BASI DI DATI II − 2 modulo
COMPLEMENTI DI BASI DI DATI
Parte III: Schemi per XML

Prof. Riccardo Torlone

Università Roma Tre

Outline

- The **purpose** of using schemas

- The schema languages **DTD** and **XML Schema**

- **Regular expressions** – a commonly used formalism in schema languages
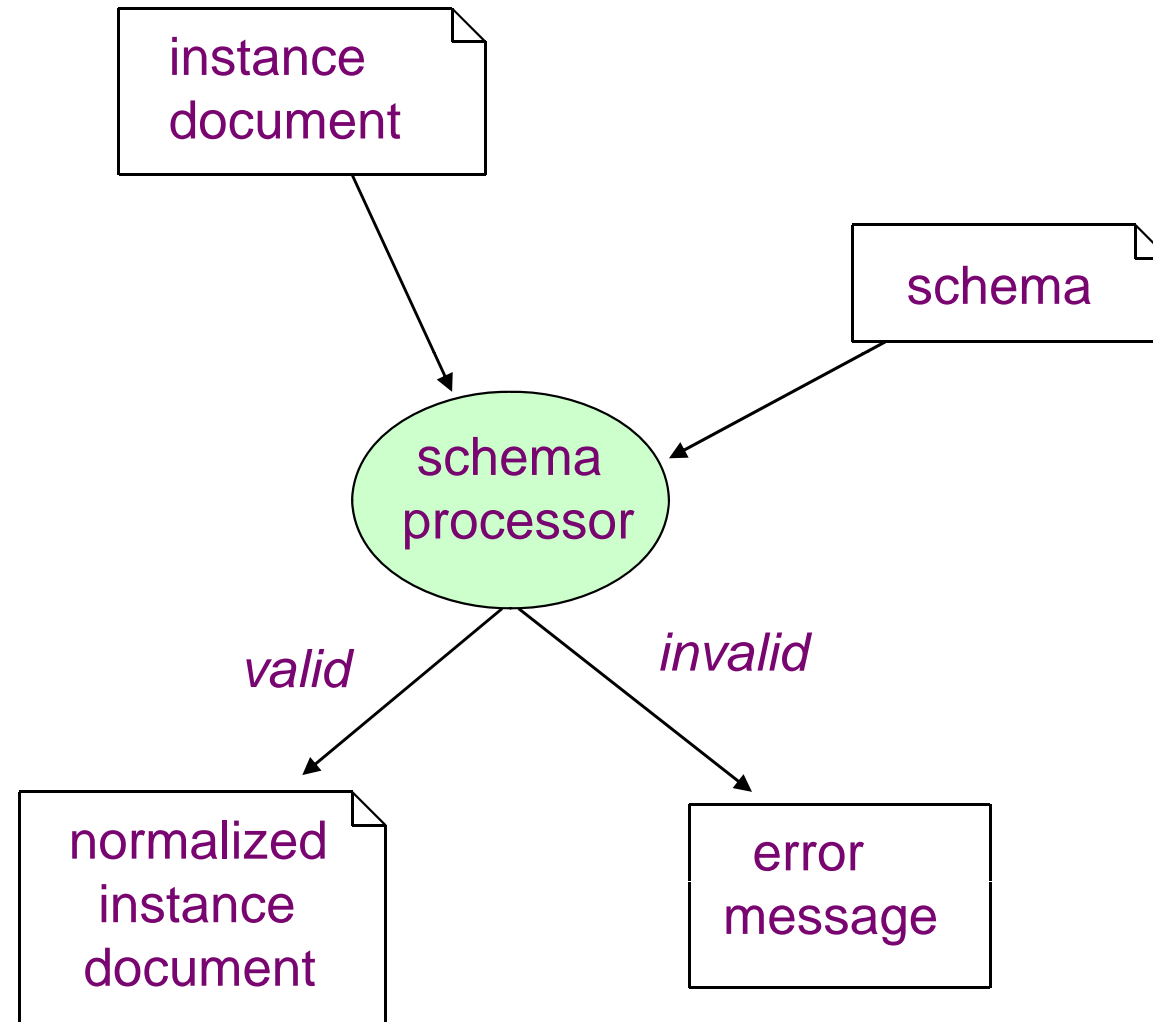
Motivation

- 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 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

Why use Schemas?

