UNICAL - A.A. 2006-2007

Gestione della Conoscenza

Prof. Massimo Ruffolo

Ing. Marco Manna

Capitolo 3

- ...
- XML: eXtensible Markup Language
  - Introduzione
  - Logical Structures
  - Document Type Definition
  - Namespace
  - XML - Schema
- ...

XML: eXtensible Markup Language

Introduzione

XML: eXtensible Markup Language (1)

- XML è nato per far fronte alle limitazioni di HTML nella realizzazione delle **nuove applicazioni Web**, in cui i **dati** costituiscono un elemento essenziale *(data-centric Web applications)*.
- XML è stato quindi il primo passo per **assegnare una semantica ai tag** permettendo lo scambio di informazioni tra database diversi.
- L' adozione di XML agevola la gestione di collezioni di documenti, e costituisce un supporto fondamentale per la pubblicazione di informazioni a livello internazionale, con il non piccolo vantaggio di essere indipendente dalla piattaforma e dal linguaggio.
- XML è stato definito “ASCII del 2000”.

XML: eXtensible Markup Language (2)

- Il documento XML è verboso ma logicamente ben strutturato, esso contiene nel contempo
  - sia i nomi dei campi
  - che i loro valori
- Il modello relazione è più semplice perché
  - una ennupla di una tabella riporta solo i valori
  - mentre lo schema (nomi e tipi dei campi) sono memorizzati una volta per tutte a parte
- Linguaggio di marcatura descrittiva del contenuto logico dei dati
  - per la visualizzazione è necessaria una descrizione a parte attraverso un ulteriore linguaggio, ad esempio
    - CSS (Cascading Style Sheets)
    - XSL (eXtensible Stylesheet Language)

Il caso HTML

- HTML *(**H**yper**T**ext **M**arkup **L**anguage)* nasce semplici documenti testuali con immagini e collegamenti ipertestuali
- L'elemento fondamentale è il **tag**
  - testo tra **'<'** e **'>'** contenente informazioni circa il testo
  - un meta-dato circa il dato vero e proprio che è nel testo
- Con il successo del web HTML viene usato per **scopi diversi** da quelli per cui era stato progettato
  - **estensioni proprietarie**
  - I parser *(browser)* **rilassano le regole sintattiche** ed interpretano anche documenti HTML "scorretti" *(in maniera differente l'uno dall'altro)*

Da HTML ad XML

- **XML** nasce dall'intento di applicare il paradigma dei **tag** in campi diversi dalla presentazione di ipertesti
- Si basa sul markup in modo simile ad HTML
- **XML** è pensato per **descrivere dati**
- I **tag** XML **non** sono **predefiniti**
- XML **non è un linguaggio**, ma un insieme di regole per costruire particolari linguaggi (metalinguaggio)

I tag in HTML

- I ***tag*** di **HTML** contengono informazioni per la visualizzazione dei dati
- La semantica di ciascun tag è nota a priori.

```
1   <html>
2
3   <body>
4
5   Note:<br>
6   to: <i>Luca</i><br>
7   from: <i>Carlo</i><br>
8   title: <b>Appuntamento</b><br>
9   Ricordati la riunione di oggi
10
11  </body>
12
13  </html>
```

Note:
to: *Luca*
from: *Carlo*
title: **Appuntamento**
Ricordati la riunione di oggi

I tag in XML (1)

- Un documento XML è simile ad un HTML, in cui però possiamo *"inventare"* i tag
- La scelta dei tag può essere effettuata a seconda delle informazioni che interessa rappresentare

```
1 <?xml version="1.0"?>
2 <note>
3     <to>Luca</to>
4     <from>Carlo</from>
5     <title>Appuntamento</title>
6     <message>Ricordati la
7      riunione di oggi</message>
8 </note>
```

```
<?xml version="1.0" ?>
- <note>
    <to>Luca</to>
    <from>Carlo</from>
    <title>Appuntamento</title>
    <message>Ricordati la
      riunione di oggi</message>
  </note>
```

XML: eXtensible Markup Language

Logical Structures

Caratteristiche base

- XML usa **tag** di **inizio** e **fine** per marcare i *campi informativi*
  - `<importo>23.45</importo>`
- Un *campo informativo* tra due **marcatori** è detto **elemento**
  - `23.45`
- Un **elemento** può essere ulteriormente arricchito dalla presenza di coppie nome/valore dette **attributi**
  - `id="ord001"`
- Regole generali x un XML **ben formato**
  - I tag devono essere inseriti correttamente uno dentro l' altro
  - Ci deve essere corrispondenza tra tag di apertura e di chiusura
  - Sono previsti elementi a campo informativo **nullo**
  - Gli attributi dei tag devono essere racchiusi tra doppi apici

