ISSGC'05

XML Schemas (XSD)

Richard Hopkins,
National e-Science Centre, Edinburgh
June 2005

National e-Science Centre

Information Society

Overview

Goals –

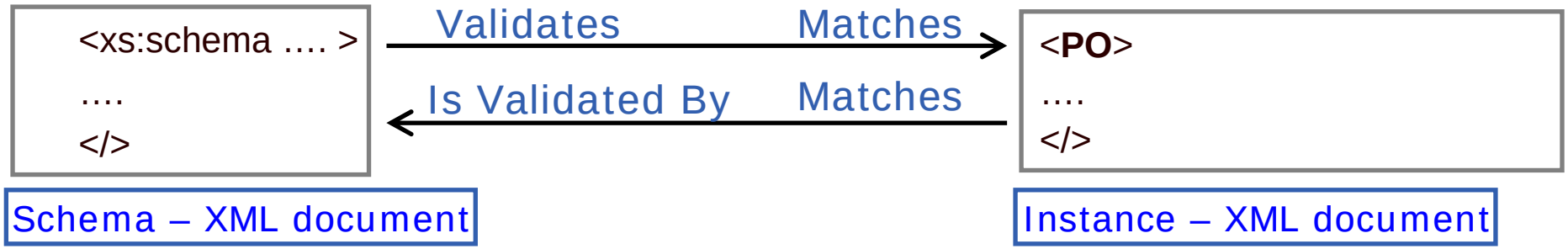
- **General appreciation of XML and Schemas**
- **Sufficient detail to understand WSDLs**

- **Structure**
  - **Schemas (XSD)**
    - **General**
    - **Elements, types and attributes**
    - **Inter-schema structures**
    - **Reality check**

Introduction to SCHEMAS

- A Schema defines the syntax for an XML language
  - An XML document can have an associated Schema
  - It is valid if it meets the syntax rules of that schema
  - This can import syntax for (parts of) other languages
- Much like programming language type declarations, But some peculiarities
  - Has declaration of attributes - needed to define XML documents
  - Three ways to define the “type” of a value
    - Giving the sub-structure directly – anonymous type
    - Referring to a Type definition
    - Referring to an Element definition
  - Allows extension points
  - Quite a complex structure –
    - Is itself an XML document
    - Easier to read than to write
  - Example – Purchase Order document –
    http://www.gs.unina.it/repository/tuesday-12

Validation / Generation



- **Schema is a type –**
  - **defines a range of possible instances**
  - **The set of all instances which are "validated by the schema"**
- **"instance is validated by the schema" – it satisfies the schema definition**
- **Alternative terminology –**
  - **"schema matches the instance"**
  - **"instance matches the schema"**

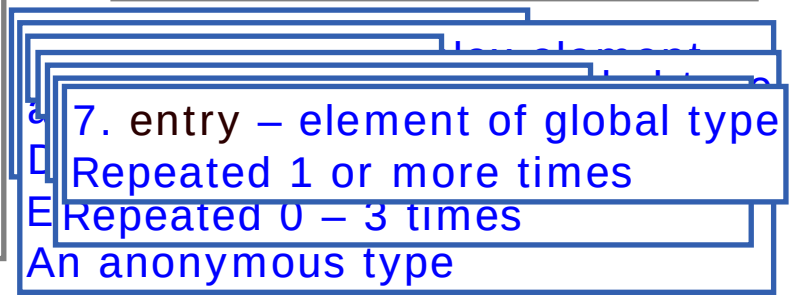
ELEMENTS etc.

- Structure
  - Schemas (XSD)
    - General
    - Elements, types and attributes
    - Inter-schema structures
    - Reality check

Main Structure

```
<xs:schema ….
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
            ….
  <xs:simpleType name="accNoT"> … </>
  <xs:complexType name="entryT"> … </>
  <xs:element name="PO">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="date"> … </>
        <xs:element name="accNo" type="accNoT"/>
        <xs:element name="customer">
          <xs:complexType> …. </>
        <xs:element ref="note"
              minOccurs="0" maxOccurs="3"/>
        <xs:element name="entry" type="entryT"
              maxOccurs="unbounded"/>
  <xs:element name="note"> … </>
```

```
<PO>
  <date>
    <USdate> 10/24/04  </></>
  <accNo> Z135-ACE</>
  <customer>
    <bill>
      <addr>…</>
      <terms>7-day</></>
    <deliver>
      <addr>…</> </> </>
  <note> …. </>
  <note> …  </>
  <entry> … </>
  <entry> … </>
</>
```



