8

W3C XML Schemas and Reuse

Reuse is one of the most misunderstood concepts in information technology. Many technology executives assume that anything in their inventory (e.g., models, designs, application programs, databases, interfaces, and transactions) can be reused to advantage. They desire the economic benefits intuitive to the concept of reuse but may be unaware of the tactical costs required to achieve these benefits. Technology managers are challenged with aggressive deadlines and limited budgets and often avoid the modifications to their development methodology that are required to achieve effective reuse. Technology practitioners are rewarded for rapid delivery and the number of tested lines of code they develop. In other words, the goals that drive development of new applications are different from those that support reuse. However, all is not lost. The syntactical and functional capabilities afforded by W3C XML Schemas provide tremendous support and opportunities for metadata reuse. The first challenge is to develop a fundamental understanding of what reuse is all about.

From the perspective of information technology, *reuse* is a two-part process that first targets the engineering of information assets with the intent of being able to use these assets more than once and then harvesting these reusable technology assets:

- Reuse engineering
- Reuse harvesting

Fact:

XML schema reuse engineering is the set of practices, techniques, and activities required to engineer a W3C XML Schema or schema component with the specific intent of reuse.

Reuse engineering is the set of practices and techniques required to construct, engineer, and describe a technology asset with the specific intent to be reused. To engineer something for reuse, the data architect must consider how the information asset will be used initially as well as in the future. Reuse engineering includes an architectural approach in which development is not limited to meeting the initial objectives and requirements of a project. Also of importance is the determination whether there is a repeatable pattern exhibited by the technology asset that would support reuse in its current context (also known as *within-domain* or *domain-specific reuse*) and in other contexts (also known as *cross-domain reuse*).[1]

Both within-domain and cross-domain reuse are of value. The benefits of within-domain reuse are generally observed with the number of repeated reuse instances within that specific context. As a representative vocabulary, a W3C XML Schema provides a context (e.g., schemas representing a customer order transaction, a purchase order transaction, or a human resources payroll transaction). Within-domain reuse tends to limit the scope of the technology asset and therefore results in a somewhat less significant development effort. An example would be the initial use and then the further reuse of a defined data type for monetary amounts within a single W3C XML Schema vocabulary (Fig. 8.1).

Cross-domain reuse considers greater degrees of abstraction and generalization to allow for broad scale reuse and can result in a more significant development effort. Given that cross-domain reuse implies reuse opportunities in other contexts, the potential benefits can be greater than those of within-domain reuse. Cross-domain reuse of a W3C XML Schema also infers that the schema is defined as an external subschema that can then be referenced by and from within other W3C XML Schemas. Each of the referencing schemas represents a specific context (Fig. 8.2).

Fact:

XML schema reuse harvesting is a process that includes activities for identification, validation, and implementation of reusable W3C XML Schemas, subschemas, or schema components.

Reuse harvesting is a set of processes that focus on identifying opportunities for reuse, finding an information asset that is a candidate for reuse, validating the fit of that asset to the reuse opportunity (i.e., similar to pattern matching), and incorporating or referencing the reusable

[1] Karlsson, E-V. Software Reuse—A Holistic Approach. John Wiley & Sons, New York, 1995.

Figure 8.1

Within-domain reuse (internal to a schema).

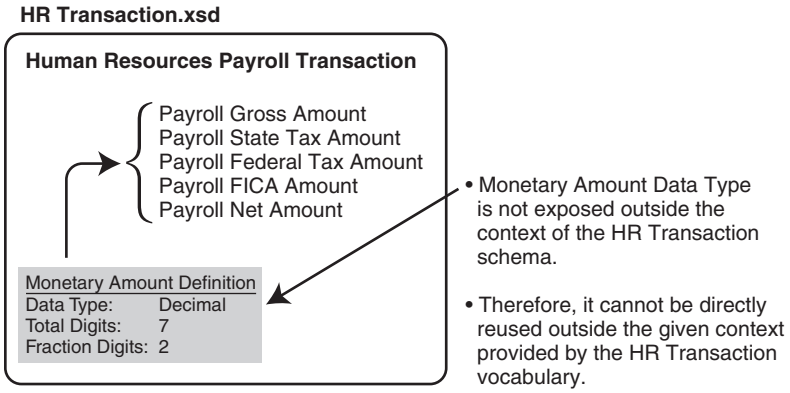
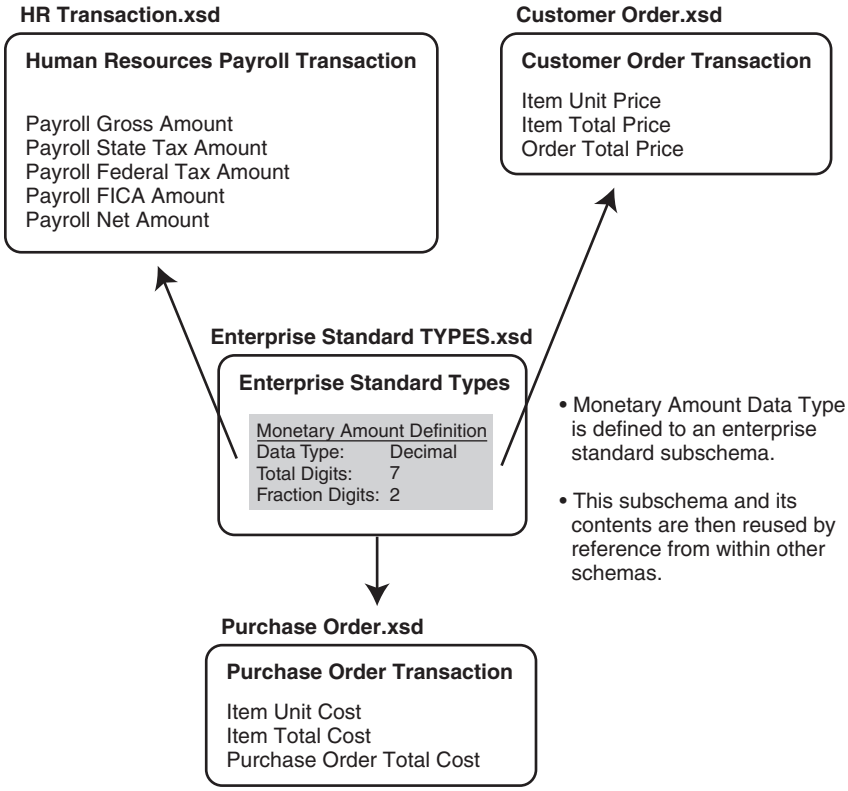


Figure 8.2

Cross-domain reuse (external to a schema).



information asset (e.g., engineering via reference and assembly rather than new development). The majority of initial reuse costs are attributed to reuse engineering. However, there are also some additional costs associated with reuse harvesting processes. Reuse harvesting costs are attributed to the additional effort required to identify a reusable asset, validating that the asset meets the project requirements, and implementing the reusable asset.

With each repeated instance of a harvested information asset the benefits of reuse become evident. The costs associated with the development and unit testing of the reusable information asset are to some degree offset by each reuse instance (e.g., future development cost avoidance as a result of reusing an information asset avoids having to redevelop and unit test a "new" asset many times). Reuse engineering is the initial part of the reuse process and where most development costs are incurred. Reuse harvesting is where reuse benefits are exploited and measured.

W3C XML Schemas present several types of reuse opportunities. The most fundamental reuse opportunities are aligned with the concept of within-domain reuse. A *domain* is a particular application system, set of related business functions, or a defined context. A W3C XML Schema-based vocabulary is a set of containers that represent a domain or some part of a domain. Containers (e.g., XML elements and attributes), groups of containers, and types (e.g., metadata characteristics and custom data types) can also be defined for reuse within a single W3C XML Schema vocabulary. Enterprise standard containers, structures, and metadata definitions can be engineered as subschemas. W3C XML Schemas can also be engineered as assemblies of other externally defined W3C XML Schemas and schema components. A data architect can engineer a W3C XML Schema vocabulary to reference the standard subschemas, rather than individually coding them.

