PPARC Summer School, May 2005

Schemas (and XML)

Richard Hopkins,
National e-Science Centre, Edinburgh

National e-Science Centre

Information Society

Overview

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

- Structure
  - XML (review & namespaces)
  - Schema Structure
  - Schema Bureaucracy
  - Extensibility (?)

An Extensible Markup Language

XML = eXtensible Markup Language

- "Markup" means document is an intermixing of
  - Content – the actual information to be conveyed - payload
  - Markup – information about the content - MetaData

    <date>22/10/1946</date>

    <date> … </date> is markup – says that the content is a date
- Self-describing document
- date is an element of a markup vocabulary / language
  - a collection of keywords used to identify syntax and semantics of constructs in an XML document

XML Review

Root Element

Prolog – standard
<? … ?> = PI – Processing Instruction
can occur elsewhere

<!-- … --> = comment
can occur elsewhere

```
<?xml version="1.0"  encoding="UTF-8" ?>
<!- - This is an example XML document - ->
<Invoice customerType="trade" dateStyle="US">
  <customer name="NESC" creditRating="A1" />
  <item>
    <date>                          10/24/04  </date>
    <price currency="Euro">         17.34     </price>
    <product code= "A1-74"/>
    <quantity>                      17.5      </quantity>
    <memo> dear <to>Joe</to> this … </memo>
  <item>                            …         </item>
<Invoice/>
```

Element –
- Start tag & matching end tag
- Nested structure of child elements
  Or
- character data
  (or Both! – mixed data)

Children -
- "Struct" – item= (date,…,quantity)
  all different names
- "Array" – Invoice = item *

Attributes in start tag
- name/value pair – simple string
- "control" information

Empty element – name, possibly attributes,
No content, no end tag, use "< …/>"

Combined array & struct
Invoice = customer, item*, …

Notation

```
<?xml version="1.0"  encoding="UTF-8" ?>   <!- - This is an example XML document - ->
<Invoice customerType="trade" dateStyle="US">
   <customer name="NESC" creditRating="A1" />
  <item>
    <date>                    10/24/04  </>
    <price currency="Euro">   17.34     </>
    <product code= "A1-74"/>
    <quantity>                17.5      </>
  </item>…          </>
</>
```

- **Use XML a lot - Schemas, Soap, WSDLs – so clearer/briefer notations**
- **Textual – Abbreviate End Tags to just </>**
  - **direct translation to actual XML**
  - **use indentation to indicate structure**
  - **always have to actually put name in end tag !!!!**
- **Tree diagram – to emphasise structure**

Multi-lingual Documents

Usually gives namespace For Children attributes

```
<businessForms:Invoice
        customerType="businessForms:trade" >
    <date> <USnotations:date> 10/22/2004 </> </>
    <product code="A1-74" businessStandards:barCode = "23-768-252" />
    <quantity> <metricMeasures:kilos>        17.53    </..> </..> </..>
```

XML = eXtensible Markup Language – the markup vocabulary is not fixed

- **XML requires explicit definition of the language - Schema**
- **One document can combine multiple languages –**
- **red, green, blue, purple**
- **Language = "namespace" identified by prefix - namespace:name**
- **Applies to – element names, attribute names, values**
- **businessForms:Invoice**
  - **An Invoice construct within the businessForms(mythical) language**
  - **A language for business interoperability**
  - **Defines structure of documents, but**
  - **Does not prescribe the language of some individual items such as dates**
    - **Taken from separate languages - USnotations:date**

Types of XML Language

- Fundamental Standards
  - E.g. SOAP - the language for soap messages
    - soap-envelope:header soap-envelope:body
    - A soap message is an XML document and its parts are identified using this vocabulary
  - Goal is a factoring that gives pick-and-mix of combinable standards
  - Associated with any WS standard will be a Schema definition of its XML language

....

- Community conventions
  - Perhaps, our BusinessForms language

....

- Specific Application Language
  - myProgram:parameter1
  - The language used in invoking particular operations of a web service

Namespaces

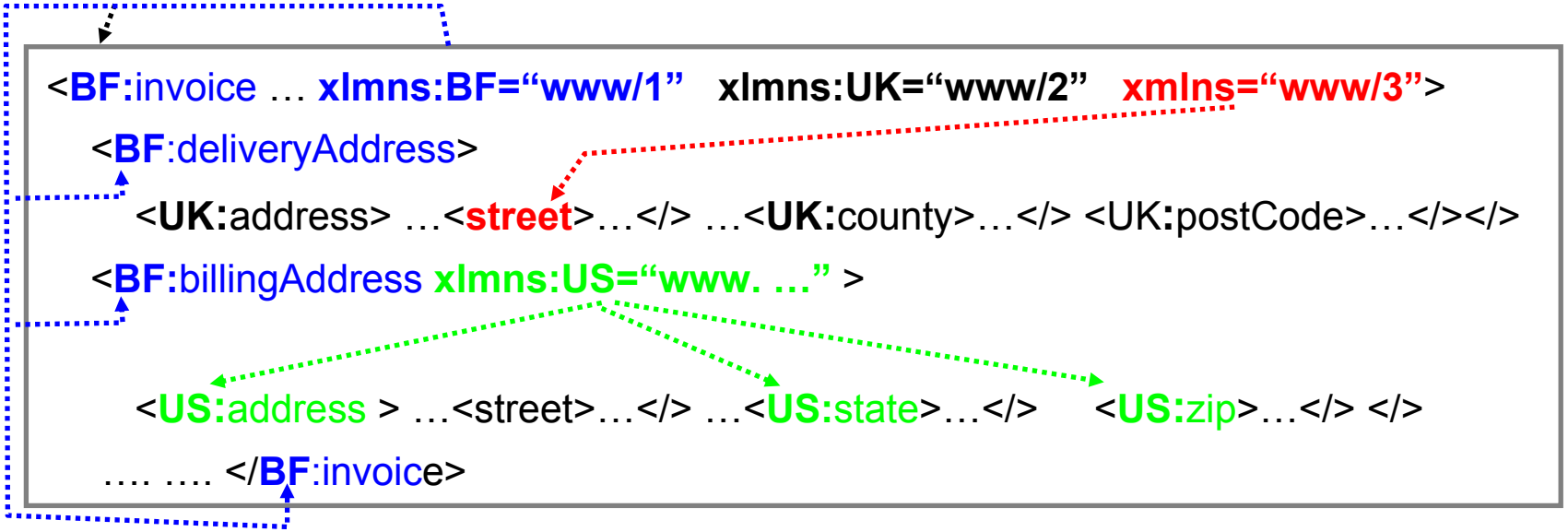
```
<invoice>                                   <!-- INT = International -->
   <deliveryAddress>
      <UK:address> …<INT:street>…</> …<UK:county>…</> <UK:postCode>…</></>
   <billingAddress>
      <US:address> …<INT:street>…</> …<US:state>…</>    <US:zip>…</> </>
    …. …. </>
```

- **A namespace (= “language”)**
  - Defines a collection of names (a vocabulary)
    - For UK : {address, county, postCode, …. }
  - Usually has an associated syntax (e.g. Schema definition)
    - address = … county, postCode, …
    - Syntax may be available to S/W processing it
  - Implies a semantics – the (programmer writing) S/W processing a UK:address knows what it means
  - Provides a unique prefix for disambiguating names from different originators
    - UK vs. US vs. INT

