Schemas

Richard Hopkins
National e-Science Centre, Edinburgh
February 23 / 24 2005

Information Society

OUTLINE

- **Goals**
  - To be able to construct and read an XML Schema
  - To be able to use the XMLspy tool for that
- **Outline**
  - General Structure
  - Simple Types
  - Miscellany
  - Extensibility
  - Concluding Remarks
  - Practical

XML(SPY)

- **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
- **XMLSPY (free edition)**
  - Provides a graphical representation of a Schema
  - Provides for checking a XML document for validity with respect to a specified Schema
  - I Will use graphical notation of XMLSPY
  - Example files (download from http://homepages.nesc.ac.uk/~gcw/WSRF/
    - POexample.xsd – a Schema
    - POexample.xml – an instance of POexample.xsd Schema

Example Schema Structure

annotation	Here is a Schema	
attribute	units	ann: Metric or Imperial
simpleType	dateT	ann: DD/MM/YYYY or MM/DD/YYYY
simpleType	accNoT	ann: Account Number format
simpleType	prodCodeT	ann: Product Code format
☺ complexType	entryT	ann: A PO entry for one ordered item
☺ element	note	ann: An annotation on the document
☺ element	addr	ann: A UK address
☺ element	PO	ann: A Purchase Order

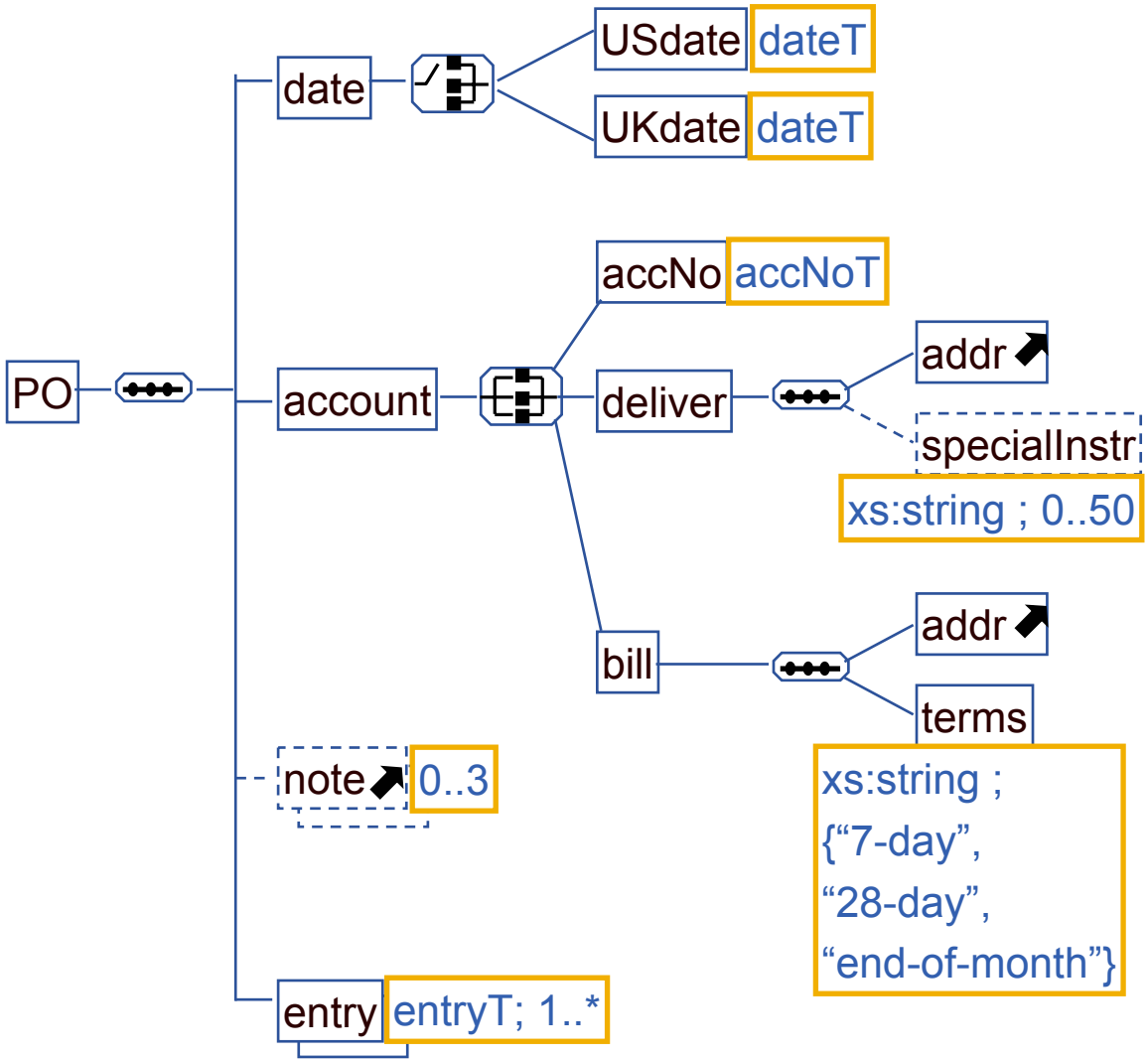
- **Top level of XMLspy -**
  - ☺ (expandable) name ann: annotation
  - Global items - can be directly referenced, here or externally
- **attribute – declares a type of attribute for use in elements**
- **annotation – supplementatry info for human / m/c processing**
- **simpleType – declares an element type without components**
- **complexType – declares an element type with components**
  - Each component is an anonymous simple type or complex type
- **element – declares an element with components – like a template**

