Tecnologie Web L-A

XML: Introduzione

Dario Bottazzi
Tel. 051 2093541,
E-Mail: dario.bottazzi@unibo.it,
SkypeID: dariobottazzi

Outline

- Introduzione
- Sintassi
- Linguaggi di Schema
  - DTD
  - XML Schema
- XML e Java
  - SAX
  - DOM
- Nelle ultime lezioni completeremo il discorso sulle tecnologie XML

Linguaggi di Markup

- Il linguaggio è per noi un mezzo con cui "trasportiamo" i dati
- Il linguaggio è anche uno strumento con cui codifichiamo i dati
- Infine il linguaggio è per noi un sistema attraverso cui i dati sono rappresentati, trasmessi e ricevuti dai destinatari

Cosa Intendiamo per Markup?

- I markup sono codici inseriti nel documento che ne governano la formattazione, la stampa o altri processi
- Esistono due categorie di linguaggi di Markup
  - Procedurali
  - Descrittivi

Linguaggi Procedurali

- Definiscono le operazioni che devono essere eseguite quando si processa un certo punto del documento

Es. Muovi il margine a sinistra di 3 pixel

Linguaggi Descrittivi

- Indicano semplicemente natura, funzione o contenuto dei dati memorizzati nel documento.
- I linguaggi descrittivi usano i codici di markup per assegnare nomi e categorizzare sezioni di documento
- Non vengono fornite indicazioni sul tipo di elaborazione dei dati che, viceversa viene delegata ad un programma separato

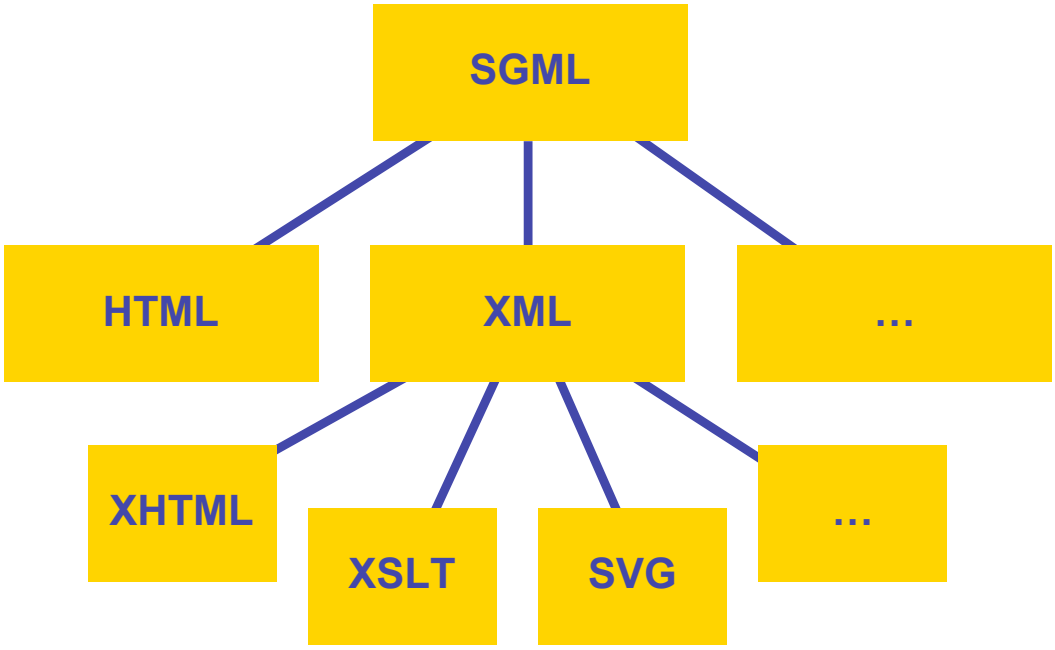
A Cosa Ci Servono i Linguaggi di Markup?

