

XML Schema Architecture Report

SERVICE AGREEMENT: NE-TRG-02

Prepared for:

**Environmental Council of States
44 N. Capital St., NW
Suite 445
Washington, DC 20001**

Prepared by:

**Systems Development Center
Science Applications International Corporation
6565 Arlington Boulevard
Falls Church, VA 22042**

CONTENT

<u>1.0</u>	<u>INTRODUCTION</u>	1
<u>1.1</u>	<u>Purpose</u>	1
<u>1.2</u>	<u>References</u>	1
<u>2.0</u>	<u>XML SCHEMA ARCHITECTURE</u>	2
<u>2.1</u>	<u>Russian Doll Design</u>	4
	<u>2.1.1 Advantages and Disadvantages</u>	7
	<u>2.1.2 Usage and Recommendations</u>	7
<u>2.2</u>	<u>Salami Slice</u>	8
	<u>2.2.1 Advantages and Disadvantages</u>	10
	<u>2.2.2 Usage and Recommendations</u>	11
<u>2.3</u>	<u>Venetian Blind</u>	11
	<u>2.3.1 Advantages and Disadvantages</u>	13
	<u>2.3.2 Usage and Recommendations</u>	13
<u>2.4</u>	<u>Garden of Eden Design</u>	14
	<u>2.4.1 Advantages and Disadvantages</u>	16
	<u>2.4.2 Usage and Recommendations</u>	16
<u>2.5</u>	<u>Summary of Architecture Characteristics</u>	16
<u>3.0</u>	<u>SCHEMA ARCHITECTURE REVIEW</u>	17
<u>3.1</u>	<u>Beaches (Notification v1.0 and Monitoring v1.1) Schema</u>	17
<u>3.2</u>	<u>Data Services (Data Services List v1.0 and Network Flow v1.0) Schema</u>	18
<u>3.3</u>	<u>Pacific Northwest Water Quality Exchange (v1.1) Schema</u>	19
<u>3.4</u>	<u>Summary Review</u>	20

APPENDIX A References and Review Schema

EXHIBITS

<u>Exhibit 1.</u>	<u>Example of Global and Local Elements and Types</u>	2
<u>Exhibit 2.</u>	<u>Russian Doll XML Schema—Graphical View</u>	4
<u>Exhibit 3.</u>	<u>Russian Doll XML Schema Text</u>	5
<u>Exhibit 4.</u>	<u>Russian Doll XML Document—Exposed Namespace</u>	6
<u>Exhibit 5.</u>	<u>Russian Doll XML Document—Hidden Namespace</u>	6

<u>Exhibit 6.</u> Salami Slice XML Schema—Graphical view	8
<u>Exhibit 7.</u> Salami Slice XML Schema—Text View	9
<u>Exhibit 8.</u> Salami Slice XML Document—Exposed Namespace.....	10
<u>Exhibit 9.</u> Venetian Blind XML Schema—Graphical view	11
<u>Exhibit 10.</u> Venetian Blind XML Schema—Text View	12
<u>Exhibit 11.</u> Garden of Eden XML Schema—Graphical view	14
<u>Exhibit 12.</u> Garden of Eden XML Schema—Text View	15
<u>Exhibit 13.</u> Summary of Characteristics for Architectures	17
<u>Exhibit 14.</u> Example of Beaches XML Schema—Graphical view	18
<u>Exhibit 15.</u> Example of Data Services XML Schema—Graphical view	19
<u>Exhibit 16.</u> Example of PNWWQX XML Schema—Graphical view	20
<u>Exhibit 17.</u> Summary of Beaches XML Schema Architecture Review	20
<u>Exhibit 18.</u> Summary of Data Services and PNWWQX XML Schema Architecture Review	21

1.0 INTRODUCTION

In the summer of 2003, on behalf of the Environmental Information Exchange Network Steering Board (NSB), ECOS initiated a Schema Review Pilot Project for technical review of eXtensible Markup Language (XML) schemas to ensure that they adhere to the values and standards of the Environmental Information Exchange Network (Exchange Network). The Exchange Network is a cooperative effort among the US Environmental Protection Agency (EPA), states, and tribes to promote data exchange and information sharing across various environmental programs. This initial review established guidelines for review of schema for compliance with World Wide Web Consortium (W3C) XML schema requirements and standards; conformance with the Exchange Network XML Design Rules and Conventions; comparison to the Core Reference Model (CRM) for overall modularity and identification of potential new data blocks that could be added to the CRM; and conformance with applicable data standards (<http://www.envdatastandards.net/>).

Subsequently, ECOS has requested that SAIC prepare a report identifying and describing commonly used XML schema architectures. The following XML schemas were reviewed to determine which, if any, of the common schema architectures were used in the development of the XML schema:

- Beaches (Notification v1.0 and Monitoring v1.1).
- Data Services v1.0.
- Pacific Northwest Water Quality Exchange v1.1.

1.1 Purpose

Section 2.0 of the report documents four state-of-the-art architectures for schema designs and Section 3.0 presents a review of the patterns used for the Beaches (Notification v1.0 and Monitoring v1.1), Data Services v1.0, and Pacific Northwest Water Quality Exchange v1.1 schema. The review and analysis will help with the further refinement of current XML schema development guideline and encourage data standardization and reusability of XML schema components.

1.2 References

The references used for this review are listed in Appendix A, References and Review Schema.

2.0 XML SCHEMA ARCHITECTURE

The most popular current XML schema architectures are Russian Doll, Salami Slice, Venetian Blind, and Garden of Eden; each represents different design characteristics. The most important characteristic differentiating between the XML schema architectures is whether elements and types are globally defined. Globally defined elements and types are the immediate child of the “<schema>” construct in the XML schema definition file. Conversely, locally defined elements and types are not the immediate child of the “<schema>” construct; locally defined elements and types are nested within other elements and types. XML schema architectures promoting global elements and types ensure that the global constructs are available for reuse and modification by other XML schemas using import or include constructs. Locally defined elements and types are not available for reuse by other XML schemas.

Exhibit 1 provides a brief example of both global and local elements. The global elements, which are named RootElement and GlobalElement, are immediate children of the “<xsd:schema>” construct. The local elements, which are named LocalElement_01 and LocalElement_02, are nested within GlobalElement.