Namespace Names

- **To get uniqueness of namespace name, use a URI**
  - **UK:**postCode is really
    HTTP://www.UKstandards.org/Web/XMLForms:postCode (mythical)
  - The URI might be a real URL, for accessing the syntax definition, documentation, ….
  - But it may be just an identifier within the internet domain owned by the namespace owner
- **But HTTP://www.UKstandards.org/Web/XML/Forms:postCode is**
  - Tediously long to use throughout the document
  - Outside XML name syntax
    - Namespaces are not part of XML
    - A supplementary standard http://www.w3.org/TR/REC-xml-names
- **In an XML document**
  - declare a namespace prefix, as an attribute of an element
    - xmlns:**UK=**“HTTP://www.UKstandards.org/Web/XML/Forms”
  - then use that for names in that namespace - UK:postCode
    - **UK:**post code is called a **QName** (qualified name)

Namespace Prefix Declarations



- **Namespace declaration occurs as an attribute of an element**
  - i.e. within a start tag
- **Scope is from beginning of that start tag to matching end tag**
  - Excluding scope of nested re-declarations of same prefix
- **Can declare a default namespace**
  - xlmns="www/3" – this is the name space for all un-qualified names in the scope of this declaration, eg. Street
  - But no defaulting for attributes – if no prefix, no namespace

Well-formed and Valid

- **Well-formed means it conforms to the XML syntax, e.g.**
  - Start and end tags nest properly with matching names
- **Valid means it conforms to the syntax defined by the namespaces used**
  - Can't check this without a definition of that syntax –
    - Normally a Schema
    - DTD (document Type Definitions) – deprecated
    - Others type definition system
      - – some more sophisticated than Schemas
- **XMLSPY – an XML editor that can use the schema to**
  - Validate – check a document against the schema
  - Anticipate – show menu of valid options

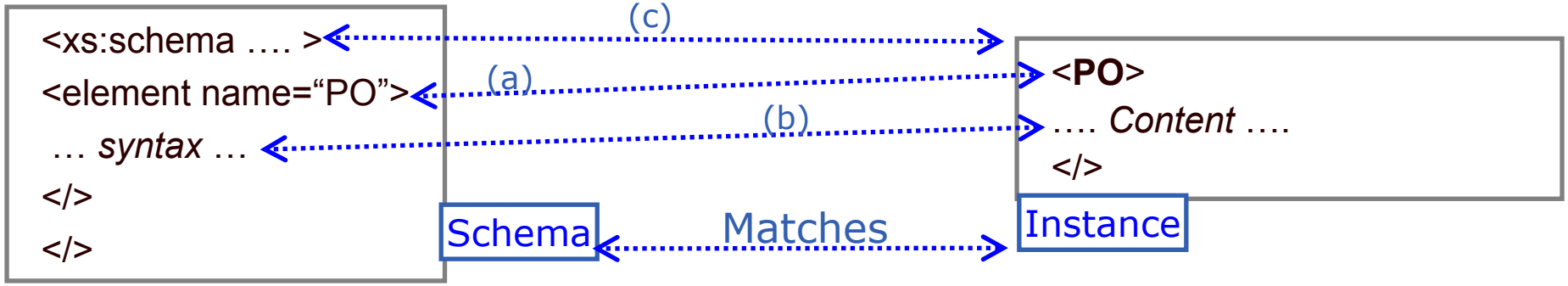
SCHEMA Structures

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

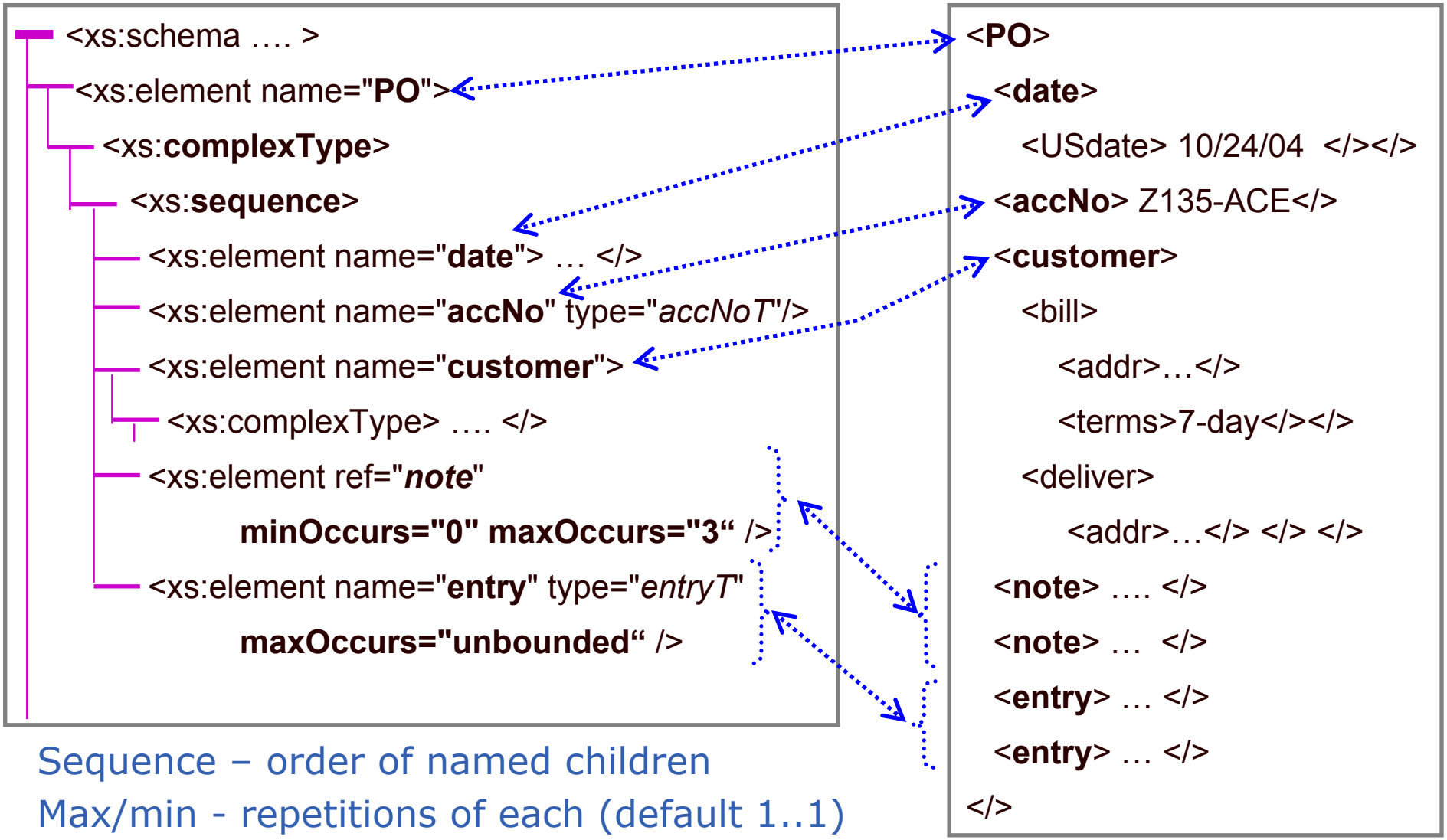
- Structure
  - XML (review & namespaces)
  - Schema Structure
  - Schema Bureaucracy
  - Extensibility

Type Validation

- **A Schema defines the syntax for an XML language in an XML document**
- **Example –** http://homepages.nesc.ac.uk/~rph/pparc/XML-Schema-Examples/
- **Schema language – xs:… or xsd:… also xsi:…**
- **Schema is a type, quite like a programming language type declaration**
  - **defines a range of possible instances**
- **"instance is validated by the schema" – it satisfies the schema definition**
- **My terminology –** "schema matches the instance"
- "instance matches the schema"
- If (a) Instance element name = a schema element name
  (b) Content matches syntax - recursively
- Then (c) Instance document matches schema