- I linguaggi di markup usano una precisa notazione per aggiungere una struttura formale al testo
- I linguaggi di markup sono costituiti da un insieme di convenzioni di markup che devono essere usate nella codifica del testo
- I linguaggi di markup specificano
  - Quali markup sono consentiti
  - Quali markup sono necessari
  - Come i markup si distinguono dal testo

Standard Generalized Markup Language

- SGML è uno standard internazionale per la rappresentazione dei dati
- SGML può essere usato per pubblicare documenti nel senso più ampio del termine
  - Pubblicare tutto: dal giornale tradizionale a contenuti multimediali sul Web
- SGML può essere usato per produrre documenti che sono allo stesso tempo interpretabili dall'uomo e facilmente scambiabili ed interpretabili dalle macchine

The Family Tree



XML

- **XML**: *e**X**tensible **M**arkup **L**anguage*
- Consente di **definire nuovi linguaggi** di **Markup**
- Ogni linguaggio è **mirato** per un **determinato dominio applicativo** ed **adotta propri tag**
- Possiamo **estendere** i **linguaggi esistenti**
- Esiste un **insieme generico** di **strumenti** per facilitare **l'elaborazione** di **documenti XML**

Obiettivi di Design di XML

Nel 1996 il W3C ha formato un gruppo di lavoro per la definizione di XML con i seguenti obiettivi:

- XML deve essere **utilizzabile** in modo semplice su **Internet**.
  - XML deve operare in maniera efficiente su Internet e soddisfare le esigenze delle applicazioni eseguite in un ambiente di rete distribuito
- XML deve **supportare** un gran **numero** di **applicazioni**

Obiettivi di Design di XML

- XML deve essere **compatibile con SGML**.
  - Un documento XML valido deve essere anche un documento SGML valido, in modo tale che gli strumenti SGML esistenti possano essere utilizzati con l'XML e siano in grado di analizzare il codice XML
- Deve essere **facile** lo **sviluppo** di **programmi** che **elaborino** documenti **XML**.
  - L'adozione del linguaggio è proporzionale alla disponibilità di strumenti e la proliferazione di questi ultimi è la dimostrazione che questo obiettivo è stato raggiunto

Obiettivi di Design di XML

- Il numero di **caratteristiche opzionali** deve essere ridotto al **minimo** possibile

- I **documenti XML** dovrebbero essere **leggibili** da un **utente**.
  - XML utilizza testo per descrivere i dati e le relazioni tra i dati, XML è più semplice da utilizzare e da leggere del formato binario che esegue la stessa operazione

Obiettivi di Design di XML

- La **progettazione** di **strutture dati XML** dovrebbe essere **rapida**.
  - XML è stato sviluppato per soddisfare l'esigenza di un linguaggio estensibile per il Web

- La progettazione di **XML** deve essere **formale** e **concisa**.
  - rendere il linguaggio il più possibile conciso, formalizzando la formulazione della specifica

- I **documenti XML** devono essere **facili** da **creare**, es tramite editor di testo

HTML VS XML (1)

- HTML
  - Orientato alla presentazione
  - Non conserva la struttura e la semantica dei dati
  - Difficile *validazione*: è sufficiente la correttezza sintattica
  - Non è estendibile
- XML
  - Orientato ai dati
  - Conserva struttura e semantica
  - Può essere *validato* tramite grammatiche
  - Può essere trasformato
  - E' estendibile

HTML VS XML (2)

- E' possibile ottenere HTML da XML tramite trasformazione
- E' praticamente impossibile estrarre dati strutturati da HTML poiché sono fusi con la logica di presentazione
- Nota: XHTML = HTML 4.0 XML compliant

Perché XML? (1)

- Standard mantenuto dal W3C
- Esistono numerosi strumenti disponibili per tutte le piattaforme
- Basato sul testo:
  - Facile utilizzo in ambiente di rete
  - Leggibile ed interpretabile anche da esseri umani
- Trasformabile con strumenti standard da un formato ad un altro
- Estendibile

Perché XML? (2)