Exhibit 1 also illustrates both global and local types. Local types are sometimes referred to as anonymous types since they are not assigned a name. The schema construct “<xsd:complexType>” is an example of the start of an anonymous type that describes the content of the element named GlobalElement. The schema construct “<xsd:simpleType name=.....>” is an example of a globally defined, named type definition. That type definition is used as to describe the local element named LocalElement_02 by assigning the named type to the attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:element name="RootElement">
    <xsd:annotation>
      <xsd:documentation>Comment describing your root element</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:element name="GlobalElement">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="LocalElement_01"/>
        <xsd:element name="LocalElement_02" type="TypeForLocalElement_01"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="TypeForLocalElement_01">
    <xsd:restriction base="xsd:string">
      <xsd:length value="10"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Exhibit 1. Example of Global and Local Elements and Types

Another related characteristic of XML schema architectures is whether the namespace in the XML instance document is hidden or exposed. If the namespace is exposed, the XML instance document must qualify every element (both local and global) with the prefix of the namespace.

Conversely, if the namespace is hidden, the XML instance document does not qualify local elements with the prefix of the namespace but must add the prefix of the namespace to all globally defined elements. Examples of XML documents with hidden and exposed namespaces are presented in Section 2.1.

The XML schema designer may select whether the XML instance document hides or exposes namespaces by setting the `elementFormDefault` schema construct. If the construct is set to `qualified`, the namespace in the XML instance document is exposed. If the construct is set to `unqualified`, the namespace in the XML instance document is hidden. Therefore, architecture requirements for hiding and exposing namespaces should be considered when selecting architecture in which to design XML schemas.

The following sections describe the characteristics of the each architecture, list the advantages and disadvantages of using it, and make recommendations regarding when to use a specific architecture. The Data Services Network Flow XML schema was modified to illustrate each of the different XML schema architectures.

The architectures will be compared based on the following characteristics:

- Compact or verbose—Describes the size of the XML schema definition file.
- Self-contained—Describes whether the XML schema components are fully described within the XML schema or are referenced from other external XML schema components.
- Coupled or decoupled—Describes whether the XML schema components are visible to other XML schemas which may be impacted by changes to this XML schema.
- Reusable—Defines the XML schema components that are available to other XML schemas.
- Single or multiple XML instance documents—Describes whether one or more different, valid XML document may be created from this XML schema.
- Option to hide namespaces—Describes whether the XML schema supports the option to eliminate the prefix of a namespace for local elements.

2.1 Russian Doll Design

In the Russian Doll architecture, the XML schema defines one single global element, which is the root element of the XML schema. All other elements are nested deeply within anonymous types like the famous stacking dolls. All type definitions are defined as local or anonymous types.

Exhibit 2 presents a graphical view of the Data Services XML schema modified to illustrate the Russian Doll design. As seen in the graphical view, the only global element on the XML Schema overview is the root element, NetworkFlow. All other locally defined elements and types are nested within the root element.

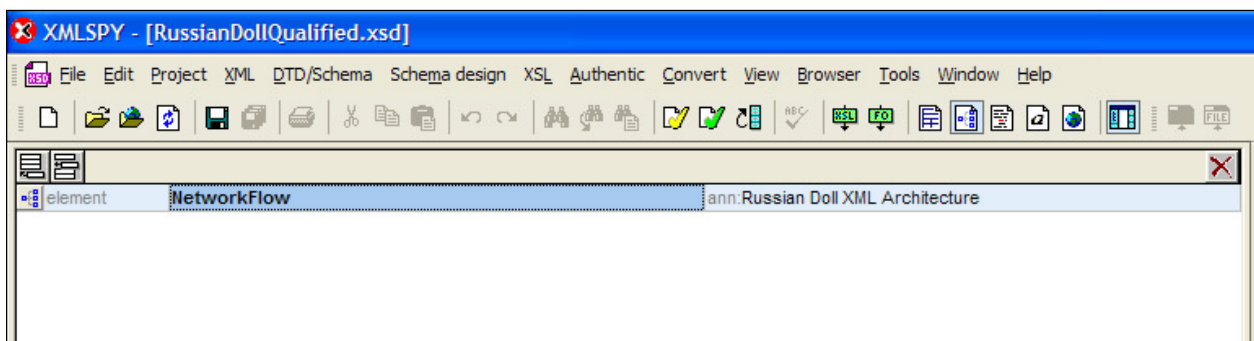


Exhibit 2. Russian Doll XML Schema—Graphical View

Exhibit 3 presents a text view of the same XML schema depicted in Exhibit 2. The root element of the XML schema, NetworkFlow, is an immediate child of the <schema> construct. The local elements named NetworkFlowName and NetworkFlowDataService are the local elements declared as child element of the root element. The remaining elements (e.g., DataServiceName, DataServiceParameters, DataServiceSolicitWithDownloadAllowedIndicator) are local elements declared as child elements of the NetworkFlowDataService and the DataServicePayloadSchema element. Since the only globally defined element is the root element, the remaining elements are not available to other XML schemas through the import or include statements.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Terrie Sutch (SAIC) -->
<xsd:schema targetNamespace="TargetNamespace" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:TN="TargetNamespace"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:element name="NetworkFlow">
    <xsd:annotation>
      <xsd:documentation>Russian Doll XML Architecture</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="NetworkFlowName"/>
        <xsd:element name="NetworkFlowDataService">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="DataServiceName"/>
              <xsd:element name="DataServiceParameters"/>
              <xsd:element name="DataServiceSolicitWithDownloadAllowedIndicator"/>
              <xsd:element name="DataServiceSolicitWithSubmitAllowedIndicator"/>
              <xsd:element name="DataServiceQueryAllowedIndicator"/>
              <xsd:element name="DataServicePayloadSchema">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="PayloadSchemaName"/>
                    <xsd:element name="PayloadSchemaTargetNamespace"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Exhibit 3. Russian Doll XML Schema Text

Exhibit 4 presents an XML document conforming to the XML schema presented in Exhibit 2. Since the elementFormDefault attribute is set to qualified, the namespace in the XML instance document is exposed. All locally defined elements (e.g., NetworkFlowName, NetworkFlowDataService) are qualified with the prefix of the namespace (i.e., “TN”).