PRINT VISIBLE COMMENTS OF PREVIOUS SLIDE

1. schema “envelope” - “xs” = schema namespace. Or “xsd”

2. PO - global complex element - a sequence of child elements - Date, accno, customer, note, Entry.
An anonymous type

3. Date – see later

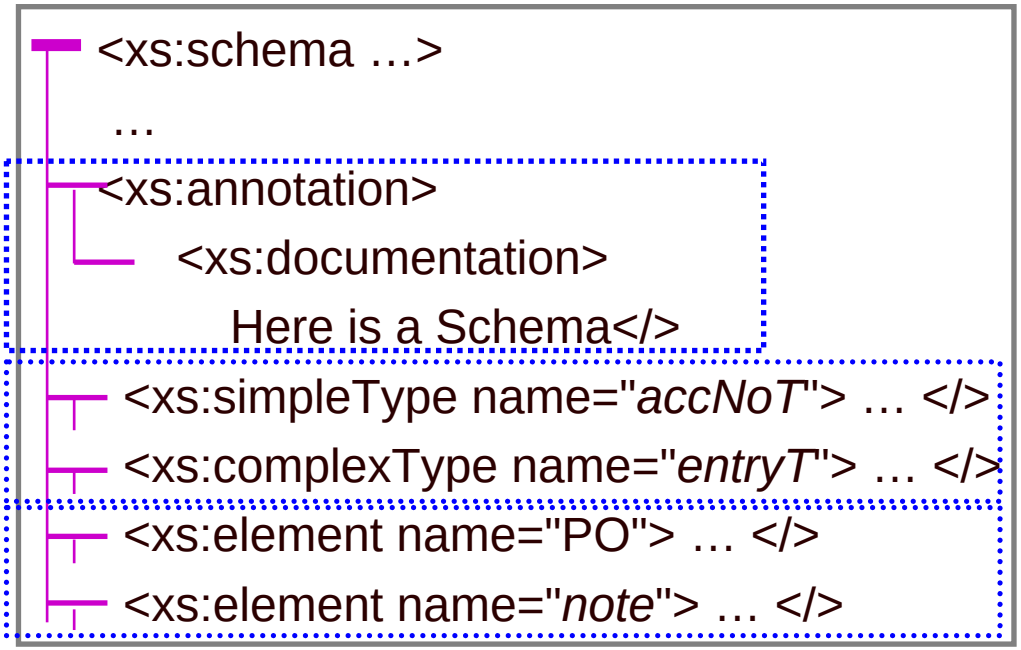
4. AccNo – element of global type

5. customer info – Nested complex element

6. note – reference to a global element – same name and type
Repeated 0 – 3 times

7. entry – element of global type
Repeated 1 or more times

Global Items



- Annotations –

Documentation (also appinfo)
Can also go deeper in
to annotate parts of structures