- Utilizzi possibili:
  - Memorizzazione di documenti
    - Testo (docx)
    - Immagini (SVG)
    - …
  - Serializzazione di oggetti
  - Memorizzazione di parti di database
- Lingua franca per scambio dati fra sistemi diversi (parser XML ovunque)
- Produzione di nuovi linguaggi

Tag (1)

- Letteralmente “**etichetta**”
- **Delimitano** i dati **contenuti** nel **documento** definendone la **struttura**
- Uno ***start tag*** è costituito da un **nome** più **eventuali attributi** (v. slide succ.) racchiusi dai simboli ‘<‘,’>’

<TagName attribute-list>

- Un ***end tag*** è costituito da un **nome** (lo stesso dello *start tag*) racchiuso da ‘</’,’>’

</TagName>

Tag (2)

- Un tag vuoto (senza contenuto) è rappresentabile come:

<TagName attribute-list />

ed è equivalente a:

<TagName attribute-list></TagName>

- I **tag sono *case sensitive*!!!**

<untag> != <UnTag>

Tag (3)

Nome del tag

Attributi: coppie nome/valore

```
<invoice    customeType="trade"     dateStyle="US">
                          ....
</invoice>
```

Start Tag

Contenuto

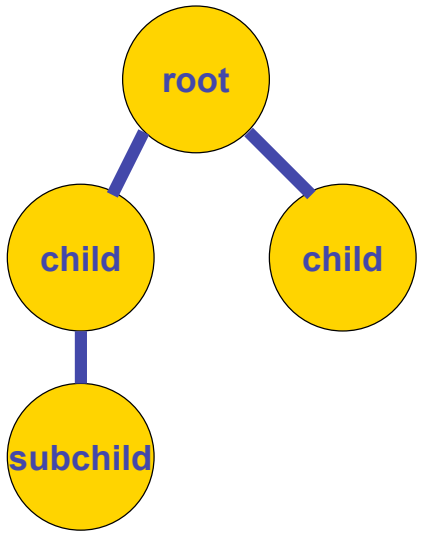
End Tag

Esempio di (Quasi) XML

(Quasi) XML Trees

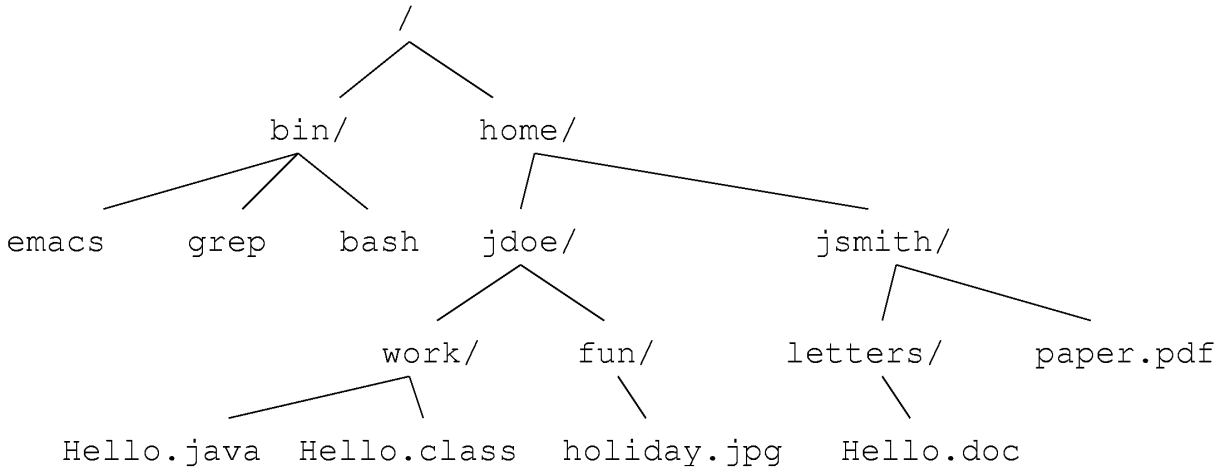
Concettualmente gli elementi di un documento XML definiscono una struttura ad albero

```
<root>
   <child>
        <subchild>
             ….
        </subchild>
   </child>
   <child>
   …
   </child>
</root>
```