Complex Type Structure

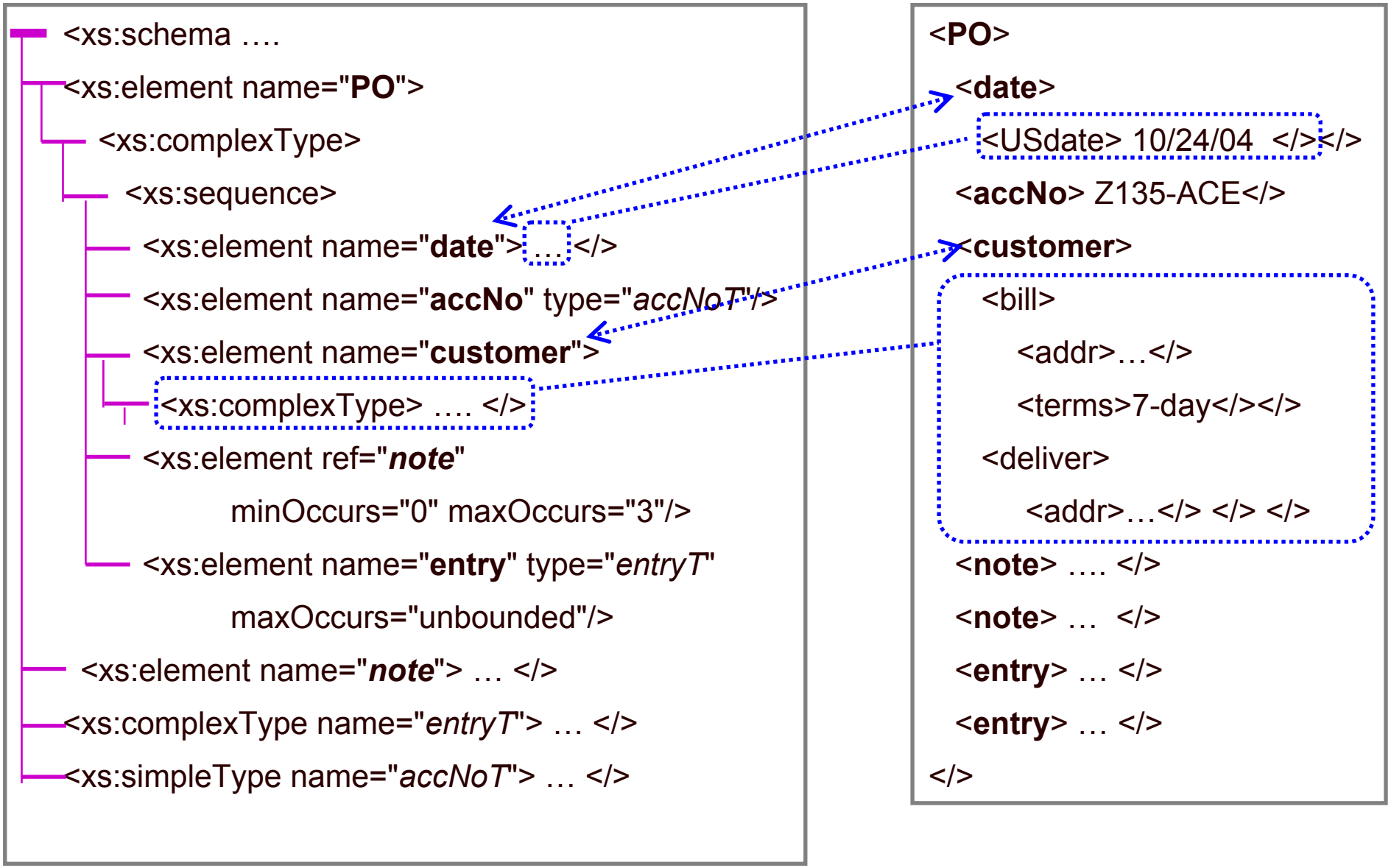


Sequence – order of named children

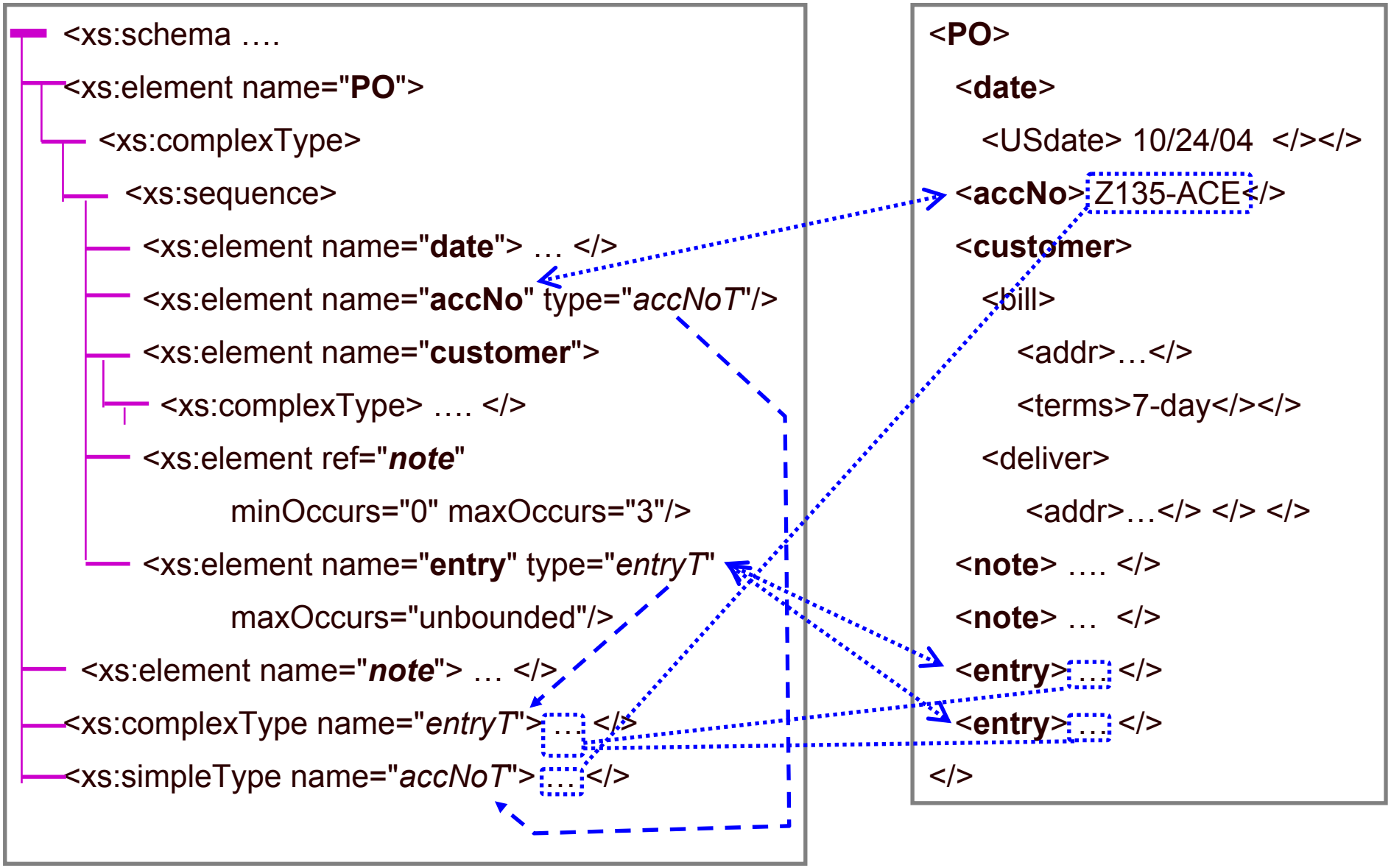
Max/min - repetitions of each (default 1..1)