- Formal but human-readable descriptions

- Data validation can be performed with existing schema processors

General Requirements

- Expressiveness

- Comprehensibility

- Efficiency

Regular Expressions

- Commonly used in schema languages to describe sequences of characters or elements

- Σ: an alphabet (typically Unicode characters or element names)

- A regular expressions (RE) over Σ:

  - Each atom in Σ is a regular expression

  - If α and β are regular expressions then:

    - $\alpha?$, α^*, $\alpha+$, $\alpha \beta$, $\alpha \mid \beta$, and (α)

    are also regular expressions

Matching

- Matching between a string of atoms in Σ and a regular expression over Σ
  - $\sigma \in \Sigma$ matches the string σ
  - $\alpha?$ matches zero or one α
  - $\alpha*$ matches zero or more α's
  - $\alpha+$ 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
  - (α) matches α

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)+

DTD – Document Type Definition

- Defined as a subset of the DTD formalism from SGML

- Specified as an integral part of XML 1.0

- A starting point for development of more expressive schema languages

- Considers elements, attributes, and character data – processing instructions and comments are mostly ignored

Document Type Declarations

- Associates a DTD schema with the instance document

```
<?xml version="1.1"?>
<!DOCTYPE collection
    SYSTEM "http://www.uniroma3.it/recipes.dtd">
    <collection>
    ...
    </collection>

<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!DOCTYPE collection [ ... ]>
```

Element Declarations

`<!ELEMENT element-name content-model>`

- Content models:
  - EMPTY
  - ANY
  - Mixed content:
    - `(#PCDATA|e1|e2|...|en)*`
  - Element content:
    - regular expression over element names (concatenation is written with ",")

- Example:
`<!ELEMENT table`
`(caption?,(col|colgroup)*,thead?,`
`tfoot?,(tbody|tr)+)>`

A valid document

```
<table>
    <caption><em>Great cities of
            the world</em></caption>
    <thead>
        <tr><td>City</td></tr>
    </thead>
    <tbody>
        <tr><td>Copenhagen</td><td>Denmark</td></tr>
        <tr><td>San Francisco</td><td>USA</td></tr>
        <tr><td>Rome</td><td>Italy</td></tr>
    </tbody>
</table>
```

An invalid document

```
<table>
   <thead>
       <tr><td>City</td></tr>
    </thead>
    <tBody>
       <tr><td>Copenhagen</td><td>Denmark</td></tr>
       <tr><td>San Francisco</td><td>USA</td></tr>
       <tr><td>Rome</td><td>Italy</td></tr>
    </tBody>
     <caption><em>Great cities of
              the world</em></caption>
</table>
```

Attribute-List Declarations

```
<!ATTLIST element-name attribute-definitions>
```

Each attribute definition consists of

- an attribute name
- an attribute type
- a default declaration

Example:

```
<!ATTLIST input
        maxlength CDATA #IMPLIED
        tabindex CDATA #IMPLIED>
```

Attribute Types

- CDATA: any value
- Enumeration: $(s_1|s_2|...|s_n)$
- ID: must have unique value
- IDREF (IDREFS): must match some ID attribute(s)
- ...

