

Schema Languages

Anders Møller & Michael I. Schwartzbach
© 2006 Addison-Wesley

Objectives

- The **purpose** of using schemas
- The schema languages **DTD** and **XML Schema** (and **DSD2** and **RELAX NG**)
- **Regular expressions** – a commonly used formalism in schema languages

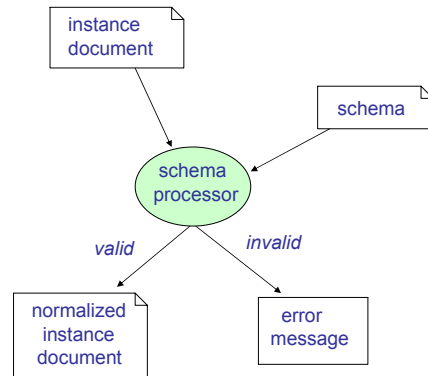
Motivation

- 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



Why use Schemas?

- Formal but human-readable descriptions
- Data validation can be performed with existing schema processors

General Requirements

- Expressiveness
- Efficiency
- Comprehensibility

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
- $\alpha?$ 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+ )
```

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.brics.dk/ixwt/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**:  $(\#PCDATA | e_1 | e_2 | \dots | e_n)^*$
- **element content**: regular expression over element names (concatenation is written with “,”)

Example:

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

## 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>
```

An Introduction to XML and Web Technologies

13

## Attribute Types

- CDATA: any value
- *enumeration*: ( $s_1 | s_2 | \dots | s_n$ )
- 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>
```

An Introduction to XML and Web Technologies

14

## Attribute Default Declarations

- #REQUIRED
- #IMPLIED (= optional)
- "value" (= optional, but default provided)
- #FIXED "value" (= required, 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">
```

An Introduction to XML and Web Technologies

15

## Entity Declarations (1/3)

- **Internal** entity declarations – a simple macro mechanism

Example:

- Schema:

```
<!ENTITY copyrightnotice "Copyright © 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.**

An Introduction to XML and Web Technologies

16

## Entity Declarations (2/3)

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

Example:

- Schema:  
`<!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.brics.dk/ixwt/widgets.xml">`

not widely used!

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

Example:

- `<!ENTITY widget-image  
SYSTEM "http://www.brics.dk/ixwt/widget.gif"  
NDATA gif >`
- `<!NOTATION gif  
SYSTEM "http://www.iana.org/assignments/media-types/image/gif">`
- `<!ATTLIST thing img ENTITY #REQUIRED>`

## Conditional Sections

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

Example:

- `<![%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)>`
- `<!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

1. Cannot constraint **character data**
2. Specification of **attribute values** is too limited
3. Element and attribute declarations are **context insensitive**
4. **Character data** cannot be combined with the **regular expression** content model
5. The content models lack an “**interleaving**” operator
6. The support for **modularity**, **reuse**, and **evolution** is too primitive
7. The normalization features lack **content defaults** and proper **whitespace** control
8. **Structured embedded self-documentation** is not possible
9. The **ID/IDREF** mechanism is too simple
10. It does not itself use an **XML syntax**
11. 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:

```
<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>
```

### Element and Attribute Declarations

Examples:

- `<element name="serialnumber" type="nonNegativeInteger"/>`
- `<attribute name="alcohol" type="r:percentage"/>`

### Simple Types (Datatypes) – Primitive

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

## Derivation of Simple Types – Restriction

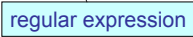
Constraining facets:

- 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>
```



## 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="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>
```

## Derivation with Complex Content

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

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

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

**Note:** restriction is not the opposite of extension!

## 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**)
- 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 must have **no namespace prefix** (i.e. **the empty namespace URI**)
  - such an attribute or element “**belongs to**” the **element declared in the surrounding global definition**
  - always change the default behavior using `elementFormDefault="qualified"`

An Introduction to XML and Web Technologies

45

## Derived Types and Subsumption

- Assume that
  - $T$  is some type
  - $T_-$  is derived from  $T$  by restriction
  - $T_+$  is derived from  $T$  by extension
- **Subsumption:** Whenever a  $T$  instance is required,
  - a  $T_-$  instance may be used instead (trivial)
  - a  $T_+$  instance may be used instead – if the instance has `xsi:type="T_+"`  
(with `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`)
- Derivation, instantiation, and subsumption can be constrained using `final`, `abstract`, and `block`

An Introduction to XML and Web Technologies

46

## Substitution Groups

- Assume  $D$  is (in some number of steps) derived from  $B$ ,  $E_D$  is an element declaration of type  $D$ , and  $E_B$  is an element declaration of type  $B$
- If  $E_D$  is in **substitution group** of  $E_B$  then an  $E_D$  element may be used whenever an  $E_B$  is required
- (This is subsumption based on element declarations, not on types)

An Introduction to XML and Web Technologies

47

## 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="."/>
 <field xpath="@manu"/>
 <field xpath="@prod"/>
 </keyref>
</element>
```

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

only a “downward” subset of XPath is used

in every widget, for each annotation, (manu, prod) must match a my\_widget\_key

**unique:** as key, but fields may be absent

An Introduction to XML and Web Technologies

48

## Other Features in XML Schema

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

– *read the book chapter*

## 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)

```
<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)

```
<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)

```
<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)

