


# Introduction to XML Schema

## Scratching the Surface

# Agenda

- Purpose of Class
- History, Level-Setting
- Limitations of DTDs
- XML Schema "Limitations"
- About Namespaces
- XML Schema Structures
  - Simple and Complex Types
  - Elements and Attributes
- XML Schema Datatypes
- Pulling It All Together

# Purpose

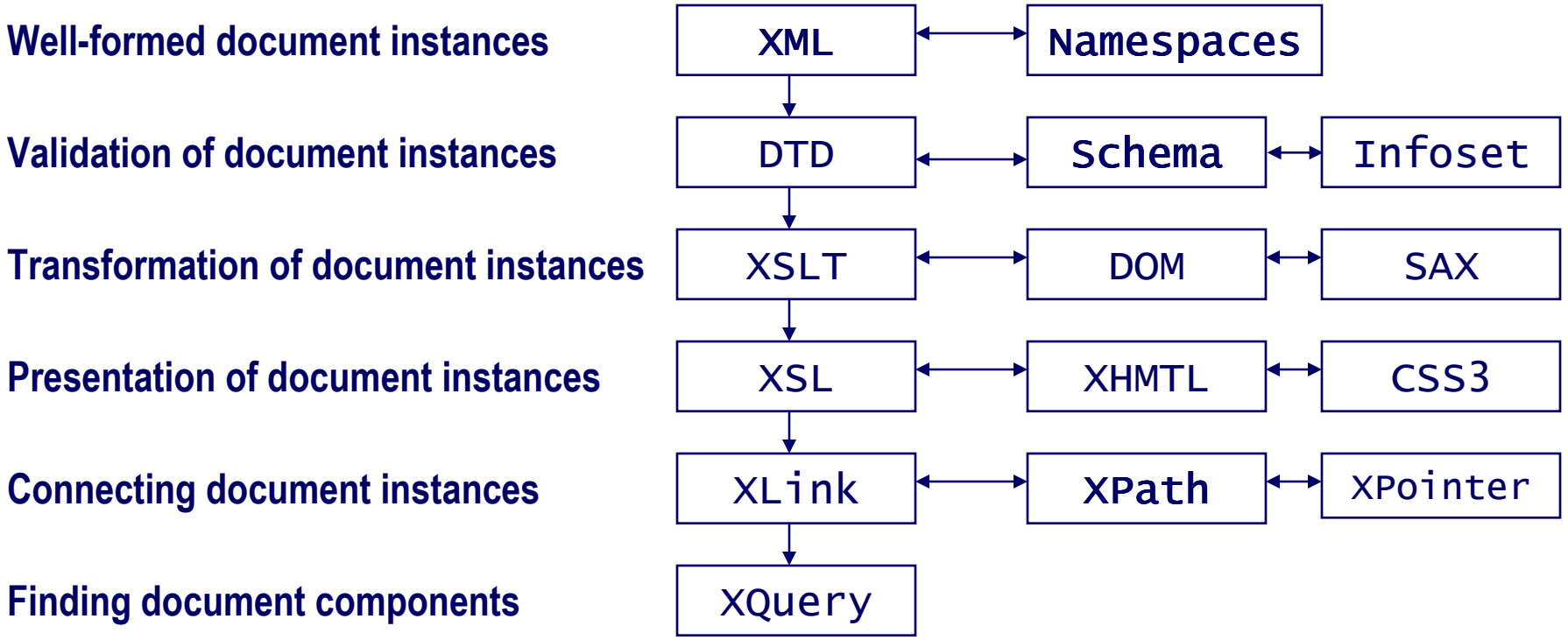
- Highlight basics of XML Schema
  - High-level comparison to DTD
  - Core set of declarations and definitions
  - Aimed at the person who may have to review a schema from a standards in order to vote on it
  - Introductory level for the person who may have to be able to read schema in order to troubleshoot XML instances
- The class does not cover everything you need to know about XML schemas
  - Would require hundreds of slides to cover each feature and debate each approach

# History

- XML, the eXtensible Markup Language, which was released in February 1998 by the W3 Consortium, was designed to permit the exchange of formatted information over the World Wide Web in providing a standardized framework for the description of both data and metadata.

- Early adopters of XML documents have made use of a DTD (Document Type Definition) to define a document's structure and syntactic organization.
  - Can be used for validation by a parser
  - Allows default values to be specified

# The XML Family of Standards




© *The SGML Centre*

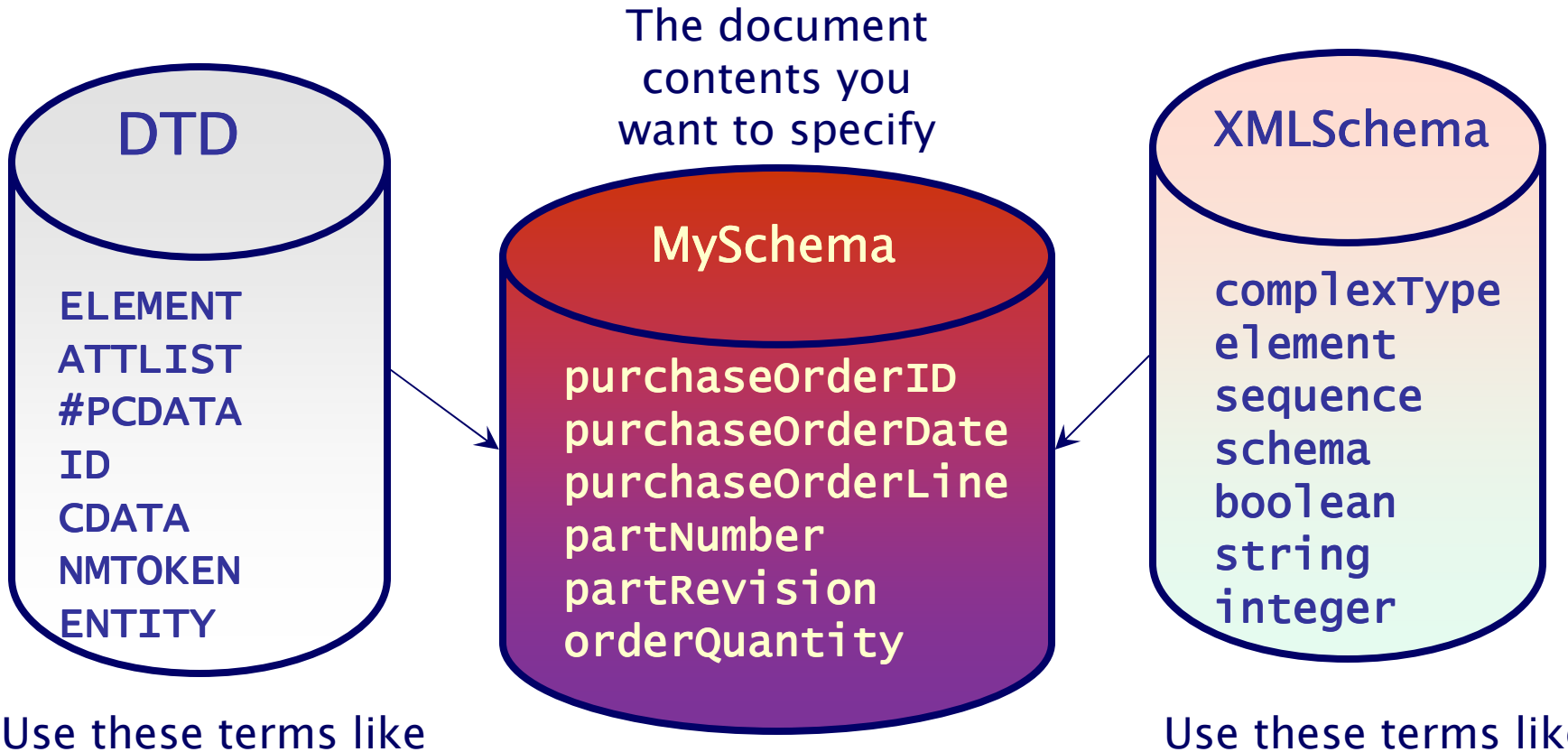
# Some Level Setting

- XML was designed for text markup systems, not as a Data Base Management System language
  - *It wasn't designed originally for B2B commercial transacting!!!*
    - *But … never mind …*
- XML 1.0 specifies the box but says nothing about what's inside the box. The box looks like this*:
  - <>…</>
- Need something else to describe what's in the box … and what's allowed to be in the box.
  - Legacy XML**: Document Type Definition (.dtd)
  - Next generation: XML Schema (.xsd)

* Okay, so we're exaggerating.