- Global (named) Types –

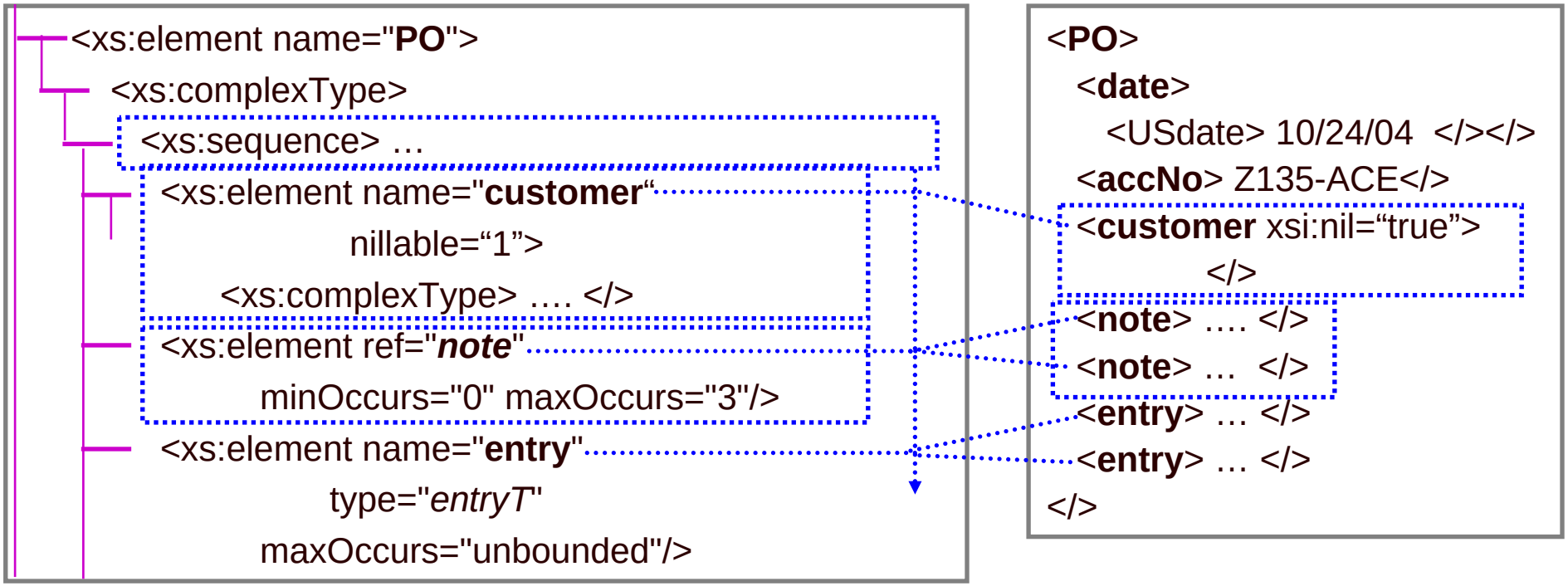
Simple or complex –
Use in giving type of element

- Elements - two roles
- The instance document can have an instance of this as its root element
  - PO or Note – not what's intended !
- Can be referenced from elsewhere as another way of giving "type" –
  - but must use same name

- Other things e.g. attributes, groups

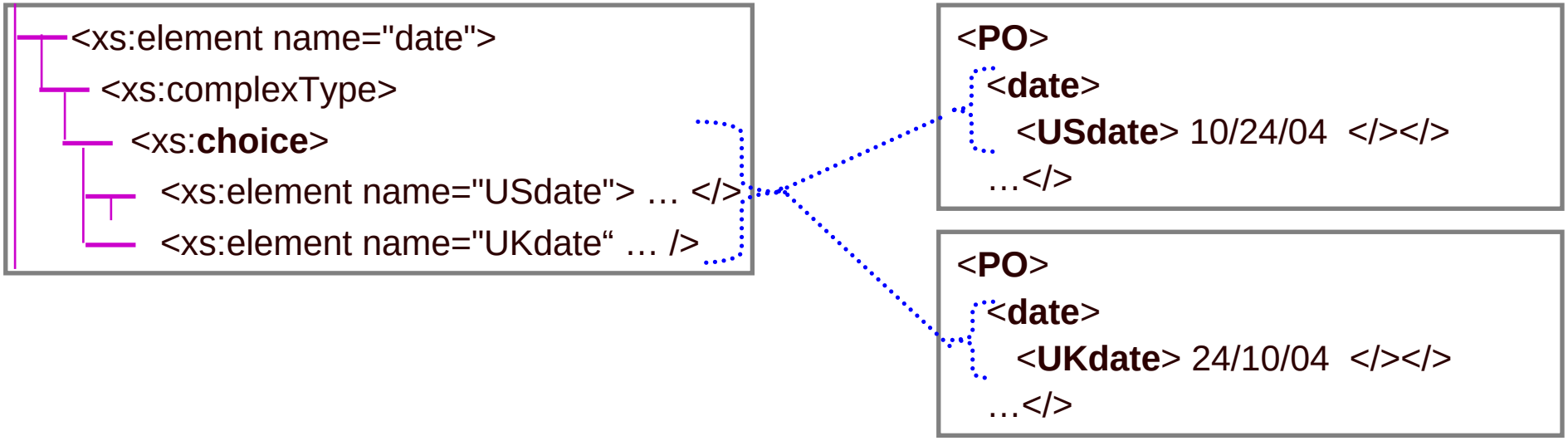
Order of global items is not significant