Example Schema Structure

annotation	Here is a Schema	
attribute	units	ann: Metric or Imperial
simpleType	dateT	ann: DD/MM/YYYY or MM/DD/YYYY
simpleType	accNoT	ann: Account Number format
simpleType	prodCodeT	ann: Product Code format
☺ complexType	entryT	ann: A PO entry for one ordered item
☺ element	note	ann: An annotation on the document
☺ element	addr	ann: A UK address
☺ element	PO	ann: A Purchase Order

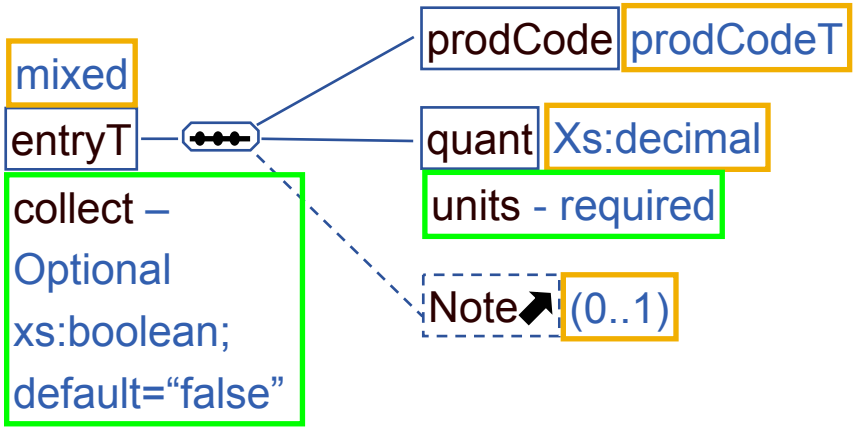
- **An element is a "element type" that**
  - could be the root element of the XML document – PO
  - Can be referenced from elsewhere as a way of giving the type of a component – addr and note –
    - an alternative to defining types addrT and noteT

Element Structuring



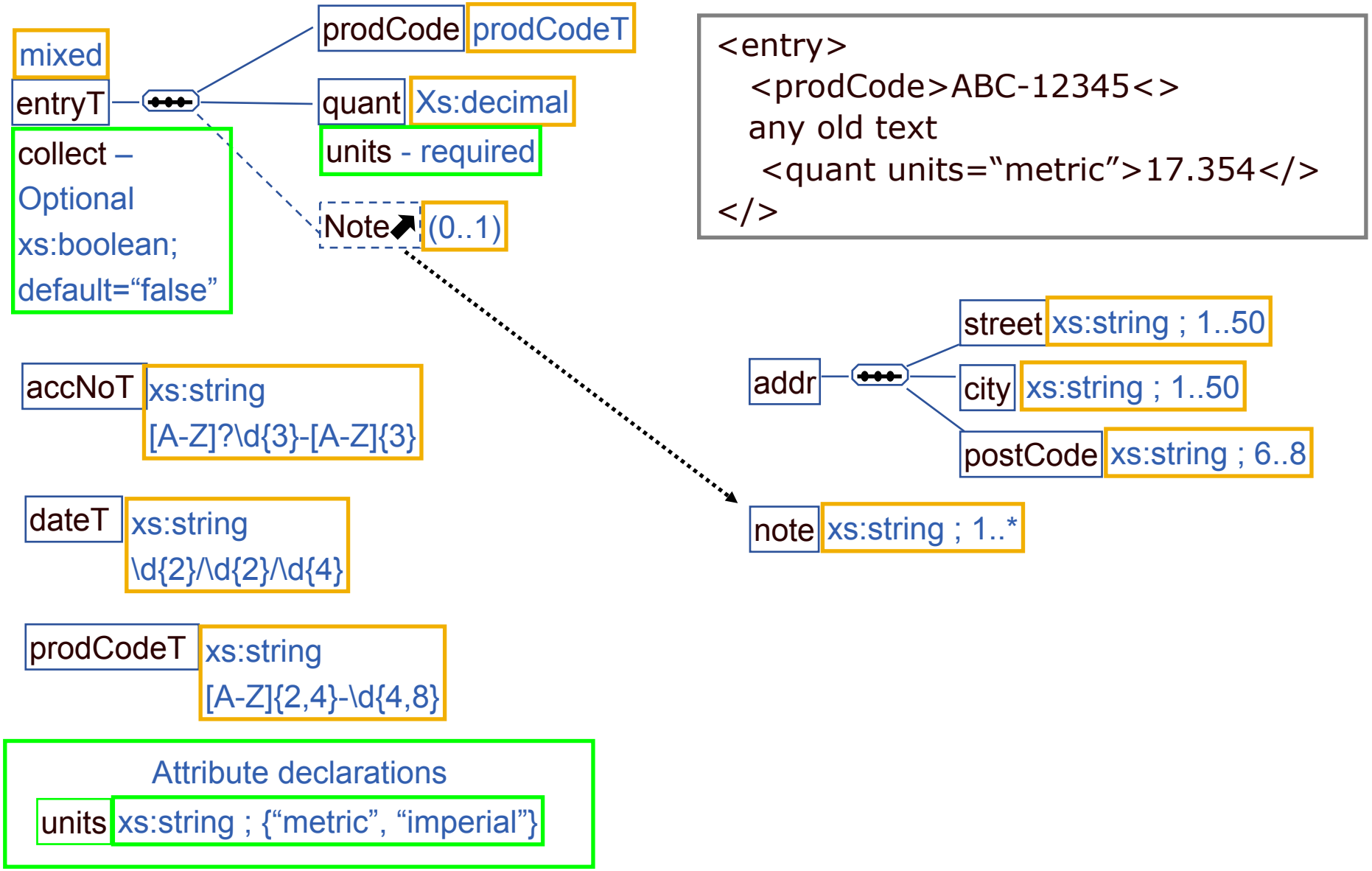

```
<PO>
  <date>
    <USdate> … </>
  </>
  <account> ….
    <accNo> … </>
    <bill>
       <addr>…</>
       <terms>7-day</>
    </>
    <deliver>
        <addr>…</>
    </>
  </>
  <note> …. </>
  <note> …  </>
  <entry> … </>
  <entry> … </>
  ….
<PO>
```