Internal W3C XML Schema Reuse

Reuse within a W3C XML Schema can take several forms. Within a W3C XML Schema, elements can be either locally or globally defined. Locally defined element containers are defined by name at the point in the schema where they are declared (Fig. 8.3). These locally defined elements and attributes are generally not reusable outside of their point of declaration. Alternatively, globally defined elements represent a funda-

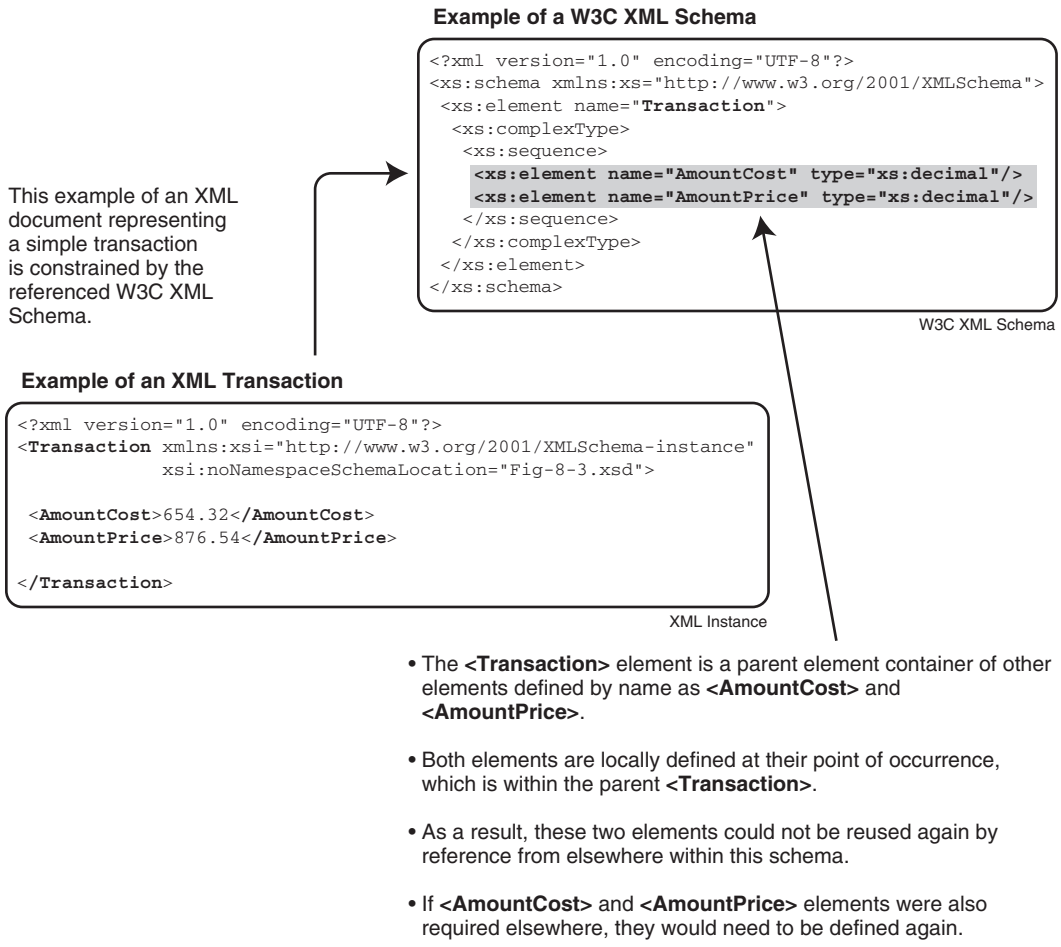


Figure 8.3

Local W3C XML Schema element definitions.

mental form of reuse. They are defined to the overall schema rather than at a specific point of occurrence. Globally defined elements are containers that can be defined with the intent of being reused by reference from other places in the schema (Fig. 8.4).

It may be advantageous or necessary to define and reference a collection of similar or related elements rather than individual elements. This can be accomplished using the W3C XML Schemas syntax for

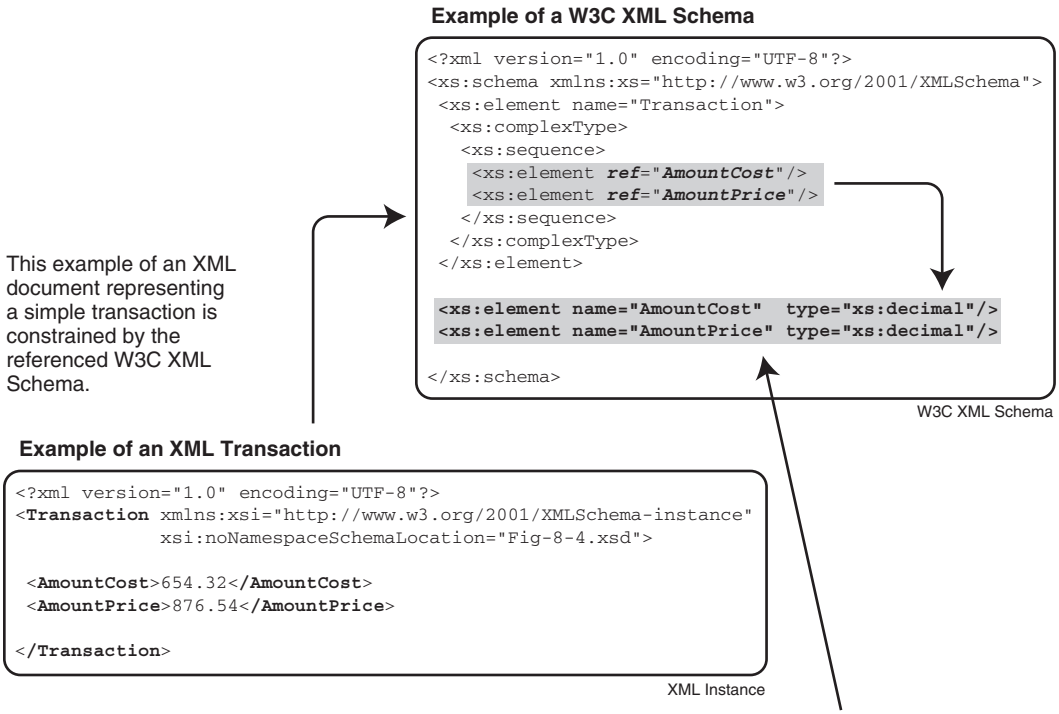


Figure 8.4

Global W3C XML Schema element definitions.

either a "complexType" or a "group." A *complexType* is a defined set of element containers that when named may be reused by reference (using the "extension" syntax within an element). Similarly, a W3C XML Schema *group* is a defined set of containers (also known as an element model group) that are specifically defined with intent of being reused.

Recommendation:

Unless intentionally prohibited from being reused, all XML element containers should be defined globally, allowing for reuse by reference.

Recommendation:

Unless there are obvious advantages to using global "complexTypes," collections of related element containers intentionally targeted for reuse should be globally defined as "groups."

The previous example of two globally defined amount elements could be defined and reused as a single collection of elements rather than as two individual element references. A complexType can be defined globally to the schema and then referenced by name as the extension of an element (Fig. 8.5).

Like a complexType, a group is also a collection of element containers. The concept of a group is similar to that of a complexType, but a slightly different syntax is used. A group is defined with the specific intent of being reused by reference. While a complexType can be defined to be reused as an extension, it can also be defined locally and excluded from reuse. The content of a group can include a list of elements or complexTypes (Fig. 8.6).

Both a complexType and a group allow for a compositor. A *compositor* specifies the sequence and selective occurrence of the containers defined within a complexType or a group. Compositors include sequence, all, and choice (Table 8.1). The *sequence* compositor declares that the individual elements defined within a complexType or a group must occur in the same order in the corresponding XML document. The *all* compositor declares that all of the elements contained within the complexType or group must be present in the corresponding XML document, but may occur in any order. The *choice* compositor states that only one of the elements defined to a complexType or group may occur in the corresponding XML document.

In addition to a compositor, most elements defined or referenced by complexTypes and groups may repeat. The degree of cardinality can be specified for repeating elements using the minOccurs and maxOccurs attributes. The *minOccurs* attribute defines the minimum degree of cardinality or the minimum number of occurrences for a referenced repeating element. A minOccurs attribute with a value of zero (i.e., "0") denotes that the element is optional. The *maxOccurs* attribute defines the maximum degree of cardinality or the maximum number of occurrences for the referenced element. Both the minimum and maximum degree of cardinality can be specific (e.g., a specific value such as "3" or "200"). The maximum degree of cardinality may also be defined as infinite (i.e., a value of "unbounded"). It is important to note that in some cases, the rules of the compositor may constrain cardinality and repeating elements (e.g., the "all" compositor does not allow for repeating elements).