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-->
<TN:NetworkFlow xmlns:TN="TargetNamespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="TargetNamespace
C:\MYDATA~2\CURREN~4\ECOS2\Architecture\RussianDollQualified.xsd">
  <TN:NetworkFlowName>Text</TN:NetworkFlowName>
  <TN:NetworkFlowDataService>
    <TN:DataServiceName>Text</TN:DataServiceName>
    <TN:DataServiceParameters>Text</TN:DataServiceParameters>
    <TN:DataServiceSolicitWithDownloadAllowedIndicator>Text</TN:DataServiceSolicitWithDownloadAllowedIndicator>
    <TN:DataServiceSolicitWithSubmitAllowedIndicator>Text</TN:DataServiceSolicitWithSubmitAllowedIndicator>
    <TN:DataServiceQueryAllowedIndicator>Text</TN:DataServiceQueryAllowedIndicator>
    <TN:DataServicePayloadSchema>
      <TN:PayloadSchemaName>Text</TN:PayloadSchemaName>
      <TN:PayloadSchemaTargetNamespace>Text</TN:PayloadSchemaTargetNamespace>
    </TN:DataServicePayloadSchema>
  </TN:NetworkFlowDataService>
</TN:NetworkFlow>
```

Exhibit 4. Russian Doll XML Document—Exposed Namespace

Exhibit 5 presents the same XML document assuming that the XML schema designer has set the elementFormDefault construct to unqualified. As illustrated, the same locally defined elements (e.g., NetworkFlowName, NetworkFlowDataService) that were qualified in Exhibit 4 are not qualified in Exhibit 5. The globally declared root element (i.e., NetworkFlow), however, is qualified in both examples.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-->
<TN:NetworkFlow xmlns:TN="TargetNamespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="TargetNamespace
C:\MYDATA~2\CURREN~4\ECOS2\Architecture\RussianDollUnqualified.xsd">
  <NetworkFlowName>Text</NetworkFlowName>
  <NetworkFlowDataService>
    <DataServiceName>Text</DataServiceName>
    <DataServiceParameters>Text</DataServiceParameters>
    <DataServiceSolicitWithDownloadAllowedIndicator>Text</DataServiceSolicitWithDownloadAllowedIndicator>
    <DataServiceSolicitWithSubmitAllowedIndicator>Text</DataServiceSolicitWithSubmitAllowedIndicator>
    <DataServiceQueryAllowedIndicator>Text</DataServiceQueryAllowedIndicator>
    <DataServicePayloadSchema>
      <PayloadSchemaName>Text</PayloadSchemaName>
      <PayloadSchemaTargetNamespace>Text</PayloadSchemaTargetNamespace>
    </DataServicePayloadSchema>
  </NetworkFlowDataService>
</TN:NetworkFlow>
```

Exhibit 5. Russian Doll XML Document—Hidden Namespace

2.1.1 Advantages and Disadvantages

The advantages to using Russian Doll as the XML schema architecture are as follows:

- **Compact**—The file size of XML schemas using this architecture is less than the file size of the same XML schema designed using other architectures.
- **Self-contained**—Since all declarations and definitions are in the same component, the XML schema has all of its parts in one place and does not interact with other XML schema components.
- **Decoupled**—The content of the XML schema is not visible to other XML schema or within the same XML schema thus changes to the XML schema are decoupled from other XML schema components.
- **Single valid XML document**—Since there is a single global element in the XML schema, there is only one valid XML instance document.
- **Option to hide namespace**—The XML schema designer is given the option to require an XML instance document to include the namespace prefix for all locally defined elements.

The disadvantages to using Russian Doll as the XML schema architecture are as follows:

- **Not reusable**—Since only the root element is globally defined, it is the only element available for use or modification by other XML schemas. The other locally declared elements are not available to other XML schemas

2.1.2 Usage and Recommendations

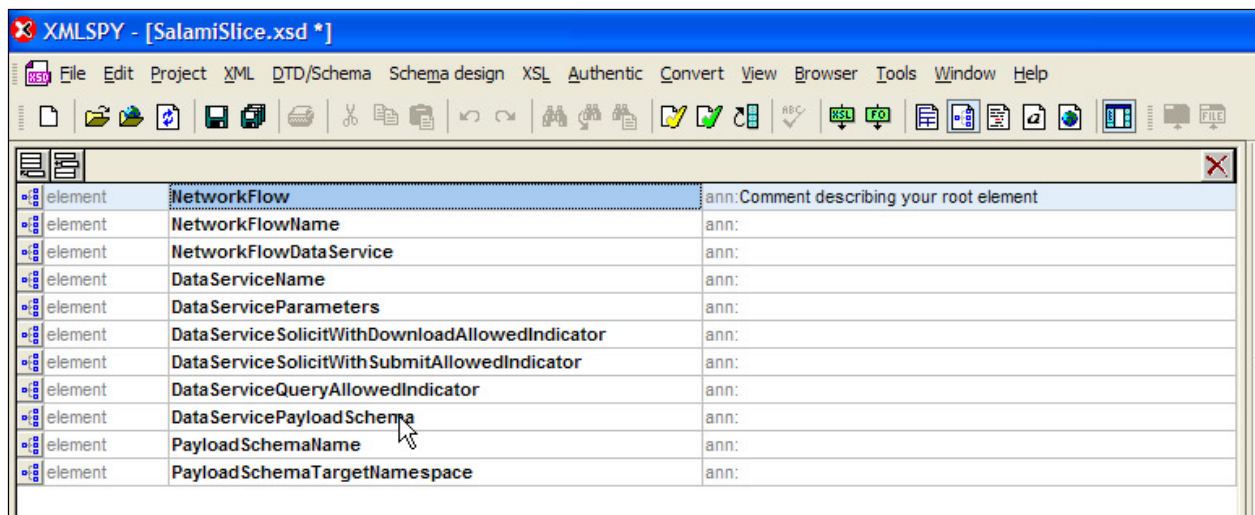
The Russian Doll architecture would be appropriate to apply for any one-time dataflow such as the migration of data from a legacy system to its replacement system. It might also be appropriate for use within a single application or when a data flow does not require any sharing of components. For example, if a single application extracts data into an XML document to present the information in a browser.