Sintassi formale semplificata

- <document> ::= <prolog> <element> <Misc>*
- <prolog> ::= <XMLDecl> <Misc>* (<doctypedecl> <Misc>*)?
- <XMLDecl> ::= '<?xml version="1.0" encoding="ISO-8859-1"?>'
- <element> ::= <EmptyElemTag> | <STag> <content> <ETag>
- <EmptyElemTag> ::= '<' <Name> (<S> <Attribute>)* <S>? '/>'
- <STag> ::= '<' <Name> (<S> <Attribute>)* <S>? '>'
- <content> ::= <CharData>? ((<element> | <Comment>) <CharData>?)*
- <ETag> ::= '</' <Name> <S>?'>'
- <Attribute> ::= <Name> <Eq> <AttValue>
- <Name> ::= [a-zA-Z_] ([a-zA-Z_0-9.] | '-')*
- <Eq> ::= '='
- <CharData> ::= [^<&]
- <AttValue> ::= '"' [^<&"]* '"'
- <Misc> ::= <Comment> | <S>
- <Comment> ::= '<!--' <Char>+ '-->'
- <S> ::= (#x20 | #x9 | #xD | #xA)+
- <Char> ::= [#x1-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]

Sintassi senza “separators”

- <document> ::= <prolog> <element> <Misc>*
- <prolog> ::= <XMLDecl> <Misc>* (<doctypedecl> <Misc>*)?
- <XMLDecl> ::= '<?xml version="1.0" encoding="ISO-8859-1"?>'
- <element> ::= <EmptyElemTag> | <STag> <content> <ETag>
- <EmptyElemTag> ::= '<' <Name> <Attribute>* '/>'
- <STag> ::= '<' <Name> <Attribute>* '>'
- <content> ::= <CharData>? ((<element> | <Comment>) <CharData>?)*
- <ETag> ::= '</' <Name> '>'
- <Attribute> ::= <Name> <Eq> <AttValue>
- <Name> ::= [a-zA-Z_] ([a-zA-Z_0-9.] | '-')*
- <Eq> ::= '='
- <CharData> ::= [^<&]
- <AttValue> ::= '"' [^<&"]* '"'
- <Misc> ::= <Comment>
- <Comment> ::= '<!--' <Char>+ '-->'
- <Char> ::= [#x1-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]

Attributi o elementi? (1)

- Spesso le stesse informazioni possono essere rappresentate sia tramite attributi che tramite (sotto)elementi.
- La scelta tra attributi o elementi è soggettiva, tuttavia le due soluzioni non sono in genere equivalenti.

```
<?xml version="1.0" ?>
- <doc>
- <note>
    <to>Luca</to>
    <from>Carlo</from>
    <title>Appuntamento</title>
    <message>...</message>
  </note>
- <note title="Appuntamento">
    <to>Luca</to>
    <from>Carlo</from>
    <message>...</message>
  </note>
</doc>
```

Attributi o elementi? (2)

- Problemi con gli attributi:
  - Non possono contenere **valori multipli**
  - Sono difficilmente **espandibili** *(aggiunta di sottoelementi)*
  - Non possono descrivere **strutture**
  - Sono difficili da controllare rispetto ad un formato di documento
- È opportuno usare gli attributi per informazioni essenziali per l'elemento, come ad esempio gli **identificativi** (ID)

XML: eXtensible Markup Language

Document Type Definition

XML ben formati ed XML validi

- I due diversi livelli di standardizzazione generano due livelli di “correttezza”:
  - XML **ben formato**: un file XML è ben formato quando obbedisce a tutte le regole di XML, ad esempio deve avere il nesting corretto dei tag, un unico elemento radice e non deve avere errori “di sintassi”
  - XML **valido**: per essere valido un XML deve essere ben formato e, inoltre, deve presentare i tag corretti per la specifica applicazione, nel giusto ordine e con i giusti attributi
- Esempio ben formato ma non “tipicamente” valido
  - `<title><book/><book>Titolo</book></title>`
- La buona formazione può essere valutata conoscendo solo XML
- La validazione dipende dalla specifica applicazione e cioè dallo specifico linguaggio

Document Type Definition

- DTD (Document Type Definition) fa parte dello standard XML e permette di specificare le regole di validazione del particolare linguaggio
- Il cuore di una applicazione XML è il parser, ovvero quel modulo che legge il documento e ne crea una rappresentazione interna adatta all'elaborazione
- Un parser validante, in presenza di un DTD, è in grado di verificare la validità del documento
- Un parser non validante, invece, anche in presenza di un DTD può solo controllare la buona forma
- Visto dal DTD un file XML è composto da elementi, tag, attributi, entità, PCDATA e CDATA

Esempio di file XML

```
<Autori>
    <Autore>
          <Nome>Alessandro</Nome>
          <Cognome>Manzoni</Cognome>
          <DataNascita>
                    <Giorno>7</Giorno>
                    <Mese>3</Mese>
                    <Anno>1785</Anno>
          </DataNascita>
    </Autore>
    <Autore>
          <Nome>Giacomo</Nome>
          <Cognome>Leopardi</Cognome>
          <DataNascita>
                    <Giorno>29</Giorno>
                    <Mese>6</Mese>
                    <Anno>1798</Anno>
          </DataNascita>
    </Autore>
</Autori>
```

```
- <Autori>
  - <Autore>
      <Nome>Alessandro</Nome>
      <Cognome>Manzoni</Cognome>
    - <DataNascita>
        <Giorno>7</Giorno>
        <Mese>3</Mese>
        <Anno>1785</Anno>
      </DataNascita>
    </Autore>
  - <Autore>
      <Nome>Giacomo</Nome>
      <Cognome>Leopardi</Cognome>
    - <DataNascita>
        <Giorno>29</Giorno>
        <Mese>6</Mese>
        <Anno>1798</Anno>
      </DataNascita>
    </Autore>
</Autori>
```