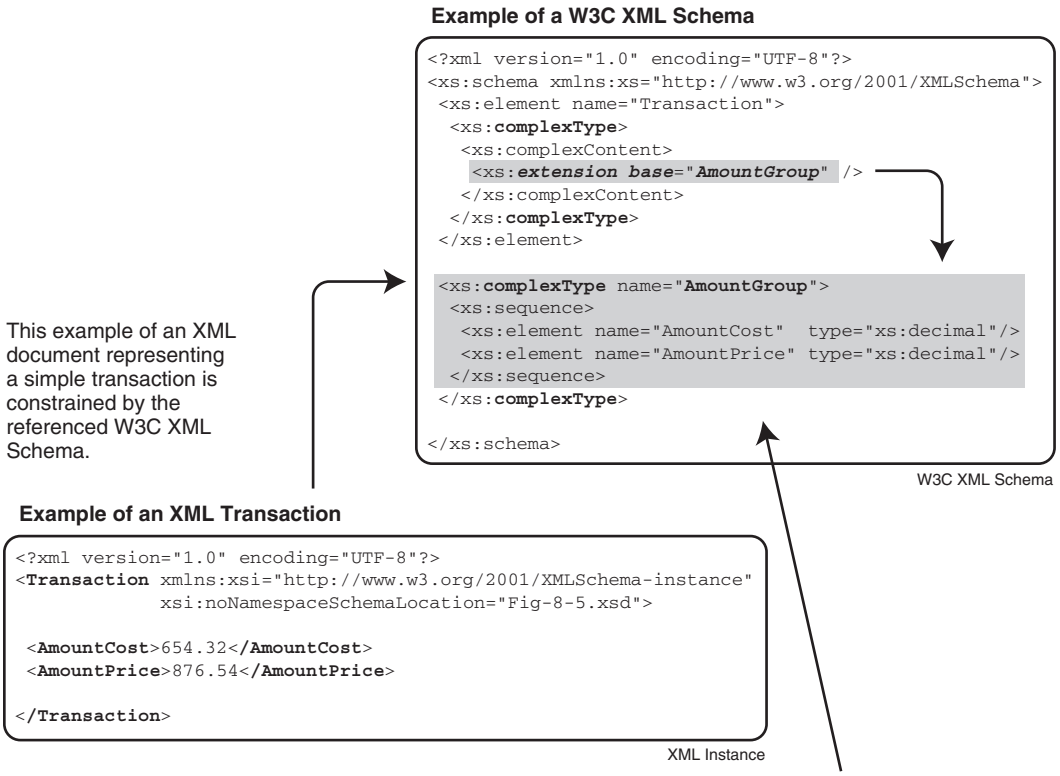


Figure 8.5

Global W3C XML Schema complexType definition.

As described in Chapter 4 (W3C XML Schema Types vs Database Data Types), W3C XML Schemas provide extensive data type support, including numerous built-in and derived data types that can be applied as a constraint to any element or attribute. Custom data types can also be defined by creating a simpleType with one of the supported

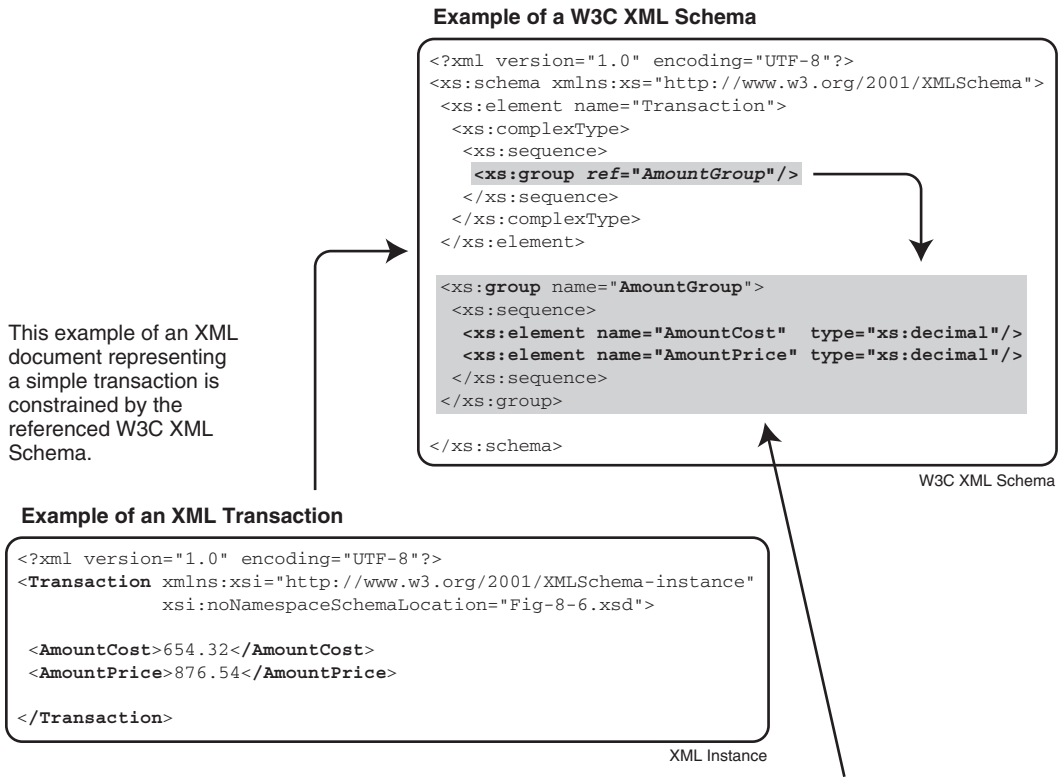


Figure 8.6

Global W3C XML Schema group definition.

Table 8.1 Compositors

Compositor type	Description
sequence	Child elements must occur in the corresponding XML document in the listed order. Specified cardinality of child elements may determine whether a specific child element must occur or may occur and the degree to which it repeats.
all	All child elements must occur in the corresponding XML document. They may occur in any order but cannot repeat (i.e., a degree of cardinality of maxOccurs greater than one cannot be specified).
Choice	Any one (but only one) of the child elements may occur.

Technique:

W3C XML Schema simpleTypes present a powerful method for defining enterprise standard data types and allowable value constraints for element and attribute containers.

data types as a base and adding constraining facets. The ability to define custom data types is a powerful form of reuse. Many organizations have a set of enterprise standard data element definitions. When new data elements of the same type are defined, the data architect is required to apply the enterprise standard metadata characteristics (e.g., data type, length, decimalization, and allowable values).

Common examples of enterprise standard data elements include those for monetary amounts, identifiers (e.g., as in primary key or unique identifiers), text descriptions, and standard code values. These same enterprise standards can be defined as custom data types and implemented as W3C XML Schema simpleTypes. Similar to an element, a *simpleType* can be defined locally and referenced by an element or attribute, or it can be defined globally to the schema with the intent of being reused by reference. When a custom data type is defined using a simpleType, it will include a declared W3C XML Schema data type and any applicable facets (Fig. 8.7). The examples of monetary amount custom data types are defined as a decimal data type, with totalDigits and fractionDigits facets. A simpleType is not limited to monetary amounts or decimal data types. Any of the supported W3C XML Schema data types and facets can be applied.

Reuse of elements, groups of elements, and data types within a single W3C XML Schema is a powerful capability. However, reuse of these internal constructs outside the context of the defining W3C XML