Niente Paura… Nulla di Nuovo

- Siamo abituati a lavorare con strutture ad albero, es. il file system



Struttura dei Documenti XML

- **Prologo**: stabilisce una **dichiarazione XML** ed il **riferimento** (**opzionale**) ad **altri documenti** che ne **stabiliscono** la **struttura**
- **Corpo**: è il **documento XML vero e proprio**. Include **numerosi elementi** (anche eventualmente ricorsivamente innestati) che **possono** contenere **attributi**
- **XML consente** inoltre di includere nel documento **spazi bianchi**, **commenti**, **direttive** di **elaborazione** del documento,…

Esempio

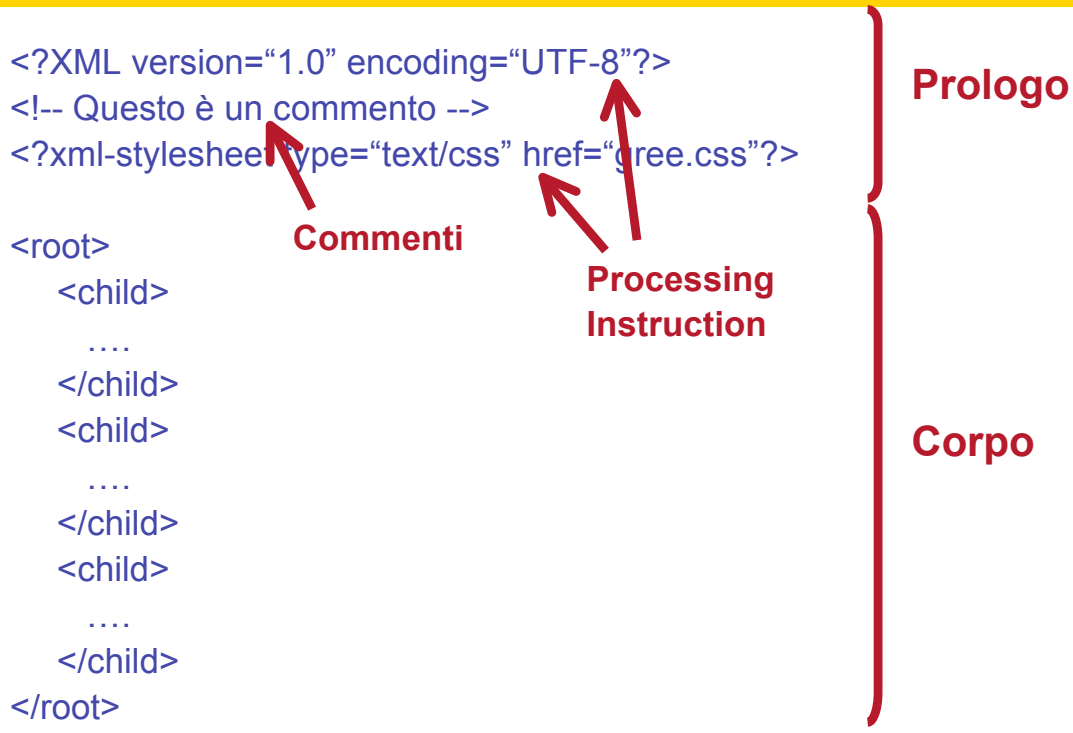
```
<?XML version="1.0" encoding="UTF-8"?>
<!-- Questo è un commento -->
<?xml-stylesheet type="text/css" href="gree.css"?>

<root>
    <child>
      ….
    </child>
    <child>
      ….
    </child>
    <child>
      ….
    </child>
</root>
```

Prologo

Corpo

Esempio



Prologo – XML Declaration

La **prima linea** di **ogni** documento **XML** inizia **sempre** con la **dichiarazione** XML.