Features of Complex Elements



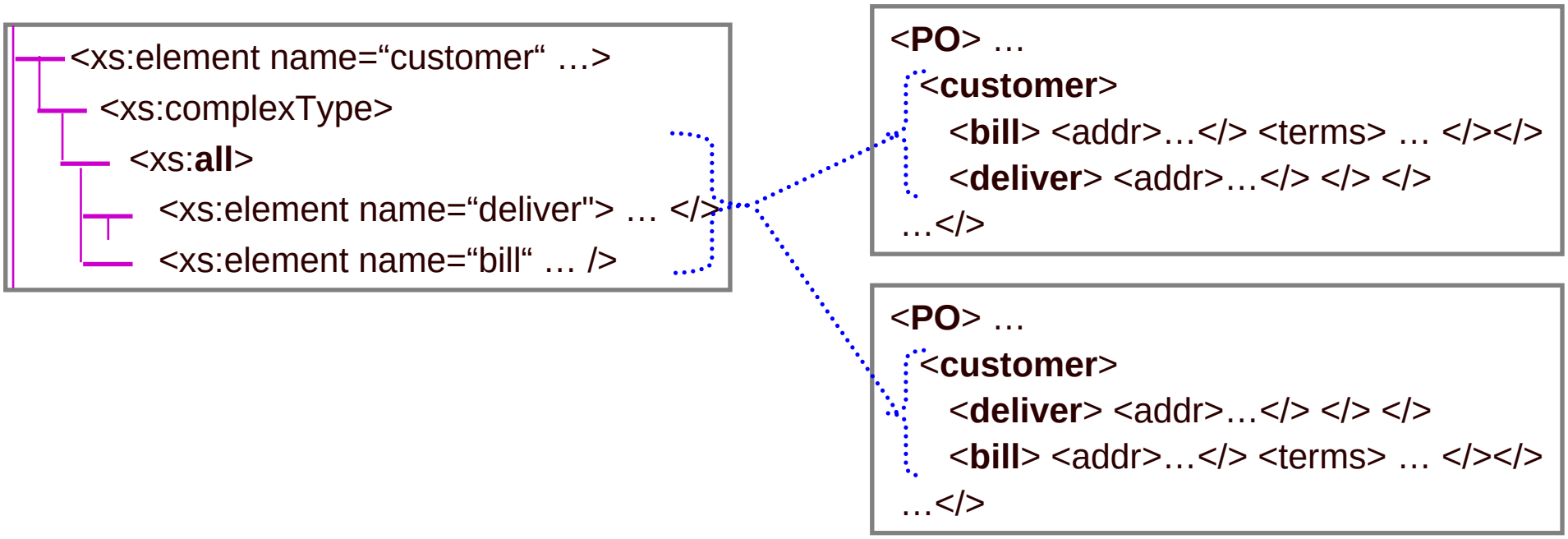
- Nillable – can match element with attribute xsi:nil = "true", and no content
- Occurrences – minOccurs , maxOccurs. Default is 1..1. max can be "unbounded"
  - This schema item can match N occurrences of the element, Min<=N<=Max
- Model (feature of type)–
  - **Sequence** – each component matched in this order
    - But each component may actually match no elements or multiple elements
    - If there are any notes – after customer and before first entry

Complex Types – Models



- Model –
  - **Sequence** – each component matched in this order
  - **Choice** – one and only one component is matched
    - But each component may actually match no elements or multiple elements
  - **All** – each component matched in any order
    - Each component must match one or zero elements – maxOccurs=“1”

Complex Elements – Models



- Model –
  - **Sequence** – each component matched in this order
  - **Choice** – one and only one component is matched
    - But each component may actually match no elements or multiple elements
  - **All** – each component matched in any order – use for "Struct"
    - Each component must match one or zero elements – maxOccurs="1"

Complex Types - Extension

```
<xs:complexType name="entryT">
  <xs:sequence>
    <xs:element ref="note" …/>
    <xs:element name="prodCode" …/>
    <xs:element name="quant"> …</></>
  ….
<xs:complexType name="XEntryT">
  <xs:complexContent>
    <xs:extension base="entryT">
      <xs:sequence>
        <xs:element name="notify" type="xs:string"/>
        <xs:element name="urgency" type="xs:string"/>
          </></></></></></>
```

```
<entry> …
  <note>….</>
  <prodCode> …</>
  <quant>… </></>
```

```
<Xentry> …
  <note>….</>
  <prodCode> …</>
  <quant>… </>
  <notify>caretaker</>
  <urgency>very</></>
```

- **XentryT inherits the components of entryT - extend the sequence**

Complex Types - Extension

```
<xs:complexType name="entryT">
  <xs:sequence>
    <xs:element ref="note" …/>
    <xs:element name="prodCode" …/>
    <xs:element name="quant"> …</></></>
  ….
<xs:complexType name="UrgentEntryT">
  <xs:complexContent>
    <xs:extension base="entryT">
      <xs:sequence>  </></></></></></>
```