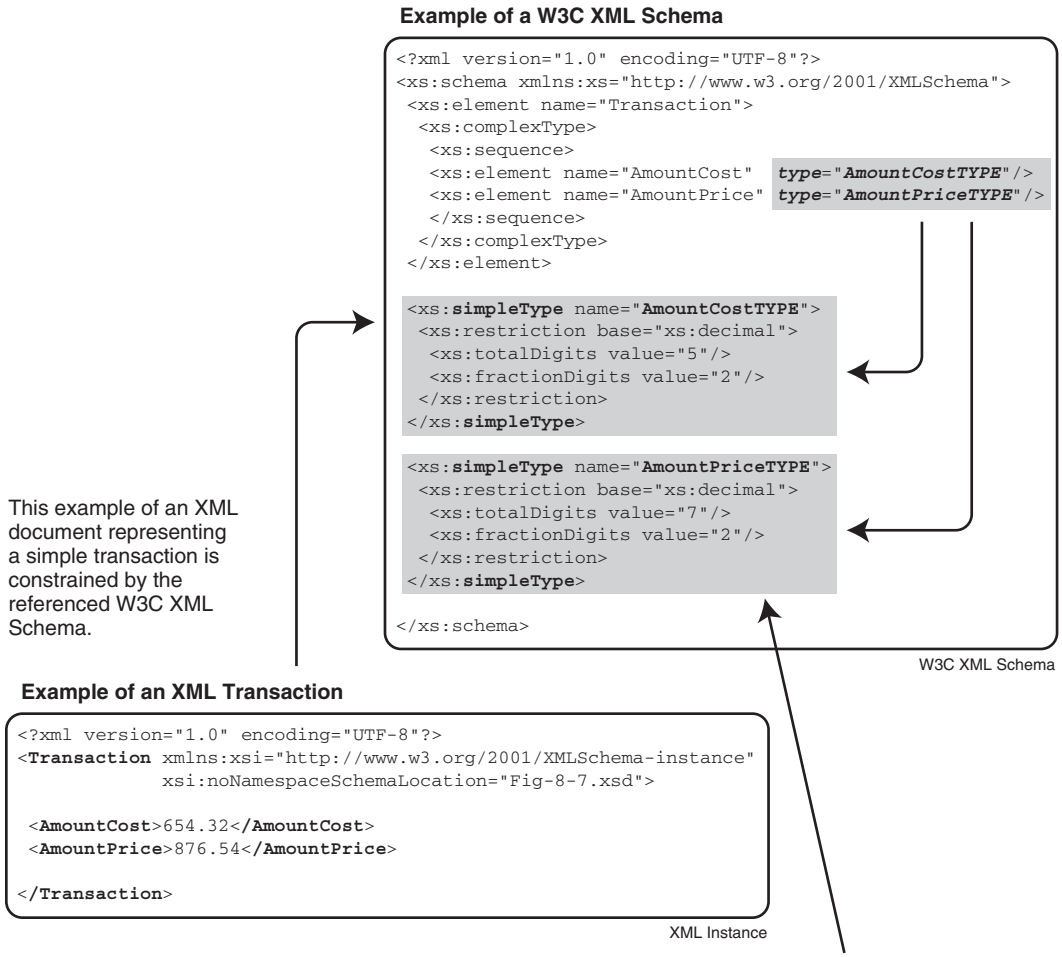


Figure 8.7

Global W3C XML Schema simpleType definitions.

Schema may be complex and in some cases not possible. An attempt to reuse these same elements, groups, and types within other schemas would probably require repetition of the W3C XML Schema syntax in each of the other schemas (as a form of copy and paste). Although there is some advantage to this approach, there are also increased costs and risks. The obvious advantage is that enterprise standard elements and data types are to some degree proliferated and reused. However, the cost of maintaining each of these schemas will escalate. Also, the potential for errors resulting from future modifications that are not synchronized to all copies increases over time. Reuse of externally defined W3C XML Schemas presents a more effective approach.

External W3C XML Schema Reuse (Component Subschemas)

From the perspective of reuse harvesting, W3C XML Schemas also provide extensive capabilities in the area of cross-domain reuse. Leveraging W3C XML Schemas for cross-domain reuse implies that a schema (or subschema) can represent a repeatable pattern, and it can be used in different contexts and by different applications. The method of implementation is to define modular W3C XML Schemas as external subschemas. Other W3C XML Schemas (potentially of varying contexts) can then reference and reuse the elements, groups, and data types of the subschemas. The referencing schema vocabularies only need to define the containers, structures, and types exclusive to their specific context. As a form of development by assembly, reused elements, groups, and data types defined to the subschemas are then "referenced" (Fig. 8.8).

Fact:

Reuse of a W3C XML Schema or subschema is a conceptual form of development by assembly. The primary W3C XML Schema is assembled by including references to the contents of other externally defined W3C XML subschemas.

Schema reuse engineering is the process of developing W3C XML Schemas (or subschemas) with the intent of reuse. Each of the external subschemas must be defined in a manner that promotes broad-scale reuse, yet ensures adherence to enterprise structures and metadata standards, which is where the data architect plays a significant role. The identification of candidate data elements, structures, and data types is similar to traditional data architecture practices. The most common opportunities for engineering reusable subschemas include the following patterns:

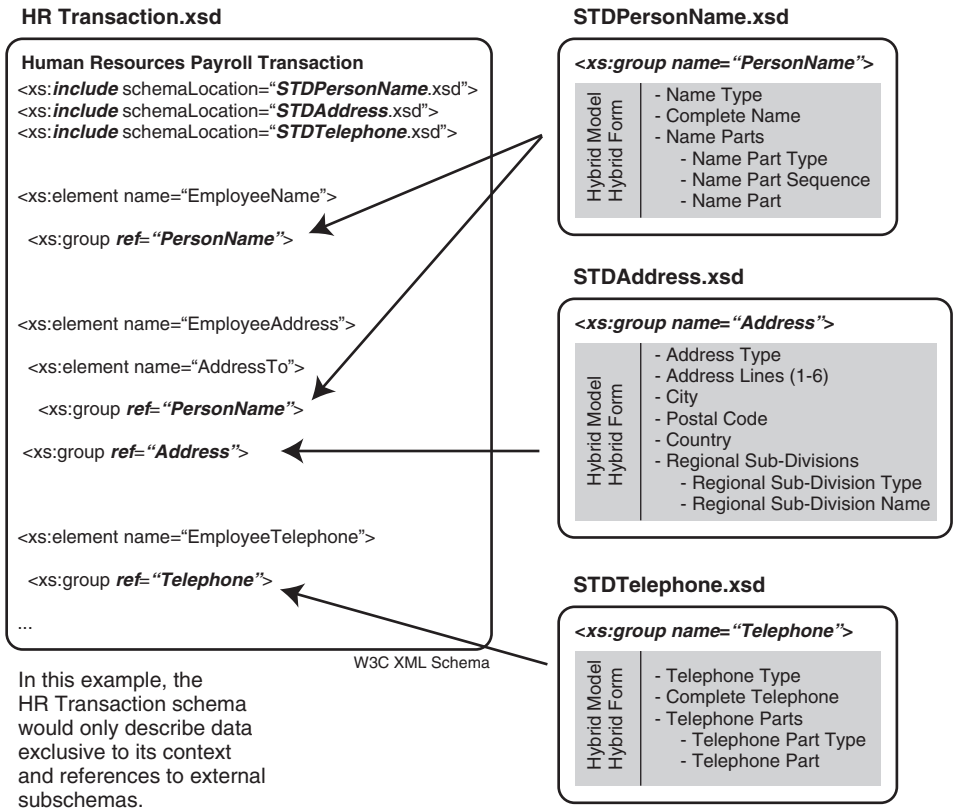


Figure 8.8

W3C XML Schema reuse by reference (e.g., "assembly").

- Highly standardized data structures (e.g., person name, postal address, telephone number, product family structures, and geographic structures)
- Standard codes and allowable values
  - Enumeration lists of internal enterprise standard code values
  - Enumeration lists of international and industry-related encoding standards (e.g., U.S. state abbreviations, country codes, and currency codes)
- Standard data types (e.g., enterprise standard data types for monetary amounts, text descriptions, and identifiers)

Fact:

The most common types of reusable subschemas include standard structures, standard codes and allowable values, and custom data types.

The W3C XML Schema syntax allows for subschemas to define elements, groups, complexTypes, and simpleTypes that can be included and referenced by other W3C XML Schemas. The concept is roughly analogous to the concept of a COBOL Copybook "include," in which data structures and file definitions are defined according to enterprise standards and are then included by reference from within other COBOL source programs. This helps to ensure that not only is there a high degree of reuse (resulting in development cost avoidance) but also enterprise structures and metadata standards are supported.

One of the greatest challenges for today's business enterprise is the diversity of data resulting from global e-commerce. When varied international cultures and locales are considered, a customer's name presents an interesting problem. As an integration transaction technology, XML can be used to exchange person name information between tactical enterprise systems such as human resources, order processing, and customer service. Similar name information can be exchanged with and imported into strategic systems such as marketing, customer relationship management, and the data warehouse. Functional uses of person name data imply repeatable patterns. However, the granularity and data formats for person name can vary from system to system. Cultural variations of person name introduce even greater complexity.

In the United States, a common format for person name is simply the combination of first, middle, and last names. However, these name parts support international and cultural variations minimally (if at all). In many countries the descriptive data element names of first, middle, and last are not applicable. Variations may include given name, surname, family name, and additional name.[2] Also, the order or sequence of name parts may differ (e.g., in some cultures the family or last name precedes the given or first name). Also, many person names are not limited to three name parts and may also include connectives (e.g., "von" or "de").