Element Structuring

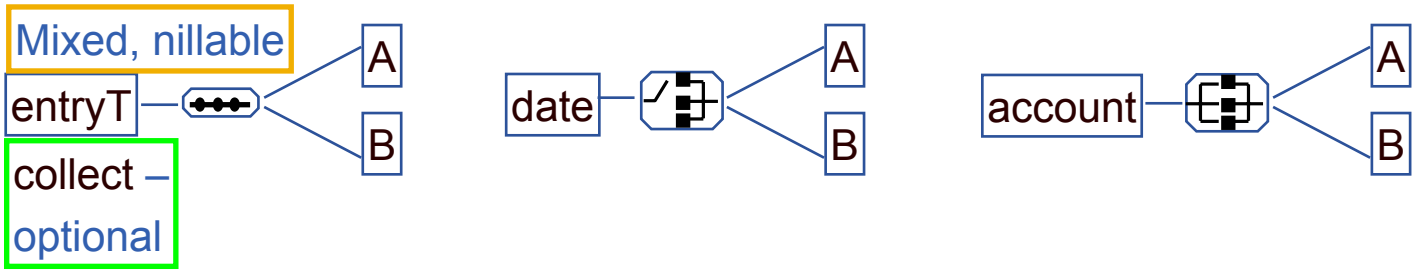



```
<entry>
  <prodCode>ABC-12345<>
  any old text
   <quant units="metric">17.354</>
</>
```

Element Structuring



Complex Content - Features



Complex Content –

- **Mixed**
  - if so text can be intermixed with element components

<entry> <prodCode>ABC-12345</> any old text <quant units="metric"> 17.354</> </>

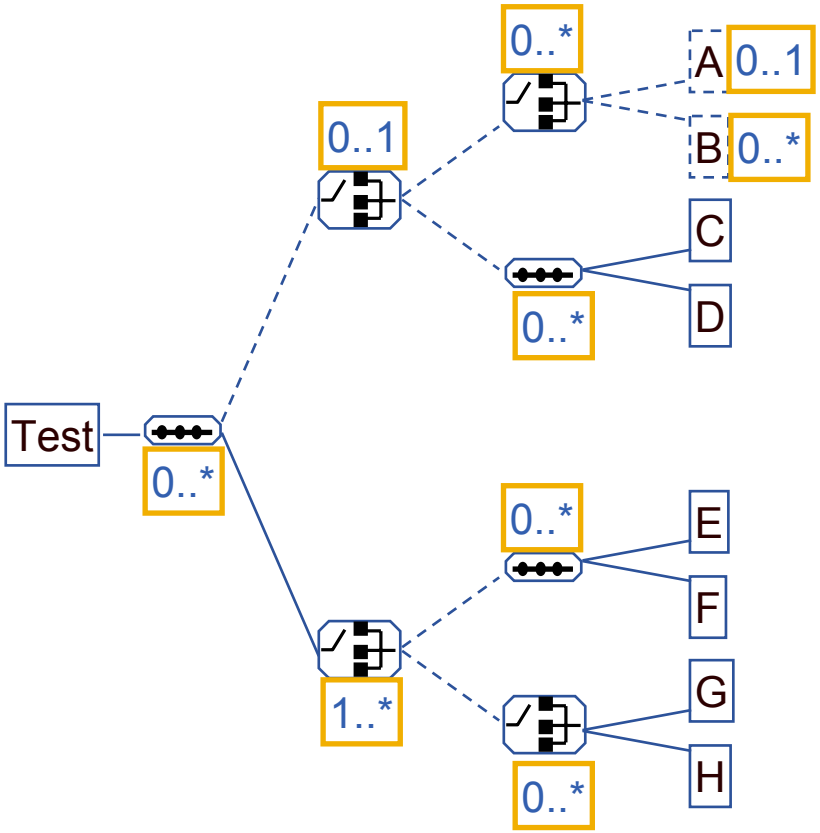
- **Nillable (element property)**
  - validated element can have attribute xsi:nil = "true" (and no content)
- **Model**


  - Sequence – All of the A, B, …components occur in that order


  - Choice – One of the A, B, … components occurs
    - For these a "component" might be empty/repeated


  - All – All of the A, B, … component occurs, in any order
    - For this, a component might be empty, but can't be repeated

Complex Content - Features

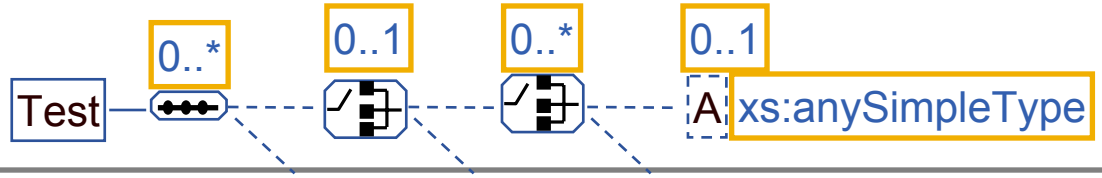


- **Multiplicities**
  - Each child element may itself represent optional and/or repeating elements
  - The constructor sequence/choice/all may itself be optional/repeating
- **Nesting**
  - The constructor may have constructor as immediate descendant
    - Except ALL can't combine with another constructor
    - Restriction is to improve parasability
- **Regular expression of child elements**

