Schema and WSDL Design Checklist

Quality Assurance

Version: ***1.0 Final***
Date: ***27/05/2008***
Distribution: ***Process Improvement***

CHANGE HISTORY

Date	Version	Change(s) Incorporated
03/12/2007	1.0 DraftA	New document
04/04/2008	1.0 DraftB	Added recommendation for 1) ID/IDREF to use form=”unqualified”, 2) Character encoding should be UTF-8, Qualified the use of xsd:choice, Added a section on the Data Binding Toolset comparison report which demonstrates some the issues this checklist guards against, Spelling errors corrected, Inconsistent schema examples in appendix corrected, Added a new section for constraining facets.
23/04/2008	1.0 DraftC	1.Removed restriction on use of xsd:date and xsd:dateTime. 2. General syntax corrections. 3. Use of venetian blind schema design pattern moved to assumptions section. 4. Added reference to RFC2119 MUST/SHOULD/MAY and used throughout the document. 5.Included more examples in the document. 6.Added attribute coverage to <xsd:documentation> recommendations. 7.Included an example which shows an alternative to using of xsd:positiveInteger. 8.Added a section on extension and restriction to the Do’s part of the document. 9.Added check for nillable – to ensure clarity where nillable was intended.
27/05/2008	1.0. Final	Add gMonthDay to section 2.5.4

TABLE OF CONTENTS

1 INTRODUCTION

As part of the deliverables in an SOA SOAP Based Web Service environment, schema and WSDL play an important part in automating the process of code generation in service implementations.

Typically Product Providers will take the schema and WSDL and use data binding tools to generate skeleton service implementation code which manages the marshalling of data between the XML pay load and native programming language constructs. These resulting constructs can then be processed by existing business logic.

Similarly, consumers of these services (for example Portals and Intermediaries), will take the same schema and WSDL and use data binding tools to generate client implementation code to access the service provided by a Product Provider.

For this reason it is necessary to ensure that any schema and WSDL which is created can be consumed by the majority of the state of the art data binding toolsets used to create Web Services based on the contracts defined by WSDL.

It should be noted that whilst data binding tools manage the marshalling of data between the XML pay load and programming language constructs, many of these tools will not provide full schema validation for the pay load.

This document contains a checklist of recommendations which, if followed, will ensure those implementing our schemas will have a good experience with Web Service tools.

Also included in this document are several recommendations which have nothing to do with the data binding issues but are simply good practice in schema design.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [10].

1.1 W3C XML Schema Patterns for Data Binding Working Group

To help meet this requirement Origo joined the W3C XML Schema Patterns For Data Binding Working Group (http://www.w3.org/2002/ws/databinding/) whose remit was to

1. Identify a set of basic schema patterns which are well supported by the majority of state of the art data binding tools.
2. Identify those advanced schema patterns which are sometimes problematic with state of the art data binding tools.
3. Document the results of a test suite which involved applying a set of schema patterns to the most of the popular data binding tools.
4. Create a patterns detection service which analyses a schema or WSDL and reports how well it conforms to the basic and advanced schema patterns.
5. Aim to encourage vendors to improve data binding tools they produce.

The first four points have been achieved successfully. However with lack of support from vendors the final point has had limited success.

Data binding tools aside, a lot of the items in this checklist are common sense. XML Schema does provide a very rich set of functions to achieve message design. With so many possible ways of defining content structure, it makes good sense to define a subset of the XML Schema functionality to use for reasons of

- consistency;
- interoperability;
- ease of understanding;
- and ease of maintenance.

Effectively this checklist provides a profile of schema functionality to be used to achieve the benefits above.

1.2 Assumptions

The following assumptions have been made during the design of this checklist document.

1. W3C XML Schema is used to define schema content.

2. This document is intended for use by those designing schemas representing message structures exchanged over a SOAP Based Web Services environment with WSDL used to define the Web Service interfaces.

3. Schemas are assumed to represent one particular type of message structure, for example either a request or response message.

4. Where possible, schema complex types and simple types will be based on the UK Governments schema library with adequate documentation to explain the use of the Government standards (http://www.govtalk.gov.uk/schemasstandards/xmlschema.asp).