** Okay, big deal, so legacy XML specification is only 5 years old (1998)

# Defining XML Vocabularies




Use these terms like these in a DTD *

Use these terms like these in a schema file **

* These terms are defined in the XML 1.0 Specification at http://www.w3.org/TR/REC-xml

** These terms are defined in the XMLSchema namespace, http://www.w3.org/2001/XMLSchema

# Limitations of DTDs

- Not expressed in XML syntax – can't process them using XML tools (such as parsers) *
  - <!ELEMENT Date (#PCDATA)>
- Limited data types
  - Incompatible set of data types with those found in databases
    - e.g. "PCDATA," "CDATA"

* The "!" means all the element definitions are just comments to an XML parser.

# Limitations of DTDs

- No constraints on element content. Example:
  - e.g. <!ELEMENT Date (#PCDATA)> allows this:
    - "October 93, -27 quintillion B.X."

- Extra information about data formats or constraints have to be included in dedicated comments which then have to be coded by users or solution providers

# XML Schemas Overcome most Limitations

- Richer capabilities than a DTD
  - Describes the possible arrangement of tags and text in a valid document
  - 44+ datatypes (as opposed to 10 in DTDs)
  - Allows attribute grouping
  - Supports use of namespaces (more about this later)
  - Supports object-oriented design
    - "Custom" data types can be derived from other data types and inherit their attributes *
  - Can specify element constraints and valid values
  - The comments are really comments (documentation), not specifications

* No, we're not encouraging defining custom tags. All this means is that standards bodies like RosettaNet and OAGI can define stuff for their respective standards that are not contained in the XMLSchema specification.

# Constraints

```
<address>
    <city>East Yahoo</city>
    <region type="usstate">NH</region>
    <name>Bob Kokko</name>
    <postCode>FahNorth</postCode>
</address>
<street>256 Easy St.</street>
```

```
<address>
    <name>Bob Kokko</name>
    <street>256 Easy St.</street>
    <city>East Yahoo</city>
    <region type="usstate">NH</region>
    <postCode>12345-6789</postCode>
</address>
```

- A constraint defines what can appear in any given context.
- Two kinds of constraints
  - Content model constraints describe the order (sequence) and nesting of elements
  - Data type constraints describe what makes bits of data valid or invalid

# DTD / Schema Simple Comparison


## DTD

```
<!ELEMENT Product
        (Model, Dealer+, Price?)>
<!ELEMENT Model (#PCDATA)>
<!ELEMENT Dealer (#PCDATA)>
<!ELEMENT Price    (#PCDATA)>
```

- Written as "comments"
- *Can* describe cardinality
- *Can't* describe what the valid values might be for each element

## Schema

```
<complexType name=product>
<element name="model" type="string"
            occurs="REQUIRED/>
<element name="dealer"
            occurs="ONEORMORE/>
            <simpleType>
            <restriction base="string">
            <enumeration value="Boils Electronics"/>
            <enumeration value="Nextbest Buy"/>
            …
            <enumeration value="Ed's Mini Mart"/>
            </restriction base>
            </simpleType>
<element name="price" type="decimal">
             <minInclusive value="0"/>
             <maxInclusive value="1000"/>
</complexType>
```

- Written in XML
- Can specify bounds and lists of valid values (enumerations)

# DTD / Schema – Another Comparison

self study

```
<!ELEMENT ShoeStore
    (Shoes)+>
<!ELEMENT Shoes (Style, Color,
    Season, Size, Designer)>
<!ELEMENT Style (#PCDATA)>
<!ELEMENT Color (#PCDATA)>
<!ELEMENT Season (#PCDATA)>
<!ELEMENT Size (#PCDATA)>
<!ELEMENT Designer
    (#PCDATA)>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.shopaholics.org"
            xmlns="http://www.shopaholics.org"
            elementFormDefault="qualified">
  <xsd:element name="ShoeStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Shoes" minOccurs="1"
   maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Shoes">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Style" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Color" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Season" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Size" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Designer" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Style" type="xsd:string"/>
  <xsd:element name="Color" type="xsd:string"/>
  <xsd:element name="Season" type="xsd:string"/>
  <xsd:element name="Size" type="xsd:string"/>
  <xsd:element name="Designer" type="xsd:string"/>
</xsd:schema>
```

# XML Schema "Limitations"

- XML Schema are not designed to be human-readable!*
  - XML Schema are intended to be read by machines – used by the parser
- Schema deciphering during troubleshooting may require special skill set **
- XML Schema are verbose

* Well, most humans can't read them!

** But wait, isn't that why we're trying to get rid of EDI?

** Okay, so they're not so much limitations as too much of a good thing!

# XML Schema "Limitations"

- Not all developers and standards bodies use Schema the same way
  - There are at least 20 ways to declare attributes
    - From local attribute with default value to global attribute with fixed value
  - At least 15 ways to declare elements
    - From local element with default value to global element with complex type
  - At least three different ways to identify hierarchy of namespaces
    - XMLSchema is the default
    - targetNamespace is the default
    - No namespace is default

# XML Schema "Limitations"


- How do we define "Shoestore"?
  - <element name="Shoestore" type="string"/>
  - <xsd:element name="ShoeStore">
    <xsd:complexType>
    <xsd:sequence>
    <xsd:element ref="Shoe"
    minOccurs="1"
    maxOccurs="unbounded"/>
    </xsd:sequence>
    </xsd:complexType>
    </xsd:element>

# Before we go any farther …


- What's the difference between:

1. 
```
<element name="ChemicalComposition">
<element ref="chemClass:elemNum"/>
<element name="element" type="string">
```

2. 
```
<xsd:element name="ChemicalComposition">
<xsd:element ref="elemNum"/>
<xsd:element name="element" type="string">
```

3. 
```
<xs:element name="ChemicalComposition">
<xs:element ref="elemNum"/>
<xs:element name="element" type="string">
```

4. 
```
<xsd:element name="ChemicalCompsition">
<xsd:element ref="chemClass:elemNum"/>
<xsd:element name="element" type="string">
```

It has to do with namespaces and identifying the where to find the specification containing the element definitions …

(con't)

# Don't be confused by namespaces

- Namespaces help avoid collisions between data element names
- The namespace identifier qualifies element names
  - Identifies the source of the vocabulary
  - Is what allows you to define a schema from more than one source
    - One of those sources is always XMLSchema – the schema that defines the vocabulary for XML schema files – "http://www.w3.org/2001/XMLSchema"

# Don't be confused by namespaces

- Qualifier is used with element names, and the qualifier is associated with a Uniform Resource Identifier (URI)
  - The URI indicates where to find the schema source
    - `xmlns:chemClass="http://www.WhatsamattaU.edu/chemClass"`
    - `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
    - `xmlns:RNpip3c3=http://www.rosettanet.org/nextgen/PIP3C3_InvoiceNotification *`
    - `xmlns="http://www.openapplications.org/oagis" **`

* Not a real namespace – just an example!

** A real namespace!

# Don't be confused by namespaces

- The choices for default are:
  1. Make the W3C XMLSchema the default, and use qualifiers on everything else
  2. Make the targetNamespace the default, and use qualifiers on everything else
     - Including everything from the XMLSchema
  3. Don't specify a default and use qualifiers on everything

There are advantages and disadvantages to each approach.

The main impact on you is that this whole thing explains why the schema are so confusing to newbie … figuring out what those "xsd:" and other prefixes mean!