  (((A? | B*)* | (C D)*)? ((E F)*|(G | H)*

  If exclude ALL and only 1..1, 0..1 and 0..*

Actual XML




```
<xs:element name="Test“ nillable=“true” >
  <xs:complexType mixed=“true”>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice minOccurs="0">
          <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="A" type="xs:anySimpleType " minOccurs="0" />
           <xs:element name="B" minOccurs="0" maxOccurs="unbounded"/></>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
           <xs:element name="C" type="xs:anySimpleType"/>
           <xs:element name="D" type="xs:anySimpleType"/></></></>
      <xs:choice maxOccurs="unbounded">
           <xs:sequence minOccurs="0" maxOccurs="unbounded">
           <xs:element name="E" type="xs:anySimpleType"/>
           <xs:element name="F" type="xs:anySimpleType"/></>
           <xs:choice minOccurs="0" maxOccurs="unbounded">
           <xs:element name="G" type="xs:anySimpleType"/>
           <xs:element name="H" type="xs:anySimpleType"/></></></></></></></></>
```

Empty Content

```
<xs:element name="Test2">
    <xs:complexType>
        <xs:attribute name="units"/>
        <xs:attribute name="quantity" type="xs:decimal"/
    </xs:complexType></>
```

```
<Test2    units=“metric” quantity=“12.3/>
```

- **No components**
- **All information is in existence of the item and its attributes (if any)**

SIMPLE TYPES

- **Goals**
  - To be able to construct and read an XML Schema
  - To be able to use the XMLspy tool for that
- **Outline**
  - General Structure
  - Simple Types
  - Miscellany
  - Extensibility
  - Concluding Remarks
  - Practical

Simple Types/Elements

- **General features**
  - minOcc, maxOcc – repetition
  - Default/Fixed –
    - Default - the value given if absent
    - Fixed – as default, but if specified, must be this value
  - Nillable – can have attribute xsi:nil="true"
  - Derivation -
    - Restriction – some restriction on a base simple type
      - String matching [A-Z]?\d{3}-[A-Z]{3} ; integer x, 4<x<23 ; …
    - List – space-separated list of instances of a base simple type
      - A44793 632981 a564
    - Union – any one of a number of different simple types
      - UKdate or USdate
      - Instance needs <Date xsi:type="USdate">12/31/2004</>

Derivation Types

- **Derivation**
  - Base type – e.g. string, integer, defined simple type
  - Facets
    - Lengths - length, maxLength,minLength
    - whiteSpace
      - *preserve*
      - *replace – tab, newline, linefeed all replaced by space character*
      - *collapse – do replace and then collapse multiple spaces to one*
    - Limits – minInclusive, maxInclusive, minExclusive, maxExclusive
    - Digits – totalDigits, fractionalDigits – (value range and accurracy)
    - pattern – regular expression
      - **[A-Z] [^a-z] [(A-Z)-[MN]] {3,6} {,7} {3} \d . | ? * +**
    - enumeration – list of allowed values

Primitive Types and their facets

List	Lengths,pattern, enumeration	
Union	pattern, enumeration	
Atomic -		
string	Lengths, pattern, enumeration, whiteSpace	
Boolean	pattern, whiteSpace	“1”, “0”, “true”, “false”
Float	pattern, enumeration, whiteSpace, Limits	“17.54E3”, “INF”, “NAN”
Double	pattern, enumeration, whiteSpace, Limits	
Decimal	Digits, pattern, whiteSpace, enumeration, Limits	“+12.34”, “17”
hexBinary	Lengths, pattern, enumeration, whiteSpace	"0FB7"
base64Binary	Lengths, pattern, enumeration, whiteSpace	“aAb9”
anyURI	Lengths, pattern, enumeration, whiteSpace	
QName	Lengths, pattern, enumeration, whiteSpace	“xsd:element”
NOTATION	Lengths, pattern, enumeration	

Primitive Types

duration	pattern, enumeration, whiteSpace, Limits	“P1Y2M3DT10H30M”
dateTime	pattern, enumeration, whiteSpace, Limits	“2002-10-10T12:00:00”
time	pattern, enumeration, whiteSpace, Limits	“13:20:00-05:00”
date	pattern, enumeration, whiteSpace, Limits	“2002-10-10”
gYearMonth	pattern, enumeration, whiteSpace, Limits	“1999-05”
gYear	pattern, enumeration, whiteSpace, Limits	
gMonthDay	pattern, enumeration, whiteSpace, Limits	
gDay	pattern, enumeration, whiteSpace, Limits	
gMonth	pattern, enumeration, whiteSpace, Limits	

Built in (derived Types)

- **anyType** **Union of them all**
  - Complex types
  - anySimpleType
    - Primitives – **decimal, string,** anyURI, QName, boolean, float, Times/Durations, Binaries
    - Derived by restriction
      - decimal –
        - Integer
        - nonPositiveInteger
        - ….
      - string
        - normalisedString each whitespace character become a space
        - token

Tokens

- **token**
  - A string with no leading or training spaces and only single spaces elsewhere
    - “This is a Token” “ This is not “
    - A tokenized string
- **Derivations of token**
  - Corresponding to various XML constructs (to ease definition and parsing of XML documents) – **name language**

