is defined global, all other elements local, and all types global, is known as the 'Venetian Blind' (VB) style of schema design.

As is known about XSD, we can create another schema in which all elements, in addition to all types, are defined global. This is illustrated as follows:

```
<xsd:schema version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" targetNamespace="MyTest" xmlns="MyTest">
  <xsd:element name="Phone" type="PhoneType" />
  <xsd:complexType name="PhoneType">
    <xsd:sequence>
      <xsd:element ref="CountryCode" />
      <xsd:element ref="TelephoneNumber" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="CountryCode" type="CountryCodeType" />
  <xsd:simpleType name="CountryCodeType">
    <xsd:restriction base="xsd:nonNegativeInteger">
      <xsd:totalDigits value="4" />
      <xsd:fractionDigits value="0" />
      <xsd:minInclusive value="1" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="TelephoneNumber" type="TelephoneNumberType" />
  <xsd:simpleType name="TelephoneNumberType">
    <xsd:restriction base="xsd:nonNegativeInteger">
      <xsd:totalDigits value="16" />
      <xsd:fractionDigits value="0" />
      <xsd:minInclusive value="1" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Figure 2: Another schema written in what is called a 'Garden of Eden' style, yet describing the same set of data instances which schema in Figure 1 describes

This style of writing, where all elements and types can be referenced or 'picked' anywhere by virtue of their global scoping, is called the 'Garden of Eden' (GoE) style of schema design, taking cue from its resemblance to a garden of elements and types blossoming within the schema.

The Universal Business Language (UBL) v1.0 (cd2) [9] as well as the upcoming UBL v2.0 from OASIS are choosing this style of schema design. Amidst the many advantages, justifications and celebrated arguments on why this approach is most suitable for UBL, we will illustrate in the subsequent sections some drawbacks that may be important to a number of users, as well as contrast the 'Venetian Blind' approach with 'Garden of Eden'.

### 3 Garden of Eden (GoE) in UBL

UBL [9] has decided that all their elements and types would be global. The decision was made after careful considerations, but is not convincingly final. Some history of developments within UBL reveals that they did stick with the Venetian Blind approach a while [4]. In any case, UBL continues this decision as they move from v1.0 to v2.0.

The way the schemas are organised within UBL v1.0 looks as follows:

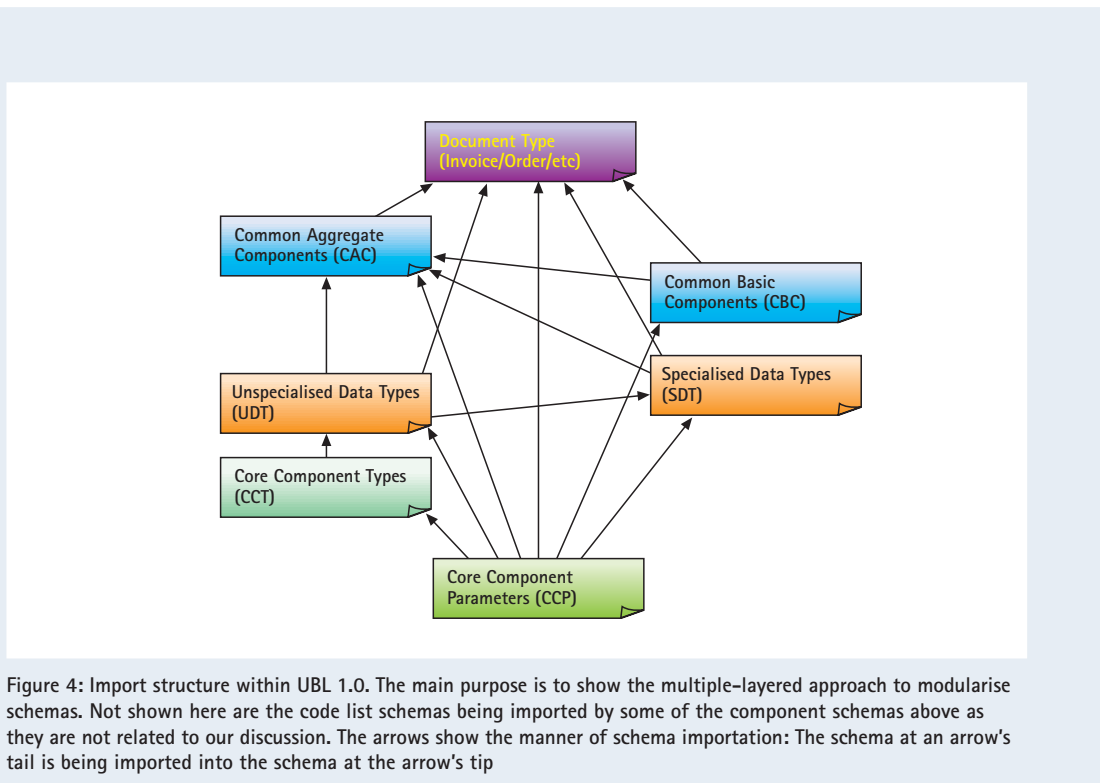


Figure 4: Import structure within UBL 1.0. The main purpose is to show the multiple-layered approach to modularise schemas. Not shown here are the code list schemas being imported by some of the component schemas above as they are not related to our discussion. The arrows show the manner of schema importation: The schema at an arrow's tail is being imported into the schema at the arrow's tip

This modularisation of schema components is a useful property. However, it has not been fully carried through because there are a lot of types and elements being defined in each of CAC, CBC, UDT and CCT. This makes the selective use of individual types difficult and cumbersome; user either imports all the types defined in CAC, for example, or else none.

But that is not the main difficulty encountered. Due to UBL's decision to define all elements global, every BIE type defined has an associated global element that references that BIE type. This, on first look, seems to make hundreds of elements globally and easily accessible to any one who wishes to reuse any of them. In a way, because of the import dependencies shown above, a user just need to import CAC into his or her document schema and



the subsequent dependencies will automatically instruct the schema validator to load all the CBC, UDT, SDT, CCT and CCP schemas. But when this happens, all the supporting and internal global elements are being exposed to the user's document schema. Supporting global elements, such as <cbc:Address> and <cbc:Floor> just to name two out of the hundreds available, are silently pulled into user's schema. Similarly, what should be UBL's internal global elements, such as <ccp:Component> and <ccp:Definition>, all get pulled into user's schema as well.

Now suppose the user's schema is just one of the UBL's main document, such as the Invoice Document within the purple document box on the top-most of Figure 4. By importing the CAC schema, all the supporting and internal global elements are also imported into the purple document box. This exposes those UBL global elements, all of which are unrelated to what the user wishes to describe, as potentially valid top-level XML instances.

Herein lies the accidental yet silent breach of the data structure description boundary mentioned in earlier sections. Instead of describing the user's Invoice data type as the only acceptable data instances, the user ends up with a huge logical schema (built from all the imported schemas) that will readily admit data instances whose top-level XML element begins with any of the now-exposed UBL global elements. In other words, if the user naively uses the purple top schema thinking, in false safety, that he or she would see only those intended Invoice data types be accepted by the schema validator against this purple top schema, the user will be in for a rather unpleasant surprise. Some illustrative examples of this are shown below.

#### 4 Examples of Overly-Lax Validation due to GoE

Consider the following incoming XML instance for schema validation:

```
<?xml version="1.0" encoding="UTF-8"?>
<CountryCode xmlns="MyTest" >65</CountryCode>
```

Figure 5: An incoming XML instance that is not really what the user expects to be a <Phone> data type, but nevertheless would get validated OK if user had used the GoE schema in Figure 3

Using the <Phone> example described earlier, it hardly seems surprising that if the user uses the GoE schema in Figure 3 to validate this incoming XML instance, the schema validator would pass this instance as OK, after all the GoE schema did define <CountryCode> as a valid global element. Yet, the user clearly could not proceed further with this piece of document since no phone number is present at all. Had the user used the almost equivalent VB schema in Figure 2 to validate this same piece of incoming document instead, the result would have been a rejection by the schema validator, since the incoming data clearly does not match the only global element present in the VB schema.

Consider some other UBL examples that illustrate this aspect as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<cac:OrderReference
  xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-1.0"
  xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:specification:ubl:schema:xsd:Invoice-1.0
    ../../xsd/maindoc/UBL-Invoice-1.0.xsd">
  <cac:BuyersID>20031234-1</cac:BuyersID>
  <cac:SellersID>154135798</cac:SellersID>
  <cbc:IssueDate>2003-02-02</cbc:IssueDate>
</cac:OrderReference>
```

Figure 6: An example of an incoming 'Invoice' data document that expects to be validated as a valid UBL Invoice document. Due to the global elements present in UBL schemas, this odd-looking 'Invoice' data document would be validated as OK by schema validators against the UBL Invoice schema

