XML Schema

Motivation

- We have designed our Recipe Markup Language
- ...but so far only **informally** described its **syntax**

- *How can we make tools that check that an XML document is a* ***syntactically correct*** *Recipe Markup Language document (and thus meaningful)?*

- Implementing a specialized validation tool for Recipe Markup Language is *not* the solution...

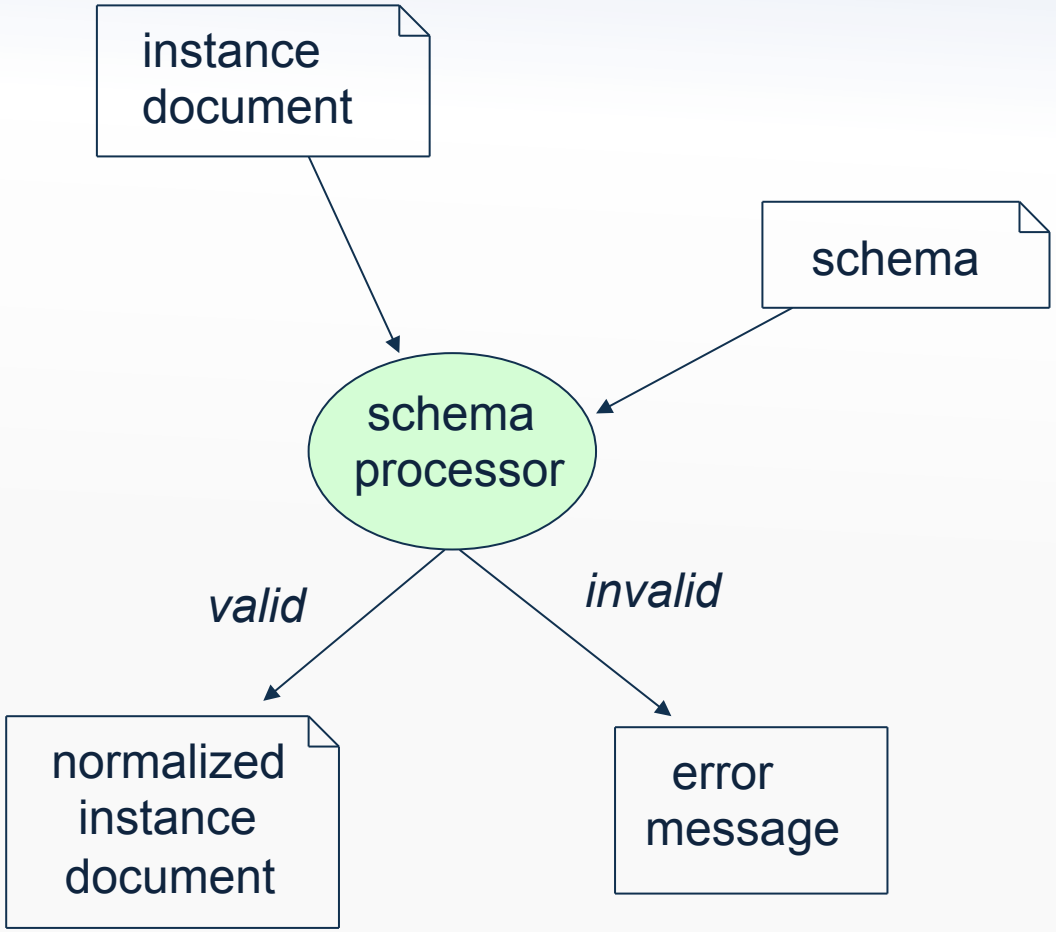
XML Languages

- ***XML language***:
  a set of XML documents with some semantics

- ***schema***:
  a formal but human readable definition of the syntax of an XML language

- ***schema language***:
  a notation for writing schemas

Validation


instance document
schema
schema processor
valid
invalid
normalized instance document
error message

Overview

- **Regular expressions**
- Simple types
  - Strings (attributes, ...)
  - Type derivations
- Complex types
  - Strings (element content)
  - Elements (element content)
- Limitations and quirks

Regular Expressions

- Consists of an ***alphabet*** Σ (a set) of symbols, e.g,
  - Unicode characters (a,b,c.. ,æ,ø,å,@,...Ξ, ”A,...)
  - Elements in html `(table, td, tr, html, …)`
- And a standard set of ***expressions*** written in a special notation using the symbols of Σ
  - Expressions ***match*** strings written with the alphabet
- For example,
  - a | b is a regular expression that matches “a” or “b”
  - (ab)+ matches ”ab” and ”abab”, and ”ababab”, and …