5. Schema design is modelled in UML initially prior to automatic production of the first draft physical schema document. This checklist is intended for use in the process of automating the creation of the final draft physical schema to ensure it is fit for purpose.

6. Use of the “Venetian Blind” schema design pattern is assumed. See Appendix A – Schema Design Patterns for more information.

2 DESIGN CHECKLIST

This checklist is the product of what has been learned from the data binding exercise, several years of schema design and input from Origo sponsors.

The purpose of the checklist is to help ensure a good quality design of schema and WSDL and to ensure it is easy to automate generation of usable source code from the WSDL or schemas for SOAP based Web Service implementations.

The checklist consists of a set of rules ("do's" and "don'ts") which try to ensure good schema/WSDL design and conformance to the Basic XML Schema Patterns [2] for data binding specification (where this is not possible, conformance to the Advanced XML Schema Patterns [3] for data binding specification should be attempted).

From a data binding toolset perspective this table shows the impact of ignoring recommendations documented in this checklist.

Impact on data binding	Meaning
None	No impact on data binding tools
Possible	Possible impact on data binding tools
High	High impact on data binding tools

Each item in the checklist will be marked with an impact level which indicates what the impact on data binding toolsets will be of ignoring that particular recommendation and possibly causing a "data binding toolset problem".

Definition of a "data binding toolset problem"

A "data binding toolset problem" can be defined as the situation where either

- Code generation will fail by producing code that will not compile or
- A schema structure being misrepresented in the host programming language.

2.1 Schema Design Patterns

As stated in the Assumptions section above, the use of the "Venetian Blind" schema design pattern is assumed. See Appendix A – Schema Design Patterns for more information.

The "Venetian Blind" schema design pattern uses a single global element to contain the payload (typically called the request or response element). This approach describes a modular way of naming and defining all type definitions globally. Each globally defined type describes an individual "slat" and can be reused by other components.

In addition, all the locally declared elements can be namespace qualified or namespace unqualified (the slats can be "opened" or "closed") depending on the "elementFormDefault" attribute setting at the top of the schema.

It is recommended that namespaces are qualified so that the local elements in the instance document must be qualified with the prefix of the namespace (see below).

2.2 WSDL Design

WSDL design should comply with the profiles specified by the Web Services Interoperability Organisation (WS-I) in particular the WS-I Basic Profile [4]. Specifics are included in the checklist where relevant.

2.3 Data Binding Toolset Comparison Report

The W3C XML Schema Patterns for Data Binding Working Group has created a report which compares how various state of the art data binding tools compare in their support for a set of XML Schema Patterns.

For those who wish to see how a particular data binding toolset copes with a particular schema pattern, a report is available which details the results of tests performed against each toolset.

The report does not include every toolset in existence but does represent most of the data binding implementations that currently exist.

The toolset report is available at http://www.w3.org/2002/ws/databinding/edcopy/report/all.html [6]

2.4 Schema/WSDL Design Rules – Do's

2.4.1 Character Encoding

Recommendation	Reason	Impact if ignored
Schema and WSDL **character encoding SHOULD** be specified as **UTF-8.**	UTF-8 and UTF-16 are recommended by the WS-I in their WS-I Basic Profile [4]. As Origo schema and WSDL will consist mainly of the ASCII character set it is safe to assume that UTF-8 will be sufficient for the purpose.	**Possible**

2.4.2 Namespaces

Recommendation	Reason	Impact if ignored
elementFormDefault and **attributeFormDefault SHOULD** be set to **“qualified”.**	For namespace clarity, it implies that instance documents must specify namespace prefixes for all components.	**Possible**
ID/IDREF attributes SHOULD be specified as form=”unqualified”. This is the only exception to the recommendation above.	Section 4.3 of the Origo Namespace Policy document [5] states that ID/IDREF attributes should be the only case where it is not appropriate to use namespace qualification.	**None**
W3C XML Schema namespace SHOULD be qualified with a prefix of **xsd.**	For consistency and ease of recognition of W3C XML Schema data types.	**None**
Each schema **SHOULD** be self contained with **all referenced complex and simple types defined locally** within the schema file.	If you do need to use include or import to share data definitions then ensure that the entity being referenced is stored in the same location as the entity from which it is included. This will maximise the chances of data binding tools working correctly.	**Possible**
Each WSDL **SHOULD** reuse schema definitions via the **import** option.	For interoperability it is advisable (see WS-I Basic Profile [4]) to import schemas into WSDL documents.	**High**
Each schema SHOULD be named in such a way that it reflects its functionality and part in a message exchange.	For clarity and ease of understanding. E.g. ProvideProductListRequest.xsd	**None**
Each schema SHOULD have a **unique namespace** which reflects both the **functionality and version status** of the schema**.**	This is necessary because data binding tools typically use the namespace URL when managing source code packaging and therefore require unique names. E.g. http://www.origostandards.com/schema/productlist/v1.0/ProvideProductListRequest	**High**