<?xml version="1.0" encoding="UTF-8"?>

Informazioni su:

- Versione: per ora solo 1.0
- Set di caratteri (opzionale):
  - ASCII: set di caratteri a 7 bit
  - UTF-8: unicode a 8 bit
  - UTF-16: unicode a 16 bit
  - ISO-8859-1: set Latin 1 con lettere accentate
  - Shift-JIS: set di caratteri Giapponesi
  - …

Prologo – XML Declaration

- La dichiarazione XML è opzionale per i browser ma generalmente richiesta da molti strumenti e quindi è bene non ometterla
- Se presente la dichiarazione **deve iniziare** al **primo carattere** della **prima riga** del documento. **Neppure uno spazio bianco può precederla**

Prologo – Processing Instructions

<?xml-stylesheet type="text/xsl“ href=“go.xslt"?>

Istruzioni interpretate a seconda del tipo di parser utilizzato – in questo caso → trasformazione XSLT

Prologo XML: riferimento a schema

- Come avremo modo di vedere è possibile definire la sintassi dei documenti XML in documenti esterni, es. documenti DTD

```
<?xml version="1.0" encoding="UTF-8" standalone=”no"?>
<!DOCTYPE book SYSTEM “book.dtd”>
```

- Stabilisce che la **sintassi dell’XML** è nel **file locale book.dtd**
- Avremmo potuto usare anche **PUBLIC** invece di SYSTEM nel caso in cui il dtd fosse reperibile ad una URL nota

Commenti

```
<!-- Questo è un commento -->
```

- I **commenti** possono **apparire ovunque** in un **documento XML**
- I **commenti** sono **utili** per
  - **Spiegare** la **struttura** del documento **XML**
  - Commentare parti del documento durante le fasi di **sviluppo** e di **test** del nostro software
- **Sequenze** di caratteri **--** **non** sono ammesse **all’interno** dei **commenti**
- I commenti **non** sono **mostrati** dai **browser** ma sono **visibili** da parte di chi guarda il **codice sorgente** del documento XML

Elementi

- Identificano “sezioni” di documento
- Delimitati da *tags* che identificano:
  - Inizio e fine dell’elemento
  - Nome dell’elemento
- Devono avere un nome valido
- Possono essere vuoti
- Possono contenere attributi
- *Tag* di chiusura NECESSARIO!
- Possono essere annidati ma non innestati

Elementi -- Esempi

1. <E>Testo</E>
Elemento con contenuto testo

2. <E />
Elemento vuoto

3. <E> <A/> </E>
Elemento con un elemento figlio

4. <E><A/> Testo</E>
Elemento “mixed”

5. <E><A> xxx </A></E>
Annidamento lecito

6. <E><A></E></A>
Innestamento **non** lecito!!

Vincolo sul nome

- I tag **non** possono avere **nomi** che **iniziano** per **XML**, XMl, Xml, xml…

```
<forbidenNames>
    <xmlTag/>
    <XMLTag/>
    <XmLTag/>
    <xMlTag/>
    <xmLTag/>
</forbidenNames>
```

Questi nomi non possono essere usati

Attributi

- Coppie (nome, valore) associate ad elementi
- Nomi unici all'interno di un elemento
- Non consentiti nomi senza valore
- Ordine non rilevante
- Utilizzati per definire "proprietà" di elementi e non "contenuto"

Attributi – Esempi

```
<Message priority="low">
    <MailInfo>
        <Sender trust="Yes">Mr G</Sender>
        <Receiver>Mr Toon<Receiver>
    </MailInfo>
    <Subject>Saluti</Subject>
    <Body>Cari saluti a Titti</Body>
</Message>
```


Elementi o Attributi? (1)

Nella modellazione come distinguiamo elementi da attributi?