This architecture would not be appropriate for use where data standardization and reusable components are required or global availability is needed.

2.2 Salami Slice

The Salami Slice architecture defines a modular approach to XML schema design by globally defining all elements. All type definitions, however, are defined locally. When an XML schema declares global elements, other XML schemas may reuse them. A global element, along with its locally defined type, provide a complete description of the of the element’s content. This information “slice” may be pieced together to construct other XML schemas.

Exhibit 6 presents a graphical view of the Data Services XML schema modified to illustrate the Salami Slice design. As seen in the graphical view, only global elements are presented in the XML schema overview. All other locally defined types are nested within the global elements and are not presented in the graphic overview of the XML schema.



element	NetworkFlow	ann:Comment describing your root element
element	NetworkFlowName	ann:
element	NetworkFlowDataService	ann:
element	DataServiceName	ann:
element	DataServiceParameters	ann:
element	DataServiceSolicitWithDownloadAllowedIndicator	ann:
element	DataServiceSolicitWithSubmitAllowedIndicator	ann:
element	DataServiceQueryAllowedIndicator	ann:
element	DataServicePayloadSchema	ann:
element	PayloadSchemaName	ann:
element	PayloadSchemaTargetNamespace	ann:

Exhibit 6. Salami Slice XML Schema—Graphical view

Exhibit 7 presents a text view of the same XML schema depicted in Exhibit 6. The root element of the XML schema, NetworkFlow, is an immediate child of the <schema> construct. Additionally, all other elements are defined as an immediate child of the <schema> construct. The XML schema does not contain any local elements. It does, however, contain local or anonymous type definitions that are nested within the global element declarations.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) (SAIC) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="NetworkFlow">
    <xs:annotation>
      <xs:documentation>Salami Slice XML Schema Architecture</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="NetworkFlowName"/>
        <xs:element ref="NetworkFlowDataService"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="NetworkFlowName"/>
  <xs:element name="NetworkFlowDataService">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="DataServiceName"/>
        <xs:element ref="DataServiceParameters"/>
        <xs:element ref="DataServiceSolicitWithDownloadAllowedIndicator"/>
        <xs:element ref="DataServiceSolicitWithSubmitAllowedIndicator"/>
        <xs:element ref="DataServiceQueryAllowedIndicator"/>
        <xs:element ref="DataServicePayloadSchema"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DataServiceName"/>
  <xs:element name="DataServiceParameters"/>
  <xs:element name="DataServiceSolicitWithDownloadAllowedIndicator"/>
  <xs:element name="DataServiceSolicitWithSubmitAllowedIndicator"/>
  <xs:element name="DataServiceQueryAllowedIndicator"/>
  <xs:element name="DataServicePayloadSchema">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="PayloadSchemaName"/>
        <xs:element ref="PayloadSchemaTargetNamespace"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="PayloadSchemaName"/>
  <xs:element name="PayloadSchemaTargetNamespace"/>
</xs:schema>

```

02
04

I

Exhibit 7. Salami Slice XML Schema—Text View

Exhibit 8 presents an XML document conforming to the XML schema presented in Exhibit 7. Since hiding a namespace only refers to local elements and all the elements are declared globally, the elements must be qualified with the prefix of the namespace, regardless of the setting of the elementFormDefault construct. This is the same XML instance document presented in Exhibit 4. If the elementFormDefault construct is set to qualified (i.e., exposed namespaces), the XML schema architecture does not change the content and structure of the XML instance document. If the elementFormDefault construct, however, is set to unqualified (i.e., hidden namespaces), the XML instance document must remove the namespace prefix from local elements based on the declaration of local element in the XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-->
<TN:NetworkFlow xmlns:TN="TargetNamespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="TargetNamespace
C:\MYDATA\2\CURRENT\4\ECOS2\Architecture\SalamiSlice.xsd">
  <TN:NetworkFlowName>Text</TN:NetworkFlowName>
  <TN:NetworkFlowDataService>
    <TN:DataServiceName>Text</TN:DataServiceName>
    <TN:DataServiceParameters>Text</TN:DataServiceParameters>
    <TN:DataServiceSolicitWithDownloadAllowedIndicator>Text</TN:DataServiceSolicitWithDownloadAllowedIndicator>
    <TN:DataServiceSolicitWithSubmitAllowedIndicator>Text</TN:DataServiceSolicitWithSubmitAllowedIndicator>
    <TN:DataServiceQueryAllowedIndicator>Text</TN:DataServiceQueryAllowedIndicator>
    <TN:DataServicePayloadSchema>
      <TN:PayloadSchemaName>Text</TN:PayloadSchemaName>
      <TN:PayloadSchemaTargetNamespace>Text</TN:PayloadSchemaTargetNamespace>
    </TN:DataServicePayloadSchema>
  </TN:NetworkFlowDataService>
</TN:NetworkFlow>
```

Exhibit 8. Salami Slice XML Document—Exposed Namespace

2.2.1 Advantages and Disadvantages

The advantages to using Salami Slice as the XML schema architecture are as follows:

- Reusable—Since all elements are globally declared, they are available for use by other XML schemas.

The disadvantages to using Salami Slice as the XML schema architecture are as follows:

- Verbose—Each element is globally declared and later referenced to describe the data flow resulting in an increase in the size of the XML schema.
- Not self-contained—The definitions and declarations describing a specific element may be contained in multiple XML schema components.
- Coupled—Since the XML schema may reference other XML schemas and may also be referenced by other XML schemas, it is no longer decoupled from other XML thus changes may impact other users.
- Multiple valid XML documents—Since many global elements may exist in the XML schema, multiple valid XML instance document may be created for the same XML schema.
- Option to hide namespace—Namespaces are always exposed since all elements are global.