Now we take a look at the example in Figure 6. This example looks bizarre enough to be claimed as a valid UBL Invoice document. But if we validate it against the UBL's Invoice document schema, the schema validator would proclaim it as OK. A few more of such examples below would show further the extent of the lax in allowing non-Invoice instances to be accepted as OK.

```
<?xml version="1.0" encoding="UTF-8"?>
<cac:BuyerParty
  xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-1.0"
  xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:specification:ubl:schema:xsd:Invoice-1.0
    ../../xsd/maindoc/UBL-Invoice-1.0.xsd">
  <cac:Party>
    <cac:PartyName>
      <cbc:Name>Bills Microdevices</cbc:Name>
    </cac:PartyName>
    <cac:Address>
      <cbc:StreetName>Spring St.</cbc:StreetName>
      <cbc:BuildingNumber>413</cbc:BuildingNumber>
      <cbc:CityName>Elgin</cbc:CityName>
      <cbc:PostalZone>60123</cbc:PostalZone>
      <cac:CountrySubentityCode>IL</cac:CountrySubentityCode>
    </cac:Address>
  </cac:Party>
</cac:BuyerParty>
```

Figure 7: A 'BuyerParty' data fragment is now a candidate for validating against the UBL Invoice document schema. The outcome of validation would be an accepted 'Invoice' data instance.



```

<?xml version="1.0" encoding="UTF-8"?>
<cbc:IssueDate
xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:specification:ubl:schema:xsd:Invoice-1.0
../Xsd/maindoc/UBL-Invoice-1.0.xsd">
2003-03-14
</cbc:IssueDate>

```

Figure 8: An 'IssueDate' data fragment is now a candidate for validating against the UBL Invoice document schema. The outcome of validation would be an accepted 'Invoice' data instance.

```

<?xml version="1.0" encoding="UTF-8"?>
<ccts:Component
xmlns:ccts="urn:oasis:names:specification:ubl:schema:xsd:CoreComponentParameters-1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oasis:names:specification:ubl:schema:xsd:Invoice-1.0"
xsi:schemaLocation="urn:oasis:names:specification:ubl:schema:xsd:Invoice-1.0
../Xsd/maindoc/UBL-Invoice-1.0.xsd"
>
  <ccts:ComponentType>BBIE</ccts:ComponentType>
  <ccts:DictionaryEntryName>Address Line. Line. Text</ccts:DictionaryEntryName>
  <ccts:Definition>An address line of unstructured text intended for use only by systems
incapable of providing structured or fully structured addresses</ccts:Definition>
  <ccts:Cardinality>1..7</ccts:Cardinality>
  <ccts:ObjectClass>Address Line</ccts:ObjectClass>
  <ccts:PropertyTerm>Line</ccts:PropertyTerm>
  <ccts:RepresentationTerm>Text</ccts:RepresentationTerm>
  <ccts:DataType>Text. Type</ccts:DataType>
  <ccts:Examples>&quot;123 Standard Chartered Tower&quot;</ccts:Examples>
</ccts:Component>

```

Figure 9: The <ccts:Component> data fragment is originally used within UBL's schemas to document a BIE's properties and other meta data. But due to the GoE style adopted by UBL, the global element makes it possible for this instance to validate as OK against the UBL Invoice document schema. The outcome of validation would be an accepted 'Invoice' data instance.

The drawbacks are not only the breakage of safety assumption that importing a UBL's type in a user schema means permitting the validation of only that imported UBL type, not just the apparent awkward absurdity that various top-level data instances which bear no relationship to being an invoice document could somehow be validated as acceptable, but also the inflow of invalid data instances further into some users' systems which rely heavily on schema validations. While it might not likely paralyse the user's processing system, it certainly introduces erroneous data instances further into the processing system before recovery could take place when such delayed error recovery need not have occurred in the first place.

For all its benefits, the existing version of XSD specification unfortunately ties elements that can be referenced with the elements that are global, and the elements that are global with the elements being one of the possible choices of data instances' top-level element name. We seem to end up with the impossibility of getting only a part of the side effects – that of only defining and reusing global elements without allowing it to become one of the possible choices of data instances' top-level element name. There may not be a good chance happening,

but until a revision of the XSD specification to separate the global characteristic of an element with its ability to allow data instances' top-level element name to be explicitly and separately defined, we have to adapt our usage of XSD by avoiding the use of global elements altogether and elect to use a type-only XSD description.

## 5 Type-Only Data Structure Description

In XIP, having observed and compared some of the differences in behaviours of schema validators on VB & GoE schemas, we would need to be more cautious in terms of importing verbatim schemas from international standards bodies such as OASIS-UBL into an XIP central depository. To ensure that user's schemas reusing schemas from the central depository would not 'leak' into accepting unrelated global elements as valid data instances, it would be proper to enforce a discipline to exclude any global element definition in any schema stored within the central depository.