Structure – three ways to define element’s structure

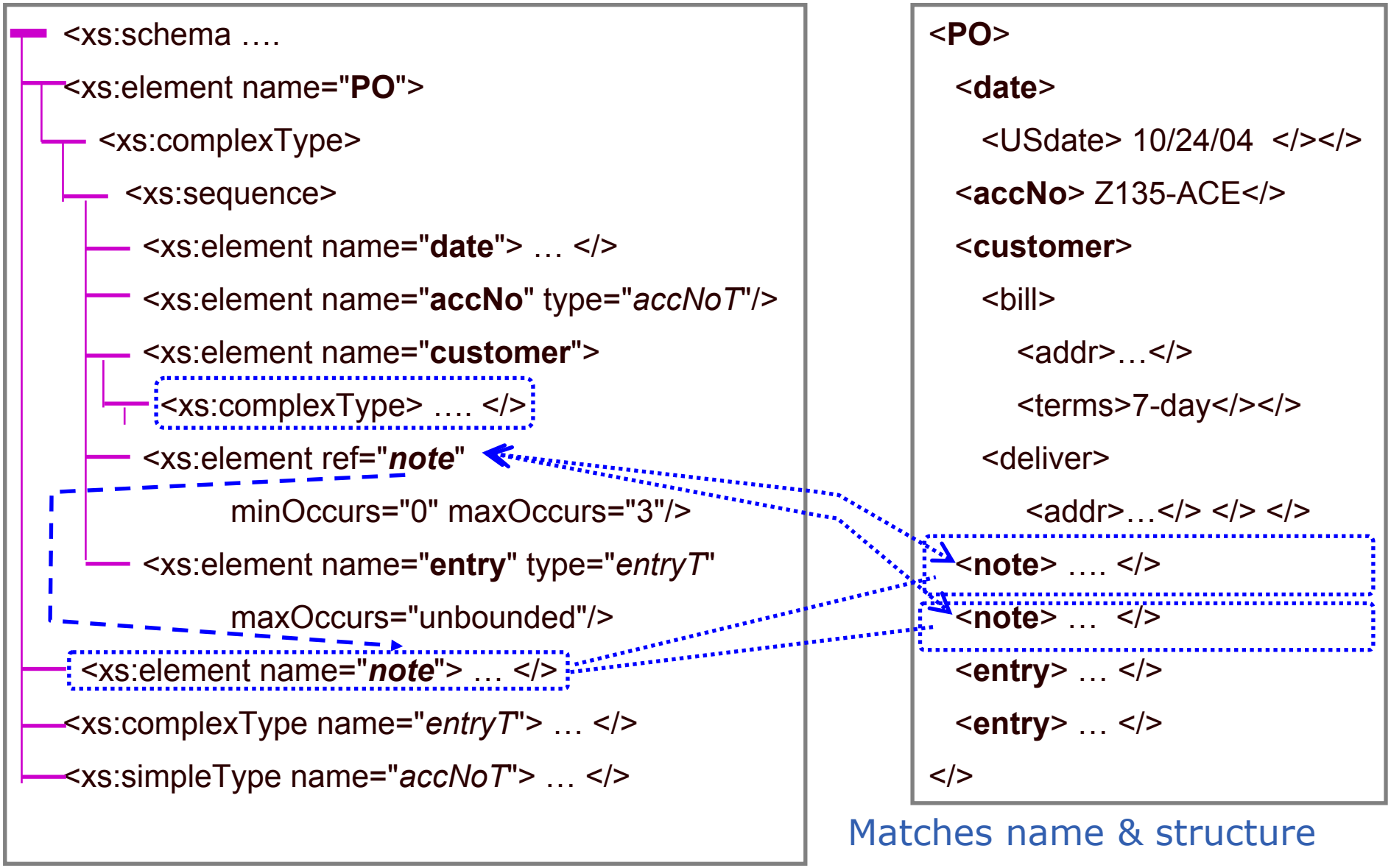
1 - Anonymous Types



2 - Named Type

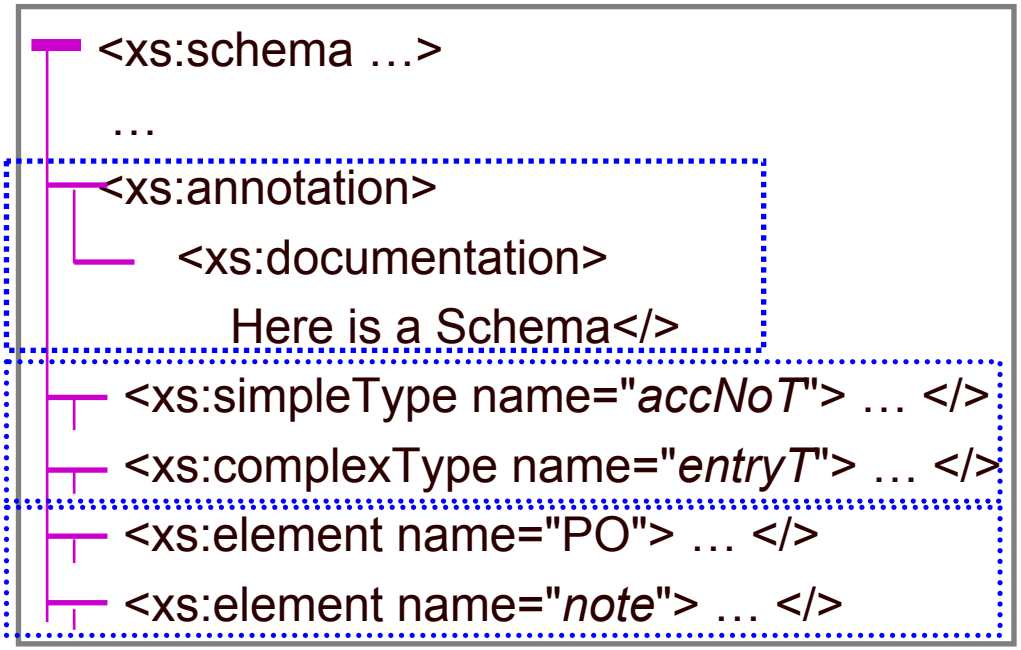

<xs:schema ….
<xs:element name="PO">
<xs:complexType>
<xs:sequence>
<xs:element name="date"> … </>
<xs:element name="accNo" type="accNoT"/>
<xs:element name="customer">
<xs:complexType> …. </>
<xs:element ref="note"
minOccurs="0" maxOccurs="3"/>
<xs:element name="entry" type="entryT"
maxOccurs="unbounded"/>
<xs:element name="note"> … </>
<xs:complexType name="entryT"> … </>
<xs:simpleType name="accNoT"> … </>
<PO>
<date>
<USdate> 10/24/04 </></>
<accNo> Z135-ACE</>
<customer>
<bill>
<addr>…</>
<terms>7-day</></>
<deliver>
<addr>…</> </> </>
<note> …. </>
<note> … </>
<entry> … </>
<entry> … </>
</>