- **Extension which adds no content – used to denote a special case**

Simple Elements/Types

```
<xs:simpleType name="accNoT">
  <xs:restriction base="xs:string">
    … optional restrictions ….</>

<xs:element name="PO">
  <xs:complexType>
    <xs:sequence> ….
      <xs:element name="accNo"
                              type="accNoT"/> ….
       <xs:element ref="note"
              minOccurs="0" maxOccurs="3"/>
          ….
<xs:element name="note">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      … optional restrictions …. </>
</>
```

```
<PO>  ….
  <accNo> Z135-ACE</> …
  <note> to collect </>  …..
</>
```

Element features

- **Occurrences (local element)**
- **Default / Fixed values**
- **Nillable**

Type features

- **Derivation as Restriction**
  - **Base simple type**
    - **ultimately a primitive xsd:type**
  - **Restrictions**
    - **patterns, enumerations, …**
- **Derivation as Union**
- **Derivation as List**

Attributes

```
<xs:complexType name="entryT">
  <xs:sequence> …
    <xs:element name="prodCode" type="prodCodeT"/>
      … </>
    <xs:attribute name="collect"
      type="xs:boolean" use="optional" default="false"/>
  </>
```

```
<PO>
      …..
    <entry collect="true">
      <prodCode>15-75-87</>
       ….
</>
```

- **Can associate attributes with a type**
  - By in-line definition
  - By naming a globally declared attribute - see later
- **Attribute has features –**
  - Some simple type
  - Default/fixed
  - Use – optional (default), prohibited, required
- **If it has an attribute, it must be a complex type**
  - For a naturally complex type, just add it in at the first level
  - For an actually simple type -

Adding attributes to simple types

Enabling Grids for E-sciencE

```
<xs:attribute name="units" default="metric">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="imperial"/>
      <xs:enumeration value="metric"/></></></></>
……
<xs:complexType entryT >
  <xs:sequence> …
    <xs:element name="quant“ type=“xs:decimal”/>
      </></>
```

```
<PO>
    …..
   <entry collect=“true”>
     <prodCode>15-75-87</>
     <quant units=“metric”>17.3</>
</>
```

```
<xs:element name="quant">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="units" use="required"/>
            </> </> … </>
```

- **Has attribute - is complex type**
- **But it is simple content**
- **Extends a simple type with an attribute - could be several**

INTER-SCHEMA STRUCTURES

- Structure
  - Schemas (XSD)
    - General
    - Elements, types and attributes
    - Inter-schema structures
    - Reality check

Target Namespace

```
<xs:schema
        …
        targetNameSpace= “http://company.org/forms/namespace”>
   <xs:element name=“PO”> … </>
```

- http://company.org/forms/namespace
- **The name of the language for which this schema defines the syntax**
- **This schema will only match an instance if its namespace matches -**

```
<?xml version="1.0" encoding="UTF-8"?>
<it:PO   xmlns:it= http://company.org/forms/namespace  it.att1=“…”> … </>
```

- **If schema has no targetNameSpace – it can only match un-qualified names**

Include

...www... /Forms/main.xsd

```
<schema targetNameSpace=
      "…www. …/forms/ns">
  <include schemaLocation=
      "…www…/Forms/PO.xsd"/>
  <include schemaLocation=
      "…www…/Forms/Inv.xsd"/>
```

...www... /Forms/PO.xsd

```
<schema targetNameSpace=
      "…www. …/forms/ns">
  <include schemaLocation=
      "…www…/Forms/Types.xsd"/>
  <element  name="PO"> ….</></>
```

...www... /Forms/Types.xsd

```
<schema targetNameSpace=
      "…www. …/forms/ns">
<simpleType  name=
      "AccNoT"> ….</>
….other types ….</>
```

...www... /Forms/Inv.xsd

```
<schema targetNameSpace=
      "…www. …/forms/ns">
  <include schemaLocation=
      "…www…/Forms/Types.xsd"/>
  <element  name="Inv"> ….</></>
```

- **All must be same target namespace**
- **Forms one logical schema as the combination of physically distinct schemas**
- **I.e. referencing main as the schema allows document to be an PO or an SE (stock enquiry)**
- **Allows individual document definitions to share type definitions**