2.2.2 Usage and Recommendations

This is one of the most commonly used schema architectures. It is easy to understand and create reusable components. All elements are declared globally and named so they can be pieced together to construct data flows. This XML schema architecture should be used to promote reusability and data standardization across diverse application systems. This architecture should not be used when the modifications to the standard elements may be required. For example, if the length, data type, permitted values of a standard element need to be changed for a specific data flow, this architecture may not be applicable.

2.3 Venetian Blind

Like the Russian Doll, the Venetian Blind architecture uses a single global element. All other element declarations are defined locally allowing the XML schema designer to expose or hide the namespaces in the XML instance document. The Venetian Blind architecture describes a modular approach to XML schema design by naming and defining all type definitions globally (as compared to the Salami slice that declares elements globally and types locally). Each globally defined type describes an individual “slat” that can be reused by other components. The “slat” can be opened or closed to expose or hide namespaces based on the setting of the elementFormDefault construct in any XML schema using that global type.

Exhibit 9 presents a graphical view of the Data Services XML schema modified to illustrate the Venetian Blind design. As seen in the graphical view, only the root element and the global type definitions are presented in the XML schema overview. All other locally declared elements are nested within the root element and are not presented in the graphic overview of the XML schema.

Category	Name	Namespace
element	NetworkFlow	ann:Venetian Blind XML Schema Architecture
complexType	NetworkFlow	ann:
complexType	DataService Type	ann:
complexType	DataServiceParametersType	ann:
complexType	DataServicePayloadSchemaType	ann:

Exhibit 9. Venetian Blind XML Schema—Graphical view

Exhibit 10 presents a text view of the same XML schema depicted in Exhibit 9. Each of the complex type definitions is named and defined globally (e.g., NetworkFlow,

DataServiceType). As in the Russian Doll architecture, the root element of the XML schema, NetworkFlow, is globally defined as an immediate child of the <schema> construct with its type attribute set to the globally defined complex type named NetworkFlow. The local elements named NetworkFlowName and NetworkFlowDataService are the local child elements of the complex type NetworkFlow. The remaining elements (e.g., DataServiceName, DataServiceParameters, DataServiceSolicitWithDownloadAllowedIndicator) are local child elements of NetworkFlowDataService and the DataServicePayloadSchema complex type.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) (SAIC) -->
<xsd:schema targetNamespace="TargetNamespace" xmlns:TN="TargetNamespace" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:element name="NetworkFlow" type="TN:NetworkFlow">
    <xsd:annotation>
      <xsd:documentation>Venetian Blind XML Schema Architecture</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name="NetworkFlow">
    <xsd:sequence>
      <xsd:element name="NetworkFlowName"/>
      <xsd:element name="NetworkFlowDataService" type="TN:DataServiceType" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="DataServiceType">
    <xsd:sequence>
      <xsd:element name="DataServiceName"/>
      <xsd:element name="DataServiceParameters" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="DataServiceSolicitWithDownloadAllowedIndicator"/>
      <xsd:element name="DataServiceSolicitWithSubmitAllowedIndicator"/>
      <xsd:element name="DataServiceQueryAllowedIndicator"/>
      <xsd:element name="DataServicePayloadSchema" type="TN:DataServicePayloadSchemaType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="DataServiceParametersType">
    <xsd:sequence>
      <xsd:element name="ParameterIndex"/>
      <xsd:element name="ParameterName"/>
      <xsd:element name="ParameterMaximumValues"/>
      <xsd:element name="ParameterDataType"/>
      <xsd:element name="ParameterRequiredIndicator"/>
      <xsd:element name="ParameterUseInstructions"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="DataServicePayloadSchemaType">
    <xsd:sequence>
      <xsd:element name="PayloadSchemaName"/>
      <xsd:element name="PayloadSchemaTargetNamespace"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Exhibit 10. Venetian Blind XML Schema—Text View

Since all elements are locally declared, the namespace may be hidden or exposed as described for the Russian Doll architecture. If the elementFormDefault attribute is set to qualified, local

elements in the XML instance document must be qualified with the prefix of the namespace. If the attribute is set to unqualified, local elements in the XML instance document must not be qualified with the prefix of the namespace. The same XML instance documents presented in Exhibits 4 and 5 illustrate that exposing and hiding namespaces for XML schema designed using the Russian Doll architecture are valid for XML schemas designed using the Venetian Blind architecture.

2.3.1 Advantages and Disadvantages

The advantages to using Venetian Blind as the XML schema architecture are as follows:

- Reusable—Since all complex and simple type definitions are defined globally, they are available for reuse by other XML schemas.
- Single valid XML document—Since there is a single global element in the XML schema, there is only one valid XML instance document.
- Option to hide namespace—The XML schema designer is given the option to require an XML instance document to include the namespace prefix for all locally defined elements.

The disadvantages to using Venetian Blind as the XML schema architecture are as follows:

- Verbose—Each type definition is defined globally and later assigned as the type to describe the data flow resulting in an increase in the size of the XML schema.
- Not self contained—The definitions and declarations describing a specific element may be contained in multiple XML schema components.
- Decoupled—Since the XML schema may reference other XML schemas and may also be referenced by other XML schemas, it is no longer decoupled from other XML thus changes may impact other users.