Another important aspect of a person's name is how it will be used or processed. Application programs that generate correspondence, mailings, and delivery labels may only require a single complete name that is composed of all name parts in the desired sequence of the individual.

[2] Bean J. XML Globalization and Best Practices. Active Education, Colorado, U.S., 2001.

Alternatively, other applications such as marketing, global customer relationship management, and data warehousing may require the granular parts of an individual's name for the purposes of sorting and grouping. A highly reusable name structure will support multiple uses of person name data.[3]

A W3C XML Schema describing the structure and format of a person name can be engineered with the intent of broad scale reuse. Other W3C XML Schemas can then reference and reuse the person name schema as a subschema, rather than including individually coded name structures parochial to their own processing. Strategic applications such as global customer relationship management (G-CRM) often act as a point of integration for customer information. In this case, the ability to accept, validate, import, and process international customer names and name parts becomes critical to success. Engineering a modular person name subschema requires that the structure can support varied processing and multiple data formats and still apply standards and constraints as necessary (Fig. 8.9).

A person name schema designed for broad scale, cross-domain reuse must support different forms of processing, as well as different structural formats (Fig. 8.10). When XML is used to describe data that will be imported into a database structure, the format, granularity, and taxonomy of that structure become important. If the data will be exchanged between systems that have different functions and purposes or the structure, form, and content of the data are variable, a flexible architecture is of value. Determination of the intended use, potential future use, variability, and granularity of the data requires an architectural perspective. To engineer a reusable schema for person name, the data architect must determine how the data will be used, structured, identified, and described.

An Architectural Approach to Reuse Engineering

W3C XML Schema reuse engineering includes activities specific to the design and development of highly reusable schemas. In this regard, the data architect can play a significant role. To ensure that a W3C XML

[3] Bean J. Engineering Global E-Commerce Sites. Morgan Kaufmann, San Francisco, 2003.

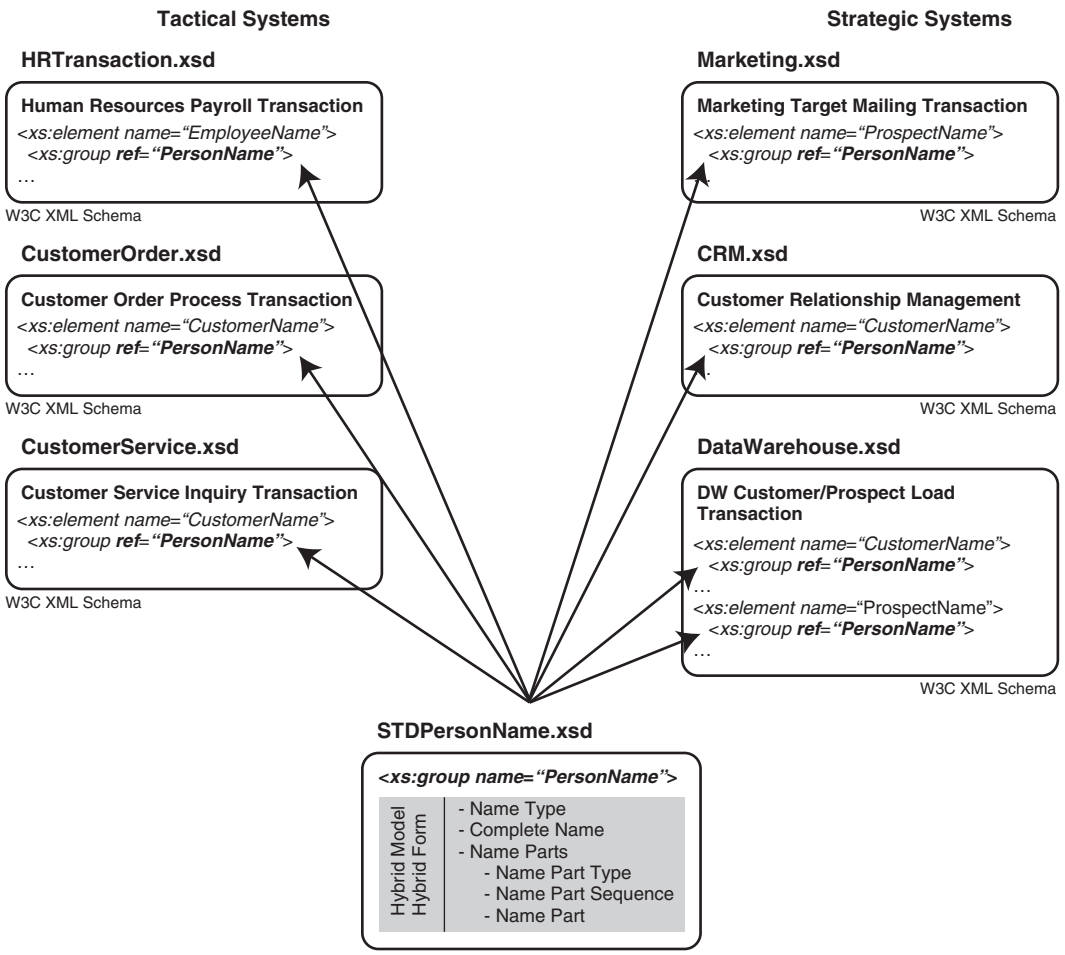


Figure 8.9

Varying use of an enterprise standard subschema for person name.

Schema is intended for broad-scale, cross-domain reuse, the data architect must apply an architectural perspective. This perspective must include several important architecture requirements:

- Schema identification (a file name allowing for later "reuse harvesting")

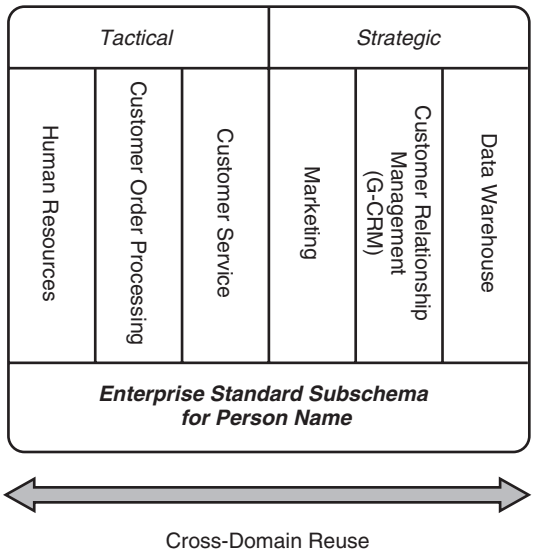


Figure 8.10

Broad-scale, cross-domain reuse of subschema for person name.

- Version management
- Variable use and processing (e.g., how the XML document and content will be "consumed")
- Application of structure models
- Adaptation of architecture container forms

Recommendation:

The external file name of a reusable W3C XML Schema (or subschema) should be intuitive, be of reasonable specificity, be of mixed character case or camel case, eliminate spaces, and be a maximum of 32 characters in length.

Identification and naming of a schema is critical to reuse. The externally defined W3C XML Schema must be named in a manner that is intuitive and simplifies the identification and harvesting of reusable schemas. The name should also align with enterprise naming standards. Similar to the taxonomy techniques applied to individual XML container names (e.g., elements and attributes), the external file name of the entire W3C XML Schema is important. Even though the schema name might be descriptive, if it is too specific or limited to a particular system, application, or process, reuse potential will be limited. Alternatively, a schema name that is too abstract can be misleading and result in inaccurate identification of reuse candidates. A schema name of "Name" is far too abstract. Name

might imply a person's name or might imply a business name, place name, trade name, or a similar form of naming. However, a schema name of "PersonName" is descriptive, depicts a context, and is generally intuitive.

At the physical level, a W3C XML Schema is a Unicode-encoded text file (most often as UTF-8 or an ASCII text file). The external name of that file must conform to the constraints of the file management and operating system (e.g., maximum character length, exclusion of special characters, character case, and support for white space). As a general guideline, file names should be no longer than 32 characters. Although somewhat arbitrary, a file name length of 32 characters is supported by most server-based operating systems (such as Windows). If the platform on which the subschema will be stored has a more restrictive maximum length, it should be used.

As to white space, some operating systems allow blanks or spaces within a file name. However, it is recommended that all white space should be removed. Also, where possible the file name should use mixed-case characters (e.g., applied in the form of camel case as described for element and attribute taxonomy). When combined, these naming techniques will result in descriptive and intuitive file names for W3C XML Schemas.

