Linguaggi di Schema (DTD e XML-Schema)

Anders Møller, Michael Schwartzbach (adattamento di Fabio Grandi)

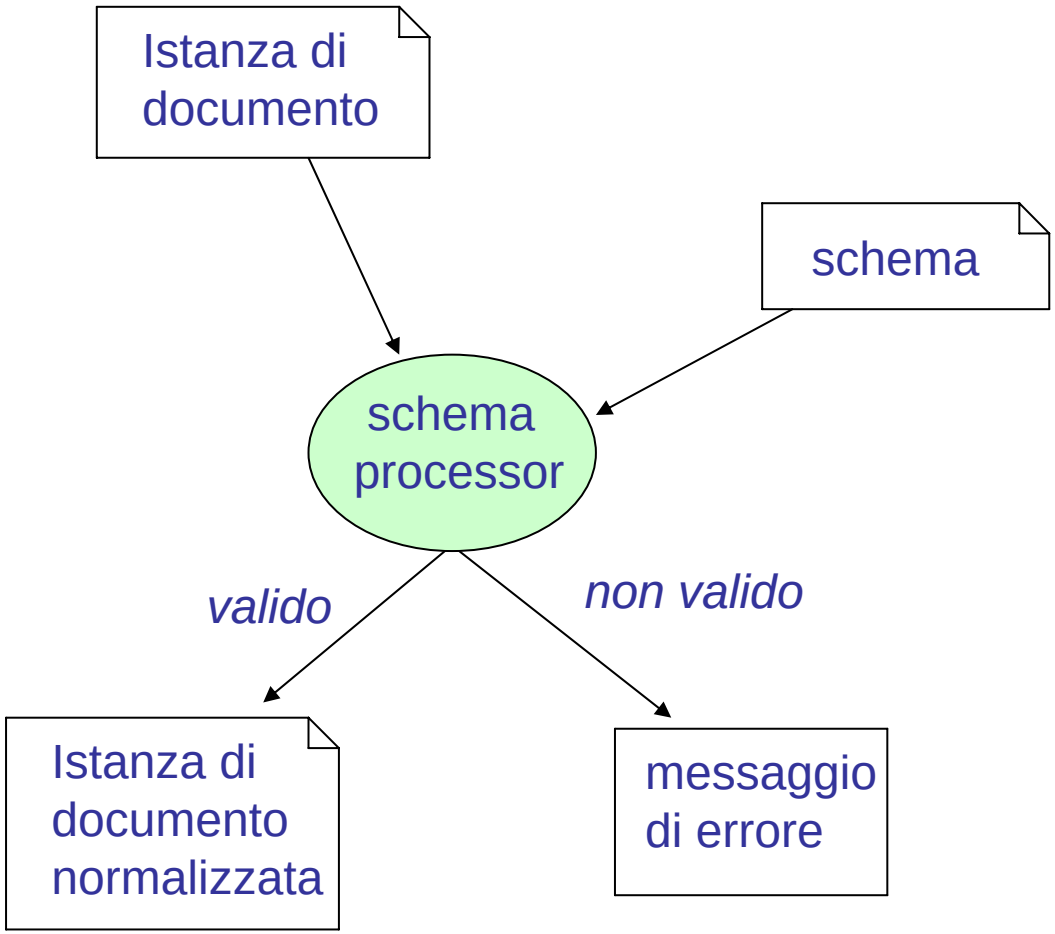
Linguaggi XML

- ***Linguaggio XML***:
  un insieme di documenti XML con una qualche semantica

- ***schema***:
  una definizione formale
  della sintassi di un linguaggio XML

- ***schema language***:
  un formalismo per scrivere schemi

Validazione



Perché usare gli schemi?

- Sono descrizioni formali ma leggibili dagli esseri umani
- La validazione dei dati può essere eseguita con gli schema processor esistenti

Requisiti Generali

- Espressività
- Efficienza
- Comprensibilità

Espressioni Regolari

- Comunemente usate nei linguaggi di schema per descrivere **sequenze** di **caratteri** o **elementi**

- Σ: un alfabeto (tipicamente caratteri Unicode, nomi di elementi)
- $\sigma \in \Sigma$ corrisponde alla stringa σ
- $\alpha?$ corrisponde a zero o una occorrenza di α
- $\alpha*$ corrisponde a zero o più occorrenze di α
- $\alpha+$ corrisponde a uno o più occorrenze di α
- $\alpha\ \beta$ corrisponde ad una concatenazione di un α e un β
- $\alpha \mid \beta$ corrisponde all'alternativa fra un α e un β

Esempi

- Un’espressione regolare che descrive numeri **interi**:

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

- Un’espressione regolare che descrive un contenuto valido of degli elementi **`table`** in XHTML:

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

DTD – Document Type Definition

- Definito per XML come sottoinsieme del formalismo DTD di SGML
- Specificato come parte integrante di XML 1.0
- Un punto di partenza per lo sviluppo di linguaggi di schema più espressivi
- Considera elementi, attributi e character data – processing instructions e commenti sono per lo più ignorati