The TargetNamespace SHOULD be specified and SHOULD be the same as the default namespace.	For namespace clarity. The default namespace need not be specified if using elementFormDefault and attributeFormDefault with value of “qualified”. If the default namespace is specified then it should be the same as the TargetNamespace. The namespace also includes the schema version status which allows instances to be related to the appropriate schema version.	**Possible**

2.4.3 Naming Conventions

Recommendation	Reason	Impact if ignored
Authors SHOULD use meaningful element, attribute and **type names.**	E.g. complexType: ProductProviderDetails and element: product_provider_details.	**Possible**

Authors SHOULD ensure that complex and simple types start with upper case and element names start with lower case.
 	
The only exception to this is where the type has been taken directly from existing Government Schema Standards documented on the govtalk web site (http://www.govtalk.gov.uk/schemasstandards/xmlschema.asp) where type names should be preserved in the Origo schemas.
	For a consistent approach to naming of components. E.g.
	
	`<xsd:simpleType name="GeographicRegion">`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:enumeration value="England"/>`
	`  <xsd:enumeration value="Scotland"/>`
	`  <xsd:enumeration value="Northern Ireland"/>`
	`  <xsd:enumeration value="Wales"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	`<xsd:element name="geographic_region" type="ns:GeographicRegion"/>`
		None

2.4.4 Constraining Facets

Recommendation	Reason	Impact if ignored
Be aware that **constraining facets** MAY be ignored by data binding tools.	Many data binding tools do not provide full schema validation checking so constraining facets may simply be ignored by data binding tools. For example *maxInclusive* and *minInclusive*, whilst data binding tools allow code to be successfully generated to represent the data structure, they may not always guarantee full schema validation of content.	**Possible**

2.4.5 Documentation

Recommendation	Reason	Impact if ignored

Documentation SHOULD be present for every simpleType, complexType, element and attribute defined in the schema.
 	
Providing business terms, definitions, purposes and dependency information is also necessary – although this is usually specified in the models used to create the physical schemas.
	This allows inclusion of business terminology, description of construct, purposes and dependencies. E.g.
	
	`<xsd:element minOccurs="0" name="firmFSARef" type="ns:FSARefType">`
	`  <xsd:annotation>`
	`  <xsd:documentation>Firm FSA Reference</xsd:documentation>`
	`  <xsd:documentation>A number given by the UK FSA.</xsd:documentation>`
	`  <xsd:documentation>May be used by the Lender to identify the MI.</xsd:documentation>`
	`  <xsd:documentation>Required if the MI is an Appointed Rep.</xsd:documentation>`
	`  </xsd:annotation>`
	`</xsd:element>`
		None

2.4.6 Use of Attributes

Recommendation	Reason	Impact if ignored
Authors SHOULD use attributes correctly.	Schemas must be designed so that elements are the main holders of information content in the XML instances. Attributes are more suited to holding ancillary metadata – simple items which provide more information about the element content. Attributes should not be used to qualify other attributes where this could cause ambiguity. E.g. a valid use of attributes would be when specifying a monetary currency.	**None**

2.4.7 Extension/Restrictions

Recommendation	Reason	Impact if ignored

If authors require the use of **extension facilities** in XML Schema **they SHOULD do so simply by extending a complexType or a simpleType.**	This is the simplest case of extension and is well supported by most toolkits.
	
	For example the current postal address complex type extends the postal address complex type.
	