2.3.2 Usage and Recommendations

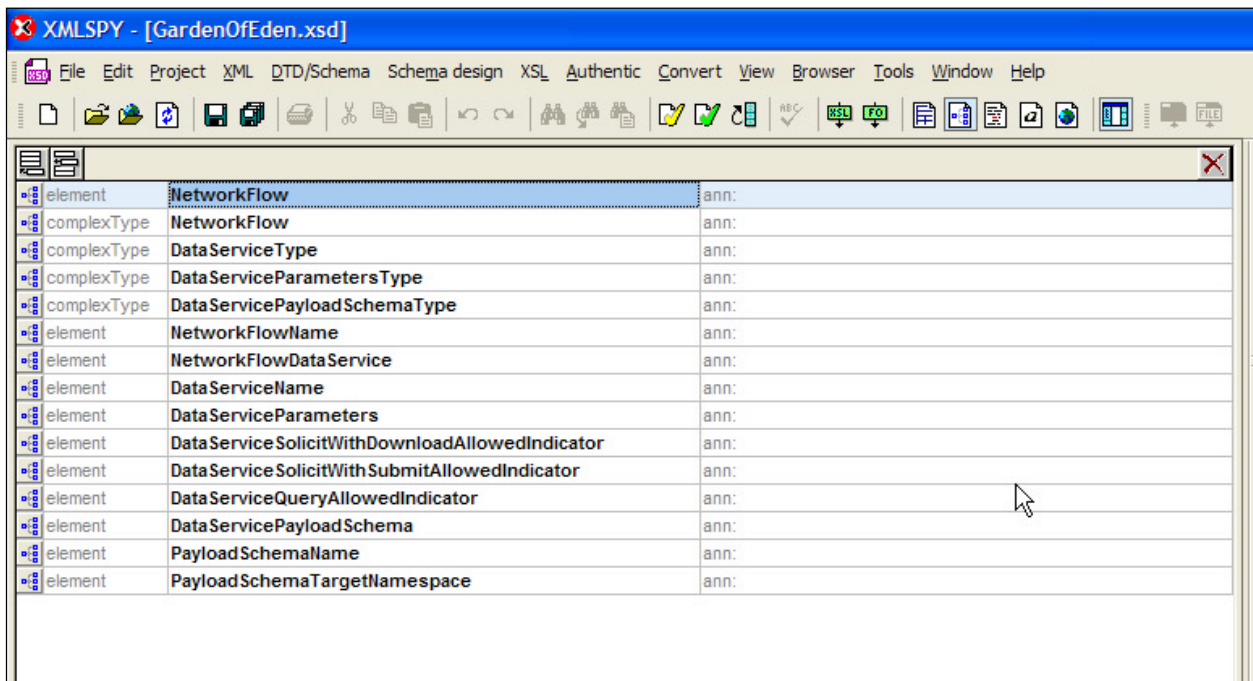
The Venetian Blind schema should be considered for use when flexibility, reuse, and namespace exposure are important. Unlike the Russian Doll where all components are locally declared and Salami Slice where elements are all globally declared, the Venetian Blind architecture uses a combination of global and local types to specify content. This provides the developer with the flexibility to provide an XML schema for most business needs because the types can be assigned to elements and extended or restricted as needed. This architecture

should be used when data is transferred among diverse organizations and agencies providing each participant the flexibility to modify the data flow to satisfy its specific requirements.

2.4 Garden of Eden Design

The Garden of Eden architecture implements characteristics of both the Venetian Blind and the Salami Slice architecture. As in the Salami Slice architecture, the Garden of Eden architecture describes a modular approach to XML schema design by defining all elements globally. As in the Venetian Blind architecture, all type definitions are declared globally.

Exhibit 11 presents a graphical view of the Data Services XML schema modified to illustrate the Garden of Eden design. As seen in the graphical view, all elements and all type definitions are globally defined and presented in the XML schema overview.



Category	Name	Namespace
element	NetworkFlow	ann:
complexType	NetworkFlow	ann:
complexType	DataServiceType	ann:
complexType	DataServiceParametersType	ann:
complexType	DataServicePayloadSchemaType	ann:
element	NetworkFlowName	ann:
element	NetworkFlowDataService	ann:
element	DataServiceName	ann:
element	DataServiceParameters	ann:
element	DataServiceSolicitWithDownloadAllowedIndicator	ann:
element	DataServiceSolicitWithSubmitAllowedIndicator	ann:
element	DataServiceQueryAllowedIndicator	ann:
element	DataServicePayloadSchema	ann:
element	PayloadSchemaName	ann:
element	PayloadSchemaTargetNamespace	ann:

Exhibit 11. Garden of Eden XML Schema—Graphical view

Exhibit 12 presents a text view of the same XML schema depicted in Exhibit 11. Each of the complex type definitions is named and defined globally. Each element is globally defined as an immediate child of the <schema> construct and its type attribute may be set to one of the named complex types. For example, the type attribute of the global element NetworkFlow is set to the complex type named NetworkFlow.

```

<?xml version="1.0" encoding="UTF-8"?>|
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) (SAIC) -->
<xsd:schema targetNamespace="TargetNamespace" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="TargetNamespace"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:element name="NetworkFlow" type="NetworkFlow"/>
  <xsd:complexType name="NetworkFlow">
    <xsd:sequence>
      <xsd:element ref="NetworkFlowName"/>
      <xsd:element ref="NetworkFlowDataService" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="DataServiceType">
    <xsd:sequence>
      <xsd:element ref="DataServiceName"/>
      <xsd:element ref="DataServiceParameters" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="DataServiceSolicitWithDownloadAllowedIndicator"/>
      <xsd:element ref="DataServiceSolicitWithSubmitAllowedIndicator"/>
      <xsd:element ref="DataServiceQueryAllowedIndicator"/>
      <xsd:element ref="DataServicePayloadSchema"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="DataServiceParametersType">
    <xsd:sequence>
      <xsd:element name="ParameterIndex"/>
      <xsd:element name="ParameterName"/>
      <xsd:element name="ParameterMaximumValues"/>
      <xsd:element name="ParameterDataType"/>
      <xsd:element name="ParameterRequiredIndicator"/>
      <xsd:element name="ParameterUseInstructions"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="DataServicePayloadSchemaType">
    <xsd:sequence>
      <xsd:element ref="PayloadSchemaName"/>
      <xsd:element ref="PayloadSchemaTargetNamespace"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="NetworkFlowName"/>
  <xsd:element name="NetworkFlowDataService" type="DataServiceType"/>
  <xsd:element name="DataServiceName"/>
  <xsd:element name="DataServiceParameters"/>
  <xsd:element name="DataServiceSolicitWithDownloadAllowedIndicator"/>
  <xsd:element name="DataServiceSolicitWithSubmitAllowedIndicator"/>
  <xsd:element name="DataServiceQueryAllowedIndicator"/>
  <xsd:element name="DataServicePayloadSchema" type="DataServicePayloadSchemaType"/>
  <xsd:element name="PayloadSchemaName"/>
  <xsd:element name="PayloadSchemaTargetNamespace"/>
</xsd:schema>

```

Exhibit 12. Garden of Eden XML Schema—Text View

Since all elements are globally declared, the namespace must be exposed regardless of the setting of the elementFormDefault attribute. The same XML instance document presented in Exhibit 4 illustrates an XML instance document conforming to the XML schema presented in Exhibit 12. Using the Garden of Eden architecture, if the elementFormDefault construct is set to qualified (i.e., exposed namespaces), the XML schema architecture does not change the content and structure of the XML instance document. If the elementFormDefault construct, however, is set to unqualified (i.e., hidden namespaces), the XML instance document must

remove the namespace prefix from local elements based on the declaration of local element in the XML schema.