Esempio di DTD

```
<!—-Doc Type Declaration-->

<!DOCTYPE Autori [
   <!ELEMENT Autori (Autore+)>
        <!ELEMENT Autore (Nome, Cognome, DataNascita)>
              <!ELEMENT Nome (#PCDATA)>
              <!ELEMENT Cognome (#PCDATA)>
              <!ELEMENT DataNascita (Giorno, Mese, Anno)>
                    <!ELEMENT Giorno (#PCDATA)>
                    <!ELEMENT Mese (#PCDATA)>
                    <!ELEMENT Anno (#PCDATA)>
]>
```

Contenuto degli elementi

- Per ogni tipo di elemento viene indicato il tipo di contenuto, che può essere:
  - **Any content**: indica che ogni contenuto è ammissibile
    - `<!ELEMENT memo ANY>`
  - **Empty content**: un elemento vuoto non può contenere alcun testo tra il tag di inizio e quello di chiusura e può quindi essere rappresentato da un tag vuoto
    - `<!ELEMENT br EMPTY>`
  - **Simple content**: è un elemento il cui contenuto è composto da testo. In questo caso #PCDATA è acronimo di “Parsed Character Data”
    - `<!ELEMENT message (#PCDATA)>`
  - **Element content**: è il caso tipico in cui il contenuto è composto da sotto-elementi
    - `<!ELEMENT note (to, from, title, message)>`
  - **Mixed content**: sono elementi che contendono testo misto ad altri elementi

Element Content Complessi

- I costrutti possono combinarsi dando origine ad espressioni regolari
  - `<!ELEMENT sezione (titolo, abstract?, para+)>`
    ogni sezione ha un titolo, può avere un abstract opzionale, seguito da almeno un paragrafo
  - `<!ELEMENT sezione (titolo, (abstract | para)+)>`
    Dentro all'elemento sezione ci deve essere un titolo, seguito da almeno un abstract o un para, che poi possono ripetersi in qualunque ordine e numero
  - `<!ELEMENT sezione (titolo, abstract*, para+)>`
    Ogni elemento sezione è composto da un titolo, da una sequenza opzionale di abstract e da una sequenza si para composta da almeno un para
  - `<!ELEMENT sezione (titolo, (sottotitolo | abstract)?, para+)>`
    Ogni sezione è data da un titolo, da uno tra sottotitolo ed abstract, che possono però anche mancarem e da una serie di para
  - `<!ELEMENT sezione (titolo, sottotitolo?, abstract?, para+)>`
    Come sopra, ma sottotitolo ed abstract possono coesistere

Mixed Content

- In XML il contenuto di testo #PCDATA ed il contenuto di elementi possono combinarsi solo nella forma seguente:
  - `<!ELEMENT para (#PCDATA | bold | italic)*>`
- Ad esempio ogni paragrafo contiene un testo in cui si possono trovare, opzionalmente, degli elementi **<bold>** ed **<italic>**:
  - `<para><bold>Questo</bold>testo contiene delle sezioni in <bold>grassetto</bold> ed in <italic>corsivo</italic>, ma potrebbe anche non averne</para>`