This would result in type-only schemas being stored and reused. Even though these types may be defined as global types throughout for easier references, their presence in the global space would not interfere with user's decision to accept any sort of data instances as valid. When we do this consistently throughout, we would basically be requiring central depository schemas to be VB-styled less any global element definition.

## 6 How User Document Schema Can Be Constructed

The XIP central depository of schemas would only be useful if it facilitates users in constructing and validating the kinds of data sets the users desire. Thus, when the user needs to create a new document schema to validate his or her particular application's data instances, what he or she needs to do is simply to create a new schema, import all the schema types from the XIP central depository which apply to the application, and finally define a single global element in his or her schema to serve as the document root. The construction is illustrated below:

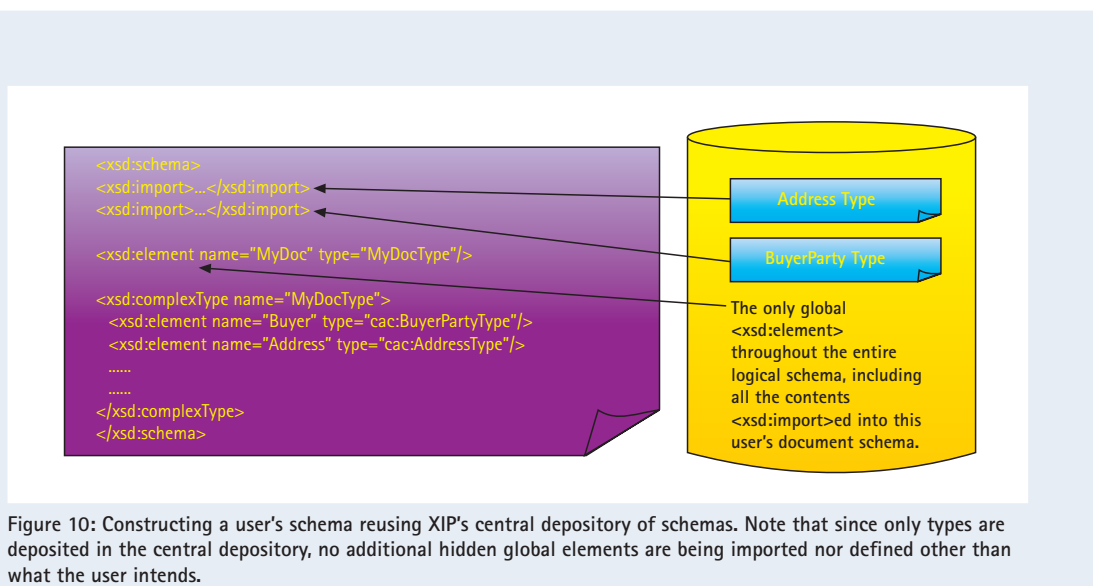


Figure 10: Constructing a user's schema reusing XIP's central depository of schemas. Note that since only types are deposited in the central depository, no additional hidden global elements are being imported nor defined other than what the user intends.

In the example illustrated in Figure 10, the user's 'MyDocType' would describe all the data instances that bear the structure described in the <xsd:complexType> shown. In particular, it would begin with a <Buyer>, followed by an <Address> and so on. Any incoming data instance set for validation must necessarily contain and only contain <MyDoc> as the root element; no other root element would be accepted as valid by the schema validator.



The manner of selecting from the central depository of schemas what type is 'most suitable' for a particular user's application is most likely a mix of skills and art. However, it is possible to systematically follow a schema type selection criterion and arrive at a very good, if not the most optimal, choice of reusable types from the central depository of schemas. Article [7] suggests an atomic model in treating the type schemas. By having the user decide at what granularity his or her BIE is, and using that level of granularity as an atomic guide (because user's granularity cannot be broken down further by user's definition) to select the closest fitting schema from the central depository, it is possible to construct practical and usable schemas.

## 7 Other Considerations of Central Depository

To accept into the XIP central depository only VB-styled schemas without any global element definitions is only one aspect of constructing a useful central depository. There are a number of other aspects, such as managing the namespace values, choice of using elements or attributes, naming of elements and attributes (if used), versioning, file storage structures, type naming and so on which would all influence the extent of ease in maintaining the schemas, reusing the schemas, upgrading the schemas, and their gracefulness in allowing users to upgrade from an older type schema to a newer schema of the same type with minimal necessity to migrate their data from old representation to new representation.