Examples:
```
<!ATTLIST p
     align (left|center|right|justify)
     #IMPLIED>

<!ATTLIST recipe id ID #IMPLIED>

<!ATTLIST related ref IDREF #IMPLIED>
```

Attribute Default Declarations

- `#REQUIRED`
- `#IMPLIED` (optional)
- "value" (optional, but default provided)
- `#FIXED` "value" (if present must have this value)

- Examples:
```
<!ATTLIST form
    action CDATA #REQUIRED
    onsubmit CDATA #IMPLIED
    method (get|post) "get"
    enctype CDATA
            "application/x-www-form-urlencoded">

<!ATTLIST html
    xmlns CDATA #FIXED "http://www.w3.org/1999/xhtml">
```

Entity Declarations (1/3)

- Internal entity declarations – a simple macro mechanism

- Example:

  - Schema:
    ```
    <!ENTITY copyrightnotice
            "Copyright &#169; 2005 Widgets'R'Us.">
    ```

  - Input:
    ```
    A gadget has a medium size head and a big gizmo
    subwidget. &copyrightnotice;
    ```

  - Output:
    ```
    A gadget has a medium size head and a big gizmo
    subwidget. Copyright &#169; 2005 Widgets'R'Us.
    ```

Entity Declarations (2/3)

- Internal parameter entity declarations – apply to the DTD, not the instance document

Example:

```
<!ENTITY % Shape "(rect|circle|poly|default)">

<!ATTLIST area shape %Shape; "rect">
```

corresponds to:

```
<!ATTLIST area shape (rect|circle|poly|default) "rect">
```

Entity Declarations (3/3)

- External parsed entity declarations – references to XML data in other files

    Example:

    ```
    <!ENTITY widgets
        SYSTEM "http://www.uniroma3.it/widgets.xml">
    ```

- External unparsed entity declarations – references to non-XML data

    Example:

    ```
    <!ENTITY widget-image
        SYSTEM "http://www.uniroma3.it/widget.gif"
        NDATA gif >
    <!NOTATION gif
        SYSTEM "http://www.iana.org/assignments/
                        media-types/image/gif">
    ```

Conditional Sections

■ Allow parts of schemas to be enabled/disabled by a switch

Example:

☐ External DTD:

```
<![%person.simple; [
  <!ELEMENT person (firstname,lastname)>
]]>
<![%person.full; [
  <!ELEMENT person (firstname,lastname,email+,phone?)>
  <!ELEMENT email (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]]>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

☐ Internal DTD:

```
<!ENTITY % person.simple "INCLUDE" >
<!ENTITY % person.full "IGNORE" >
```

Checking Validity with DTD

- A DTD processor (also called a validating XML parser)

- parses the input document (includes checking well-formedness)

- checks the root element name

- for each element, checks its contents and attributes

- checks uniqueness and referential constraints (ID/IDREF(S) attributes)

RecipeML with DTD (1/2)

```
<!ELEMENT collection (description,recipe*)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT recipe
    (title,date,ingredient*,preparation,comment?,
     nutrition,related*)>
<!ATTLIST recipe id ID #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT ingredient (ingredient*,preparation)?>
<!ATTLIST ingredient name CDATA #REQUIRED
                     amount CDATA #IMPLIED
                     unit CDATA #IMPLIED>
```

RecipeML with DTD (2/2)

```
<!ELEMENT preparation (step*)>
<!ELEMENT step (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT nutrition EMPTY>
<!ATTLIST nutrition
                calories CDATA #REQUIRED
                carbohydrates CDATA #REQUIRED
                fat CDATA #REQUIRED
                protein CDATA #REQUIRED
                alcohol CDATA #IMPLIED>
<!ELEMENT related EMPTY>
<!ATTLIST related ref IDREF #REQUIRED>
```

Problems with the DTD description

- calories should contain a non-negative number
- protein should contain a value on the form N% where N is between 0 and 100;
- comment should be allowed to appear anywhere in the contents of recipe
- unit should only be allowed in an elements where amount is also present
- nested ingredient elements should only be allowed when amount is absent
- our DTD schema permits in some cases too much and in other cases too little!

Limitations of DTD

- Cannot constraint character data
- Specification of attribute values is too limited
- Element and attribute declarations are context insensitive
- Character data cannot be combined with the regular expression content model
- The content models lack an "interleaving" operator
- The support for modularity, reuse, and evolution is too primitive
- The normalization features lack content defaults and proper whitespace control
- Structured embedded self-documentation is not possible
- The ID/IDREF mechanism is too simple
- It does not itself use an XML syntax
- No support for namespaces

Requirements for XML Schema

- W3C's proposal for replacing DTD
- Design principles:
    - ☐ More expressive than DTD
    - ☐ Use XML notation
    - ☐ Self-describing
    - ☐ Simplicity
- Technical requirements:
    - ☐ Namespace support
    - ☐ User-defined datatypes
    - ☐ Inheritance (OO-like)
    - ☐ Evolution
    - ☐ Embedded documentation
    - ☐ ...

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

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:

```xml
<schema xmlns:xsd="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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bc:card xmlns:bc="http://businesscard.org"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://businesscard.org
                       business_card.xsd">
   <bc:name>John Doe</bc:name>
   <bc:title>CEO, Widget Inc.</bc:title>
   <bc:email>john.doe@widget.com</bc:email>
   <bc:phone>(202) 555-1414</bc:phone>
   <bc:logo bc:uri="widget.gif"/>
</bc:card>
```

Element and Attribute Declarations

Examples:

□ `<element`

       `name="serial number"`
       `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

☐ ...

Complex Types with Complex Contents

- Content models as regular expressions:
  - ☐ Element reference   `<element ref="..."/>`
  - ☐ Ordered sequence   `<sequence> ... </sequence>`
  - ☐ Union   `<choice> ... </choice>`
  - ☐ Unordered sequence   `<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"/>
<attribute name="id" type="unsignedInt"/>

<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>
```

Derivation with Simple Content

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

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

Derivation with Complex Content

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

```
<complexType name="extended_type">          <complexType name="further_derived">
 <complexContent>                            <complexContent>
  <extension base=                            <restriction base=
           "b:basic_card_type">                          "b:extended_type">
   <sequence>                                  <sequence>
    <element ref="b:title"/>                   <element ref="b:name"/>
    <element ref="b:email"                     <element ref="b:title"/>
             minOccurs="0"/>                   <element ref="b:email"/>
   </sequence>                                 </sequence>
  </extension>                                </restriction>
 </complexContent>                           </complexContent>
</complexType>                               </complexType>
```

Note: restriction is <u>not</u> the opposite of extension!

Global vs. Local Descriptions

Global (top-level) 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 (in-lined) style:

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

Global vs. Local Descriptions

- Local type definitions are **anonymous**
- Local element/attribute declarations can be **overloaded** − a simple form of *context sensitivity*
- Only globally declared elements can be starting points for validation (e.g. **roots**)
- Local definitions permit an alternative **namespace** semantics (explained later...)

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

- ...

Namespaces

- `<schema targetNamespace="..." ... >`

- Prefixes are also used in certain attribute values!

- Unqualified Locals:

  - □ if enabled, the name of a locally declared element or attribute in the instance document has no namespace prefix

  - □ such an attribute or element belongs to the element declared in the surrounding global definition

  - □ always change the default behaviour using `elementFormDefault="qualified"`

Uniqueness, Keys, References

```
<element name="w:widget" xmlns:w="http://www.widget.org">
   <complexType>

   ...
   </complexType>
   <key name="my_widget_key">
      <selector xpath="w:components/w:part"/>
      <field xpath="@manufacturer"/>
      <field xpath="w:info/@productid"/>
   </key>
   <keyref name="annotation_references" refer="w:my_widget_key">
      <selector xpath=".//w:annotation"/>
      <field xpath="@manu"/>
      <field xpath="@prod"/>
   </keyref>
</element>
```

in every widget, each part must have unique (manufacturer, productid)

in every widget, for each annotation,
(manu, prod) must match a my_widget_key

unique: as key, but fields may be absent

Other Features in XML Schema

- Groups
- Nil values
- Substitution groups
- Annotations
- Defaults and whitespace
- Modularization