3 - Referenced Element



Matches name & structure

Global Items



- Annotations –

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

- Global (named) Types –

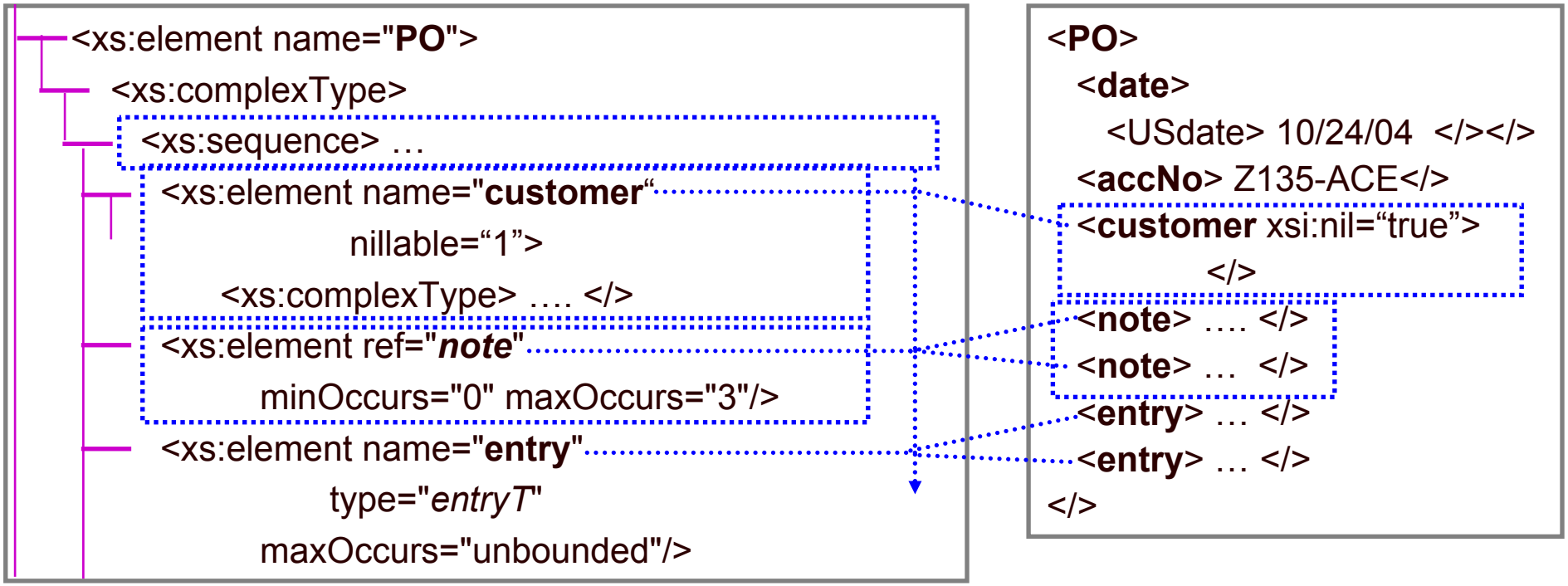
Simple or complex –
Use in giving type of element

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

- Other things e.g. attributes, groups

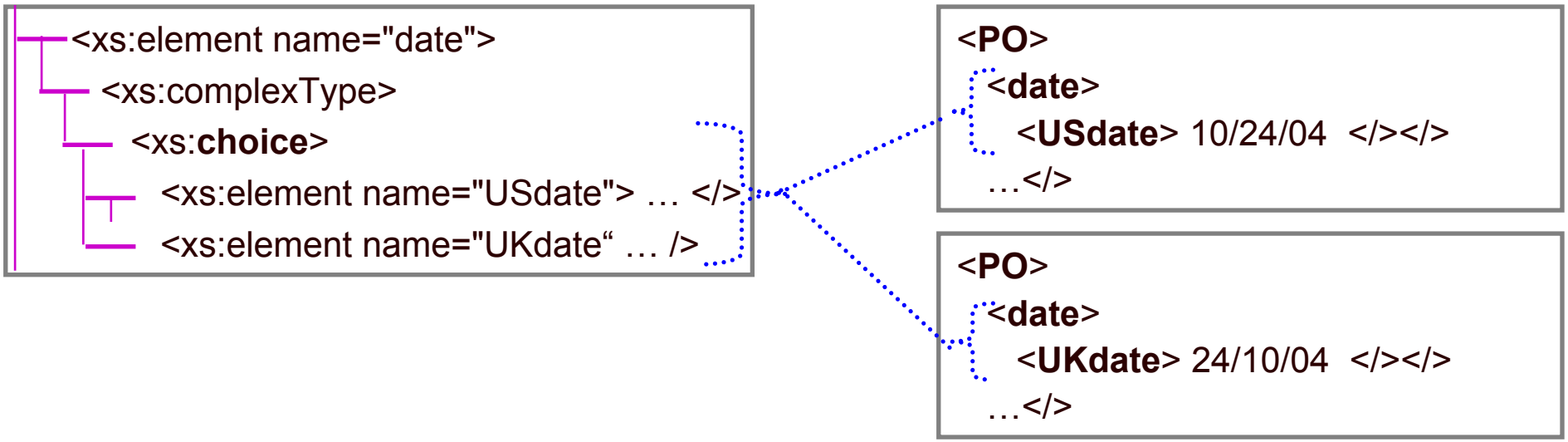
Order of global items is not significant

Features of Complex Elements



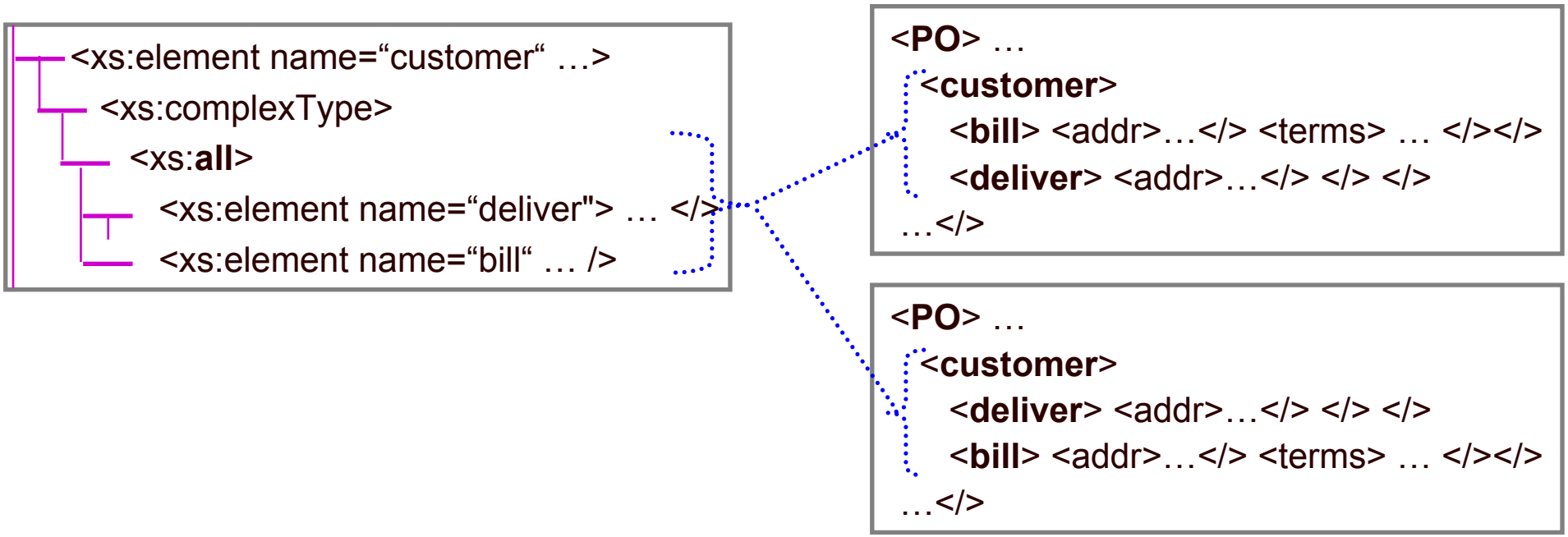
- Nillable – can match element with attribute xsi:nil = "true", and no content
- Occurrences – minOccurs , maxOccurs. Default is 1..1. max can be "unbounded"
  - This schema item can match N occurrences of the element, Min<=N<=Max
- Model (feature of type)–
  - **Sequence** – each component matched in 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

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"