# What should be the default namespace?

self study

- Differences in examples 2 slides ago is the result of different choices for default namespace

1. 
```
<element name="ChemicalComposition">
<element ref="chemClass:elemNum"/>
<element name="element" type="string">
```

   - Default namespace is "XMLSChema", targetNamespace is qualified

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.WhatsamattaU.edu"
xmlns:chemClass="http:// www.WhatsamattaU.edu"
elementFormDefault="qualified">
<include schemaLocation="chemicals.xsd"/>
<element name="elemNum">
```

No qualifier after "xmlns" means this is the default namespace. All unqualified elements are from this specification.

Qualified, meaning it's not the default namespace. Elements from this namespace have to be identified with a qualifier.

# Don't be confused by namespaces

self study

- **Default namespace is "XMLSChema", targetNamespace is qualified**

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.WhatsamattaU.edu"
xmlns:chemClass="http://www.WhatsamattaU.edu"
elementFormDefault="qualified">
<include schemaLocation="chemicals.xsd"/>
<element name="elemNum">
```

- **Default namespace is targetNamespace,"XMLSchema" is qualified**

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.WhatsamattaU.edu"
xmlns="http://www.WhatsamattaU.edu"
elementFormDefault="qualified">
<xsd:include schemaLocation="chemicals.xsd"/>
<xsd:element name=" elemNum">
```

OAGI and most others uses this choice; some use "xsd:" prefix, others use "xs:" prefix

- **No default – qualify both**

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.WhatsamattaU.edu"
xmlns:chemClass="http://www.WhatsamattaU.edu" elementFormDefault="qualified">
<xsd:include schemaLocation="chemicals.xsd"/>
<xsd:element name=" elemNum">
```

# Resulting Schema Wrapper – OAGIS

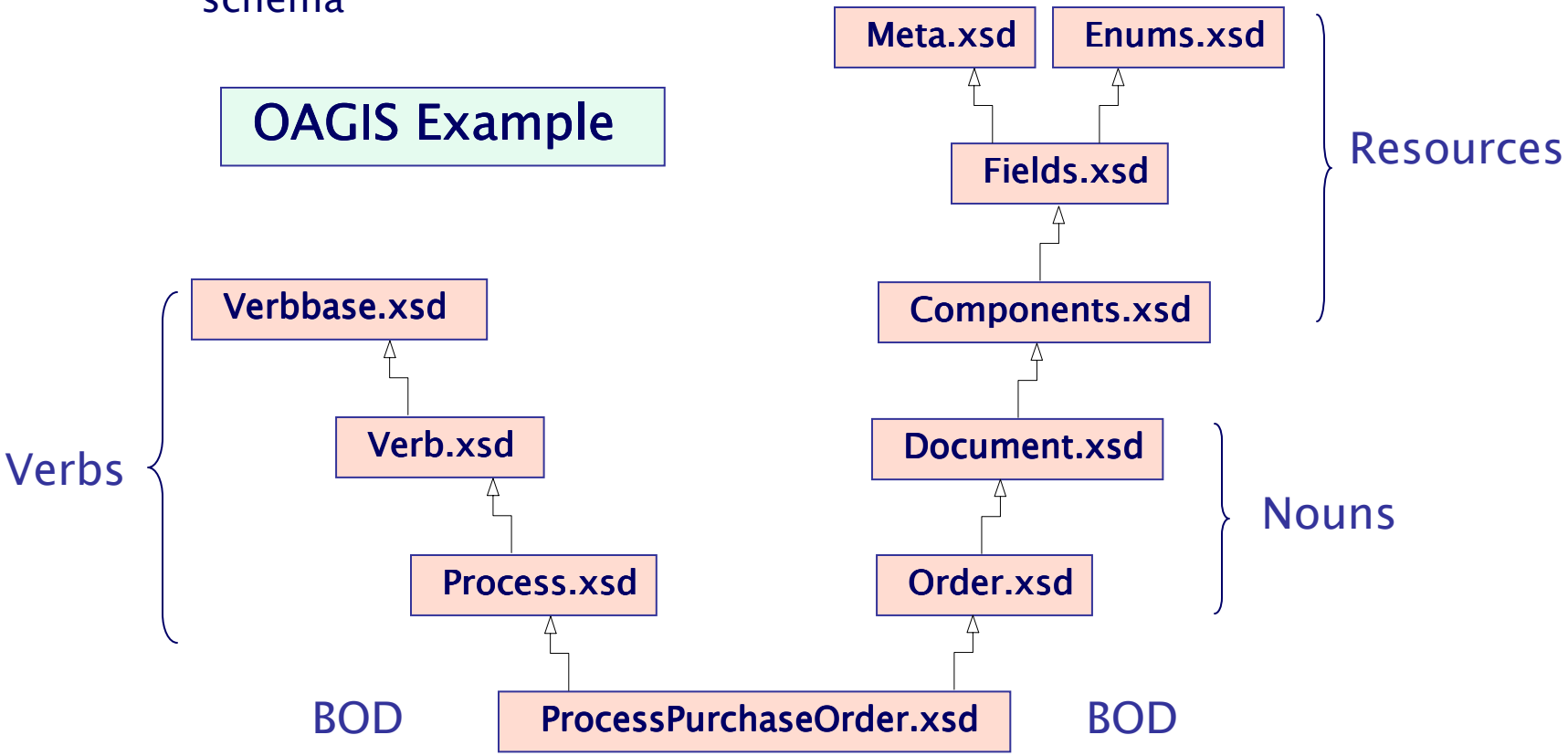
```
<xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.openapplications.org/oagis"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
….
</xs:schema>
```

xmlns:xs – 'xs' prefix selected to reference elements defined in a schema from the namespace where the XMLSchema lives; prefix will be used with everything that comes from that schema.

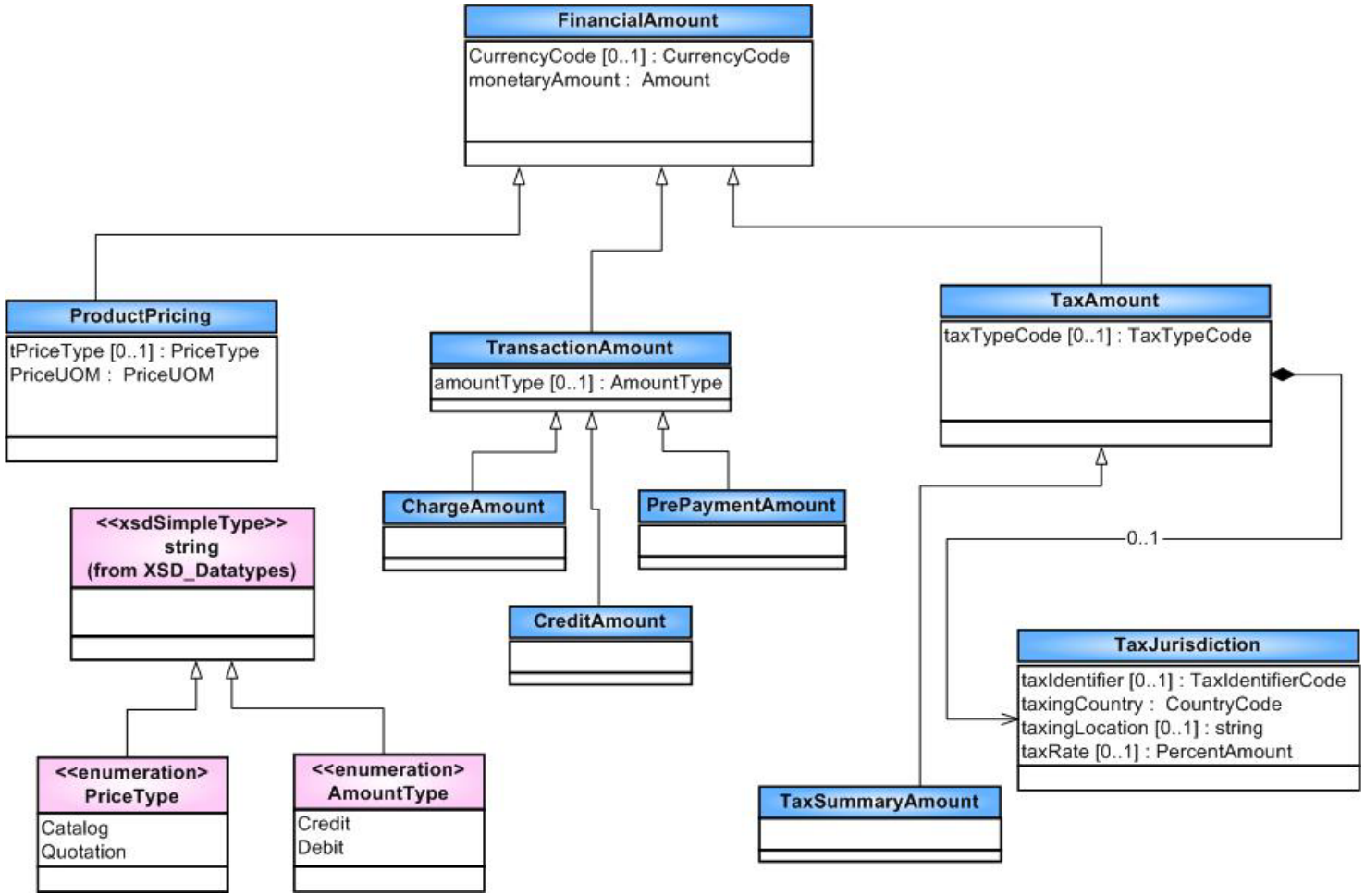
targetNamespace – All the elements and types defined in *this schema* come from *this namespace,* and will not use a prefix . Use this URI to import or include these definitions in other schemas.