To go into all these discussions would entail the examination of what is known as the Naming and Design Rules (NDR) [10] in UBL. UBL Technical Committee (TC) has started a good effort in coming up with guidance in these areas in an attempt to make uniform all the schemas that are generated. The NDR has helped ensure that even though volunteers participating in UBL TC come with all kinds of ideas about what style is the best schema design style, eventually the normative schemas that are generated by the TC would still look and feel consistent and uniform.

On the aspect of global element or otherwise, the UBL's NDR 1.0 [10] requires all elements to be defined global. This seems to be the case for UBL's NDR 2.0 as well. The XIP central depository's NDR would most likely differ in this aspect by forbidding any definition of global element for all schemas to be stored in the central depository. There is also a need to examine various other aspects of XSD schema design guidelines and suggestions in UBL NDR so that the aspects discussed in NDR would accommodate requirements in XIP thereby improving the level of utility of schemas within XIP and those of users' organisations.

## 8 Summary and Conclusion

In XIP, we seek to build a central depository of schemas that would allow users to reuse already defined BIE types, be they incorporated from international standards bodies, or created locally due to specific local needs. Because of the emphasis to store only types and no global element (the Venetian Blind style) in schemas within the central depository, we expect to give users the safe assumption that the imported schemas from the central depository would not introduce unwanted data instances due to unwanted global elements. This requirement differs from those of UBL 1.0 [9], which requires all schema elements to be defined global (the Garden of Eden style). As the latter introduces acceptance of unrelated top-level data instances, we have to deviate from the way UBL advises and adopt UBL BIE types into XIP's central depository by stripping off all the global element definitions.

With the existing XSD specification, it is only by enforcing a discipline of no-global-element can we reliably assure the central depository users that no extraneous data instances could be introduced into users' set of acceptable

data instances, and that whatever global elements users see in their own schemas will be what they get from data instances.

## 9 Acknowledgements

The author acknowledges the useful availability of UBL 1.0cd2 archive by UBL TC of OASIS, as well as the on-list and off-list discussions with people subscribed to the ubl-dev mailing list. The author thanks the generous editorial suggestions and proof-reading given by Dr Stephane Bressan from NUS, as well as Dr Lee Eng Wah from SIMTech and Chair of Information Exchange Technical Committee of ITSC. The author is also grateful to The Editor and his/her Editorial Team for the patience and help in moving this article through the whole editorial process. Finalising the article would have been much more difficult had it not been the good preparation work done by the ITSC Secretariat to whom the author is much grateful for the help rendered in the entire submission process.

## 10 References

- [1] XML Schema Part 1: Structures. W3C Recommendation 2 May 2001. Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn. <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>
- [2] XML Schema Part 2: Datatypes. W3C Recommendation 2 May 2001. Paul V. Biron, Ashok Malhotra. <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>
- [3] Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation 4 February 2004. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau. <http://www.w3.org/TR/REC-xml/>
- [4] Eve Maler, Schema Design Rules for UBL ... and Maybe for You. XML Conference & Exposition 2002. [http://www.idealliance.org/papers/xml02/dx\\_xml02/papers/05-01-02/05-01-02.html](http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-01-02/05-01-02.html)
- [5] XML Interop Platform (v1.0), Information Technology Standards Committee 2002. [http://www.itsc.org.sg/downloads/xip/xip\\_index.html](http://www.itsc.org.sg/downloads/xip/xip_index.html)
- [6] XIPv2.0 - e-Business Interoperability for Large & Small Business, Information Technology Standards Committee Synthesis 2005. [http://www.itsc.org.sg/synthesis/2005/2\\_XIPv2.0.pdf](http://www.itsc.org.sg/synthesis/2005/2_XIPv2.0.pdf)
- [7] Tapping Standards from Nature - Atomic Model for Creating Electronic Message Schemas, Information Technology Standards Committee Synthesis 2004. [http://www.itsc.org.sg/synthesis/2004/4\\_TappingStd.pdf](http://www.itsc.org.sg/synthesis/2004/4_TappingStd.pdf)
- [8] Compound XML document profiles for rich content: Part 1: Exploring extensibility alternatives using XML Schema, Steve Speicher, Kevin E. Kelly. 13 September 2005. IBM. <http://www-128.ibm.com/developerworks/xml/library/x-cxdp1/>
- [9] Universal Business Language (UBL) 1.0, OASIS Standard, 15 September 2004. OASIS. <http://docs.oasis-open.org/ubl/cd-UBL-1.0/>
- [10] Universal Business Language (UBL) Naming & Design Rules (NDR) v1.0, OASIS Standard, 15 November 2004. OASIS. <http://www.oasis-open.org/committees/download.php/10323/cd-UBL-NDR-1.0Rev1c.pdf>