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 </>  …..
</>
```

Simple Type –
- named or anonymous

Element features
- Occurrences (local element)
- Default / Fixed values
- Nillable

Type features
- Derivation as Restriction
  - Base simple type
    - ultimately a
      - primitive type - xs:type
  - Restrictions
    - patterns, enumerations, …
- Derivation as Union
- Derivation as List
- White Space handling

Attributes

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

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

- **Can associate attributes with an element**
  - By naming a globally declared attribute
  - By in-line definition
- **Attribute has features –**
  - Some simple type
  - Default/fixed
  - Use – optional (default), prohibited, required

SCHEMA Bureaucracy

- Goals –
  - General appreciation of XML and Schemas
  - Sufficient detail of XML and Schemas to understand WSDLs
- Structure
  - XML (review & namespaces)
  - Schema Structure
  - Schema Bureaucracy
  - Extensibility

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 a PO or an Inv**
- **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**

Odd & Ends

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY  … -->
<xs:schema
     elementFormDefault="qualified"
     attributeFormDefault="unqualified"
     xmlns:xs=
          “http://www.w3.org/2001/XMLSchema”
     xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
  <xs:element> …</>
  ….. </>
```

An XML document –
Although schema can be e.g. part of a WSDL document

Can put in XML level comments as well as schema level annotations

Whether a child/attribute needs to repeat the namespace qualification

```
<it:outer
      it:attr=“it:value
      xmlsns:it=“…”>
   <it:inner> …. </>
```

Standard namespace prefixes
- xs / xsd – XML Schema Definintion
- xsi – XML Schema instances

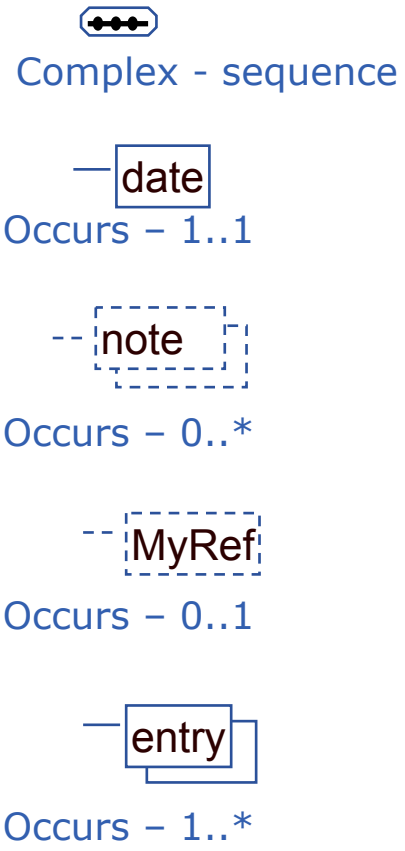
Extensibility

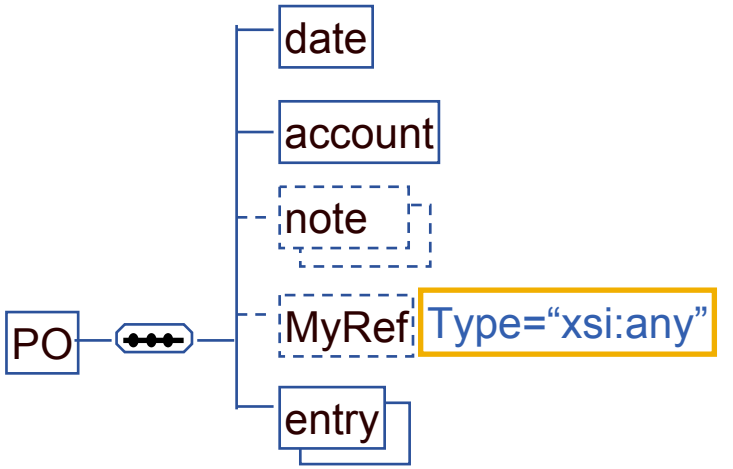
- Goals –
  - General appreciation of XML and Schemas
  - Sufficient detail of XML and Schemas to understand WSDLs

- Structure
  - XML (review & namespaces)
  - Schema Structure
  - Schema Bureaucracy
  - Extensibility

Don't Care Content

Use XMLSpy notation

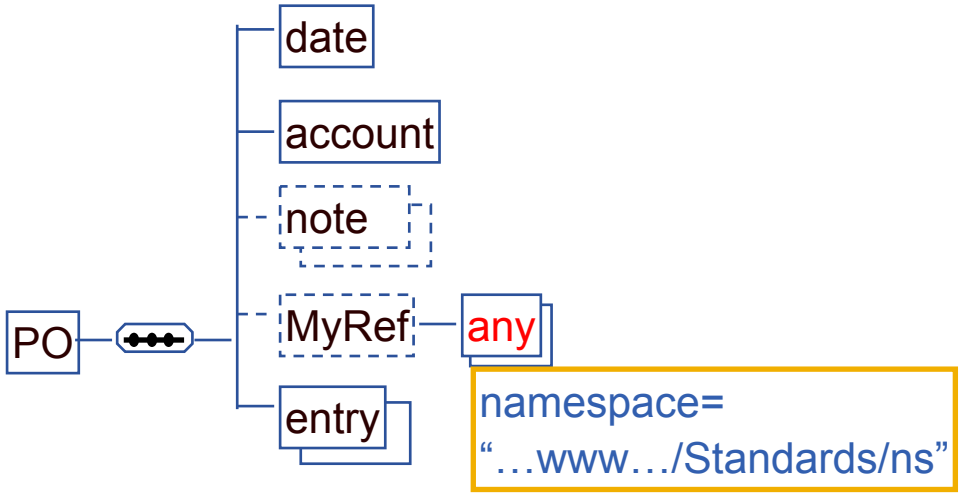





```
xlmns:me = "…."
Xlmns:you="…"
- - - - - - - - - - - - - - - - -
<you:PO>
  <you:date> …   </>
  <you:account> … </>
  <you:MyRef>
     <me:authority>…</>
     <me:chargeCode> </>
   </>
  <you:entry> ….</>
</you:PO>
```

- **Allow the originator to include their own information**
  - MyRef's do not need to be understood by this application
  - Just copied back in the invoice/statement as YourRef
- **This style, using "any" type**
  - Completely unconstrained
  - Requires a containing element, called MyRef

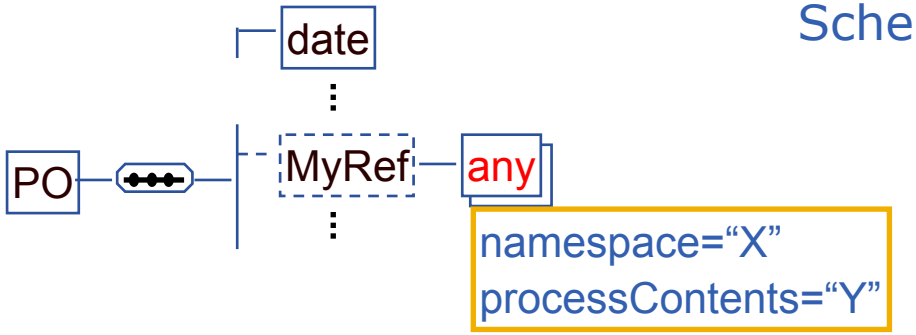
Don't Care too much Content




```
xlmns:st = "… standards/ns""
Xlmns:you="…"
- - - - - - - - - - - - - - - -
<you:PO>
  <you:date> …   </>
  <you:account> … </>
  <you:MyRef>
    <st:authority>…</>
    <st:chargeCode> </>
  </>
  <entry> ….</>
</you:PO>
```

- **Use a new kind of component,**
  - <any namespace="…" …./> instead of <element name="X" …> … </>
  - This is an Extension point – a place where this languages can be extended with an element from some other language
- **This style, using "any" element**
  - Constrained – what can be provided should be defined in the specified namespace