Import

- **Include is to distribute the definition of this namespace (language) over multiple Schema definitions**
- **Import is to allow use of other namespaces (languages) in the definition for this language.**

…www… /Forms/PO.xsd

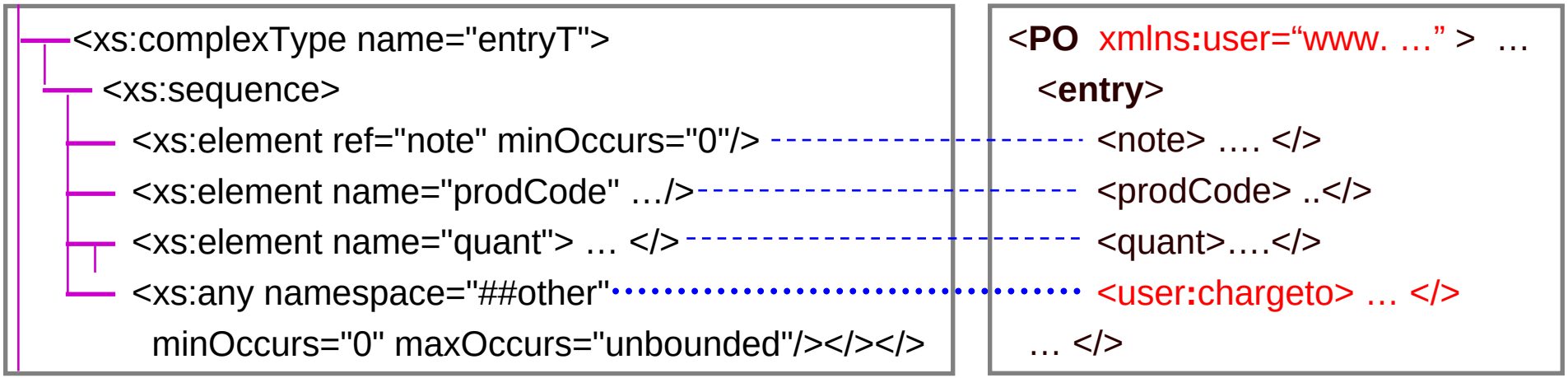
```
<schema
    targetNameSpace= "…www. …/forms/ns"
    xmlns:st ="…www…/Standards/ns" >
  <import
      namespace= "…www…/Standards/ns"
      schemaLocation= "…www… /Standards.xsd" >
  <element    name="PO"> ….
          <name="USdate" type="st:USdateT"\>…</>
</></>
```

…www… /Standards.xsd

```
<schema targetNameSpace=
      "…www. …/Standards/ns" >
<simpleType    name=
      "USdateT"> ….</>
….other types ….</>
```

- **Must have namespace definition for import's namespace**

Extension points



- **Web Services needs to accommodate extensibility**
  - A PO can be "extended" with client's own information about an order item – e.g. how the user allocates the cost -This is just reflected back in the Invoice
  - This element will be from the client's namespace – unknown when writing the schema
  - Covered by an Any element ("wildcard")
  - The Any element can define the allowed namespaces
- **Extensibility covers two kinds of language enhancement –**
  - Specialisation – namespace="##other" anything but this names space – could view user-PO as a specialisation of PO
  - Versioning – namespace="##local" this namespace

REALITY CHECK

- Structure
  - XML
    - Philosophy
    - Detailed XML Format
    - Namespaces
  - Schemas (XSD)
    - General
    - Elements, types and attributes
    - Inter-schema structures
    - Reality check

Schema Generality

- **Schemas are a moderately sophisticated type language**
  - choice; all; any
  - union; pattern; …

  More sophisticated than e.g. Java type language
- **Usage of the Schema generality depends on inter-operability issues**
- **Low interoperability**
  - A configuration table for your particular application
  - Schema only for human consumption
    - User does not write programs that use the table
  - Could use full generality of schema definition language
- **High Interoperability**
  - WSDL for your web service
  - Schema used to define the structure of SOAP messages
  - Schema must be usable by any web services toolkit
  - Type structure must be translatable into any programming language type scheme
  - Lowest common denominator – WS-I

Schemas within WSDLs

- **Complex Elements**
  - Sequence for distinctly-named component fields
  - Standard type for Array
- **Simple Elements**
  - A collection of standard types

Excludes

- **Any**
- **Extensions**
- **Choice**
- **All**
- **Repitition / optionality (maxoccurrs, minoccurrs)**
- **Mixed content**
- **…..**