# Including Other Schema

- XML Schema allow you to include one schema inside another, so that the included schema is part of the current schema
  - Included schema must be in the same namespace
  - Makes it challenging to interpret any schema that inherits from other schema



# Brief digression: Inheritance and Composition*




*This example is not from OAGIS

# Brief Digression: Inheritance and Composition

## Result using Composition

```
<ChargeAmount>
      <TransactionAmount>
          <FinancialAmount>
              <CurrencyCode>EUR</CurrencyCode>
              <MonetaryAmount>45.00</MonetaryAmount>
           </FinancialAmount>
           <AmountType>Debit</AmountType>
       </TransactionAmount>
</ChargeAmount >
```

## Result using Inheritance

```
<ChargeAmount>
      <CurrencyCode>EUR</CurrencyCode>
      <MonetaryAmount>45.00</MonetaryAmount>
      <AmountType>Debit</AmountType>
</ChargeAmount >
```




# XML Schema: Structures

Reference:
XML Schema Part 1: Structures
W3C Recommendation 2 May 2001

http://www.w3.org/TR/xmlschema-1/

# What can you put in an XML file?

- Elements
  - Form <element attribute="*value*">*value*</element>
  - Some elements contain other elements, some do not
  - Some elements contain attributes, some do not
- Attributes
  - Form <element attribute="*value*">*value*</element>
  - To "qualify" elements
  - Can't contain other attributes
- Comments
  - Form <!--*comment* -->
  - Can't be parsed; don't use them in business document instances
  - Only useful as documentation in schemas and DTDs

# OAGIS XML Purchase Order Fragment

```
<Schedule>
   <LineNumber>1</LineNumber> …
   <OrderQuantity uom="PCE">100</OrderQuantity>
   <UnitPrice>
      <Amount currency="EUR">12.95</Amount>
      <FunctionalAmout currency="EUR">12.95</FunctionalAmout>
      <PerQuantity uom="PCE">12.95</PerQuantity>
   </UnitPrice>
   …
</Schedule>
```

For this OAGIS fragment, the schema needs a declaration for the **element** `Schedule`, which contains the **elements** `LineNumber`, `OrderQuantity` and `UnitPrice` (among others). Unit Price contains the elements `Amount`, `FunctionalAmount`, and `PerQuantity`.

For the elements `OrderQuantity` and `PerQuantity`, the schema needs a declaration for the **attribute** `uom`, and for the elements `Amount` and `FunctionalAmount`, the schema needs a declaration for the attribute `currency`.

# What can you put in an XML Schema*?


- Type Definitions ***<simpleType>*** **<complexType>**
  `<element> <group> <all> <choice> <sequence> <attribute> <attributeGroup>`
- Element Declarations **<element>**
  *<simpleType>* `<complexType>`
- Attribute Declarations **<attribute>**
  *<simpleType>*
- Attribute Group Definitions **<attributeGroup>**
  `<attribute> <attributeGroup>`
- Model Group Definitions **<group>**
  `<element> <group> <all> <choice> <sequence>`
- Notation Declarations **<notation>**
- Annotations **<annotation>**
  `<appinfo> <documentation>`

* Only the fundamental parts of the first three covered in this class.

# Simple and Complex Types

- XML Documents can have two types of content
  - Simple Types – elements that can contain only text
    - No child elements and no attributes
    - Numeric, dates and times, URI's, etc.
    - Create a new simple type if you want to refine an existing simple type
  - Complex Types – elements that contain other elements or attributes
    - Can contain child elements and attributes
    - Most of created elements are Complex Types

# Simple and Complex Types

- Have to locate the element's type definition to determine whether it's a `simpleType` or a `complexType`
  - Declared by `type="`*type*" in element declaration
  - "Type" is how you can specify inheritance
  - Attributes only contain text so are always `simpleType`
- Some `complexType` and `simpleType` definitions do not have a "name=" statement
  - This is an "anonymous" type, which is which is always defined inline (inside an element) and applies only to that element (can't be re-used)

# Primitive and Derived Types

- Primitive Types are the basic, atomic data types that are already built into the XMLSchema

- Derived types are created when by extending or restricting another type – the derived type specializes the base type, using inheritance
  - Several derived types are already built into the XMLSchema – these are derived from the primitive types *
  - Other derived types are those that you create within your schema, which can be derived from primitive types, built-in derived types, or other derived types in your schema

* Primitives and derived types that are built into the XMLSchema are covered later in the class.

# Creating New Derived Types

- You can derive a new type which adds more elements to a base
  - Can only do this with a `complexType`
- You can derive a new type which narrows ranges or reduces alternatives
  - Create a type that restricts the range of values to a subset of the base
  - Create a type which has a more restrictive cardinality than the base type
  - Permissible with both `complexType` and `simpleType`
    - A `simpleType` is specialized by using `<restriction>`
    - A `complexType` is specialized by using `complexContent` or `simpleContent`, which each can contain `<restriction>` or `<extension>`

# complexType Definitions

- complexType Definitions
  - An complexType definition may include these information items (only the most frequently used in OAGIS listed here):
    - abstract=
    - name=
  - Nested within a complexType definition, you may find these:
    - <complexContent>
    - <simpleContent>
    - <sequence>
    - <choice>

# Simple and Complex Types

self study

OAGIS `simpleType` and `complexType` examples:

```
<xs:element name="LineNumber" type="LineNumber" minOccurs="0">
(found in document.xsd)
<xs:simpleType name="LineNumber">
(found in fields.xsd)
```

```
<xs:element name="EffectivePeriod" type="TimePeriod" minOccurs="0">
(found in Components.xsd)

<xs:complexType name="TimePeriod">
    <xs:sequence>
        <xs:element name="From" type="DateTime" minOccurs="0"></xs:element>
        <xs:choice>
            <xs:element name="Duration" type="Duration" minOccurs="0">
            </xs:element>
            <xs:element name="To" type="DateTime" minOccurs="0"></xs:element>
        </xs:choice>
    </xs:sequence>
<xs:attribute name="inclusive" type="xs:boolean" use="optional" default="true" />
</xs:complexType>
(found in fields.xsd)
```

# complexContent versus simpleContent

- <complexContent> allows you to you extend or restrict a complexType
  - base= statement must identify a complexType
- <simpleContent> allows you to extend (<extension>) or restrict (<restriction>) a simpleType
  - base= statement must identify a simpleType
- <extension> allows you extend the base type by adding one or more new elements or attributes – to specialize the base

Modified OAGIS Example – **<simpleContent>** and **<restriction>** :

```
<xs:complexType name="Amount">
   <xs:simpleContent>
      <xs:extension base="xs:decimal">
         <xs:attribute name="currency" type="Currency"
            use="required">
         </xs:attribute>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

(found in fields.xsd)

This type definition is extending the definition of the simpleType "decimal" in the XMLSchema namespace by adding the required use of the attribute "currency"

# complexType Definition

## restriction

- <sequence> Allows you to specify the sequence in which elements must appear
- <restriction> Allows you to specify the complexType that is being restricted; a restriction narrows the ranges or reduces alternatives of elements and/or attributes; like <extension>, this specializes the base

---

Modified OAGIS Example – **<simpleContent>** and **<restriction>** :

```
<xs:complexType name="Order">
  <xs:complexContent>
    <xs:restriction base="Document">
      <xs:sequence>
        <xs:element ref="Header" minOccurs="0" />
        <xs:element ref="Line" minOccurs="0" maxOccur
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

(found in Order.xsd)

The complexType "Order" is a restriction of the complexType "Document". The element "Header" must appear sequentially before the element "Line".

# complexType Definition

## choice

- <choice> Allows you to list a series of choices that can populate a given element

Modified OAGIS Example – **<simpleContent>** and **<restriction>** :

```
<xs:complexType name="Attachment">
  <xs:sequence>
    <xs:choice>
      <xs:element name="EmbeddedData" type="EmbeddedData" minOccurs="0">
      </xs:element>
      <xs:element name="URI" type="xs:anyURI" minOccurs="0"></xs:element>
      <xs:element name="FileName" type="xs:string" minOccurs="0"></xs:element>
      <xs:element name="ISBN" type="ISBN" minOccurs="0"></xs:element>
    </xs:choice>
    <xs:element name="DocumentDate" type="DateTimeAny" minOccurs="0"></xs:element>
    …
    <xs:element name="FileType" type="xs:string" minOccurs="0"></xs:element>
    <xs:element name="FileSize" type="xs:decimal" minOccurs="0"></xs:element>
    …
    </xs:sequence>
  <xs:attribute name="inline" type="xs:boolean" use="optional"
</xs:complexType>
```

(found in Components.xsd)

The Attachment element can contain any one of the four elements listed. (EmbeddedData, URI, etc.