Attributi

- Il DTD permette anche di **vincolare gli attributi** dei singoli tag, cioè dei singoli elementi
- Gli attributi vengono specificati dal costrutto **`ATTLIST`**:
  - ```
    <!ATTLIST elemento
          attributo1   tipo1    modificatore1
          attributo2   tipo2    modificatore2
          attributo-n  tipo-n   modificatore-n>
    ```
- I tipi definiscono l'insieme o la tipologia dei valori assumibili dall'attributo
- I modificatori identificano le condizioni di *obbligatorietà* o *opzionalità* dell'attributo ed, eventualmente, un *valore di default* per lo stesso.
    ```

Attributi Stringa

- Esempio
  - `<!ATTLIST messagge`
    `lang CDATA "Italiano">`
- In questo caso l'attributo "*lang*" è una stringa
- Se l'attributo è presente nel file il suo valore è quello specificato
  - `<note>…<message lang="English">Ricordati l'appuntamento</message></note>`
- Altrimenti viene assunto il valore di default *"Italiano"*
  - `<note>…<message>Ricordati l'appuntamento </message></note>`

Tipi di attributi predefiniti

- DTD definisce alcuni tipi speciali, che aiutano il progettista soprattutto per quanto riguarda le relazioni tra elementi
  - **ID**: identificativo univoco all'interno del file
    - `<!ATTLIST User login ID #REQUIRED>`
  - **IDREF**: riferimento ad un identificativo univoco definito nel file
    - `<!ATTLIST User userClass IDREF #REQUIRED>`

Modificatori

- Valore di **default**: espresso da una stringa indica il valore da assegnare all'attributo in mancanza di diverse indicazioni
- Valore **fisso**: definito da **`#FIXED`** più il valore. L'attributo assume obbligatoriamente il valore assegnato e l'autore del documento XML non può modificarlo:
  - `<!ATTLIST persona numeroGambe CDATA #FIXED "2">`
- Specifica di **obbligatorietà**: **`#REQUIRED`**. Indica che l'attributo deve essere sempre presente in ogni elemento
  - `<!ATTLIST misura val CDATA #REQUIRED>`
- Specifica di **opzionalità**: **`#IMPLIED`**. Indica che l'attributo è opzionale e può non essere specificato dall'autore del documento. *(Se combinato con ID indica che il sistema genererà un identificativo automaticamente).*

DTD: Sintassi formale semplificata

- <prolog> ::= <XMLDecl> <Misc>* (<doctypedecl> <Misc>*)?
- <doctypedecl> ::= '<!DOCTYPE' <S> <Name> <S>?
  ('[' <intSubset> ']' <S>?)? '>'
- <intSubset> ::= (<markupdecl> | <S>)*
- <markupdecl> ::= <elementdecl> | <AttlistDecl> | <Comment>
- <elementdecl> ::= '<!ELEMENT' <S> <Name> <S> <contentspec> <S>? '>'
- <contentspec> ::= 'EMPTY' | 'ANY' | <Mixed> | <children>
- <Mixed> ::= '(' <S>? '#PCDATA' (<S>? '|' <S>? <Name>)* <S>? ')*' |
  '(' <S>? '#PCDATA' <S>? ')'
- <children> ::= (<choice> | <seq>) ('?' | '*' | '+')?
- <choice> ::= '(' <S>? <cp> (<S>? '|' <S>? <cp>)+ <S>? ')'
- <seq> ::= '(' <S>? <cp> (<S>? ',' <S>? <cp>)* <S>? ')'
- <cp> ::= (<Name> | <choice> | <seq>) ('?' | '*' | '+')?
- <AttlistDecl> ::= '<!ATTLIST' <S> <Name> <AttDef>* <S>? '>'
- <AttDef> ::= <S> <Name> <S> <AttType> <S> <DefaultDecl>
- <AttType> ::= 'CDATA' | 'ID' | 'IDREF'
- <DefaultDecl> ::= '#REQUIRED' | '#IMPLIED' |
  (('#FIXED' <S>)? <AttValue>)

Sintassi DTD senza “separators”

- <prolog> ::= <XMLDecl> <Misc>* (<doctypedecl> <Misc>*)?
- <doctypedecl> ::= '<!DOCTYPE' <Name>
  ('[' <intSubset> ']')? '>'
- <intSubset> ::= <markupdecl>*
- <markupdecl> ::= <elementdecl> | <AttlistDecl> | <Comment>
- <elementdecl> ::= '<!ELEMENT' <Name> <contentspec> '>'
- <contentspec> ::= 'EMPTY' | 'ANY' | <Mixed> | <children>
- <Mixed> ::= '(#PCDATA' <Name>* ')*' | '(#PCDATA)'
- <children> ::= (<choice> | <seq>) ('?' | '*' | '+')?
- <choice> ::= '(' <cp> ('|' <cp>)+ ')'
- <seq> ::= '(' <cp> (',' <cp>)* ')'
- <cp> ::= (<Name> | <choice> | <seq>) ('?' | '*' | '+')?
- <AttlistDecl> ::= '<!ATTLIST' <Name> <AttDef>* '>'
- <AttDef> ::= <Name> <AttType> <DefaultDecl>
- <AttType> ::= 'CDATA' | 'ID' | 'IDREF'
- <DefaultDecl> ::= '#REQUIRED' | '#IMPLIED' |
  (('#FIXED')? <AttValue>)

XML: eXtensible Markup Language

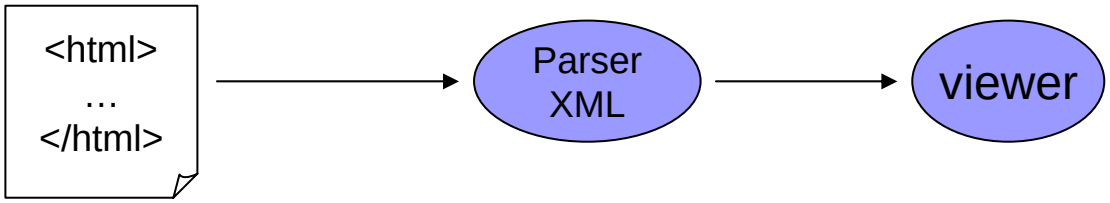
Namespace

Applicazioni e vocabolari

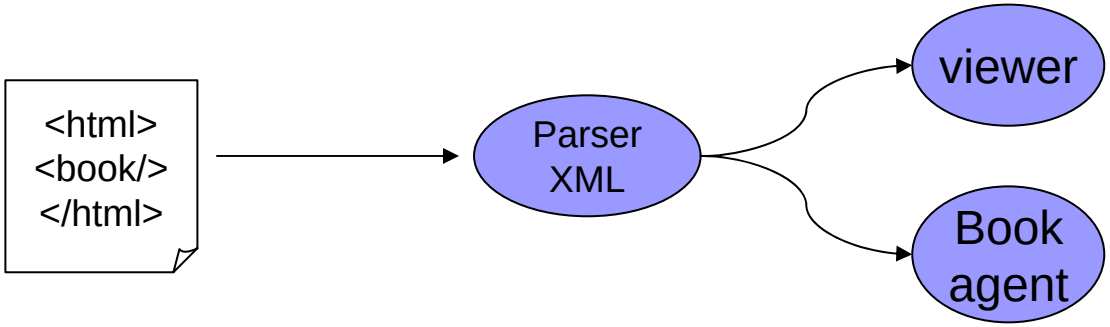
- I **tag** sono **meta-dati** che vengono interpretati dalla particolare applicazione costruita su XML
- Il **parser** XML deve fornire alle singole applicazioni le informazioni necessarie per individuare quali tag sono di loro competenza
- Occorre un meccanismo semplice che permetta al livello XML di **associare** il **tipo** ad ogni **tag** in maniera “universale”
- Soluzioni classiche:
  - Estensioni di file
  - Metadati proprietari
- In XML si può sfruttare la tecnica dei **metadati** per rappresentare queste informazioni

Mischiare i tag (1)

- Sistema di elaborazione semplice: il documento viene elaborato da una sola applicazione (esempio il viewer)



- Sistema complesso: parti diverse del documento vengono elaborate da applicazioni diverse



Mischiare i tag (2)

- Cosa accade se due applicazioni definiscono lo stesso elemento?