MISCELLANY

- **Goals**
  - To be able to construct and read an XML Schema
  - To be able to use the XMLspy tool for that
- **Outline**
  - General Structure
  - Simple Types
  - Miscellany
  - Extensibility
  - Concluding Remarks
  - Practical

Attributes Declarations

- **Attribute has properties –**
  - Some simple type
  - Default/fixed
  - Use – optional (default), prohibited, required

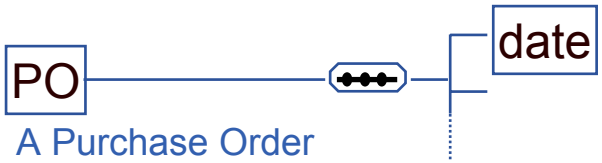
```
<xs:attribute name="TestA" use="required" fixed="fixation">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="22"/>
      <xs:minLength value="1"/>
      <xs:maxLength value="4"/>
      <xs:whiteSpace value="replace"/>
      <xs:pattern value="a|b"/>
      <xs:enumeration value="type1"/>
      <xs:enumeration value="type2"/></></></></>
```

Annotations

- **To annotate a schema for the benefit of**
  - human readers – a **documentation** element
  - Applications – an **appinfo** element

```
<xs:element name="PO">
   <xs:annotation>
      <xs:documentation>A Purchase Order</>
      <xs:appinfo>How to do it</></> …. </>
```

annotation	Here is a Schema	
attribute	units	ann: Metric or Imperial
simpleType	dateT	ann: DD/MM/YYYY or MM/DD/YYYY
….		



Namespaces –Target Namespace

```
<?xml version="1.0" encoding="UTF-8"?><!-- edited with XMLSPY … -->
<xs:schema elementFormDefault=“unqualified“
attributeFormDefault="unqualified"
     xmlns:xs=http://www.w3.org/2001/XMLSchema
     targetNameSpace= “http://company.org/forms/namespace”
     xmlns=“http://company.org/forms/namespace”>
<xs:element name=“outer”> ….<xs:element name=“inner”> ….</> …. </>
<xs:attribute name=“att1“ …>…</> </>
```

- **The name of the language for which this schema defines the syntax**
- **This schema will only validate an instance if its namespace matches -**

```
<?xml version="1.0" encoding="UTF-8"?><!-- edited with XMLSPY … -->
<it:outer   xmlns:it= http://company.org/forms/namespace  it.att1=“…”>
   <inner> …</>
   <inner> … </></>
```

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

Qualification Form

```
<xs:schema elementFormDefault="unqualified"    attributeFormDefault="unqualified"
<xs:element name="outer"> ….<xs:element name="inner"> ….</> …. </>
<xs:attribute name="att1" …>…</> </>
```

```
<it:outer   xmlns:it= http://company.org/forms/namespace  att1="…">
   <inner> …</> … </>
```

- **The root element name has to be qualified**
- **This requires other names to be unqualified**

```
<xs:schema elementFormDefault="qualified"    attributeFormDefault="qualified"
<xs:element name="outer"> ….<xs:element name="inner"> ….</> …. </>
<xs:attribute name="att1" …>…</> </>
```

```
<it:outer   xmlns:it= http://company.org/forms/namespace  it:att1="…">
   <it:inner> …</> … </>
```

- **This Requires other names also to be qualified**
- **Can override the defaults by defining form for an element**

Qualification Form

- **Normal is**
  - Schema requires qualified names, unqualified attributes
  - Instance uses default qualifier (only applies to element names)

```
<xs:schema elementFormDefault="qualified"     attributeFormDefault="unqualified"
<xs:element name="outer"> ….<xs:element name="inner"> ….</> …. </>
<xs:attribute name="att1" …>…</> </>
```

```
<outer   xmlns:= http://company.org/forms/namespace  att1="…">
   <inner> …</> … </>
```

- **Equivalent to**

```
<it:outer   xmlns:it= http://company.org/forms/namespace  att1="…">
   <it:inner> …</> … </>
```