# simpleType Definitions

- An attribute must be a `simpleType`
  - `Attribute` and `simpleType` are frequently confused with each other
  - Not the same – an `element` can be a `simpleType`
- An `simpleType` frequently extends or restricts a primitive type or a derived type
- An `simpleType` definition may include these information items (only the most frequently used in OAGIS listed here):
    - `id=`
    - `name=`
  - Nested within a `complexType` definition, you may find these:
    - `<restriction> (which can contain <enumeration>)`

# Code Lists

- Enumeration
- External code lists
  - Method 1: Directly integrate external code list schema into your schema
    - namespace declaration
      - xmlns:iso3166="http://www.unece.org/etrades/unedocs/repository/codelists/xml/CountryCode.xsd"
    - element declaration
      - <xs:element name="CountryIdentificationCode" type="iso3166:CountryCodeType" />
  - Method 2: Identify source in documentation, but build explicit enumeration list into your schema

# simpleType Definition

OAGIS `simpleType` and `complexType` examples:

```
<xs:simpleType name="Date">
  <xs:restriction base="xs:date">
    <xs:pattern value="\d\d\d\d-\d\d-\d\d" />
  </xs:restriction>
</xs:simpleType>
```

The `simpleType` `Date` in the OAGIS namespace is a restriction of the primitive type `date` in the XMLSchema.

```
<xs:simpleType name="Rating">
  <xs:annotation>
    <xs:documentation
    source="http://www.openapplications.org/oagis">
    Business Party ratings.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Good" />
    <xs:enumeration value="Bad" />
    <xs:enumeration value="Ugly" />
  </xs:restriction>
</xs:simpleType>
```

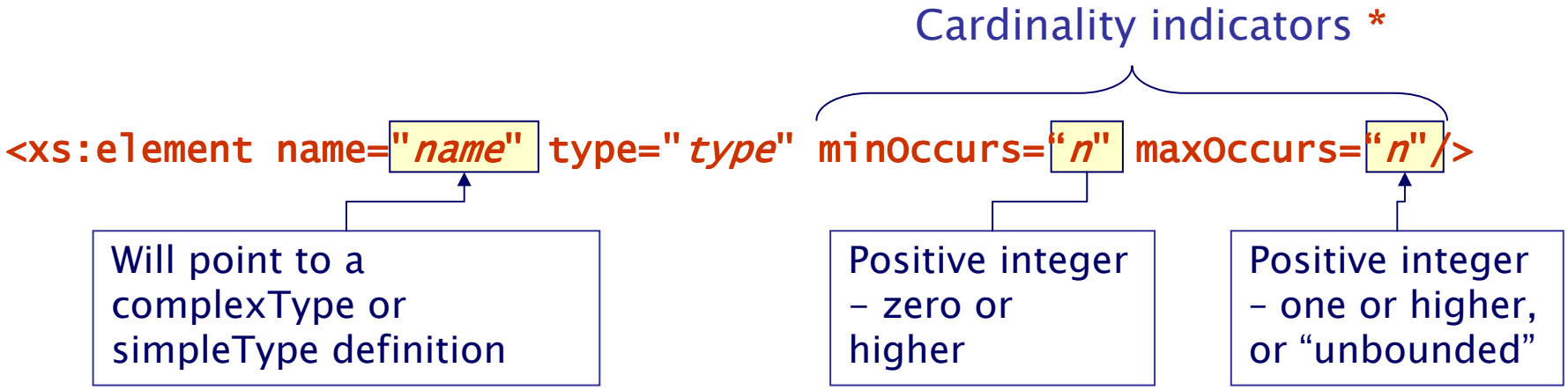
We're not making any of this up, you know!

# Element Declarations

- Any element that appears in your XML instance must have an element declaration somewhere in the schema
  - `<element name=...` or `<prefix:element name=...`
- An element declaration may include these information items (only the most frequently used in OAGIS listed here):
  - `abstract=`
  - `maxOccurs=`
  - `minOccurs=`
  - `name=`
  - `ref=`
  - `substitutionGroup=`
  - `type=`

# Frequently Used Forms of Element Declarations

## Basic




OAGIS Examples:

```
<xs:element name="PerQuantity" type="Quantity">
```
(found in fields.xsd)

```
<xs:element name="FunctionalAmount" type="Amount" minOccurs="0">
```
(found in fields.xsd)

```
<xs:element name="ShipNote" type="Note" minOccurs="0"
      maxOccurs="unbounded">
```
(found in Order.xsd)

* Cardinality examples on the next slide!

# Indicating Cardinality

self study

- In schema, cardinality is defined using "minoccurs" and "maxoccurs"
  - If one of these is not present in the declaration, its value defaults to "1"

| | UML Notation* | DTD Notation | XML Schema Notation |
|---|---|---|---|
| *Optional, not repeatable* | 0..1 | ? | minoccurs=0<br>(maxoccurs defaults to "1" if not declared) |
| *Optional, repeatable* | 0..n | * | minoccurs=0<br>maxoccurs="n" or maxoccurs="unbounded" |
| *Required, not repeatable* | 1 | (default) | (default – in this case minoccurs and maxoccurs do not need to be declared) |
| *Required, minimum is once, repeatable* | 1..n | + | minoccurs="1"<br>maxoccurs="n" or maxoccurs="unbounded" |
| *Required, minimum is more than once* | n..n | (can't be notated – closest is +) | minoccurs="n"<br>maxoccurs="n" or maxoccurs="unbounded" |

* UML notation is what you see in RosettaNet Message Guidelines

# Frequently Used Forms of Element Declarations

## Referencing another Element Declaration

```
<xs:element ref="ref" minOccurs="n" maxOccurs="n"/>
```

References an element defined elsewhere.
- ✓Useful if the cardinality of your element varies depending on context.
- ✓Useful if an element is used in more than one `complexType` element – define it once then refer to it with "`ref=`"
- ✓The referenced element may be defined in the current schema or in another schema
- ✓We need to find the type definition in order to determine if the element is a `simpleType` or `complexType`

OAGIS Examples:

```
<xs:element ref="OrderStatus" minOccurs="0" />
```
(found in Order.xsd)

```
<xs:element name="OrderStatus" type="OrderStatus">
```
(found in Components.xsd)

```
<xs:element ref="PurchaseOrder" maxOccurs="unbounded" />
```
(found in ProcessPurchaseOrder.xsd)

```
<xs:element name="PurchaseOrder" … substitutionGroup="Noun">
```
(found in PurchaseOrder.xsd)

# Frequently Used Forms of Element Declarations

## Substitution Groups

```
<xs:element name="name" … substitutionGroup="substitutable element"/>
```

Means that this element is a suitable substitution for another element
- ✓The substitutable element is often an abstract element (next slide)
- ✓May be substitutable element may be declared in the current schema or another schema

OAGIS Examples:

```
<xs:element name="PurchaseOrder" … substitutionGroup="Noun">
(found in PurchaseOrder.xsd)
<xs:element name="Noun" type="Noun" abstract="true" />
(found in meta.xsd)
```

```
<xs:element name="ShipToPartyId" type="PartyAssignedPartyId"
substitutionGroup="PartyAssignedPartyId" />
(found in fields.xsd)
<xs:element name="PartyAssignedPartyId" type="PartyAssignedPartyId"
abstract="true">
(found in fields.xsd)
```

# Frequently Used Forms of Element Declarations

## Abstract Elements

```
<xs:element name="name" type="type" abstract="true"/>
```

Means that the element will never be instantiated – you'll never see the element used in an XML instance file

✓The abstract element is a kind of template or placeholder – "we need a something like this here, and that something will be identified somewhere else"

✓Can only use this if substitution is allowed in the data model – there must be non-abstract substitute elements available which are used for instantiation

✓If "abstract=" is not present, the element is not an abstract element. Only necessary to explicitly declare it when the element *is* abstract

See OAGIS Examples on previous slide:

# Frequently Used Forms of Element Declarations

## Inline complexType definition

```
<xsd:element name="name" minOccurs="n" maxOccurs="n">
    <xsd:complexType>
        ...
    </xsd:complexType>
</xsd:element>
```

In the example below, the type definition is "inline" – contained within the element declaration. The complexType example on slide 37 shows that the type definition can be in a different schema than the element declaration.