Regular Expressions

- Commonly used in schema languages to describe **sequences** of **characters** or **elements**
- Σ: an alphabet (typically Unicode characters or element names)
- $\sigma \in \Sigma$ matches the string σ
- α? matches zero or one α
- α* matches zero or more α's
- α+ matches one or more α's
- $\alpha\ \beta$ matches any concatenation of an α and a β
- $\alpha \mid \beta$ matches the union of α and β

Examples

- A regular expression describing **integers**:

```
0|-?(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*
```

- A regular expression describing the valid contents of `table` elements in XHTML:

```
caption? ( col* | colgroup* ) thead? tfoot? ( tbody+ | tr+ )
```

Clicker question

- Which string is matched by this expression: a*bb*a(a|b)
  - ababab
  - aabbaa ✓
  - aaab
  - bbb

Overview

- Regular expressions
- **Simple types**
  - **Strings (attributes, ...)**
  - **Type derivations**
- **Complex types**
  - **Strings (element content)**
  - **Elements (element content)**
- Limitations and quirks

Types and Declarations

- **Simple type definition**:
  defines a family of Unicode text strings

- **Complex type definition**:
  defines a content and attribute model

- **Element declaration**:
  associates an element name with a simple or complex type

- **Attribute declaration**:
  associates an attribute name with a simple type

Overview

- Regular expressions
- **Simple types**
  - **Strings (attributes, ...)**
  - **Type derivations**
- Complex types
  - Strings (element content)
  - Elements (element content)
- Limitations and quirks

Example (1/3)

Instance document:

```
<b:card xmlns:b="http://businesscard.org">
  <b:name>John Doe</b:name>
  <b:title>CEO, Widget Inc.</b:title>
  <b:email>john.doe@widget.com</b:email>
  <b:phone>(202) 555-1414</b:phone>
  <b:logo b:uri="widget.gif"/>
</b:card>
```

Example (2/3)

Schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org">

  <element name="card" type="b:card_type"/>
  <element name="name" type="string"/>
  <element name="title" type="string"/>
  <element name="email" type="string"/>
  <element name="phone" type="string"/>
  <element name="logo" type="b:logo_type"/>
  <attribute name="uri" type="anyURI"/>
```

Example (3/3)

```
  <complexType name="card_type">
    <sequence>
      <element ref="b:name"/>
      <element ref="b:title"/>
      <element ref="b:email"/>
      <element ref="b:phone" minOccurs="0"/>
      <element ref="b:logo" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="logo_type">
    <attribute ref="b:uri" use="required"/>
  </complexType>
</schema>
```

Connecting Schemas and Instances

```
<b:card xmlns:b="http://businesscard.org“
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://businesscard.org
                      business_card.xsd">
  <b:name>John Doe</b:name>
  <b:title>CEO, Widget Inc.</b:title>
  <b:email>john.doe@widget.com</b:email>
  <b:phone>(202) 555-1414</b:phone>
  <b:logo b:uri="widget.gif"/>
</b:card>
```

Namespaces

- `<schema targetNamespace="..." ...>`
- **Prefixes** are also used in certain **attribute *values***!
- ***Unqualified Locals***:
  - the default behavior is confusing
  - always change the default behavior using `elementFormDefault="qualified"`

Element and Attribute Declarations

Examples:

- <element name="serialnumber"
  type="nonNegativeInteger"/>

- <attribute name=”alcohol"
  type=”r:percentage"/>

Simple Types (Datatypes) – Primitive

string	*any Unicode string*
boolean	true, false, 1, 0
decimal	3.1415
float	6.02214199E23
double	42E970
dateTime	2004-09-26T16:29:00-05:00
time	16:29:00-05:00
date	2004-09-26
hexBinary	48656c6c6f0a
base64Binary	SGVsbG8K
anyURI	http://www.brics.dk/ixwt/
QName	rcp:recipe, recipe
…	

Derivation of Simple Types – Restriction

Constraining facets:

- `length`
- `minLength`
- `maxLength`
- `pattern`
- `enumeration`
- `whiteSpace`
- `maxInclusive`
- `maxExclusive`
- `minInclusive`
- `minExclusive`
- `totalDigits`
- `fractionDigits`

Examples