Include

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

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

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

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

Importation

- **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="date" type="st:USdateT"\>…</>
</></>
```

…www… /Standards.xsd

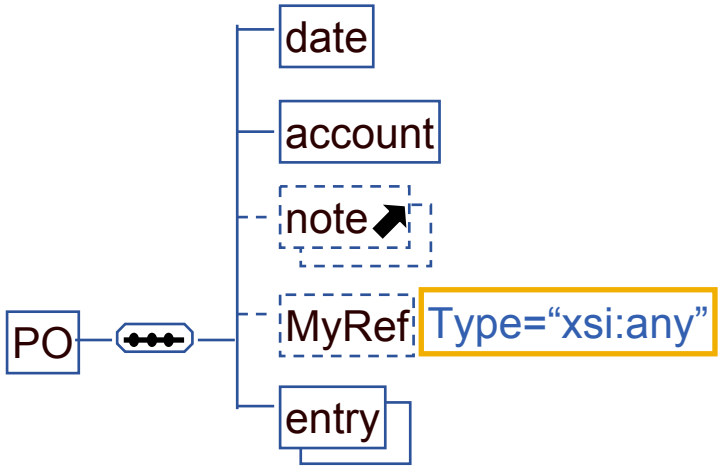
```
<schema targetNameSpace=
      "…www. …/Standards/ns" >
<simpleType    name=
      "USdateT"> ….</>
….other types ….</>
```

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

EXTENSIBILITY

- **Goals**
  - To be able to construct and read an XML Schema
  - To be able to use the XMLspy tool for that
- **Outline**
  - General Structure
  - Simple Types
  - Miscellany
  - Extensibility
  - Concluding Remarks
  - Practical

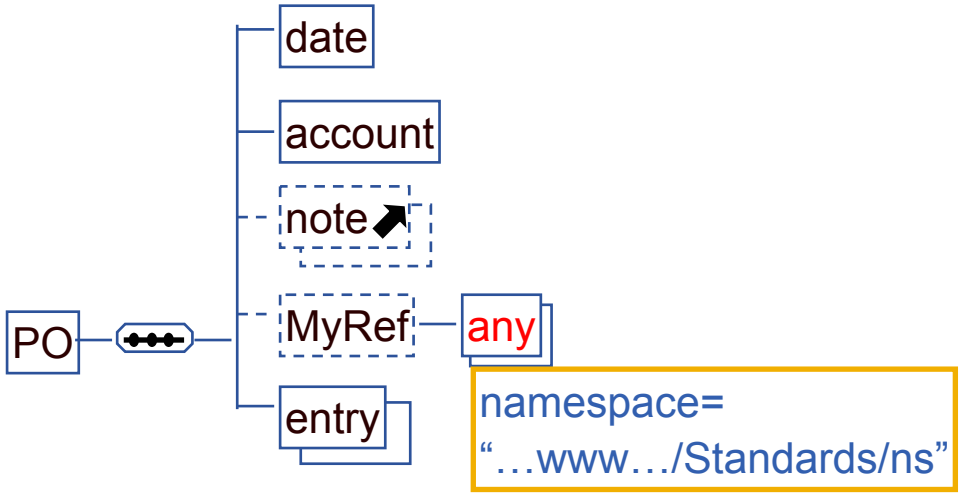
Don't Care Content




```
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 appication
  - Just copied back in the invoice/statement as YourRef
- **This style, using "any" type**
  - Completely unconstrained
  - Requires a containing element, called MyRef