Recommendation:

The external file name of a reusable W3C XML Schema (subschema) should include a version. The version may be prefixed or suffixed depending upon enterprise standards.

Version management is rarely the responsibility of the data architect. However, it plays a significant role in reuse engineering as part of the schema name (the external file name). As with any technology asset, evolution and change will be experienced. Although the intent of reuse engineering is to develop a reusable schema structure that is also an enterprise standard, over time, modifications will be required. Being able to identify different versions from the external file name becomes critical. During the reuse harvesting process the developer or data architect will need to easily determine the current version of a candidate schema. Including version identification as part of the external schema name is of significant value. The scheme used for versioning (e.g., numbers, characters, and combinations of version and release) should adhere to enterprise standards. The version should be included as part of the external file name (usually as a prefix or suffix).

Recommendation:

Reusable W3C XML subschemas should include some form of classification or type in the name (e.g., "STD," "CODES," or "TYPES").

In addition to the version and the schema name, it may be valuable to describe the schema by a classification or type. The most common types of reusable W3C XML

Schemas are either standard structures, sets of code values, or data types. The ability to identify the type of reusable schema can simplify the reuse harvesting process. As examples, each of the different schema types could be classified as one of the following:

- STD—representing enterprise standard structures
- CODES—representing enumeration lists of standard code values
- TYPES—representing enterprise standard data types

The classification for each schema type should be included in the schema external file name. When developers or data architects are searching for reusable schemas, they can limit their search to a specific type. Terminating the W3C XML Schema file name is the file type or extension (when supported by the file management and operating system). The recommended file type or extension for a W3C XML Schema is "xsd" (Fig. 8.11).

Unlike traditional data modeling, XML transactions used for data exchanges between systems can include denormalized data, abstract data element names, and even duplicate data. When the objectives of the transaction are a combination of data exchange between disparate systems and enterprise integration, the ability to describe the transaction content in different forms, by different names, and with a structure

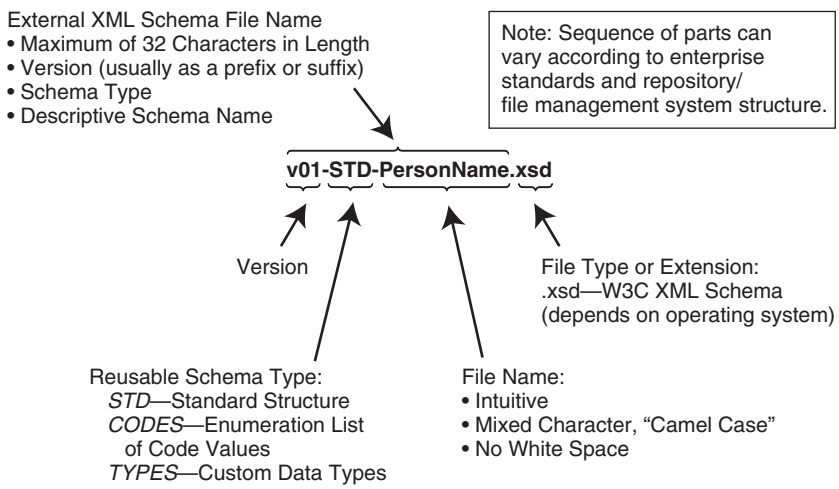


Figure 8.11

Example of a format for (external) W3C XML Schema file names.

that can be easily transformed into different formats is requisite. An example is when data of the same general context will be exchanged between different systems but with variation in how those data will be used and processed. To support the varied use and processing of these target applications, it may be necessary to describe multiple structures or formats for the same data. The data architect is challenged with extending traditional data architecture skills to address this paradigm.

Recommendation:

When used to describe a transaction for data exchange or enterprise integration, a reusable W3C XML Schema should incorporate several different structures and formats to address the potential for varied use and different application processes.

Consider a transaction carrying a customer's name. The person name data used by an order processing system and a customer relationship management system may need to support different structures and processes. The order processing system may require a customer's complete name (i.e., the concatenated set of all name parts). Typical uses of this data structure include order correspondence and documentation, the "Address To" line of the order delivery information, and use for customer contact. Alternatively, a G-CRM or marketing application may require a highly variable structure to utilize individual parts of the customer's name as well as the complete name. This type of processing can include sorting and grouping of customer data according to a sequenced set of name parts (Fig. 8.12).

To address both types of person name processing, a single reusable W3C XML Schema should be engineered to incorporate two different name structures. One structure defined to the schema is a single data element for a complete person name. The other is a decomposed, flexible structure of individual data elements for the parts of a globally diverse name. If both structures are defined as optional rather than mandatory, this single schema can be used to describe and constrain different XML documents and transactions (Listing 8.1).

Listing 8.1

W3C XML Schema for different forms of person name data.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="CustomerData">
      <xs:complexType>
```

Listing 8.1 *Continued*

```
      <xs:sequence>
        <xs:element ref="CustomerName"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="CustomerName">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="PersonNameComplete" minOccurs="0"
        maxOccurs="1"/>
        <xs:group ref="PersonNameParts" minOccurs="0"
        maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="PersonNameComplete"
  type="PersonNameCompleteTYPE"/>

  <xs:group name="PersonNameParts">
    <xs:sequence>
      <xs:element name="PersonNamePrefix" type="PersonNamePartTYPE"
      minOccurs="0"/>
      <xs:element ref="PersonNamePart" minOccurs="1"
      maxOccurs="unbounded"/>
      <xs:element name="PersonNameSuffix" type="PersonNamePartTYPE"
      minOccurs="0"/>
    </xs:sequence>
  </xs:group>

  <xs:element name="PersonNamePart">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="PersonNamePartTYPE">
          <xs:attribute name="sequence" use="required"
          type="xs:byte"/>
          <xs:attribute name="type" type="PersonNamePartTypeCODE"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
```

Listing 8.1 *Continued*

```
    <xs:simpleType name="PersonNameCompleteTYPE">
      <xs:restriction base="xs:string">
        <xs:maxLength value="80"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="PersonNamePartTYPE">
      <xs:restriction base="xs:string">
        <xs:maxLength value="20"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="PersonNamePartTypeCODE">
      <xs:restriction base="xs:string">
        <xs:enumeration value="Family"/>
        <xs:enumeration value="Last"/>
        <xs:enumeration value="Given"/>
        <xs:enumeration value="First"/>
        <xs:enumeration value="Surname"/>
        <xs:enumeration value="Additional"/>
        <xs:enumeration value="Middle"/>
        <xs:enumeration value="Religious"/>
        <xs:enumeration value="Other"/>
      </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

When combined with a prototype XML document, the potential for varied use and different types of application processing can also help to identify applicable XML structure models and architectural container forms. The element used to contain the complete name combined with the set of name part elements is a good fit for a vertical structure model. Name part elements should also include attributes for sequence (the preferred order in which name parts should occur), as well as a type for each name part, which implies a horizontal structure model. The ability to reuse a person name structure in different contexts (e.g., customer order processing, employee, and customer contact) infers that this W3C XML Schema is a good fit for a component structure model. Additionally, the data types and allowable values of the name elements could also

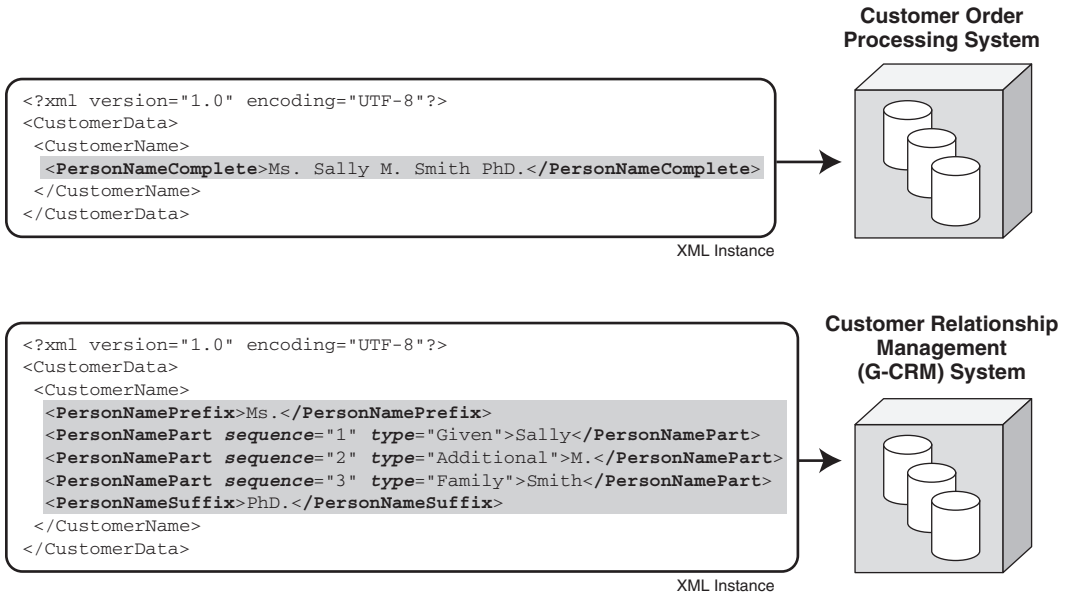


Figure 8.12

Different XML transactions containing person name data.

become enterprise metadata standards that are implemented as other modular component structure models. When considered in total, the person name schema is an excellent candidate for a hybrid structure model, which incorporates advantageous characteristics of the other model types.