	`<xsd:complexType name="PostalAddress">`
	`  <xsd:sequence>`
	`  <xsd:element name="line" type="ns:AddressLineType" maxOccurs="5"/>`
	`  <xsd:element name="postcode" type="ns:PostCodeType" minOccurs="0"/>`
	`  </xsd:sequence>`
	`</xsd:complexType>`
	
	`<xsd:complexType name="CurrentPostalAddress">`
	`  <xsd:complexContent>`
	`  <xsd:extension base="ns:PostalAddress">`
	`  <xsd:sequence>`
	`  <xsd:element name="residential_status" type="xsd:string"/>`
	`  <xsd:element name="start_date" type="xsd:date"/>`
	`  </xsd:sequence>`
	`  </xsd:extension>`
	`  </xsd:complexContent>`
	`</xsd:complexType>`
		Possible

If authors require the use of **restriction facilities** in XML Schema **they SHOULD do so simply by restricting a complexType or a simpleType.**	This is the simplest case of restriction and is reasonably well supported by most toolkits.
	
	For example the simple type AddressLineType used above could be defined as follows.
	
	`<xsd:simpleType name="AddressLineType">`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:minLength value="1"/>`
	`  <xsd:maxLength value="35"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	
	continued...
		Possible

	The complexType RestrictedPostalAddress could be defined as follows to ensure a post code is entered.
	
	`<xsd:complexType name="RestrictedPostalAddress">`
	`  <xsd:complexContent>`
	`  <xsd:restriction base="ns:PostalAddress">`
	`  <xsd:sequence>`
	`  <xsd:element name="line" type="ns:AddressLineType" maxOccurs="5"/>`
	`  <xsd:element name="postcode" type="ns:PostCodeType"/>`
	`  </xsd:sequence>`
	`  </xsd:restriction>`
	`  </xsd:complexContent>`
	`</xsd:complexType>`	

2.4.9 Data Types

Recommendation	Reason	Impact if ignored
Authors SHOULD use xsd:int, xsd:decimal and xsd:double to represent numeric values.	This is acceptable to all data binding tools. For example use **xsd:int** as opposed to **xsd:integer**. The definition of the 32 bit integer type in schema is xsd:int and this includes the integer range -2147483648 to 2147483647 which can be mapped very easily to host languages like the Java primitive type “int” or C# value type “int”.	**High**
Authors SHOULD use xsd:date to represent dates.	This is acceptable to the majority of data binding tools.	**High**
Authors SHOULD use xsd:boolean when true/false or yes/no are the only options required.	xsd:boolean should be sufficient without the need to define additional simple type or complex types.	**None**

2.4.10 Schema/Instance Data Re-use

Recommendation	Reason	Impact if ignored
Eliminate redundant XML data with the use of ID/IDREF.	ID/IDREF allows one copy of a data structure to be held and referenced as required within an instance document. Most data binding tools support this feature quite well.	**None**

2.4.11 Optional/Required Elements (Must/Should/May)

Recommendation	Reason	Impact if ignored
Must/Should/May (optional and required ELEMENTS) representations SHOULD be specified accordingly.	**Must** – represented by required elements. **Should** – represented by required elements using the nillable attribute. **May** – represented by optional elements. Unfortunately some data binding tools do not support the use of the nillable attribute very well. Don't use attributes for 'Should' cases as these cannot be implemented as nillable.	**Possible**

If a string element is intended to be nillable then this SHOULD be deliberately specified rather than implied by default.	When support for an empty string element is intended the author should ensure that the element has the @nillable attribute set to true. This makes it clear that nillable is intended by explicitly setting it rather than being implied by a default @minLength attribute value of zero.
	
	Example 1: using @minLength – without the minimum length specified the value will default to zero and effectively allow empty elements. Specifying the minimum length of 1 indicates that nillable is **not** allowed, so the intent is clear.
	
	`<xsd:simpleType name="LongText">`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:minLength value="1"/>`
	`  <xsd:maxLength value="500"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	`<xsd:element name="long_text" type="ns:LongText"/>`
	
	Example 2: using @nillable to denote that empty elements are allowed. The missing @minLength attribute on the simple type implies that nillable **is** intended but using the @nillable attribute makes the intent clear.
	