- Un elemento è estendibile in termini di contenuto (con elementi figli) e di attributi
- Un elemento è un'entità a sé stante (un oggetto?)
- Un attributo è strettamente legato ad un elemento
- Un attributo non è estendibile → può solo modellare una proprietà di un elemento in termini di valore
- Un attributo può solamente contenere un valore "atomico"

Elementi o Attributi? (2)

Non c'è una regola facilmente definibile. La scelta dipende da diversi fattori:

- Leggibilità
- Semantica
- Tipo di applicazione
- Efficienza della soluzione
- …

Elementi o Attributi? (3)

```
<libro isbn="1324AX" titolo="On the road" />
```

```
<libro isbn="1324AX">
        <titolo>On the road</titolo>
</libro>
```

```
<libro>
        <isbn>1324AX</isbn>
        <titolo>On the road</titolo>
</libro>
```

Entità – Riferimenti a Entità

- E' possibile definire aggregazioni di caratteri che possono essere condivise
- Alcuni caratteri non sono permessi poiché caratteri di controllo
- Esistono opportune entità predefinite:

Nome entità	Riferimento	Carattere
lt	&lt;	<
gt	&gt;	>
amp	&amp;	&
apos	&apos;	‘
quot	&quot;	“

Caratteri Unicode

- È possibile inserire un carattere *unicode* tramite un *character entity reference*
- Il riferimento &#dddd; inserisce il carattere avente codice decimale dddd
- Esempi:
  - ½ → ½
  - &#0232: → è

Sezioni CDATA (1)

- *CDATA* = ***Character Data***
- Il testo contenuto in una sezione CDATA NON viene analizzato dal parser

Una sezione CDATA può contenere caratteri "normalmente" proibiti

- E' delimitata dai caratteri:

<![CDATA[Contenuto della sezione]]>

Sezioni CDATA (2)

Può contenere testo "qualsiasi"

<E> **<![CDATA[<<"'!] &&]]>** </E>

caratteri contenuti sono interpretati come testo e non come elementi

<E> **<![CDATA[<Elemento/><A>Ciao</A>]]>** </E>

L'unica sequenza di caratteri che non può contenere è la sequenza di terminazione della sezione stessa

<E> **<![CDATA[No no]]> No no]]>** </E>

Problema sui nomi

- **Elementi** e **attributi possono** avere **stesso nome** e **semantica diversa**

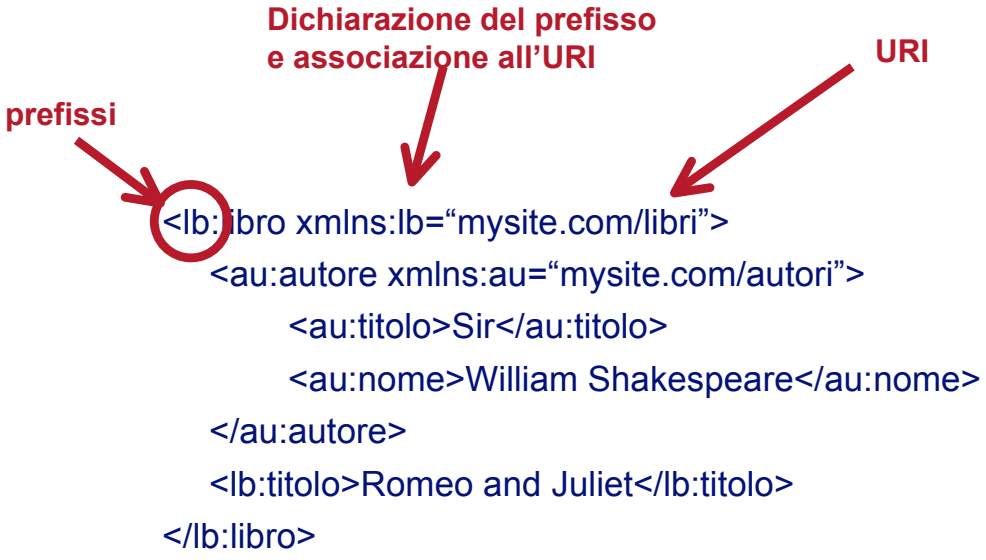
```
<libro>
    <autore>
        <titolo>Sir</titolo>
        <nome>William Shakespeare</nome>
    </autore>
    <titolo>Romeo and Juliet</titolo>
</libro>
```

→ Come distinguere?