```
<?xml version="1.0"?>
<table>
<tr>
    <td>nome</td>
    <td>cognome</td>
</tr>
<tr>
    <td>Mario</td>
    <td>Bianchi</td>
</tr>
<tr>
    <td>Luca</td>
    <td>Rossi</td>
</tr>
</table>
```

```
<?xml version="1.0"?>
<table>
<tr><td>ordine</td>
<td>prodotto</td></tr>
<tr><td>1</td>
<td>
  <table>
  <name>Comodino</name>
  <price>100</price>
  <width um="cm">50</width>
  <length um="cm">80</length>
  </table>
</td></tr>
</table>
```

```
<?xml version="1.0"?>
<table>
    <name>Comodino001</name>
    <price curr="EU">100</price>
    <width u="cm">50</width>
    <length u="cm">80</length>
</table>
```

<table> viene riconosciuto dal viewer HTML o dal gestore dell'arredo?

I namespace (1)

- Ogni nome di elemento *(tag)* XML è preceduto da un prefisso che lo rende univoco *(tag qualificato)*
- La struttura del tag è
  - **`prefisso:nometag`**

```
<?xml version="1.0"?>
<h:table>
    <h:tr>
        <h:td>ordine</h:td>
        <h:td>prodotto</h:td>
    </h:tr>
    <h:tr>
        <h:td>1</h:td>
        <h:td>
            <fur:table>
                <fur:name>Comodino</fur:name>
                <fur:price>100</fur:price>
                <fur:width u="cm">50</fur:width>
                <fur:length u="cm">80</fur:length>
            </fur:table>
        </h:td>
    </h:tr>
</h:table>
```

I namespace (2)

- Ma non basta:
- Il prefisso potrebbe ripetersi
  - Documenti diversi potrebbero identificare lo stesso **namespace** “logico” con identificativi differenti
  - Imporre l’unicità porterebbe a namespase lunghissimi ed a codici illeggibili
- I prefissi usati vengono introdotti tramite l’attributo predefinito xmlns
  - **<h:table xmlns:h="http://www.w3.org/HTML/1998/html4" />**
  - **<http://www.w3.org/HTML/1998/html4/table />**
- Quello che conta è l’URI univoco associato al prefisso:
  - **<h:table xmlns:h="http://www.w3.org/HTML/1998/html4" />**
  - **<pippo:table xmlns:pippo="http://www.w3.org/HTML/1998/html4" />**

URI, URL ed URN

- **Uniform Resource Identifier** *(URI)* è una stringa di caratteri che identifica univocamente una risorsa sulla rete
- Il caso più comune è l'**Uniform Resource Locator** *(URL)* che identifica un dominio su Internet
- **XMLNS** non fornisce alcun significato all'URI del *namespace*, è semplicemente una stringa a cui si richiede di essere univoca

Namespace di default

- Quando l'argomento di un XML è prevalentemente riferito ad un namespace è possibile definirlo come default e sottintendere la qualificazione *(prefisso)*