	`<xsd:simpleType name="LongText">`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:maxLength value="500"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	`<xsd:element name="opt_text" type="ns:LongText" nillable="true" minOccurs="0"/>`
		None

2.4.12 Error Management

Recommendation	Reason	Impact if ignored
Authors SHOULD use SOAP Faults to relay error conditions back to service requestor.	Don't include error constructs in schema definitions. Use the facilities already provided by the SOAP framework. All errors should be documented separately but the structure for communication of error information should not be documented in the schema.	**None**

2.5 Schema/WSDL Design Rules – Don’ts

2.5.1 Naming Conventions

Recommendation	Reason	Impact if ignored
Authors SHOULD avoid attribute and element name clashes.	Some data binding tools cannot cope with the situation where an attribute and an element have the same name.	**High**
Authors SHOULD avoid the use of duplicate element names where their types are different.	This is not good practice and can be confusing.	**Possible**

2.5.2 Use of Attributes

Recommendation	Reason	Impact if ignored
Authors SHOULD avoid using "default" and "fixed" attributes.	The use of the default and fixed attributes allow specification of instance data values to be documented in the schema. As most data binding tools do not perform schema validation against instance content it is not advisable to use these features. The use of "default" implies schema access is available – to retrieve the default value.	**Possible**

2.5.3 Extension/Restrictions

Recommendation	Reason	Impact if ignored
Authors SHOULD avoid the use of abstract complex types and substitutionGroup.	Some data binding tools cannot cope with the situation where complex types as defined as abstract.	**High**
Authors SHOULD avoid the use of mixed content.	Mixed content is more appropriate to documentation style XML rather than message structures. Many data binding tools do not handle mixed content.	**High**
Authors SHOULD avoid the use of anonymous types.	This relates directly to the “Venetian Blind” schema design pattern where types are defined globally.	**None**
Authors SHOULD avoid the use of xsd:anyAttribute and xsd:any.	These features effectively allow any data structures to be extended with or without a backing schema definition (depending on the @processContents attribute). For data binding tools to be most reliable we need to be more prescriptive, so for this reason try to avoid the use of these data types.	**High**
Authors SHOULD avoid the use of xsd:anyType.	Again, this feature effectively allows any data structures to be implemented. For data binding tools to function correctly we need to be more prescriptive.	**High**
Authors SHOULD avoid the use of blockDefault and finalDefault.	These forbid substitution and restriction/extension. Most data binding tools do not cope with schemas which use either of these features.	**High**
Authors SHOULD avoid the use of xsd:union.	Most data binding tools cannot cope with anything other than the most basic use of xsd:union.	**High**
Authors SHOULD avoid the use of	For interoperability it is advisable (see WS-I Basic Profile [4]) to **import** schemas into WSDL	**High**

include when reusing schema definitions **in WSDL.**	documents.	

2.5.4 Data Types

Recommendation	Reason	Impact if ignored

Authors SHOULD avoid use of xsd:nonPositiveInteger, xsd:nonNegativeInteger, xsd:positiveInteger, xsd:negativeInteger, xsd:unsignedLong, xsd:unsignedShort, xsd:unsignedInt or xsd:unsignedByte.	These types do not map directly to host language data types so many data binding tools have problems with them.
	
	If you need functionality offered by these types then it is recommended to use simpleTypes. For example instead of using xsd:positiveInteger use a simpleType like this –
	
	`<xsd:simpleType name="PositiveInteger">`
	`  <xsd:restriction base="xsd:int">`
	`  <xsd:minInclusive value="1"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
		High

Authors SHOULD avoid the use of xsd:float.	Many data binding tools do not cope with this type very well.	**High**
Authors SHOULD avoid use of the Gregorian data types xsd:gDay, xsd:gMonth, xsd:gYear, xsd:gYearMonth and gMonthDay	Some data binding tools do not cope with these types very well.	**High**

2.5.5 Schema/Instance Data Re-use

Recommendation	Reason	Impact if ignored
Authors SHOULD Avoid the use of xsd:redefine.	No data binding tools support this schema feature.	**High**

2.5.6 Sequences

Recommendation	Reason	Impact if ignored

Authors SHOULD avoid xsd:sequence with maxOccurs > 1 or minOccurs > 1	Many data binding tools do not provide schema validation checking so they will not cope with finite limitations on maxOccurs and minOccurs.
	