Namespace – Introduzione

- Utilizzo di **prefissi di qualificazione** per identificare il **vocabolario** di **appartenenza** di **elementi** ed **attributi**
- **Ogni prefisso** è **associato** ad un **URI** (Uniform Resource Identifier) **ed è un alias** per l'URI stesso
- L'URI in questione è **normalmente** un **URL** → Certezza di univocità

Namespace – Esempio (1)



Namespace – Definizione

xmlns:NamespacePrefix="NamespaceURI"

- La **definizione** può essere posta, **come attributo di un elemento**, **ovunque** all'interno del documento
- Lo ***scope*** del ***namespace*** è **l'elemento all'interno** del **quale** è **stato dichiarato** → *se si dichiara un namespace nell'elemento radice, il suo scope è l'intero documento*
- L'**URI** può essere **qualsiasi** (il **parser non** ne **controlla** l'univocità) ma **dovrebbe** essere **scelto** in modo da essere **effettivamente univoco**

Namespace – Esempio (2)

```
<DC:Docenti xmlns:DC="www.unibo.it/docenti">
  <DC:Docente DC:codAteneo="112233">
      <DC:Nome>Giuseppe</DC:Nome>
      <DC:Cognome>Bellavia</DC:Cognome>
      <CR:Corso id="123“ xmlns:CR=“www.unibo.it/corsi">
        <CR:Nome>Ingegneria del Software L-A</CR:Nome>
      </CR:Corso>
      <CO:Corso id="124" xmlns:CO="www.unibo.it/corsi">
        <CO:Nome>Laboratorio di informatica L-B</CO:Nome>
      </CO:Corso>
  </DC:Docente>
</DC:Docenti>
```

Attributi qualificati

1. **CR e CO sono prefissi “collegati” allo stesso namespace**
2. **Nel secondo elemento Corso è necessario ripetere la dichiarazione di namespace poiché ricade fuori dallo scope della prima dichiarazione**

Per evitare la seconda dichiarazione basta dichiarare il namespace in un elemento più in alto nella gerarchia

Namespace – Default

È possibile definire un ***namespace*** di **default associato** al **prefisso nullo**:

```
<Docenti xmlns="www.unibo.it/docenti">
  <Docente codAteneo="112233">
      <Nome>Giuseppe</Nome>
      <Cognome>Bellavia</Cognome>
      <CR:Corso id="123“ xmlns:CR=“www.unibo.it/corsi">
        <CR:Nome>Ingegneria del Software L-A</CR:Nome>
      </CR:Corso>
    </Docente>
</Docenti>
```

Tutti gli elementi non qualificati da prefisso appartengono al namespace di default.

Namespace – Ridefinizione dei prefissi

Un prefisso di namespace (anche quello vuoto di default) può essere associato a diversi namespace all'interno di uno stesso documento

```
<PR:Docenti xmlns:PR="www.unibo.it/docenti">
  <PR:Docente codAteneo="112233">
      <PR:Nome>Giuseppe</PR:Nome>
      <PR:Cognome>Bellavia</PR:Cognome>
      <PR:Corso id="123“ xmlns:PR=“www.unibo.it/corsi">
        <PR:Nome>Ingegneria del Software L-A</PR:Nome>
      </PR:Corso>
    </PR:Docente>
</PR:Docenti>
```

…se possibile però **eviterei** di ridefinire i prefissi nello stesso documento

Leggibilità

I documenti XML devono essere facilmente leggibili e comprensibili.

L'utilizzo di *namespace* di default o la ridefinizione di prefissi può, a volte, inficiare la leggibilità violando uno dei principi fondamentali dell'XML

Un documento XML deve essere progettato, in termini di struttura e di utilizzo dei namespace, in modo da essere leggibile e facilmente fruibile.

Esempio: Espressioni Aritmetiche

Progettare un linguaggio XML per la rappresentazione di espressioni aritmetiche basate sulle quattro operazioni fondamentali.