Any Elements



Schema

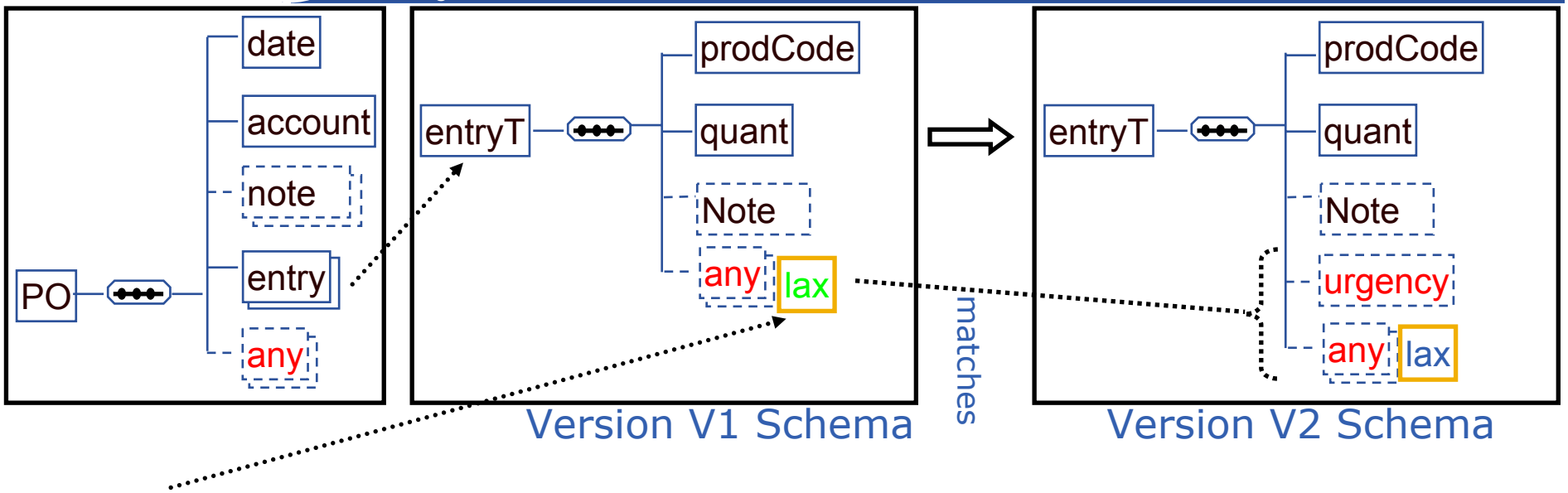
```
<xs:element name="PO">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="date">…</>
       …
      <xs:any
            namespace="X"
            processContents="Y"
            minOccurs="0"
            maxOcurrs ="ubounded"/>
      … </></></></>
```

- **Namespace options, "X" =**
  - "##any"
  - "##local" this namespace
  - "##other" anything but this namespace
  - " wwx.NS1 www.NS2 …" whitespace-separated list of namespace names, Can include "##targetnamespace
- **Processing options, "Y" =**
  - "skip" – no validation
  - "strict" – must obtain the namespace schema and validate the content
  - "lax" – validate what you can

Evolution

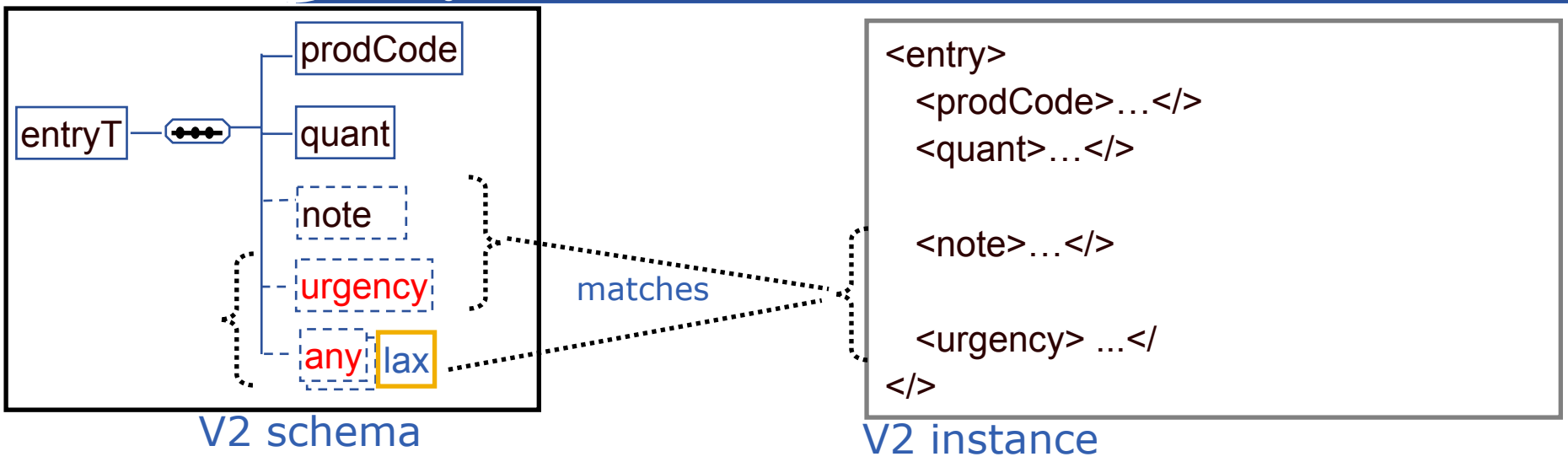
- **The loose-coupling principles of web services means that a schema should allow for change which is**
  - Forward compatible – newer versions of documents can be used by old S/W: new producer, old consumer
  - Backward Compatible – older versions of documents can be used by newer S/W : old producer, new consumer
- **Evolving may be by**
  - New Versions – the original authors enhancing the language
  - New Extensions – others enhancing the language
- **An Any element (wildcard) is an explicit extension point that allow compatibility as the language evolves**
- **Typically, for every complex element**
  - Make the last component an Any which occurs 0..* times
  - For versioning, make it ##local
  - For extensions, make it ##other

Obtaining Compatibility



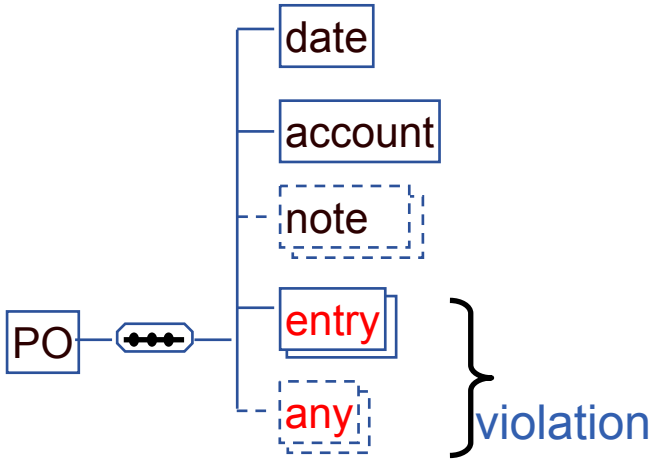
- **lax – gives forward compatibility**
  - V1 consumer (coded using V1 schema)
  - can process document produced by V2 producer
- **Optionality on new item gives backward compatibility**
  - V2 consumer
  - can process document produced by V1 producer
- **If compatibility is not the reality –**
  - use a new namespace name for the new version