## Modified OAGIS Example

```
<xs:element name="UnitPrice" type="AmountPerQuantity" minOccurs="0"/>
  <xs:complexType name="AmountPerQuantity">
    <xs:sequence>
      <xs:element name="Amount" type="Amount">
      </xs:element>
      <xs:element name="FunctionalAmout" type="Amount" minOccurs="0">
      </xs:element>
      <xs:element name="PerQuantity" type="Quantity">
      </xs:element>
    </xs:sequence>
 </xs:complexType>
</xs:element>
```

# Frequently Used Forms of Element Declarations

## Inline simpleType Definition

```
<xsd:element name="name" minOccurs="n" maxOccurs="n">
    <xsd:simpleType>
        ...
    </xsd:simpleType>
</xsd:element>
```

In this example, the type definition is inline – contained within the element declaration. Looks visually like `complexType`, but it's not – no element declarations are contained within the type definition.

---

Modified OAGIS Example:

```
<xs:element name="ProximoNumberMonth" minOccurs="0">
    <xs:simpleType>
        <xs:restriction base="xs:positiveInteger">
            <xs:minInclusive value="1" />
            <xs:maxInclusive value="12" />
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

(found in Components.xsd)

This type definition is restricting the definition of "`positiveInteger`" in the XMLSchema – values between 1 and 12; applies only to this element.

# Attribute Declarations

- Any attribute that appears in any element in your XML instance must have an attribute declaration somewhere in the schema
  - `<attribute name=…` or `<prefix:attribute name=…`
- An attribute declaration may include these information items (only the most frequently used in OAGIS listed here):
  - `name=`
  - `type=`
  - `use=`
- An attribute must be a simpleType

# Frequently Used Form of Attribute Declaration
## Along with example simpleType definition

```
<xs:attribute name="name" type="type" use="required"/>
```

Will point to a simpleType definition

Will be "optional", "prohibited" or "required"; if use= not stated, default is "optional"

OAGIS Examples*:

```
<xs:attribute name="currency" type="Currency" use="required">
```
(found in fields.xsd)

```
<xs:simpleType name="UOM">
  <xs:annotation>
    <xs:documentation source="http://www.openapplications.org/oagis">
     Standard values from ISO / SI ????
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:NMTOKEN" />
</xs:simpleType>
```
(found in enums.xsd)

* In OAGIS, attributes are generally declared within complexType definitions; there are ~20 attributes declared in the entire specification, as compared to hundreds of elements.

# Back to the OAGIS Purchase Order XML Fragment …

Armed with my new knowledge about schemas, I can now look for the elements and attributes in my XML fragment.

```
<Schedule>
    <LineNumber>1</LineNumber> …
    <OrderQuantity uom="PCE">100</OrderQuantity>
    <UnitPrice>
        <Amount currency="EUR">12.95</Amount>
        <FunctionalAmout currency="EUR">12.95</FunctionalAmout>
        <PerQuantity uom="PCE">12.95</PerQuantity>
    </UnitPrice>
    …
</Schedule>
```

This is a PO, so naturally I start with PurchaseOrder.xsd. And I find that it contains only 5 elements, and only one of those is a match to my fragment.

```
<xs:element name="PurchaseOrder" type="PurchaseOrder" substitutionGroup="Noun">
<xs:element name="Header" type="PurchaseOrderHeader">
<xs:element name="Line" type="PurchaseOrderLine">
<xs:element name="SubLine" type="PurchaseOrderSubLine">
<xs:element name="Schedule" type="PurchaseOrderSchedule">
```

**Where are all the other elements from my fragment?**

# Finding the Purchase Order Elements

```
<xs:element name="PurchaseOrder" type="PurchaseOrder" substitutionGroup="Noun">
<xs:element name="Header" type="PurchaseOrderHeader">
<xs:element name="Line" type="PurchaseOrderLine">
<xs:element name="SubLine" type="PurchaseOrderSubLine">
<xs:element name="Schedule" type="PurchaseOrderSchedule">
```

I know that the elements are contained in `Schedule`, and I see that Schedule is of the type `PurchaseOrderSchedule`, so I look for that now. I find that it appears earlier in the PurchaseOrder schema file.

```
<xs:complexType name="PurchaseOrderSchedule">
  <xs:complexContent>
    <xs:extension base="OrderSchedule">
      <xs:sequence>
        <xs:element ref="UserArea" minOccurs="0" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

(found in PurchaseOrder.xsd)

I still don't see the other elements from my fragment :-(

# Finding the Purchase Order Elements

```
<xs:complexType name="PurchaseOrderSchedule">
  <xs:complexContent>
    <xs:extension base="OrderSchedule">
      …
  </xs:complexContent>
</xs:complexType>
```

(found in PurchaseOrder.xsd)

I see that "PurchaseOrderSchedule" is an extension of OrderSchedule, so I look for that now. It's not in the PurchaseOrder schema file. But, right at the beginning of PurchaseOrder.xsd is the following:

```
<xs:include schemaLocation="Order.xsd" />
```

This means I've got to look there next, and I will look for OrderSchedule.

# Finding the Purchase Order Elements

```
<xs:complexType name="OrderSchedule">
  <xs:complexContent>
    <xs:extension base="OrderLineBase">
     <xs:sequence>
        <xs:element name="EffectivePeriod" type="TimePeriodAny" minOccurs="0">
        </xs:element>
        <xs:element name="RequiredQuantity" type="Quantity" minOccurs="0">
        </xs:element>
        ...
        <xs:element name="OverShipTolerance" type="Quantity" minOccurs="0">
        </xs:element>
        <xs:element name="UnderShipTolerance" type="Quantity" minOccurs="0">
      </xs:element>
     </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

(found in Order.xsd)

I find it. It has lots of elements, but not any from my fragment. But I note that `OrderSchedule` extends `OrderLineBase`.

# Finding the Purchase Order Elements

```
<xs:complexType name="OrderLineBase">
  <xs:complexContent>
    <xs:extension base="DocumentLine">
      <xs:sequence>
        <xs:element ref="OrderItem" minOccurs="0" />
        <xs:element name="OrderQuantity" type="Quantity" minOccurs="0">
        </xs:element>
        <xs:element name="UnitPrice" type="AmountPerQuantity" minOccurs="0">
        </xs:element>
         <xs:element name="ExtendedPrice" type="Amount" minOccurs="0">
         </xs:element>
...
         <xs:element ref="TransportationTerm" minOccurs="0" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
(found in Order.xsd)
```

Okay, I found two more of the elements from the fragment.

# Finding the Purchase Order Elements

```
<Schedule>
   <LineNumber>1</LineNumber> …
   <OrderQuantity uom="PCE">100</OrderQuantity>
   <UnitPrice>
      <Amount currency="EUR">12.95</Amount>
      <FunctionalAmout currency="EUR">12.95</FunctionalAmout>
      <PerQuantity uom="PCE">12.95</PerQuantity>
   </UnitPrice>
   …
</Schedule>
```

I still have several elements unaccounted for. I note from the previous slide that `OrderLineBase` extends `DocumentLine`. I'm still in Order.xsd. It's not there. But, Order.xsd includes Document.xsd, so I look there, and find one more of my elements.

```
<xs:complexType name="DocumentLine">
  <xs:sequence>
    <xs:element name="LineNumber" type="LineNumber" minOccurs="0"></xs:element>
  </xs:sequence>
</xs:complexType>
```

(found in Document.xsd)

# Finding the Purchase Order Elements

```
<Schedule>
   <LineNumber>1</LineNumber> …
   <OrderQuantity uom="PCE">100</OrderQuantity>
   <UnitPrice>
      <Amount currency="EUR">12.95</Amount>
      <FunctionalAmout currency="EUR">12.95</FunctionalAmout>
      <PerQuantity uom="PCE">12.95</PerQuantity>
   </UnitPrice>
   …
</Schedule>
```

Now I have all the elements that are 1st level relative `Schedule`, but I need the elements contained in `UnitPrice`, and the attributes `uom` and `currency`. `UnitPrice` is of the type `AmountPerQuantity`.

```
<xs:element name="UnitPrice" type="AmountPerQuantity" minOccurs="0">
```

Neither order.xsd nor document.xsd contain the complexType `AmountPerQuantity`. But document.xsd includes components.xsd, so I look there. It's not in components.xsd, but components.xsd includes fields.xsd.

# Finding the Purchase Order Elements

```
<xs:complexType name="AmountPerQuantity">
  <xs:sequence>
    <xs:element name="Amount" type="Amount"></xs:element>
    <xs:element name="FunctionalAmout" type="Amount" minOccurs="0">
    </xs:element>
    <xs:element name="PerQuantity" type="Quantity"></xs:element>
  </xs:sequence>
</xs:complexType>
```

(found in Fields.xsd)

There are the rest of the *elements* in the fragment. Now I need to find the attributes uom and currency.