```
<?xml version="1.0"?>
<table xmlns="http://www.w3.org/HTML/1998/html4" xmlns:fur="http://www.furniture.it">
    <tr>
        <td>ordine</td>
        <td>prodotto</td>
    </tr>
    <tr>
        <td>1</td>
        <td>
            <fur:table>
                <fur:name>Comodino</fur:name>
                <fur:price>100</fur:price>
                <fur:width u="cm">50</fur:width>
                <fur:length u="cm">80</fur:length>
            </fur:table>
        </td>
    </tr>
</table>
```

Scope dei prefissi

- I prefissi possono essere definiti all'interno di ogni tag di inizio o vuoto.
- L'associazione tra prefisso e NS dura per tutto l'elemento
- Se un prefisso viene ridefinito in un sottoelemento nel rispetto delle regole di visibilità il *blocco interno* nasconde quello esterno

```
<?xml version="1.0"?>
<table xmlns="http://www.w3.org/HTML/1998/html4">
    <tr>
        <td>ordine</td>
        <td>prodotto</td>
    </tr>
    <tr>
        <td>1</td>
        <td>
            <table xmlns="http://www.furniture.it">
                <name>Comodino</name>
                <price>100</price>
                <width u="cm">50</width>
                <length u="cm">80</length>
            </table>
        </td>
    </tr>
</table>
```

Attributi

- Anche gli attributi, essendo legati alla particolare applicazione, possono essere qualificati con un prefisso

```
<fur:table fur:xmlns="http:furniture.it">
  <fur:name>Comodino</fur:name>
  <fur:price>100</fur:price>
  <fur:width fur:u="cm">50</fur:width>
  <fur:length fur:u="cm">80</fur:length>
</fur:table>
```

Namespace e DTD

- Il DTD fa parte delle specifiche XML 1.0, è quindi nato insieme ad XML
- I namespace sono stati introdotti successivamente
- L'uso di namespace con DTD è macchinoso
- Occorre:
  - Esprimere i nomi qualificati come nomi validi, bloccando così il prefisso
  - Forzare l'assegnamento del prefisso per gli elementi definiti nel DTD, definendo l'attributo xmlns
- Esempio:

```
<!ELEMENT fur:table (fur:name, fur:price, fur:width, fur:length)>
<!ELEMENT fur:name (#PCDATA) >
…
<!ATTLIST fur:table xmlns:fur CDATA #FIXED "http://furniture.it">
```

XML: eXtensible Markup Language

XML - Schema

XML ben formati ed XML validi

- La buona forma di un documento XML è una proprietà puramente sintattica
  - Tutti i tag sono chiusi, propriamente innestati ed esiste un'unica radice
- La validazione è invece già "semantica", nel senso che ha a che fare con il significato dei dati e l'utilizzo del documento

Perché XML-S ?

- DTD è poco pratico e poco espressivo per le esigenze di comunicazione
- Limiti dei DTD
  - Namespace: come già visto è difficile far coesistere DTD e *namespace* perché i primi sono nati con XML, mentre i *namespace* sono stati introdotti successivamente
  - Elementi di testo: non è possibile imporre vincoli al contenuto testuale e, soprattutto, agli attributi. Non esiste il concetto di testo “tipizzato”. Esempio:

```
<corso codice="Ing.Conoscenza">
  <numeroIscritti>Marco</numeroIscritti>
</corso>
```

  - Content model misti: è possibile comporli solo come (#PCDATA|..|..)*
  - Documentazione: con i DTD posso solo inserire i commenti XML, che però possono essere ignorati dal parser
  - I DTD NON sono scritti in XML!!!

XSD

- **XSD** *(XML Schema Definition)* è una particolare applicazione **XML** ***(linguaggio)*** per descrivere le regole di validità di un altro linguaggio
- Risposta all'inadeguatezza di DTD
  - Supporto estensivo per la qualificazione tramite ***namespace***
  - Un ***sistema di tipi*** gerarchico
    - Tipizzazione del testo
    - Tipizzazione dei contenuti
  - Definizione di ***frammenti di specifica*** riutilizzabili
  - Permette di specificare vincoli per elementi strutturati ed offre grande flessibilità per ***Content Model*** *misti*
  - **Documentazione esplicita**
  - Scritto in XML
- Più complesso e "prolisso" del DTD (fattore 1: 4)

Struttura di un XML Schema

- XSD fa riferimento al **namespace** "http://www.w3.org/2001/XMLSchema"
- Un documento XSD è racchiuso in un elemento **<schema>**
  - **<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">**
  - **</xsd:schema>**
- Il documento si articola in una serie di **definizione di tipi ed elementi**
- XML Schema usa i tipi per specificare i vincoli sul contenuto degli elementi e degli attributi
  - **<xsd:element name="..." type="..."/>**
  - **<xsd:attribute name="..." type="..."/>**
- I tipi possono essere:
  - **Semplici**: un tipo semplice non può contenere markup o attributi. Si tratta di restrizioni di **#PCDATA** e **CDATA**
  - **Complessi**: un tipo complesso è l'analogo dei tipi strutturati (element content) e misti (mixed content) del DTD

Tipi semplici

- Si dividono in
  - **built-in**
  - **user-defined**
- Sono tutti qualificati. Esempio
  - `xsd:string`
- Tipi **built-in**
  - `string`, `boolean`, `decimal`, `float`
  - `Date` *(es: '2004-01-10')*, `time` *(es: '13:00:00+01:00')*
  - `ID`, `IDREF` *(stesso significato del DTD)*
  - …
- Tipi **user-defined** *(derivati)*
  - `<xsd:simpleType name="...">...</xsd:simpleType>`

Derivazione per restrizione (1)

- Il metodo classico per derivare tipi user-defined è quello di partire da un tipo già noto e **restringere i valori assumibili**
- Ogni tipo semplice ha delle **caratteristiche** *(factes)* che possono essere usate nella restrizione
- **Facets**
  - **`length, minLength, maxLength`**
  - **`minExclusive, minInclusive, maxExclusive, maxInclusive`**
  - **`enumeration`**
  - **…**

Derivazione per restrizione (2)

- Esempio 1