	They also have issues with maintaining element order. E.g.
	
	`<xsd:element name="books" type="ns:Books"/>`
	`<xsd:complexType name="Books">`
	`  <xsd:sequence maxOccurs="unbounded"/>`
	`  <xsd:element name="pub_date" type="xsd:date"/>`
	`  <xsd:element name="pub_ISBN" type="xsd:string"/>`
	`  </xsd:sequence>`
	`</xsd:complexType>`
	
	Many data binding tools will not handle this data structure correctly.
	See ref [1]
	
	The alternative in the example above would be to specify the maxOccurs at the element level.
	
	`<xsd:element name="books" type="ns:Books" maxOccurs="unbounded"/>`
	`<xsd:complexType name="Books">`
	`  <xsd:sequence>`
	`  <xsd:element name="pub_date" type="xsd:date"/>`
	`  <xsd:element name="pub_ISBN" type="xsd:string"/>`
	`  </xsd:sequence>`
	`</xsd:complexType>`
		High

Authors SHOULD avoid the use of xsd:all.	This option allows for a sequence of elements to appear in any order. Some data binding tools have a problem supporting this construct.	**Possible**

2.5.7 Choice

Recommendation	Reason	Impact if ignored

Authors SHOULD avoid xsd:choice with maxOccurs > 1 or minOccurs > 1 if possible.	Not every data binding tool manages repeating choice correctly.
	
	For example this structure will be problematic:
	
	`<xsd:element name="case_reference" type="ns:CaseReference"/>`
	`<xsd:complexType name="CaseReference">`
	`  <xsd:choice maxOccurs="2">`
	`  <xsd:element name="lender_case_reference" type="xsd:string"/>`
	`  <xsd:element name="mi_case_reference" type="xsd:string"/>`
	`  </xsd:choice>`
	`</xsd:complexType>`
	
	And this structure will be more acceptable:
	
	`<xsd:element name="case_reference" type="ns:CaseReference" maxOccurs="2"/>`
	`<xsd:complexType name="CaseReference">`
	`  <xsd:choice>`
	`  <xsd:element name="lender_case_reference" type="xsd:string"/>`
	`  <xsd:element name="mi_case_reference" type="xsd:string"/>`
	`  </xsd:choice>`
	`</xsd:complexType>`
	
	The data binding toolset report [6] shows that a few older toolsets demonstrate this problem.
		Possible

3 GLOSSARY

WSDL	Web Service Description Language – defines the contract/interface details required to communicate with an implemented SOAP based Web Service.
SOAP	SOAP is a protocol for exchanging XML based messages over networks (normally using HTTP). SOAP forms the foundation layer of the Web Services stack (WS-*) providing a basic messaging framework upon which abstract layers can be built.
HTTP	Hypertext Transfer Protocol is a communications protocol used to transfer information on the internet.

4 REFERENCES

[1] .Net is just one of the tools having problems with xsd:sequence with maxOccurs > 1
http://www.w3.org/2002/ws/databinding/edcopy/report/report_dotnet_cs_2.0.50727.42.html#SequenceMinOccurs0MaxOccursUnbounded101

[2] The W3C XML Schema Patterns for Data Binding Working Group's "Basic XML Schema Patterns" document.http://www.w3.org/TR/xmlschema-patterns/

[3] The W3C XML Schema Patterns for Data Binding Working Group's "Advanced XML Schema Patterns" document.http://www.w3.org/2002/ws/databinding/edcopy/advanced/advanced.html

[4] The Web Services Interoperability Organisation's Basic Profile document.

http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile

[5] Origo Namespace Policy Document.

https://www.origostandards.com/DownloadResource.asp?/StandardsISEF/Technical/Standards_Management/1.0/NamespacesAndOrigoMessages.pdf