RecipeML with XML Schema (1/5)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:r="http://www.brics.dk/ixwt/recipes"
        targetNamespace="http://www.brics.dk/ixwt/recipes"
        elementFormDefault="qualified">
  <element name="collection">
    <complexType>
      <sequence>
        <element name="description" type="string"/>
        <element ref="r:recipe" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
    <unique name="recipe-id-uniqueness">
      <selector xpath=".//r:recipe"/>
      <field xpath="@id"/>
    </unique>
    <keyref name="recipe-references" refer="r:recipe-id-uniqueness">
      <selector xpath=".//r:related"/>
      <field xpath="@ref"/>
    </keyref>
  </element>
```

RecipeML with XML Schema (2/5)

```xml
<element name="recipe">
  <complexType>
    <sequence>
      <element name="title" type="string"/>
      <element name="date" type="string"/>
      <element ref="r:ingredient" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="r:preparation"/>
      <element name="comment" type="string" minOccurs="0"/>
      <element ref="r:nutrition"/>
      <element ref="r:related" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="NMTOKEN"/>
  </complexType>
</element>
```

RecipeML with XML Schema (3/5)

```xml
<element name="ingredient">
  <complexType>
    <sequence minOccurs="0">
      <element ref="r:ingredient" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="r:preparation"/>
    </sequence>
    <attribute name="name" use="required"/>
    <attribute name="amount" use="optional">
      <simpleType>
        <union>
          <simpleType>
            <restriction base="r:nonNegativeDecimal"/>
          </simpleType>
          <simpleType>
            <restriction base="string"> <enumeration value="*"/>
            </restriction>
          </simpleType>
        </union>
      </simpleType>
    </attribute>
    <attribute name="unit" use="optional"/>
  </complexType>
</element>
```

RecipeML with XML Schema (4/5)

```xml
<element name="preparation">
  <complexType>
    <sequence>
      <element name="step" type="string"
               minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
<element name="nutrition">
  <complexType>
      <attribute name="calories" type="r:nonNegativeDecimal"
                                  use="required"/>
      <attribute name="protein" type="r:percentage" use="required"/>
      <attribute name="carbohydrates" type="r:percentage"
                                       use="required"/>
      <attribute name="fat" type="r:percentage" use="required"/>
      <attribute name="alcohol" type="r:percentage" use="optional"/>
  </complexType>
</element>
<element name="related">
  <complexType>
    <attribute name="ref" type="NMTOKEN" use="required"/>
  </complexType>
</element>
```

RecipeML with XML Schema (5/5)

```xml
<simpleType name="nonNegativeDecimal">
  <restriction base="decimal">
    <minInclusive value="0"/>
  </restriction>
</simpleType>

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

</schema>
```

Problems with the XML Schema description

- `calories` should contain a non-negative number
- `protein` should ~~contain a~~ value on the form *N%* where *N* is between 0 and 100;

  solved

- *comment should be allowed to appear anywhere in the contents of `recipe`*
- *`unit` should only be allowed in an elements where `amount` is also present*
- *nested `ingredient` elements should only be allowed when `amount` is absent*

– *even XML Schema has insufficient expressiveness!*

Limitations of XML Schema

- The details are extremely complicated (and the spec is unreadable)
- Declarations are (mostly) context insensitive
- It is impossible to write an XML Schema description of XML Schema
- With mixed content, character data cannot be constrained
- Cannot require specific root element
- The type system is overly complicated
- xsi:type is problematic
- Simple type definitions are inflexible

Strengths of XML Schema

- Namespace support

- Data types (built-in and derivation)

- Modularization

- Type derivation mechanism

Summary

- **schema**: formal description of the syntax of an XML language

- **DTD**: simple schema language

  ☐ elements, attributes, entities, ...

- **XML Schema**: more advanced schema language

  ☐ element/attribute declarations

  ☐ simple types, complex types, type derivations

  ☐ global vs. local descriptions

  ☐ ...

Essential Online Resources

- http://www.w3.org/TR/xml11/
- http://www.w3.org/TR/xmlschema-1/
- http://www.w3.org/TR/xmlschema-2/