2.4.1 Advantages and Disadvantages

The advantages to using Garden of Eden as the XML schema architecture are as follows:

- Reusable—Since both type definitions and element declarations are globally defined, both are available for reuse this architecture offers the maximum reusable content.

The disadvantages to using Garden of Eden as the XML schema architecture are as follows:

- Verbose—Each element is globally declared and later referenced to describe the data flow resulting in an increase in the size of the XML schema.
- Not self contained—The definitions and declarations describing a specific element may be contained in multiple XML schema components.
- Decoupled—Since the XML schema may reference other XML schemas and may also be referenced by other XML schemas, it is no longer decoupled from other XML thus changes may impact other users.
- Multiple XML documents—Since there are multiple global elements in the XML schema, there may be multiple valid XML instance document.
- Option to hide namespace—Namespaces are always exposed since all elements are global.

2.4.2 Usage and Recommendations

The Garden of Eden is a relatively new XML schema architecture. It has been recommended for use by Organization for the Advancement of Structured Information Standards (OASIS) Universal Business Language (UBL) and provides global specification for both elements and types. It provides semantic clarity when it is used. By globally defining both elements and types, the XML schema designer has the most flexibility. The XML schema developer can choose to use either the globally declared element, or if that element does not exactly fit the specifications, the global type can be modified as required.

2.5 Summary of Architecture Characteristics

Exhibit 13 is a comparison of some of the characteristics noted for the XML schema architectures described above. Each architecture has specific characteristics and the choice of

architectures is dependent on the need for reusability, semantic specification, compact design, and namespace clarity. For a simple in-house application that is used only a few times for a very specific task, the Russian Doll design might be appropriate. For a flexible, multi-user application with complex information transfer, however, the Venetian Blind, Salami Slice, or the newer, Garden of Eden architecture would be more appropriate. The following table summarizes the characteristics of the each architecture.

Characteristic	Russian Doll	Salami Slice	Venetian Blind	Garden of Eden
Compact	Yes	No	No	No
Self-contained	Yes	No	No	No
Coupled	No	Yes	Yes	Yes
Reusable Elements	No	Yes	No	Yes
Reusable Types	No	No	Yes	Yes
Valid XML Documents	Single	Multiple	Single	Multiple
Option to Hide Namespace	Yes	No	Yes	Yes

Exhibit 13. Summary of Characteristics for Architectures

3.0 SCHEMA ARCHITECTURE REVIEW

XML schemas submitted to the Exchange Network have been constructed using a variety of XML schema architectures. XML schemas from the following data flows have been reviewed to determine the schema architecture used in their construction. The analysis is summarized in Section 3.4.

3.1 Beaches (Notification v1.0 and Monitoring v1.1) Schema

The design of the Beaches XML schemas is most consistent with the Venetian Blind architecture because it defines global types and does not define global elements (except the root element). Exhibit 14 depicts an overview of the Beaches Notification XML schema. It clearly illustrates a single root element and the specification of named complex and simple types. The current Beaches XML schemas define some complex and simple types locally. To be fully structured according the Venetian Blind architecture, all types must be global.

The Beaches XML schemas have set the elementFormDefault construct to expose namespaces in the XML instance document, which is consistent with the options for the Venetian Blind architecture. A target namespace has not been declared in the Beaches XML schemas. Although a target namespace is not required by the Venetian Blind architecture, it is typically declared to enhance reusability of global types.

element	BeachDataSubmission	ann:
complexType	MailingAddressType	ann:
complexType	ProgramInterestType	ann:
complexType	OrganizationType	ann:
complexType	PersonType	ann:
complexType	ElectronicAddressType	ann:
complexType	TelephoneType	ann:
complexType	ProcedureType	ann:
complexType	ActivityType	ann:
simpleType	Char2	ann:
simpleType	Char5	ann:
simpleType	Char12	ann:
simpleType	Char30	ann:
simpleType	Char50	ann:
simpleType	Char60	ann:
simpleType	Char255	ann:
simpleType	Char100	ann:
simpleType	TransactionType	ann:
simpleType	StatusType	ann:
simpleType	Number14	ann:

Exhibit 14. Example of Beaches XML Schema—Graphical view

3.2 Data Services (Data Services List v1.0 and Network Flow v1.0) Schema

The design of the Data Services XML schemas is consistent with the Garden of Eden architecture because it defines both elements and types globally. Exhibit 15 depicts an overview of the Data Services List XML schema. It clearly illustrates the global specification of named elements and types. The current XML schemas do not define any elements or types locally.

The Data Services XML schemas have accepted the default value of unqualified for the elementFormDefault construct, which hides namespaces in the XML instance document. This setting is consistent with the options for the Garden of Eden architecture. A target namespace is declared in the Data Services XML schemas, which enhances the reusability of global types.