Dischiarazioni di DTD nei documenti

- Associa una DTD ad una istanza di documento

- ```
  <?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 [ ... ]>
  ```
  ```

Dichiarazione di Elementi

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

Content-models:

- **EMPTY**
- **ANY**
- ***mixed content***: (#PCDATA|e_1|e_2|...|e_n)*
- ***element content***: espressione regolare sui nomi di element
  (concatenazione espressa usando "**,**")

Esempio:

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

Dichiarazione di Lista di Attributi

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

Ciascuna dichiarazione di attributo consiste di

- il nome dell'attributo
- il *tipo* dell'attributo
- una dichiarazione di *default*

Esempio:

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

Tipi di Attributo

- `CDATA`: un qualunque valore
- *enumerazione*: $(s_1|s_2|...|s_n)$
- `ID`: deve avere valore unico nel documento
- `IDREF` (/ `IDREFS`): deve corrispondere ad un qualche attributo `ID`
- ...

Esempi:

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

<!ATTLIST recipe id ID #IMPLIED>
<!ATTLIST related ref IDREF #IMPLIED>
```

Dichiarazione di Valori di Default

- #REQUIRED
- #IMPLIED (= opzionale)
- *"value"* (= opzionale, con valore di default assegnato)
- #FIXED *"value"* (= deve obbligatoriamente avere questo valore)

Esempi:

```
<!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">
```

Dichiarazione di ENTITY (1/3)

- ENTITY ***Interne*** – semplice definizione di macro

  Esempio:

  - 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.
    ```

Dichiarazione di ENTITY (2/3)

- Dichiarazioni ***Internal parameter*** – si applicano alla DTD, non alle istanze di documento

  Esempio:

  - Schema:

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

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

    corrisponde a:

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

Dichiarazione di ENTITY (3/3)

- Dichiarazioni ***External parsed*** –
  fanno riferimento a dati XML in altri files

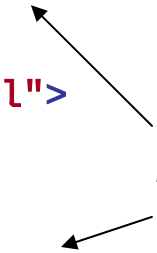
  Esempio:

  ```
  <!ENTITY widgets
    SYSTEM "http://www.brics.dk/ixwt/widgets.xml">
  ```

- Dichiarazioni ***External unparsed***–
  fanno riferimento a dati **non-**XML

  Esempio:

  ```
  <!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>
  ```



non molto usate!

Controllo di Validità tramite DTD

Un processore DTD, altrimenti detto un analizzatore (parser) XML validante:

- analizza il docuemento in input (controllando anche la well-formedness)
- controlla il nome dell'elemento radice
- per ogni element, controlla il suo contenuto e i suoi attributi
- controlla vincoli di unicità e referenziali (sui valori degli attributi `ID`/`IDREF`(S))

Esempio di DTD - RecipeML (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>
```

Esempio di DTD - RecipeML (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>
```

Requisiti per XML Schema

- Proposta del W3C per rimpiazzare le DTD

Principi progettuali:

- Più espressivo delle DTD
- Utilizzo di notazione XML
- Auto-esplicativo
- Semplice

Requisiti tecnici:

- Supporto di namespace
- Tipi di dato definibili dall'utente
- Ereditarietà (stile O-O)
- Evoluzione
- Documentazione integrata
- ...

Tipi e Dichiarazioni

- **Definizione di Simple type**:
  definisce una famglia di stringhe di testo Unicode
- **Definizione di Complex type**:
  definisce un modello di contenuto e attributi

- **Dichiarazione di Element**:
  associa un element name con un simple o complex type
- **Dichiarazione di Attribute**:
  associa un attribute name con un simple type

Esempio (1/3)

Istanza di documento:

```
<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>
```

Esempio (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"/>
```

Esempio (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>
```

Collegamento fra Schemi e Istanze

```
<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>
```

Dichiarazioni di Element e Attribute

Esempi:

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

Simple Type (tipi di dato) – Primitivi

string	*una qualunque stringa Unicode*
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`
…	

Simple Type Derivati – Restrizione

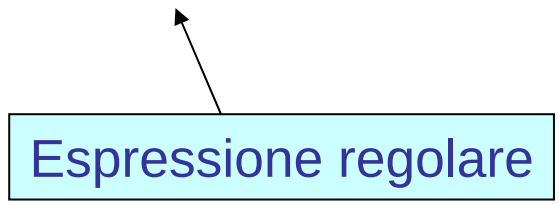
Definiti imponendo vincoli aggiuntivi sulle sfaccettature (facets) disponibili:

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

Esempi

```
<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 Derivati – Liste

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

Corrisponde a liste di `integers` separati da spazi bianchi

Simple Type Derivati – Unione

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

Simple Type Derivati Predefiniti

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

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

Complex Type – Sequenza ordinata

```
<complexType name="personalRecord">
  <sequence>
    <element name="name" type="string"/>
    <element name="address" type="string"/>
    <element name="city" type="string"/>
    <element name="birthDate" type="date"/>
    <element name="phone" type="phoneNumber"
             maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