```
<simpleType name="score_from_0_to_100">
  <restriction base="integer">
    <minInclusive value="0"/>
    <maxInclusive value="100"/>
  </restriction>
</simpleType>

<simpleType name="percentage">
  <restriction base="string">
    <pattern value="([0-9]|[1-9][0-9]|100)%"/>
  </restriction>
</simpleType>
```

regular expression

Simple Type Derivation – List

```
<simpleType name="integerList">
  <list itemType="integer"/>
</simpleType>
```

matches whitespace separated lists of `integers`

Simple Type Derivation – Union

```
<simpleType name="boolean_or_decimal">
  <union>
    <simpleType>
      <restriction base="boolean"/>
    </simpleType>
    <simpleType>
      <restriction base="decimal"/>
    </simpleType>
  </union>
</simpleType>
```

Built-In Derived Simple Types

- normalizedString
- token
- language
- Name
- NCName
- ID
- IDREF
- integer

- nonNegativeInteger
- unsignedLong
- long
- int
- short
- byte
- ...

Overview

- Regular expressions
- Simple types
  - Strings (attributes, ...)
  - Type derivations
- **Complex types**
  - **Strings (element content)**
  - **Elements (element content)**
- Limitations and quirks

Complex Types with Complex Contents

- Content models as **regular expressions**:
  - Element reference `<element ref="name"/>`
  - Concatenation `<sequence> … </sequence>`
  - Union `<choice> … </choice>`
  - All `<all> … </all>`
  - Element wildcard: `<any namespace="…" processContents="…"/>`
- Attribute reference: `<attribute ref="…"/>`
- Attribute wildcard: `<anyAttribute namespace="…" processContents="…"/>`

Cardinalities: `minOccurs`, `maxOccurs`, `use`

Mixed content: `mixed="true"`

Example

```
<element name="order" type="n:order_type"/>

<complexType name="order_type" mixed="true">
  <choice>
    <element ref="n:address"/>
    <sequence>
      <element ref="n:email"
               minOccurs="0" maxOccurs="unbounded"/>
      <element ref="n:phone"/>
    </sequence>
  </choice>
  <attribute ref=”n:id" use="required"/>
</complexType>
```

Complex Types with Simple Content

```
<complexType name="category">
  <simpleContent>
    <extension base="integer">
      <attribute ref=”r:class”/>
    </extension>
  </simpleContent>
</complexType>
```

```
<complexType name="extended_category">
  <simpleContent>
    <extension base="n:category">
      <attribute ref=”r:kind"/>
    </extension>
  </simpleContent>
</complexType>
```