```
<xsd:simpleType name="anno">
   <xsd:restriction base="xsd:integer">
     <xsd:minInclusive value="0" />
     <xsd:maxInclusive value="9999" />
   </xsd:restriction>
</xsd:simpleType>
```

- Esempio 2

```
<xsd:simpleType name="anno">
   <xsd:restriction base="xsd:nonNegativeInteger">
     <xsd:precision value="4" />
   </xsd:restriction>
</xsd:simpleType>
```

- Esempio 3

```
<xsd:simpleType name="salutation">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="Mr"/>
     <xsd:enumeration value="Mrs"/>
     ...
   </xsd:restriction>
</xsd:simpleType>
```

Restrizione tramite pattern

- Particolare metodo di restrizione che utilizza una sintassi tramite espressioni regolari *(JAVA, Perl)*
  - **a?**
  - **a+**
  - **a***
  - **[abcd]/(a|b|c|d)**
  - **[a-z]**
  - **a{2,4}**
  - **[^0-9]+**
- Esempio

```
<xsd:simpleType name="telefono">
   <xsd:restriction base="xsd:string">
    <xsd:pattern value="(0039-)?0[0..9]{1,3}-[0..9]+)" />
   </xsd:restriction>
</xsd:simpleType>
```

Derivazione per unione

- I valori assumibili sono l'unione dei valori assumibili da due tipi semplici

```
<xsd:simpleType name="Tpositivo">
   <xsd:restriction base="xsd:decimal">
     <xsd:minExclusive value="0.0" />
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Tgratis">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="gratis" />
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Tprezzo">
   <xsd:union membersTypes="Tpositivo Tgratis" />
</xsd:simpleType>
```

Derivazione per lista (1)

- Oltre ai tipi *"scalari"* è possibile definire come ***tipo semplice*** la lista (divisa da spazi) di ***altri tipi semplici***.

```
<xsd:simpleType name="Tpositivo">
  <xsd:restriction base="xsd:decimal">
    <xsd:minExclusive value="0.0" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TlistaDiPositivi">
  <xsd:list itemType="TPositivo" />
</xsd:simpleType>

<xsd:element name="valore" type="TlistaDiPositivi">

<valore>1 2 34 88</valore>
```

Derivazione per lista (2)

- Con i tipi lista il facet “length” si riferisce ai componenti

```
<xsd:simpleType name="Tpositivo">
  <xsd:restriction base="xsd:decimal">
    <xsd:minExclusive value="0.0"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TlistaDiPositivi">
  <xsd:list itemType="TPositivo" />
</xsd:simpleType>

<xsd:simpleType name="TlistaDiSeiPositivi">
  <xsd:restriction base="TlistaDiPositivi">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
```

Tipi complessi

- I tipi complessi sono
  - I content vuoti e generici *(**EMPTY** ed **ANY** del DTD)*
  - **Element** content
  - **Mixed** content
  - Qualunque elemento con **attributi**
- Il concetto centrale è quello di **aggregazione** *(tipi strutturati)*

Content model ANY ed EMPTY

- Si costruiscono sulla base dei tipi predefiniti
  - **xsd:anyType**
  - **xsd:complexType**
- **ANY** content è definito come **xsd:anyType**

```
<xsd:element name="memo" type="xsd:anyType" />
```

- **EMPTY** content è un complexType per cui non si specifica nessun componente

```
<xsd:complexType name="empty" />

<xsd:element name="br" type="empty" />
```

Element content (1)

- XSD utilizza degli elementi appositi per esprimere la struttura dei sottoelementi di un element content (DTD usava le espressioni regolari)
- Dato che XSD gestisce separatamente tipi ed istanze occorre assegnare ad ogni sottoelemento sia il nome (tag) che il tipo (struttura del contenuto e degli attributi)
- **Sequenza**

```
<xsd:sequence>
   <xsd:element name="A" type="tipoA" />
   <xsd:element name="B" type="tipoB" />
   ...
</xsd:sequence>
```

- Esempio

```
<xsd:complexElement name="note">
   <xsd:sequence>
     <xsd:element name="title" type="xsd:string" />
     <xsd:element name="from" type="xsd:string" />
     <xsd:element name="to" type="xsd:string" />
   </xsd:sequence>
</xsd:complexElement>
```

Element content (2)

- Choice