Complex Type – Unione

```
<complexType name="transportation">
  <choice>
    <element name="train" type="string"/>
    <element name="plane" type="string"/>
    <element name="car" type="string"/>
    <element name="bike" type="string"/>
  </choice>
</complexType>
```

Complex Type – Sequenza non ordinata

```
<complexType name="unorderedRecord">
  <all>
    <element name="name" type="string"/>
    <element name="address" type="string"/>
    <element name="city" type="string"/>
    <element name="birthDate" type="date"/>
    <element name="phone" type="phoneNumber"
             maxOccurs="unbounded"/>
  </all>
</complexType>
```

Vincoli sull'uso di <all>

- Elementi dichiarati dentro <all> devono avere un valore maxOccurs pari a "1" (minOccurs può essere sia "0“ che "1")
- Se un Complex Type usa <all> ed estende un altro tipo, allora il tipo genitore deve avere contenuto empty content.
- L'elemento <all> non può essere annidato dentro <sequence>, <choice>, o un altro <all>
- Il contenuto di <all> deve essere di soli element. Non può contenere <sequence> o <choice>

Esempio di tipo complesso

```
<complexType name=“life">
   <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="work" type="xsd:string"/>
            <xsd:element name="eat" type="xsd:string"/>
        </xsd: sequence>
        <xsd:choice>
            <xsd:element name="work" type="xsd:string"/>
            <xsd:element name="play" type="xsd:string"/>
        </xsd:choice>
        <xsd:element name="sleep" type="xsd:string"/>
    </xsd:sequence>
</complexType>
```

Corrisponde a expr. regolare: ((work, eat)*, (work | play), sleep)*

Complex Type a Contenuto Complesso

- Modelli di contenuto come **espressioni regolari**:
  - Elemento reference `<element ref="name"/>`
  - Concatenazione `<sequence> ... </sequence>`
  - Unione `<choice> ... </choice>`
  - All `<all> ... </all>`
  - Elemento wildcard: `<any namespace="..." processContents="..."/>`
- Attributo reference: `<attribute ref="..."/>`
- Attributo wildcard: `<anyAttribute namespace="..." processContents="..."/>`

Vincoli di cardinalità: `minOccurs`, `maxOccurs`, `use`

Contenuto Mixed: `mixed="true"`

Esempio

```
<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 Type a Contenuto Semplice

```
<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>
```

Derivazione con Contenuto Complesso

```
<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>
```

Nota: `restriction` ***non è l'opposto di*** `extension`***!***

Descrizioni Globali vs. Locali

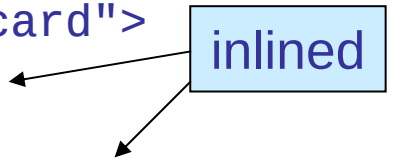
Stile Globale (a livello top):

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

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

Stile Locale (inlined):

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



Descrizioni Globali vs. Locali

- Le definizioni di tipo locale sono **anonime**
- Le dichiarazioni di element/attribute locali possono essere **overloaded**
  – una semplice forma di *dipendenza dal contesto* (particolarmente utile per gli attributi!)
- Solamente elementi dichiarati globalmente possono essere il punto di partenza per la validazione (es. **roots**)
- Le definizioni locali consento una semantica alternativa per i **namespace** (spiegato più avanti...)

Requisiti per Complex Type

- **Due dichiarazioni di element** che hanno lo **stesso nome** e compaiono **nello stesso complex type** devono avere **tipi identici**

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

  - Questo requisito rende più semplici le implementazioni

- `all` può solamente contenere `element` (es. non `sequence`!)
- ...

Namespace

- `<schema targetNamespace="..." ...>`
- **Prefissi** usati anche in certi **valori di attributi**!
- ***Unqualified Locals***:
  - se abilitati, il nome di un elemento/attributo **dichiarato localmente** nelle istanze di documento non deve avere **alcun prefisso di namespace** (ovvero **empty namespace URI**)
  - tale elemento/attributo **"appartiene" all'element dichiarato nella definizione globale che lo racchiude**
  - il comportamento di default cambia sempre usando `elementFormDefault="qualified"`

Unicità, Chiavi, Riferimenti

```
<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 ogni `widget`, ciascun `part` deve avere unico (`manufacturer`, `productid`)

Si usa un sottoinsieme di XPath

in ogni `widget`, per ogni `annotation`, (`manu`, `prod`) deve corrispondere a `my_widget_key`

`unique`: come `key`, ma possono mancare i field

Altre caratteristiche di XML Schema

- Gruppi
- Valori Nil
- Annotazioni
- Gestione di default e spazi bianchi
- Modularizzazione

– *non li trattiamo...*

RecipeML con 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 con 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 con 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 con 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 con 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>
```

Punti di Forza di XML Schema

- Supporto ai Namespace
- Tipi di Dato (predefinti e derivazione)
- Modularizzazione
- Meccanismo per la derivazione dei tipi

Essential Online Resources

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