Customer relationship management applications will often need to sort, group, and select from customer data by customer name. Because full names may be specified in varied form, using a complete person name element is largely inadequate for this type of processing. Name prefixes and suffixes are rarely used for sorting and may complicate grouping of like names. As a result, prefixes and suffixes are a good fit for specifically named rigid container forms. Individual name parts present an effective use of name data for sort, group, and select processes. Processing names of international customers, for which the number, sequence, and type of name parts can vary widely, requires an abstract container form that can dynamically expand and contract to fit culturally diverse name formats (Fig. 8.13).

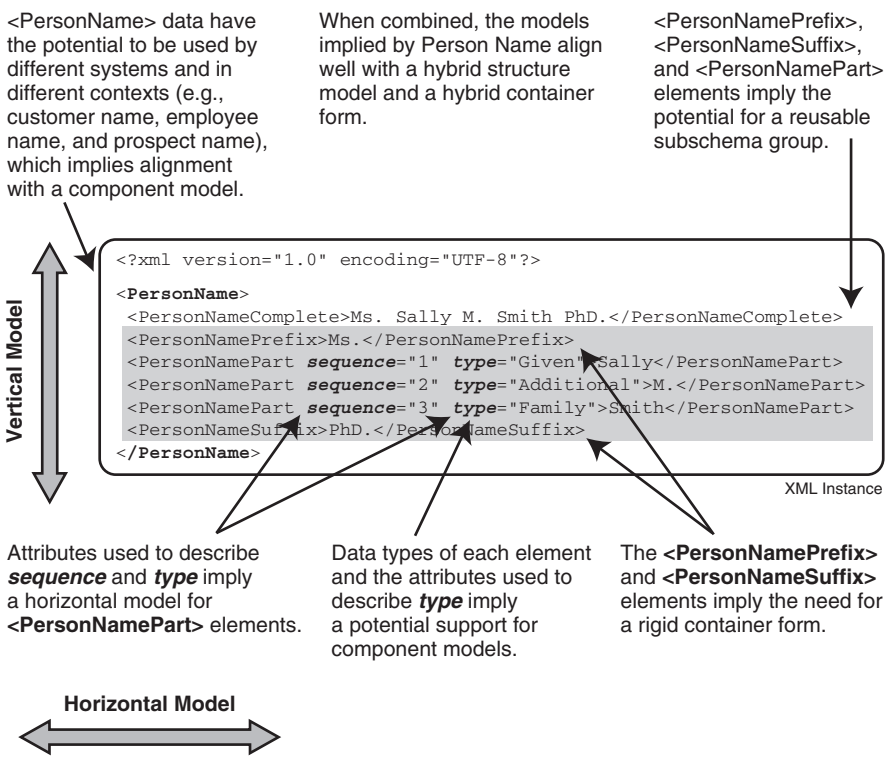


Figure 8.13

Structure models and container forms implied by person name data.

Custom data types and enumerated lists of standard code values also present the opportunity to further decompose the person name schema. Subschemas to represent the enterprise standard person name structure, data types specific to person name data, and applicable standard code values (e.g., Person Name Part "types"), are all excellent candidates for reuse (Fig. 8.14).

The <PersonNameComplete> element is defined by a string data type with a maximum allowable length. The <PersonNamePrefix>, <PersonNamePart>, and <PersonNameSuffix> elements are also defined by a string data type but with a shorter maximum length. Both types are declared as simpleTypes and are included in a subschema that contains all data types for person name data. Similarly, the "type" attribute of the <PersonNamePart> element is described by a set of

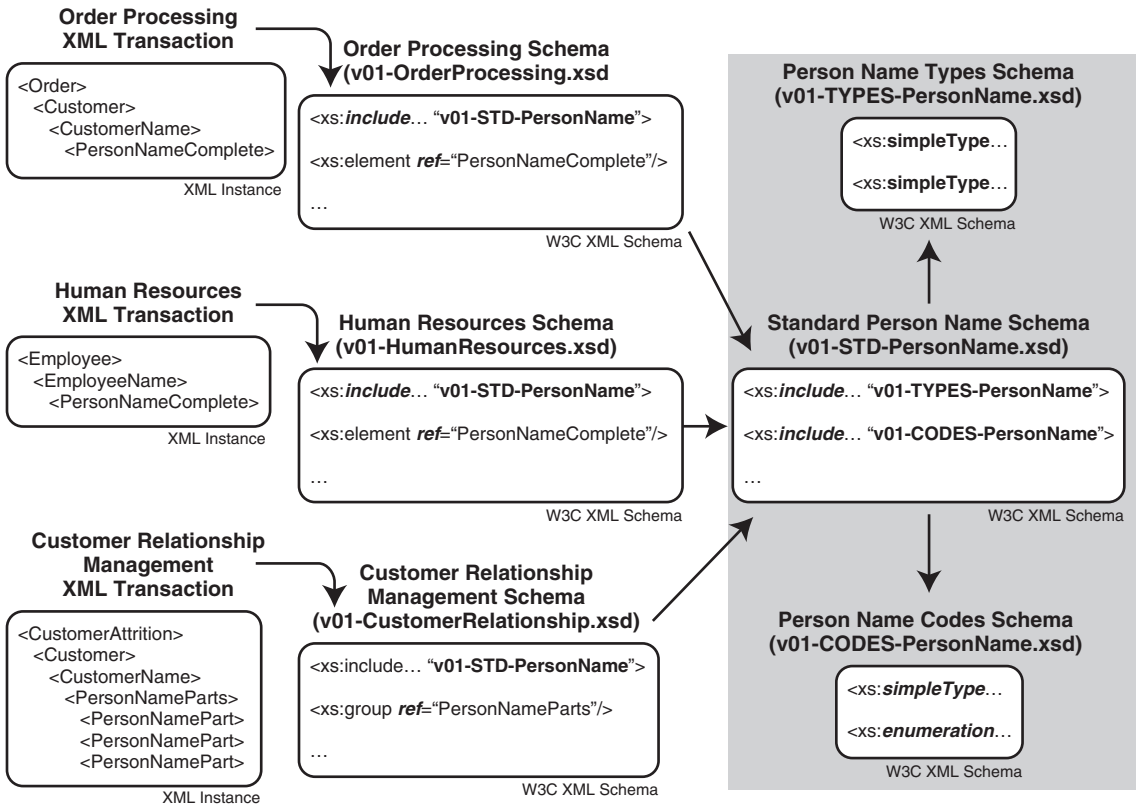


Figure 8.14

Reference of component subschemas.

standard code values. These values are declared as an enumeration list and are defined to a subschema containing all codes and allowable values for person name data. The result is a set of three component schemas. The Person Name Standard Structure Schema references and uses both the Person Name Types and Person Name Codes schemas. Each of the schemas can be reused in whole or in part:

- Person name structure schema (Listing 8.2)
- Person name types schema (Listing 8.3)
- Person name codes schema (Listing 8.4)

Listing 8.2

W3C XML Schema for a person name structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:include schemaLocation="v01-TYPES-PersonName.xsd"/>
    <xs:include schemaLocation="v01-CODES-PersonName.xsd"/>