```
<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

1. The details are extremely **complicated** (and the spec is unreadable)
2. Declarations are (mostly) **context insensitive**
3. It is impossible to write an **XML Schema description of XML Schema**
4. With **mixed content**, **character data** cannot be constrained
5. **Unqualified local elements** are bad practice
6. Cannot require specific **root element**
7. **Element defaults** cannot contain markup
8. The **type** system is overly complicated
9. **xsi:type** is problematic
10. **Simple type definitions** are inflexible

## Strengths of XML Schema

---

- Namespace support
- Data types (built-in and derivation)
- Modularization
- Type derivation mechanism

## Document Structure Description 2.0

---

– *read the book chapter*

## RELAX NG

---

- OASIS + ISO competitor to XML Schema
- Validation only (no normalization)
- Designed for simplicity and expressiveness, solid mathematical foundation

## Processing Model

---

- For a valid instance document, the **root** element must match a designated **pattern**
- A **pattern** may match **elements**, **attributes**, or **character data**
- Element patterns can contain **sub-patterns**, that describe contents and attributes

## Patterns – Regular Hedge Expressions

- `<element name="..."> ... </element>`
- `<attribute name="..."> ... </attribute>`
- `<text/>`
- `<group> ... </group>` (concatenation)
- `<optional> ... </optional>`
- `<zeroOrMore> ... </zeroOrMore>`
- `<oneOrMore> ... </oneOrMore>`
- `<choice> ... </choice>` (union)
- `<empty/>`
- `<interleave> ... </interleave>`
- `<mixed> ... </mixed>`

## Example

```
<element name="card">
 <element name="name"><text/></element>
 <element name="title"><text/></element>
 <element name="email"><text/></element>
 <optional>
 <element name="phone"><text/></element>
 </optional>
 <optional>
 <element name="logo">
 <attribute name="uri"><text/></attribute>
 </element>
 </optional>
</element>
```

## Grammars

- Pattern **definitions** and **references**  
allow description of *recursive* structures
- ```
<grammar ...>
  <start>
    ...
  </start>

  <define name="...">
    ...
  </define>
  ...
</grammar>
```

Other Features in RELAX NG

- Name classes
 - Datatypes (based on XML Schema's datatypes)
 - Modularization
 - An alternative compact, non-XML syntax
- *read the book chapter*

RecipeML with RELAX NG (1/5)

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  ns="http://www.brics.dk/ixwt/recipes"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <start>
    <element name="collection">
      <element name="description"><text/></element>
      <zeroOrMore><ref name="element-recipe"/></zeroOrMore>
    </element>
  </start>

  <define name="element-recipe">
    <element name="recipe">
      <optional><attribute name="id">
        <data datatypeLibrary="http://relaxng.org/..." type="ID"/>
      </attribute></optional>
    </element>
  </define>
</grammar>
```

RecipeML with RELAX NG (2/5)

```
<interleave>
  <group>
    <element name="title"><text/></element>
    <element name="date"><text/></element>
    <zeroOrMore><ref name="element-ingredient"/></zeroOrMore>
    <ref name="element-preparation"/>
    <element name="nutrition">
      <ref name="attributes-nutrition"/>
    </element>
    <zeroOrMore><ref name="element-related"/></zeroOrMore>
  </group>
  <optional><element name="comment"><text/></element></optional>
</interleave>
</element>
</define>
```

RecipeML with RELAX NG (3/5)

```
<define name="element-ingredient">
  <element name="ingredient">
    <attribute name="name"/>
    <choice>
      <group>
        <attribute name="amount">
          <choice><value>*</value><ref name="NUMBER"/></choice>
        </attribute>
        <optional><attribute name="unit"/></optional>
      </group>
      <group>
        <zeroOrMore><ref name="element-ingredient"/></zeroOrMore>
        <ref name="element-preparation"/>
      </group>
    </choice>
  </element>
</define>
```

RecipeML with RELAX NG (4/5)

```
<define name="element-preparation">
  <element name="preparation">
    <zeroOrMore><element name="step"><text/></element></zeroOrMore>
  </element>
</define>

<define name="attributes-nutrition">
  <attribute name="calories"><ref name="NUMBER"/></attribute>
  <attribute name="protein"><ref name="PERCENTAGE"/></attribute>
  <attribute name="carbohydrates"><ref name="PERCENTAGE"/></attribute>
  <attribute name="fat"><ref name="PERCENTAGE"/></attribute>
  <optional>
    <attribute name="alcohol"><ref name="PERCENTAGE"/></attribute>
  </optional>
</define>
```

RecipeML with RELAX NG (5/5)

```
<define name="element-related">
  <element name="related">
    <attribute name="ref">
      <data datatypeLibrary="http://relaxng.org/..." type="IDREF"/>
    </attribute>
  </element>
</define>

<define name="PERCENTAGE">
  <data type="string">
    <param name="pattern">([0-9]|[1-9][0-9]|100)%</param>
  </data>
</define>

<define name="NUMBER">
  <data type="decimal"><param name="minInclusive">0</param></data>
</define>

</grammar>
```

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/xml111/>
- <http://www.w3.org/TR/xmlschema-1/>
- <http://www.w3.org/TR/xmlschema-2/>