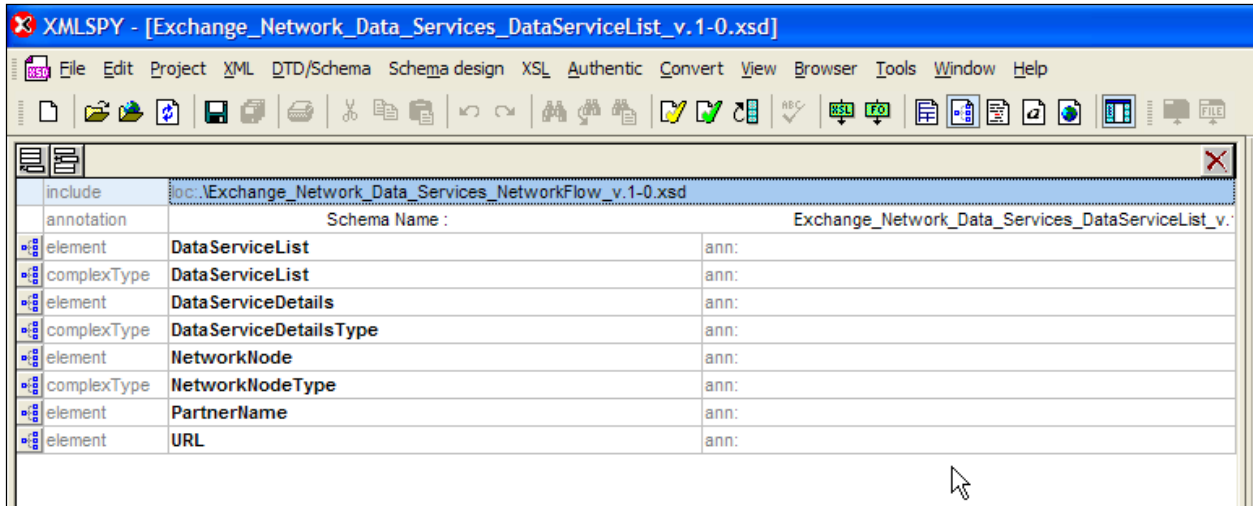


Exhibit 15. Example of Data Services XML Schema—Graphical view

3.3 Pacific Northwest Water Quality Exchange (v1.1) Schema

The design of the Pacific Northwest Water Quality Exchange (PNWWQX) XML schemas is most consistent with the Garden of Eden architecture because it defines both elements and types globally. Exhibit 16 depicts an overview of the PNWWQX Mailing Address XML schema. It clearly illustrates the global specification of named elements and types. The current XML schemas do not define any elements or types locally.

The PNWWQX XML schemas have accepted the default value of unqualified for the elementFormDefault construct, which hides namespaces in the XML instance document. This setting is consistent with the options for the Garden of Eden architecture. A target namespace has not been declared in the PNWWQX XML schemas as should be done when using the Garden of Eden architecture.

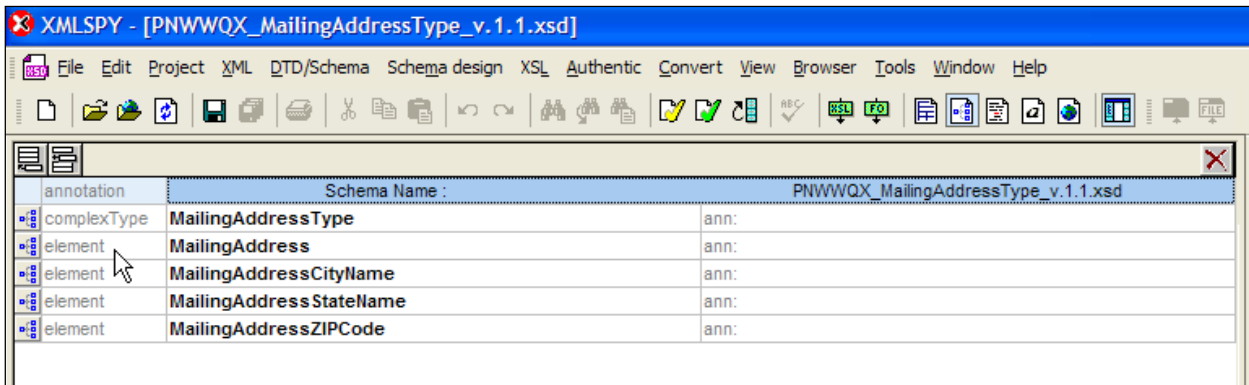


Exhibit 16. Example of PNWWQX XML Schema—Graphical view

3.4 Summary Review

Exhibits 17 and 18 summarize the schema architectural analysis described above. XML schemas developed using the *XML Design Rules and Conventions for the Environmental Information Exchange Network* dated September 2004 (e.g., Data Services and PNWWQX) will be structured most closely to the Garden of Eden architecture. Both the guidelines and the architecture require all elements and types to be globally defined. Additionally, the design guidelines require specification of a target namespace, which should also be declared when using the Garden of Eden architecture.

Characteristic	Beaches	Venetian Blind
Compact	No	No
Self-contained	No	No
Coupled	Yes	Yes
Reusable Elements	No	No
Reusable Types	Yes	Yes
Valid XML Documents	Single	Single
Option to Hide Namespace	Exposes namespace	Yes

Exhibit 17. Summary of Beaches XML Schema Architecture Review

Characteristic	Data Services	PNWWQX	Garden of Eden
Compact	No	No	No
Self-contained	No	No	No
Coupled	Yes	Yes	Yes
Reusable Elements	Yes	Yes	Yes
Reusable Types	Yes	Yes	Yes
Valid XML Documents	Multiple	Multiple	Multiple
Option to Hide Namespace	Hide namespace	Hide namespace	Yes

Exhibit 18. Summary of Data Services and PNWWQX XML Schema Architecture Review

APPENDIX A

References and Review Schema

References

- Chelsea Valentine, Lucinda Dykes, and Ed Tittel, *XML Schemas*, SYBEX, Inc, 2002.
 - Eve Maler, *Schema Rules for UBL – and Maybe for You*, XML Conference, December 12, 2002, <http://www.ebxml.org/presentations/ubl-schema-rules-xml2002.pdf>
 - *MediBiquitous XML Schema Design*, Version 1.2, April 30, 2004, MediBiquitous Technical Steering Committee, http://www.medbiq.org/technology/tech_architecture/xmldesignguidelines.pdf
 - Johan Peeters, *WSDL Tales From the Trenches*, August 5, 2003, <http://webservices.xml.com/pub/a/ws/2003/08/05/wsdl.html>
- X *XML Design Rules and Conventions for the Environmental Information Exchange Network*, US Environmental Protection Agency, September 2004.

Schema Reviewed

- Beaches (Notification v1.0 and Monitoring v1.1).
- Data Services v1.0.
- Pacific Northwest Water Quality Exchange v1.1 (includes 31 individual schema).