```
<xsd:choice>
  <xsd:element name="A" type="tipoA" />
  <xsd:element name="B" type="tipoB" />
  ...
</xsd:choice>
```

- Set

```
<xsd:all>
  <xsd:element name="A" type="tipoA" />
  <xsd:element name="B" type="tipoB" />
  ...
</xsd:all>
```

Element content (3)

- Recurrences
  - **`xsd:minOccurs`** numero di occorrenze minime
    - Valore di default il valore **`"1"`**
  - **`xsd:maxOccurs`** numero massimo di occorrenze
    - può essere **`"unbounded"`**
    - Valore di default il valore **`"1"`**
- Recurrence A?
  - **`<xsd:element name="…" type="…" minOccurs="0" />`**
- Recurrence A+
  - **`<xsd:element name="…" type="…" maxOccurs="unbounded" />`**
- Recurrence A*
  - **`<xsd:element name="…" type= "…" maxOccurs="unbounded" minOccurs="0" />`**

Element content complessi

- Esempio con DTD

```
<!ELEMENT sezione (titolo, (sottotitolo | abstract)?, para+)>
```

- Esempio con xsd

```
<xsd:element name="sezione">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="titolo" type="xsd:string" />
      <xsd:choice minOccurs="0">
        <xsd:element name="sottotitolo" type="xsd:string"/>
        <xsd:element name="abstract" type="xsd:string"/>
      </xsd:choice>
      <xsd:element name="para" type="xsd:string" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Mixed content (1)

- Esempio con DTD

```
<!ELEMENT testo (#PCDATA | bold | italic)*>
```

- Esempio con xsd

```
<xsd:complexType name="TipoTesto" mixed="true">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="bold" type="xsd:string" />
    <xsd:element name="italic" type="xsd:string" />
  <xsd:choice>
</xsd:complexType>

<xsd:element name="testo" type="TipoTesto">
```

Mixed content (2)

- Forzare l'ordine degli elementi

```
<xsd:complexType name="testo" mixed="true">
  <xsd:sequence>
    <xsd:element name="bold" type="xsd:string" minOccurs="0" />
    <xsd:element name="italic" type="xsd:string" minOccurs="0" />
  <xsd:sequence>
</xsd:complexType>
```

- Avere tutti i sottoelementi, anche se inframezzati da testo

```
<xsd:complexType name="testo" mixed="true">
  <xsd:all>
    <xsd:element name="bold" type="xsd:string" />
    <xsd:element name="italic" type="xsd:string" />
  <xsd:all>
</xsd:complexType>
```

Derivazione di tipi complessi

- Derivazione per restrizione: si limitano i valori assumibili dall'elemento all'interno del documento XML
  - Rafforzamento dei vincoli **`minOccurs`** e **`maxOccurs`**
  - Tipizzazione più precisa di un sottoelemento o di un attributo
  - Assegnamento di un valore preciso a sottoelementi o attributi
- Derivazione per estensione :
  - Si aggiungono sottoelementi e/o attributi

Derivazione per restrizione

- Il **TestoConEffetti** è un testo con almeno un elemento **bold** o **italic**

```
<xsd:complexType name="TipoTestoConEffetti" mixed="true">
  <xsd:restriction base="TipoTesto">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="bold" type="xsd:string" />
      <xsd:element name="italic" type="xsd:string" />
    <xsd:choice>
  </xsd:restriction>
</xsd:complexType>

<xsd:element name="testo" type="TipoTesto">
```

- Può essere usato ogni volta che ci si aspetterebbe **TipoTesto**
- Altre restrizioni sono:
  - Impostare un default per gli elementi
  - Assegnare un valore fisso o specificare il tipo
  - Restringere i minOccurs-maxOccurs

Derivazione per estensione

- Si aggiungono elementi e/o attributi

```
<xsd:complexType name="TipoTestoInternazionale"
   mixed="true">
  <xsd:extension base="TipoTesto">
    <xsd:sequence>
      <xsd:element name="estratto" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="language" type="xsd:string" />
  </xsd:extension>
</xsd:complexType>

<xsd:element name="testo" type="TipoTesto">
```

Utilizzo degli XSD

- Il file XML può dichiarare lo schema a cui si riferisce sfruttando il namespace "http://www.w3.org/2001/XMLSchema-instance"
- Se lo schema **non definisce** un **targetNamespace** lo si può utilizzare per gli elementi **non qualificati**

```
<?xml version="1.0"?>
   <note
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="http://www.aaa.it/note.xsd">
     ...
    </note>
```

- Se lo schema **definisce** un **target** occorre utilizzare il namespace corretto

```
<?xml version="1.0"?>
   <note:note xmlns:note=http://www.aaa.it
     xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
     xsi:schemaLocation="http://www.aaa.it/note.xsd">
     ...
   </note:note>
```

Useful Links

- Guide
  - http://www.unicode.org/versions/Unicode5.0.0
  - http://www.w3.org/XML
  - http://www.w3.org/TR/xml11
  - http://www.w3.org/TR/xml
  - http://www.w3.org/TR/xml-names
- Validatori
  - http://www.stg.brown.edu/service/xmlvalid
  - http://validator.aborla.net