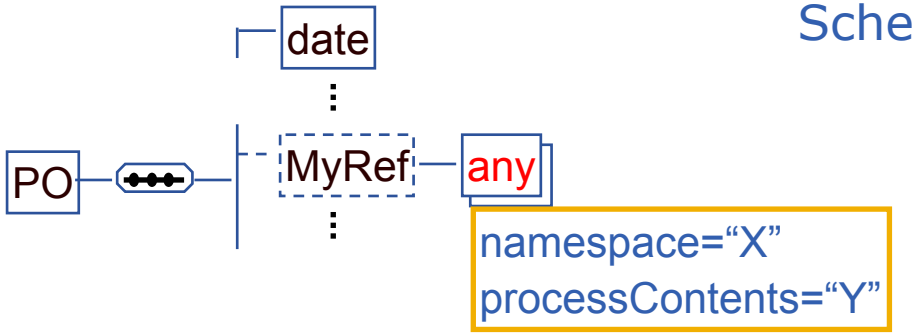
Don't Care too much Content




```
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

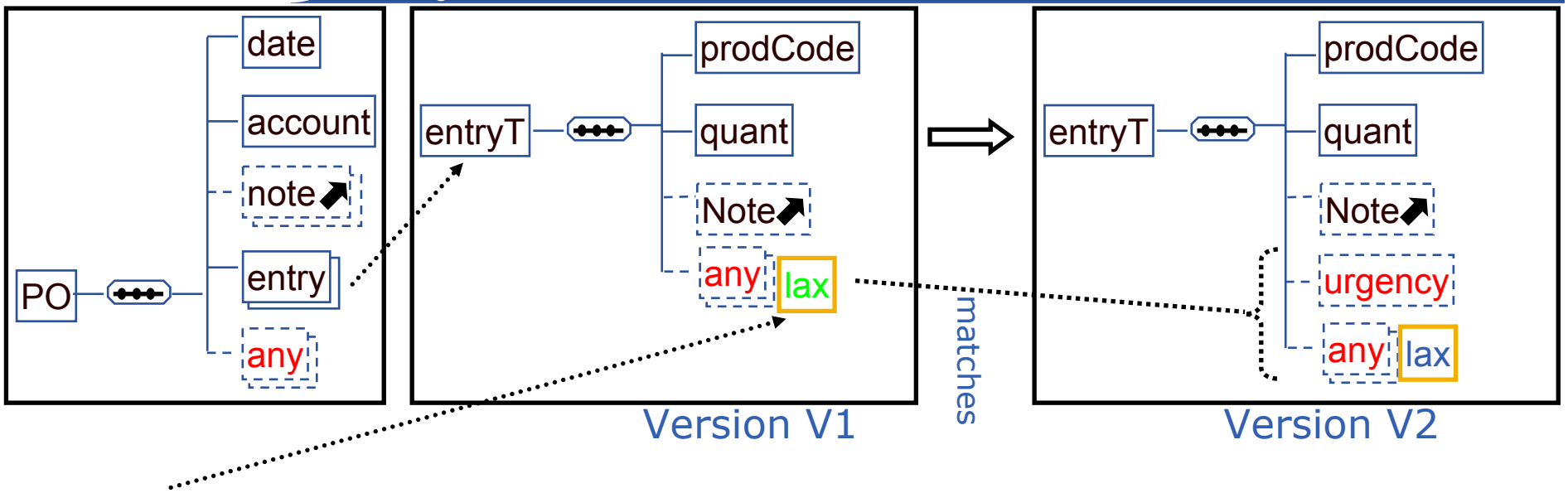
```
<xs:element name="PO">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="date">…</>
       …
      <xs:any
            namespace="X"
            processContents="Y"
            minOccurs="0"
            maxOcurrs ="ubounded"/>
      … </></></></>
```

- **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 conten
  - "lax" – validate what you can

Evolution

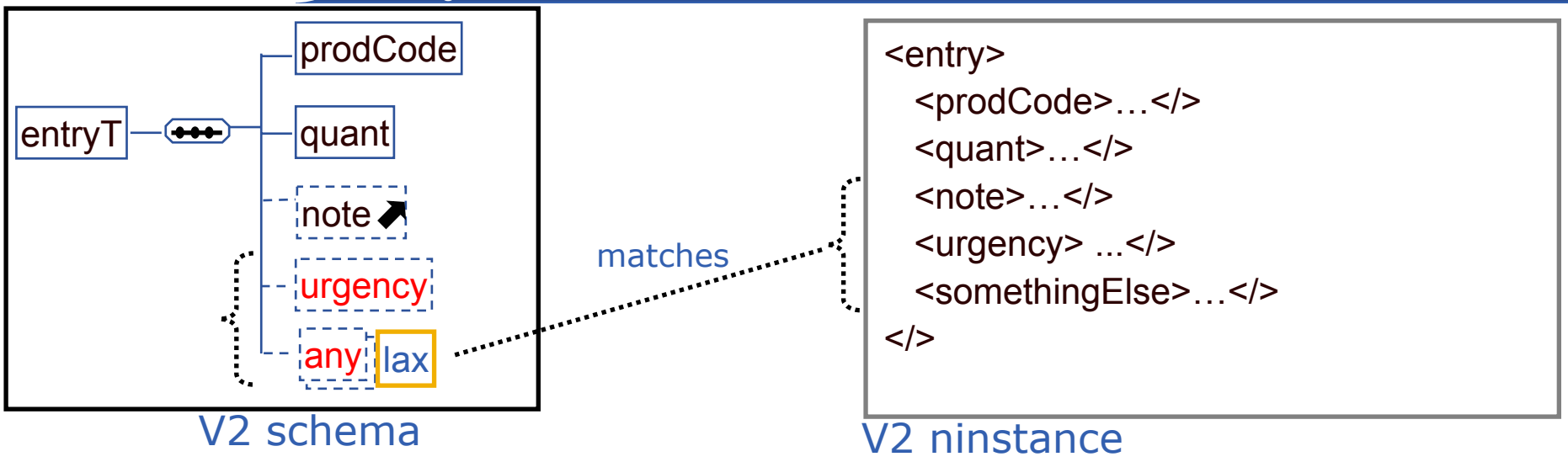
- **The loose-coupling principles of web services means that a schema should allow for change which is**
  - Forward compatible – newer versions of documents can be used by old S/W: new producer, old consumer
  - Backward Compatible – older versions of documents can be used by newer S/W : old producer, 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 compatability 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



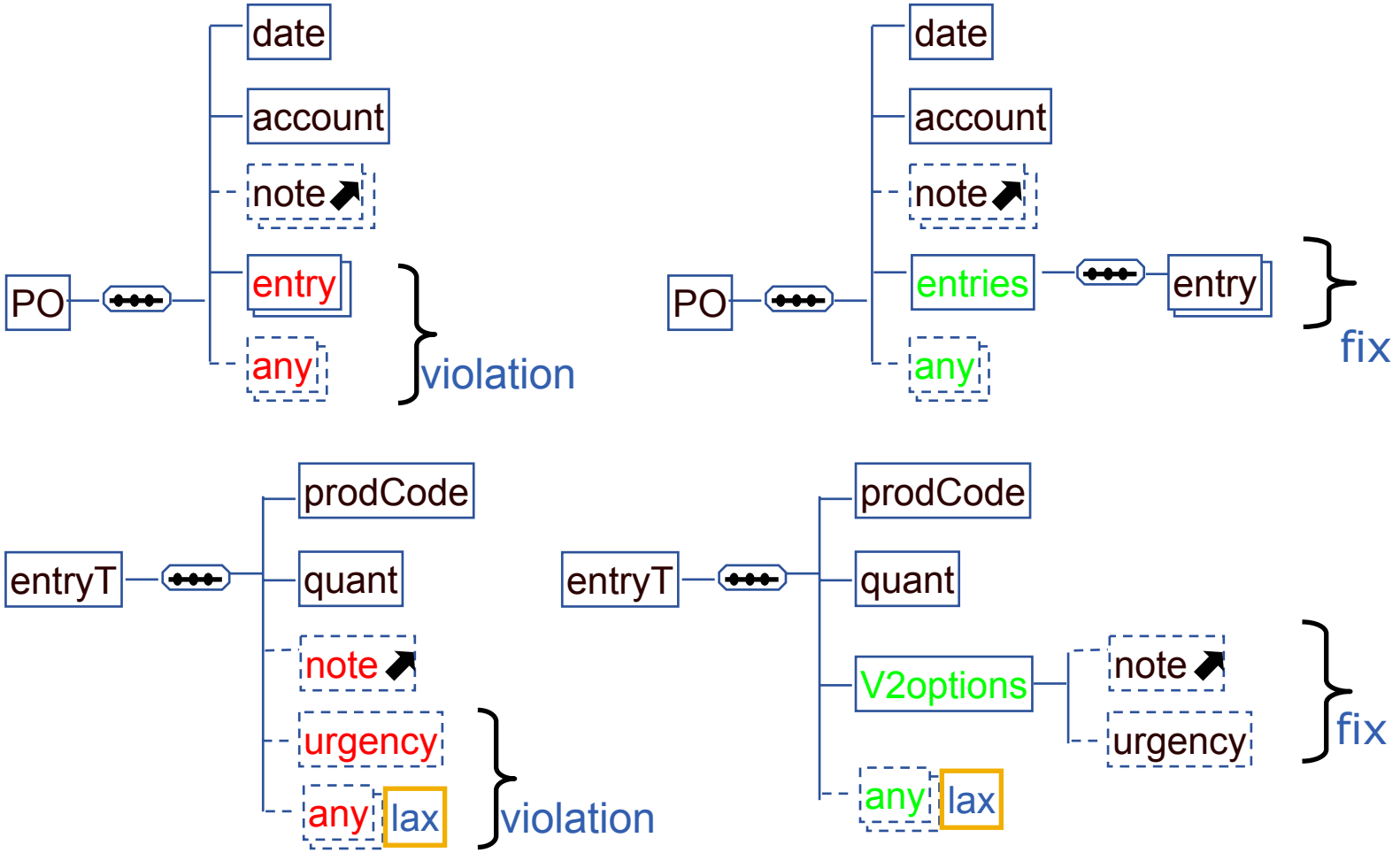
- **lax – gives forward compatiblity**
  - V1 consumer (coded using V1 schema)
  - can process document produced by V2 producer
- **Optionality on new item gives backward compatibility**
  - V2 consumer
  - can process document produced by V1 producer
- **If compatibility is not the reality –**
  - use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

Determinism Requirement