[6] Databinding Toolset Comparison Report

http://www.w3.org/2002/ws/databinding/edcopy/report/all.html

[7] RFC 2119

http://www.ietf.org/rfc/rfc2119.txt

5 APPENDIX A – SCHEMA DESIGN PATTERNS

5.1 Russian Doll

In this design the schema has one single global element - the root element. All other elements and types are nested progressively deeper giving it the name due to each type fitting into the one above it. Since the elements in this design are declared locally they will not be reusable through the import or include statements. This will not change if the elements are namespace qualified or namespace unqualified.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="TargetNamespace" xmlns:TN="TargetNamespace"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
    <xs:element name="BookInformation">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Title"/>
                <xs:element name="ISBN"/>
                <xs:element name="PeopleInvolved">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Author"/>
                            <xs:element name="Publisher">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="CompanyName"/>
                                        <xs:element name="ContactPerson"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

The advantages of the Russian Doll approach are: The schema is self contained as it has all of its parts in the schema and does not interact with other schemas. In as much as it is self contained it is also decoupled. Since the content of the schema is not visible to other schemas, changes to the schema are decoupled from other schema components.
The disadvantage is that it is not reusable.
This type of approach would be appropriate for use within a single application or for migration of data from legacy systems.

5.2 Salami Slice

In this approach all elements are defined globally but the type definitions are defined locally. This way other schemas may reuse the elements. With this approach, a global element with its locally defined type provide a complete description of the elements content. This information 'slice' is declared individually and then aggregated back together and may also be pieced together to construct other schemas.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="TargetNamespace" xmlns:TN="TargetNamespace"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
    <xs:element name="BookInformation">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="TN:Title"/>
                <xs:element ref="TN:ISBN"/>
                <xs:element ref="TN:PeopleInvolved"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Title"/>
    <xs:element name="ISBN"/>
    <xs:element name="PeopleInvolved">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="TN:Author"/>
                <xs:element ref="TN:Publisher"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Author"/>
    <xs:element name="Publisher"/>
</xs:schema>
```

The advantage is that the schema is reusable since the elements are declared globally.
The disadvantages are: the schema is verbose since each element is declared globally and then referenced to describe the data which leads to larger schema size. This approach also is not self contained and is coupled. The elements defined may be contained in other schemas and because of this the schema is coupled to other schema and thus changes to one schema will impact other schemas.
This type of approach is commonly used since it is easy to understand and create reusable components. It would be an appropriate design to promote reusability and data standardisation across differing applications. This approach is not, however, recommended when modifications to the standard elements will be necessary. If the length, data types, restrictions or other modifications of the elements need to be changed then this will cause added work as well as a larger impact to other systems.

5.3 Venetian Blind (Recommended)

This approach is similar to the Russian Doll approach in that they both use a single global element. The Venetian Blind approach describes a modular approach by naming and defining all type definitions globally (as opposed to the Salami Slice approach which declares elements globally and types locally). Each globally defined type describes an individual "slat" and can be reused by other components. In addition, all the locally declared elements can be namespace qualified or namespace unqualified (the slats can be "opened" or "closed") depending on the elementFormDefault attribute setting at the top of the schema. If the namespace is unqualified then the local elements in the instance document must not be qualified with the prefix of the namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="TargetNamespace" xmlns:TN="TargetNamespace"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
    <xs:element name="BookInformation" type="TN:BookInformation"
maxOccurs="unbounded"/>
    <xs:complexType name="BookInformation">
        <xs:sequence>
            <xs:element name="Title"/>
            <xs:element name="ISBN"/>
            <xs:element name="PeopleInvolved" type="TN:PeopleInvolvedType"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PeopleInvolvedType">
        <xs:sequence>
            <xs:element name="Author"/>
            <xs:element name="Publisher"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

The advantages are that since all complex and simple types are defined globally they are available for reuse. In addition, the option exists to hide the namespace prefix for all locally defined elements in the instance document.
The disadvantages are that the schema is verbose, it is not self contained and it may be coupled with other schemas.

This type of approach is good when flexibility, reuse and namespace exposure are important. This approach uses a combination of local and global types unlike the Russian Doll approach which all components are locally declared and the Salami Slice where all components are globally declared. This is important as it provides the flexibility to create a schema for most needs since the types can be assigned to elements and extended or restricted as needed. This would be an appropriate design when data is transferred between diverse organisations or business units since it provides each group the flexibility for modifications for each specific requirement.