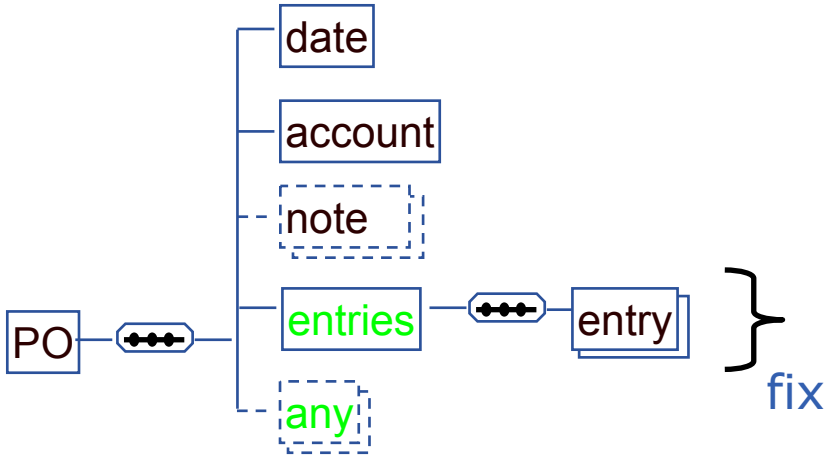
Determinism Requirement

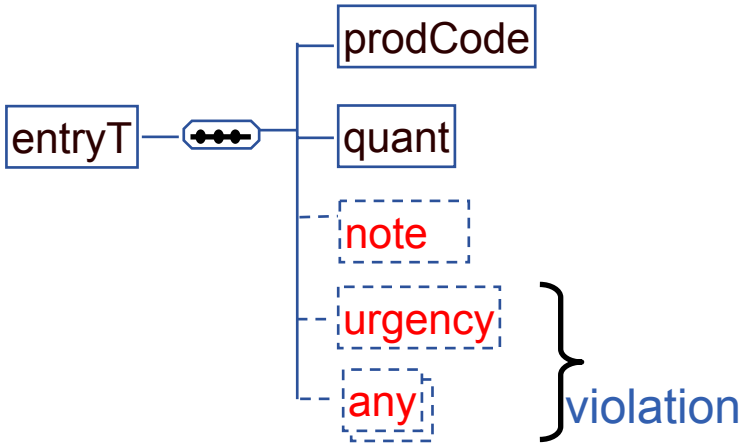


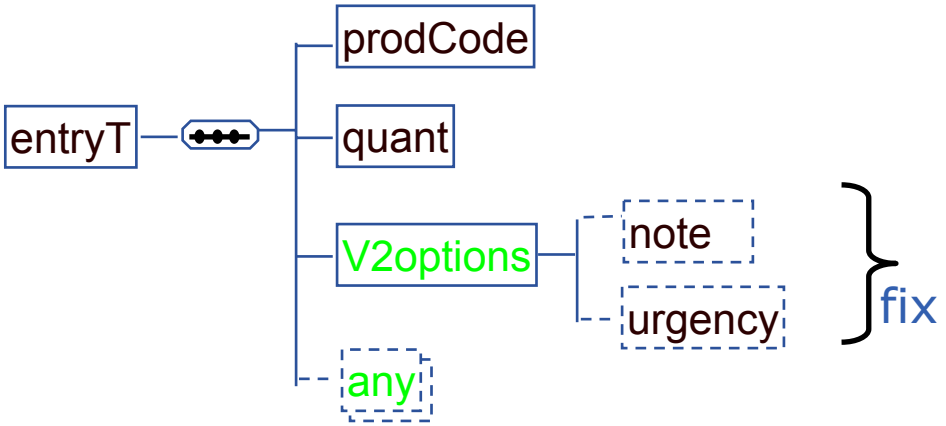
- **When "parsing" the instance, The note in instance could correspond to**
  - The note in schema
  - The any in schema
- **The Schema standard prohibits this non-determinism**
  - Can't have an Any within Choice or All
  - Can't have an Any before or after a variable occurrence component.
- **If disjoint namespaces then not a problem –**
  - <any namespace="##other">
  - The namespace will indicate whether something matches the Any

Design for Deterministic Extensibilty I



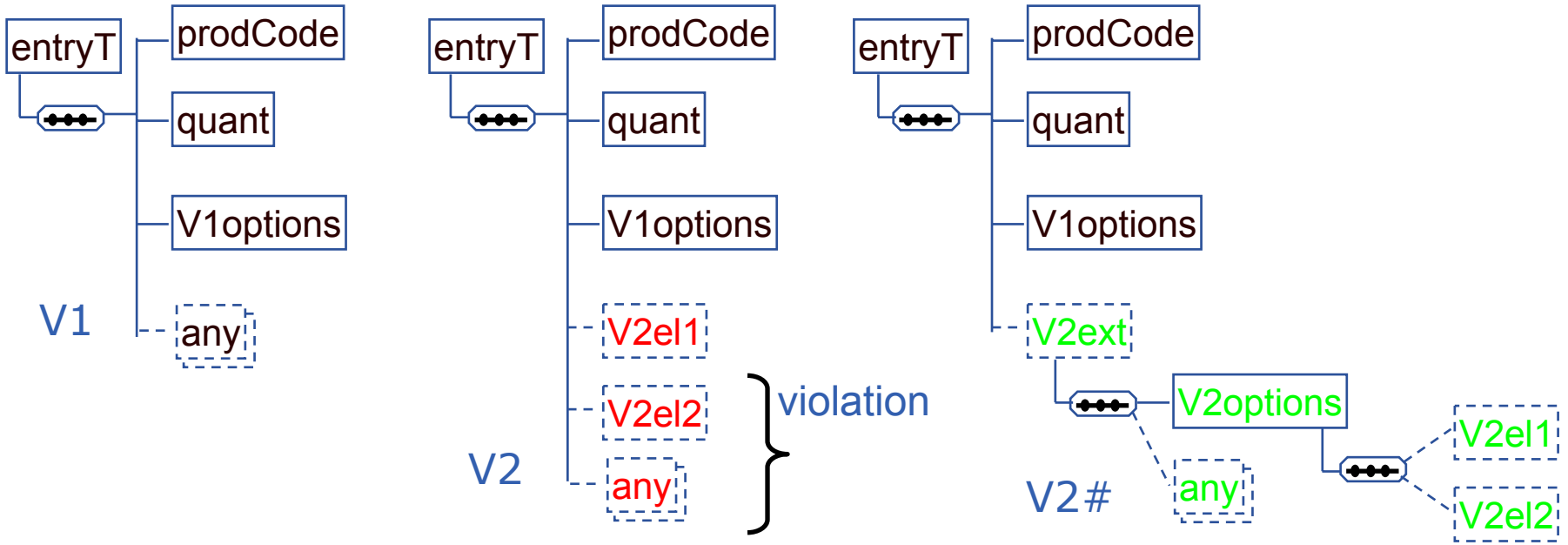






- **Put variable occurrence structure within a mandatory single-occurrence container**

Design for Deterministic Extensibilty II



- **Problem with V2 - its any for second extension**
- **Solutions (?)**
  - Make at least V2el2 mandatory, losing backward compatibility –
    - V1 document fails against V2 processor
  - Remove the extension point, losing forward compatibility
    - New schema has to be new namespace – V2 processor can't deal with V3 document
- **Solution -V2# - Nest Extensions – yes, but cumbersome**

Any Attributes

```
<xs:complexType name="entryT">
    <xs:sequence> … </xs:>
    <xs:attribute name="collect" type="xs:boolean" use="optional" default="false"/>
    <anyAttribute namespace="##any" processContents="lax">
</>
```

- **Same concept as Any elements**
  - procesContents – lax / strict / skip
  - namespace allowed – ##other etc.
- **Can't constrain how many**
- **Don't have determinism issues**
  - Because no order or repetition

Further Aspects

- **Uniqueness and key Constraints**
- **Complex Type Derivation**
- **Final and Abstract**
- **Groups**
  - Attribute
  - Element

Peculiarities

Compared with usual type definition framework

- Element component can be attribute or child
- 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
- Mixing of “struct” and “array”
- Implicit “choice” of Global elements
- Allows extension points
- Allows Mixed Content
- Quite a complex structure –
  - Is itself an XML document
  - Easier to read than to write

END

THE END