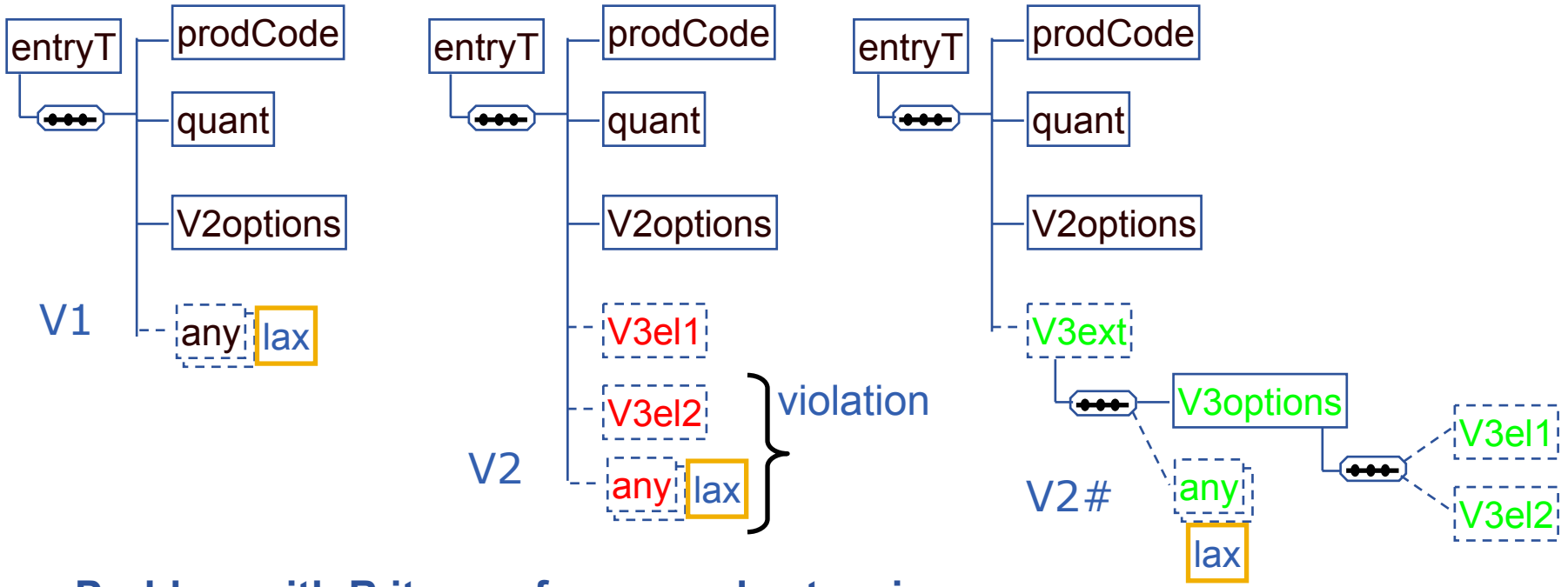
- **When "parsing" the instance, The note in instance could correspond to**
  - The note in schema
  - The any in schema
- **The Schema standard prohibits this non-determinism**
  - Can't have an Any within Choice or All
  - Can't have an Any before or after a variable occurrence component.
- **If disjoint 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 B 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 shema has to be new namespace – V1 processor can't deal with V2 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 repitition

Further Aspects

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

PRACTICAL

- **Goals**
  - To be able to construct and read an XML Schema
  - To be able to use the XMLspy tool for that
- **Outline**
  - General Structure
  - Simple Types
  - Miscellany
  - Extensibility
  - Concluding Remarks
  - Practical

Practical

- **Use XMLSPY to construct a schema for an invoice/statement document**

- **Similar to a PO document, http://homepages.nesc.ac.uk/~gcw/WSRF/**
  - Entry has
    - *Unit price*
    - *Cost*
    - *Optional VAT rate and amount*
    - *PO number*
  - Additionally a list of POs covered by the Invoice, each having the following information taken from the PO
    - *PO date*
    - *PO notes*
    - *A PO number (allocated by us)*
  - Includes Extension points – do on text representation
  - Construct an XML document with that as its schema

The End

THE END