```
<Schedule>
   <LineNumber>1</LineNumber> …
   <OrderQuantity uom="PCE">100</OrderQuantity>
   <UnitPrice>
      <Amount currency="EUR">12.95</Amount>
      <FunctionalAmout currency="EUR">12.95</FunctionalAmout>
      <PerQuantity uom="PCE">12.95</PerQuantity>
   </UnitPrice>
   …
</Schedule>
```

# Finding the Purchase Order Elements

`OrderQuantity` and `PerQuantity` both contain the attribute `uom` and are both of the type `Quantity`, so that's what I look for next.

```
<xs:complexType name="Quantity">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="uom" type="UOM" use="required" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

(found in Fields.xsd)

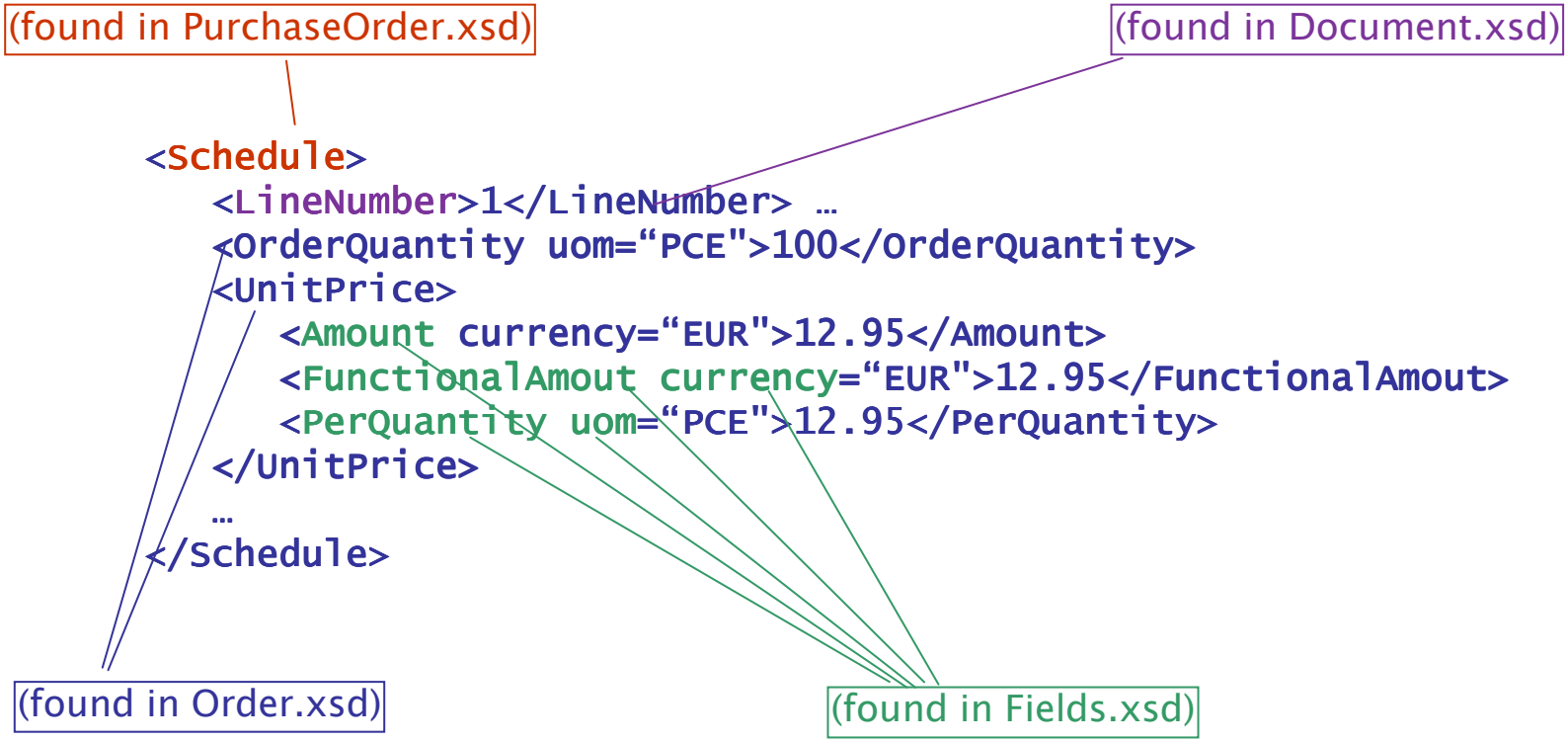
`Amount` and `FunctionalAmount` both contain the attribute `currency` and are both of the type `Amount`.

```
<xs:complexType name="Amount">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="currency" type="Currency" use="required">
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

(found in Fields.xsd)

# Finding the Purchase Order Elements

Now I have found all my elements and attributes




From here I can start the process of validating my XML fragment. But I need to know more about data types and constraints.




# XML Schema: Datatypes and Datatype Constraints

Reference:

**XML Schema Part 2: Datatypes**
**W3C Recommendation 02 May 2001**

http://www.w3.org/TR/xmlschema-2/

# Validating Element and Attribute Contents

Earlier in the class, we saw how to express cardinality, which are rules as to whether an element is mandatory or optional, can appear once or more than once, etc.

We also need to know the rules for the contents of each element and attribute.

We've already seen how to create derived types, which are created from other types. Ultimately, there is a core set of datatypes from which all others get derived. These are primitive datatypes.

# Built-in Primitive and Derived Types in XMLSchema

XMLSchema has 19 built-in primitive datatypes. Most frequently used:

- string
  - "EIDX has lots of characters"
- boolean
  - "True" or "False"
- decimal
  - 3.14, -2.2, etc.
- dateTime
  - 2000-03-04T20:00:00Z
- time
  - 13:20:00-05:00
    - "-05:00" is optional time zone indicator
- date
  - 2002-10-30

To keep designers from reinventing things, XMLSchema has 25 built-in derived types. Most frequently used:

- Integer
  - 1, -21, 50
- positiveInteger
  - 1, 21, 50
- Language
  - "EN" (English), "GA" (Irish), "IU" (Inuktitut)

# Refining a Datatype with Constraining Facets

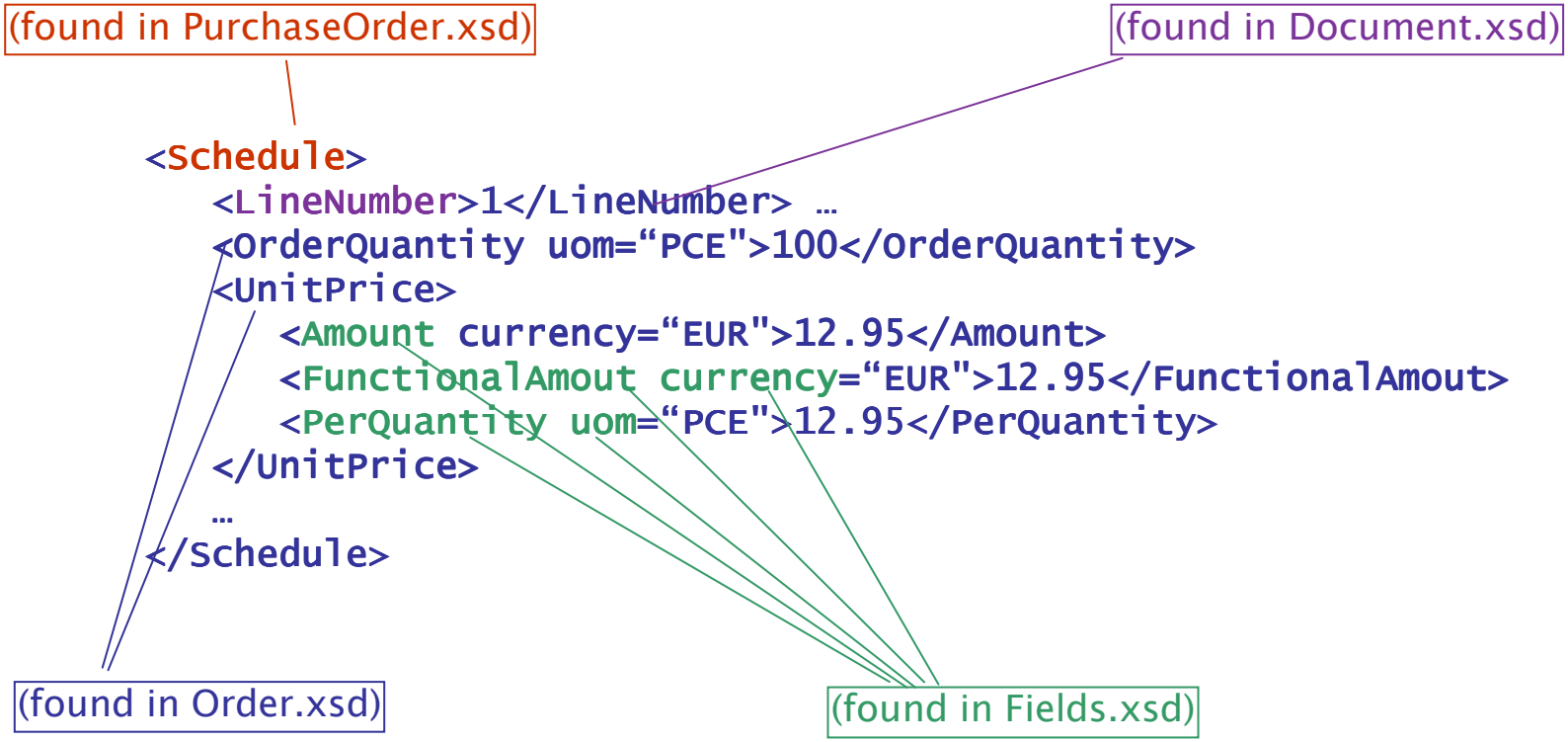
- Facets are additional properties that add further define a datatype in the context of a particular simpleType
  - Length constraints
    - `length, minlength, maxlength`
  - String and Numeric constraints
    - `pattern, enumeration, whitespace`
  - Numeric constraints
    - `minExclusive, minInclusive, maxExclusive, maxInclusive, totalDigits, fractionDigits`
  - Only some facets are applicable to each datatype.
    - e.g. Can use `minlength` with `string`, but can't use it with `date`

# Built-in Types and Applicable Facets


- string and language
  - Use length, minLength, maxLength, pattern, enumeration, whiteSpace
- boolean
  - Uses pattern, whiteSpace
- decimal, integer and positiveInteger
  - Use totalDigits, fractionDigits, pattern, whiteSpace, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive dateTime
- date and time
  - Use pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive

# Validating the Purchase Order Elements

Now I have found all my elements and attributes




From here I can start the process of validating my XML fragment. But I need to know more about data types and constraints.

# Validating the Purchase Order Elements

## Example

When I found `OrderQuantity` in order.xsd, it was declared with a type="`Quantity`," so I look for that to find out what the datatype of `OrderQuantity` should be. From the following, I know that the OrderQuantity can contain a decimal number. There are no constraints on the length of the field nor the number of decimal places allowed.

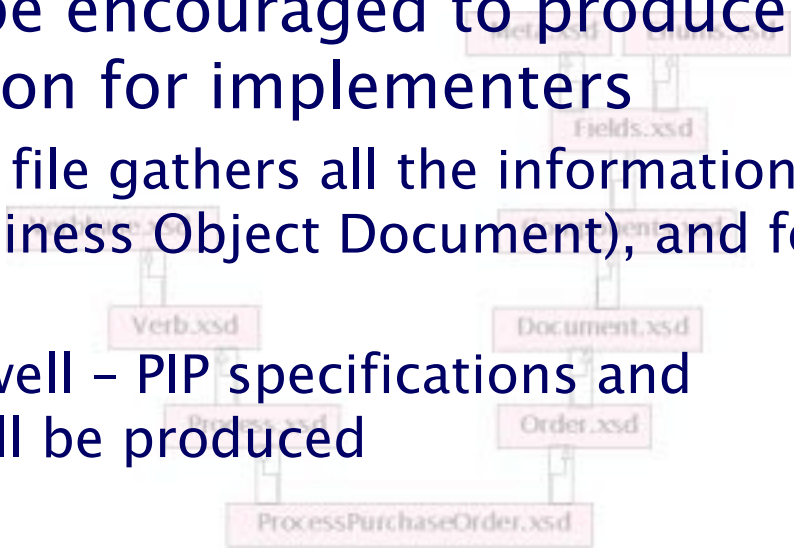
```
<xs:complexType name="Quantity">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="uom" type="UOM" use="required" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