    <xs:element name="PersonNameComplete"
    type="PersonNameCompleteTYPE"/>
    <xs:element name="PersonNameParts">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="PersonNamePartsGroup"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:group name="PersonNamePartsGroup">
      <xs:sequence>
        <xs:element name="PersonNamePrefix"
        type="PersonNamePartTYPE" minOccurs="0"/>
        <xs:element ref="PersonNamePart" minOccurs="1"
        maxOccurs="unbounded"/>
        <xs:element name="PersonNameSuffix"
        type="PersonNamePartTYPE" minOccurs="0"/>
      </xs:sequence>
    </xs:group>

    <xs:element name="PersonNamePart">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="PersonNamePartTYPE">
            <xs:attribute name="sequence"
            use="required" type="xs:byte"/>
            <xs:attribute name="type" type="PersonNamePartTypeCODE"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

Listing 8.3

W3C XML Schema for person name types.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:simpleType name="PersonNameCompleteTYPE">
      <xs:restriction base="xs:string">
        <xs:maxLength value="80"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="PersonNamePartTYPE">
      <xs:restriction base="xs:string">
        <xs:maxLength value="20"/>
      </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

Listing 8.4

W3C XML Schema for descriptive person name codes.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=
"http://www.w3.org/2001/XMLSchema">

    <xs:simpleType name="PersonNamePartTypeCODE">
      <xs:restriction base="xs:string">
        <xs:enumeration value="Family"/>
        <xs:enumeration value="Last"/>
        <xs:enumeration value="Given"/>
        <xs:enumeration value="First"/>
        <xs:enumeration value="Surname"/>
        <xs:enumeration value="Additional"/>
        <xs:enumeration value="Middle"/>
        <xs:enumeration value="Religious"/>
        <xs:enumeration value="Other"/>
      </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

Syntax for Referencing a Component W3C XML Schema

For a W3C XML Schema to be reused, it must be referenced by another schema. As previously described, the process of referencing a schema is conceptually similar to the COBOL Copybook include syntax. When the primary W3C XML Schema is validated during the parsing process, the parser will locate referenced subschemas as resources, include the declarations of the referenced subschemas, and then resolve referenced constructs. Subschema references are not limited to a single nesting layer or reference. A subschema can reference another subschema, which can reference another subschema, and so on. There is no specifically documented limit to the number of nested schema references, but caution is advised. Significant nesting has the potential to slow the parsing process. The W3C XML Schema syntax for referencing one schema (e.g., a subschema) from within another can take on any of three syntactical forms:

- Include
- Redefine
- Import

Recommendation:

When the primary or referencing W3C XML Schema is of a single context (e.g., namespace), the "include" syntax is the desired form for referencing an externally defined subschema.

When the primary W3C XML Schema is representative of a single context, the preferred method of referencing a subschema is to use the "include" syntax. In this case, the included schema becomes part of the overall namespace of the primary or referencing schema (e.g., the "master" schema). When the primary W3C XML Schema is composed of several contexts and the desired subschema reference must match one of those contexts, the "import" syntax may be a better fit. The import syntax targets a specific context using a namespace to provide uniqueness. The "redefine" syntax is similar to the include syntax (e.g., targeting a single context and namespace). However, the redefine syntax includes a referenced subschema and allows for modification of referenced constructs. Regardless of the chosen syntax, references to desired subschema constructs must be made and resolved.

The include syntax incorporates both a reference to the name and location of the desired subschema and one or more internal references to components declared in the subschema (Fig. 8.15). The W3C XML Schema syntax for a schema include is simple and intuitive. The primary

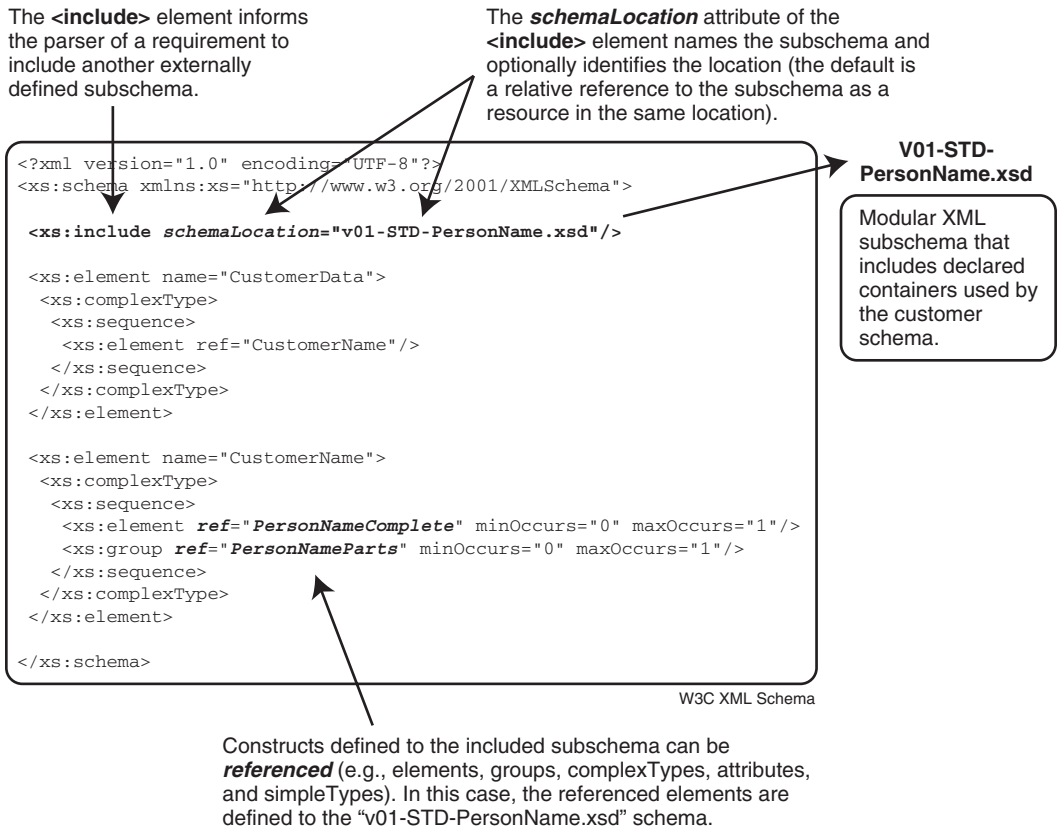


Figure 8.15

W3C XML Schema include syntax.

XML Schema uses an "include" element in the primary or referencing schema (e.g., <xs:include>). The "schemaLocation" attribute of the include element is valued with the name and location of a referenced subschema. The value of the schemaLocation attribute can be a relative resource location or an absolute resource location. A *relative resource location* requires that the referenced subschema is located in the same place or directory as the primary or referencing schema. An *absolute resource location* not only names the referenced subschema as a resource but also identifies a specific location for the subschema (e.g., the server, the directory, the node path, or a similar location of the referenced

Figure 8.16

W3C XML Schema import and redefine syntax.

XML Schema "Import" Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:import schemaLocation="schema-location-goes-here"
      namespace="namespace-goes-here"/>


</xs:schema>
```

W3C XML Schema

XML Schema "Redefine" Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:redefine schemaLocation="schema-location-goes-here">

  ... revised complexTypes, simpleTypes, groups go here
  ...
  ...

 </xs:redefine>

</xs:schema>
```

W3C XML Schema

subschema). During the validation process, the parser identifies the include element, interrogates the schemaLocation attribute, searches for the named resource, and includes or brings in the referenced schema content.

Although the W3C XML Schema include syntax is an effective method for reusing other W3C XML Schemas by reference, caution is advised. If the content and context of the referenced subschema are unknown, there is the potential to include constructs of the referenced schema that are not applicable. Also, a referenced W3C XML Schema may reference and include other W3C XML Schemas and so on. This could introduce an inefficient assembly and processing chain for the parser.

The W3C XML Schema syntax for import and redefine are somewhat similar to include, although the resulting methods of reference are different. The import syntax requires an import element being defined to the primary schema (e.g., <xs:import>). Additionally, schemaLocation and namespace attributes are included. The schemaLocation attribute identifies the location of the referenced subschema. The namespace attribute defines the context. The redefine syntax includes a "rede-

fine" element (e.g., <xs:redefine>) and a schemaLocation attribute. Like the include and import syntax, the schemaLocation attribute identifies the location and name of the referenced subschema. Additionally, the redefine syntax allows specific containers and constructs to be defined differently (Fig. 8.16).

So far, the important data architecture and metadata activities required to engineer flexible, extensible, and reusable W3C XML Schemas have been described. Next we will identify how data architects can leverage their roles, and how these activities will be integrated with the application development process.