```
<complexType name="restricted_category">
  <simpleContent>
    <restriction base="n:category">
      <totalDigits value="3"/>
      <attribute ref=“r:class" use="required"/>
    </restriction>
  </simpleContent>
</complexType>
```

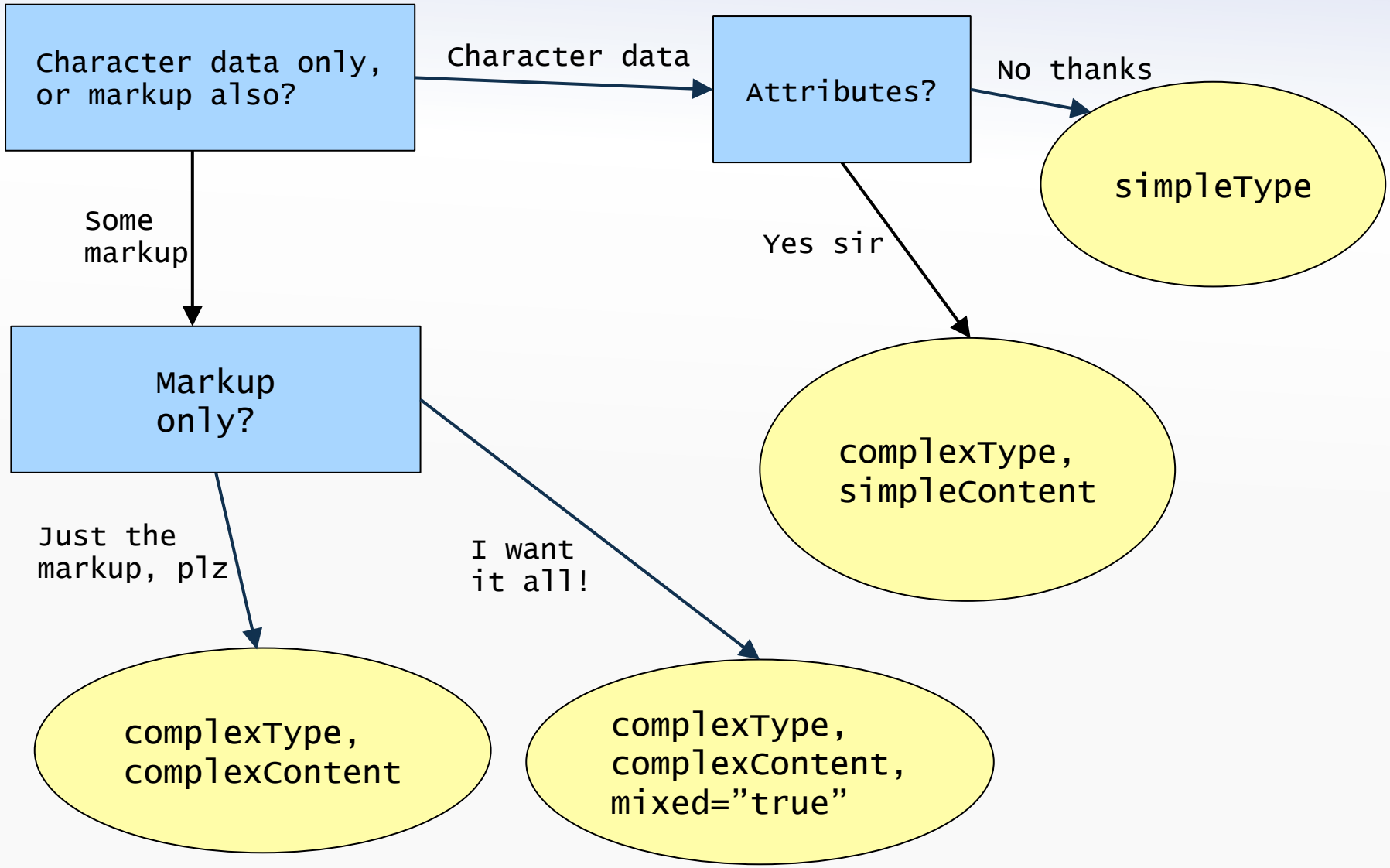
Clicker question

```
<element name="person">
   <complexType>
      <choice>
         <element name="job"><complexType/></element>
      </choice>
   </complexType>
</element>
```

Which XML fragment is valid?

1. `<person><job/></person>`
2. `<person><job>clown</job></person>`
3. `<person></person>`

How to choose?



Overview

- Regular expressions
- Simple types
  - Strings (attributes, ...)
  - Type derivations
- Complex types
  - Strings (element content)
  - Elements (element content)
- **Limitations and quirks**

Global vs. Local Descriptions

Global (toplevel) style:

```
<element name="card“
         type="b:card_type"/>
<element name="name“
         type="string"/>

<complexType name="card_type">
  <sequence>
    <element ref="b:name"/>
    ...
  </sequence>
</complexType>
```

Local (inlined) style:

```
<element name="card">
  <complexType>
    <sequence>
      <element name="name"
               type="string"/>
      ...
    </sequence>
  </complexType>
</element>
```

inlined

Global vs. Local Descriptions

- Local type definitions are **anonymous**
- Local element/attribute declarations can be **overloaded**
  – a simple form of *context sensitivity*
  (particularly useful for attributes!)
- Only globally declared elements can be starting points for validation (e.g. **roots**)

Requirements to Complex Types

- **Two element declarations** that have the **same name** and appear **in the same complex type** must have **identical types**

```
<complexType name="some_type">
  <choice>
    <element name="foo" type="string"/>
    <element name="foo" type="integer"/>
  </choice>
</complexType>
```

  - This requirement makes efficient implementation easier

- `all` can only contain `element` (e.g. not `sequence`!)
  - so we cannot use `all` to solve the problem with `comment` in RecipeML
- ...

Clicker question

- What is a simple type in XML Schema?
  - A description of a set of Unicode strings. ✓
  - A predefined type built into the XML Schema specification.
  - An association between an attribute name and its possible values.
  - A type that cannot act as base for derivations.

Summary

- Regular expressions
- Simple types
  - Strings (attributes, ...)
  - Type derivations
- Complex types
  - Strings (element content)
  - Elements (element content)
- Limitations and quirks

Essential Online Resources

- http://www.w3.org/TR/xmlschema-0/