(found in Fields.xsd)

# Do I have to do that for the entire PO?

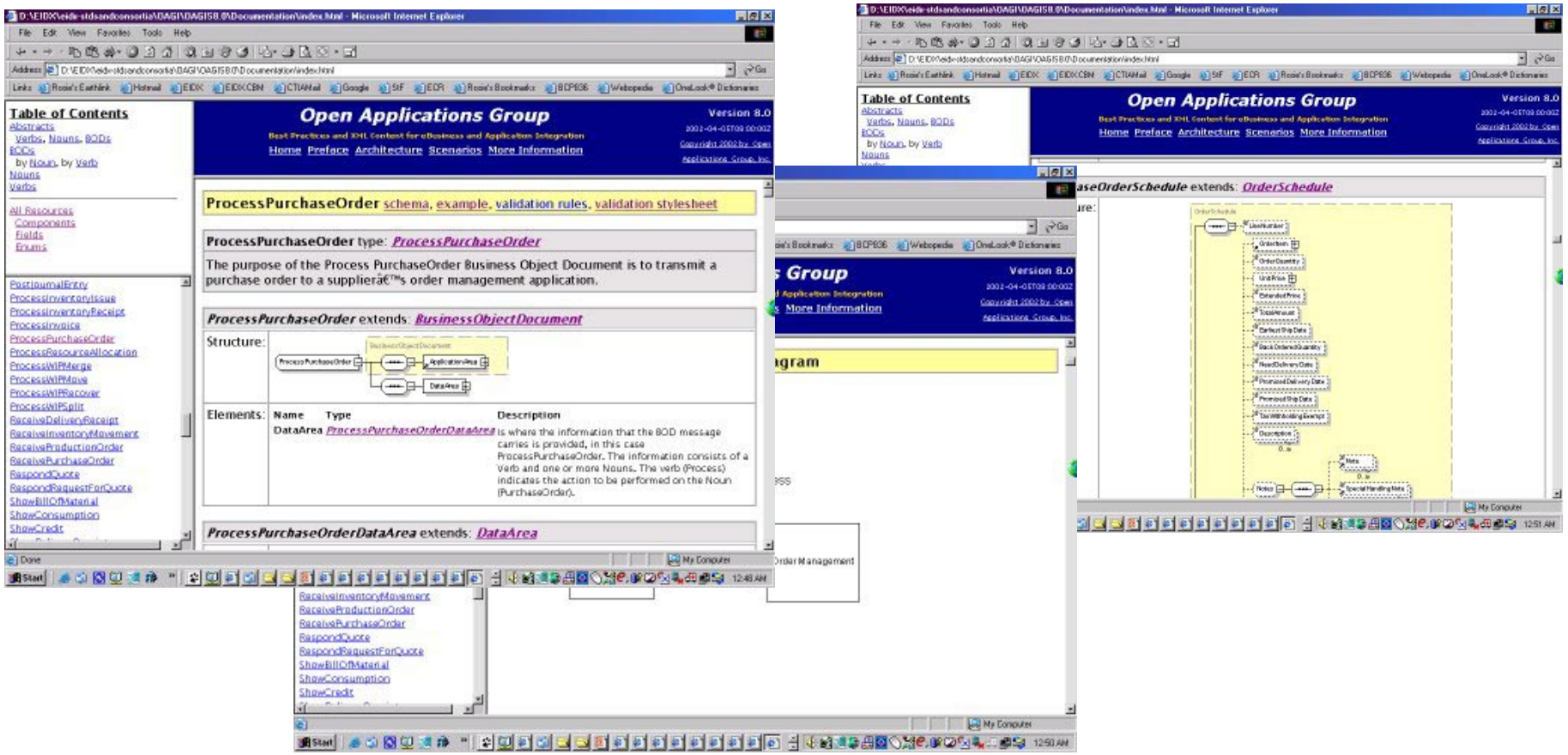
- With an object-oriented design, it can be challenging to piece together what should end up in a single message definition
  - 10 schema files in OAGIS needed to figure out what the possible contents of a purchase order are

- Standards bodies should be encouraged to produce user-friendly documentation for implementers
  - OAGI does this. One .html file gathers all the information together for each BOD (Business Object Document), and for each business document
  - RosettaNet will do this as well – PIP specifications and message guidelines will still be produced



# Pulling It All Together – Tools*

Screen shots from OAGIS8.0\Documentation\BODs\ProcessPurchaseOrder.html




** There is a list of tools on the W3C web site, at http://www.w3c.org/XML/Schema

# Questions? Comments?

Feedback is welcome!