I possibili linguaggi di rappresentazione sono *virtualmente infiniti!!!*

Qual è la rappresentazione migliore?

- Quella più leggibile
- Quella che evidenzia in modo corretto tutte le informazioni

Espressione da modellare: 5 * (8 – 2) / 3

Espressioni Aritmetiche (1)

<Exp>

5 * (8 – 2) / 3

</Exp>

Modellazione lecita ma…

- L'espressione è in formato testo: occorre scrivere un algoritmo che l'analizzi
- Non si sfruttano le potenzialità descrittive di XML

Espressioni Aritmetiche (2)

<Exp>

5 <Mul/> <Par> 8 <Minus/> 2 </Par> <Div/> 3

</Exp>

Ma…

…Necessità di introdurre un tag di rappresentazione delle parentesi per mantenere la priorità delle operazioni

→ Non è stata sfruttata la capacità di rappresentazione "gerarchica" di XML

Espressioni Aritmetiche (3)

```
<Exp>
 <Mul>
  <Num>5</Num>
   <Div>
    <Minus> <Num>8</Num> <Num>2</Num> </Minus>
    <Num>3</Num>
   </Div>
 </Mul>
</Exp>
```

- Modellazione con associatività a destra.
- Ogni *token* dell'espressione è delimitato da un *tag*
- Elevata espressività ma anche elevata verbosità

…e gli attributi dove sono finiti?!

Espressioni Aritmetiche (3)

```
<Exp>
 <Mul>
  <Num val="5"/>
   <Div>
    <Minus> <Num val="8"/> <Num val="2"/> </Minus>
    <Num val="3"/>
   </Div>
 </Mul>
</Exp>
```

Linguaggio assolutamente equivalente al precedente

Documenti *well formed*

- Rispettano le regole già dettate per elementi, attributi e namespace
- Contengono una XML Declaration corretta
- Contengono un solo elemento radice

→ Documenti *well formed* sono sintatticamente corretti… e la semantica?

Validazione

- Un documento well formed (quindi sintatticamente corretto) può non essere semanticamente valido; può, cioè, contenere informazioni estranee al dominio:

```
<Message priority="low">
    <Lasagne>Si, grazie!!</Lasagne>
    <Subject>Saluti</Subject>
    <Body>Cari saluti a Titti</Body>
</Message>
```

- Occorre una descrizione formale della struttura "desiderata" per una certa classe di documenti XML
- Un documento XML *well formed* munito di una grammatica può essere validato

Il Documento è Well-Formed?

```
<example>
    <isLower>
       23 < 46
    </isLower>
    <ampersand>
      Willey & sons
    </ampersand>
</example>
```

Il Documento è Well-Formed?

```
<example>
   <right-bracket> both > and &gt; permitted</right-bracket>
   <double-quote> both " and &quot; permitted</double-quote>
   <apostrophe> both ' and &apos; permitted</apostrophe>
   Useful in: <el value=" &apos; &quot; &apos; "/>
</example>
```


Lettura di doc. XML

- Disponibili ormai per tutte le piattaforme diversi tipi di *parser* (processore)
- Parser:
  - ***Validante***: verifica che il doc. sia sintatticamente corretto e che rispetti una struttura specificata tramite una grammatica

    → GARANZIA DI CORRETTEZZA SEMANTICA
  - ***Non validante***: verifica solo la correttezza sintattica

    → Può essere sufficiente?

    Dipende dall'applicazione!

Abbiamo visto un po' di XML…

- Ma quali sono le effettive applicazioni di XML?
- I domini applicativi sono limitati dalla fantasia
- Classificazione semplificata
  - Data-oriented languages
  - Document-oriented languages
  - Protocols and programming languages
  - Hybrids

XHTML: World Wide Web

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Hello world!</title></head>
  <body>
    <h1>This is a heading</h1>
    This is some text.
  </body>
</html>
```

CML: Chimica

```
<molecule id="METHANOL">
 <atomArray>
  <stringArray builtin="id">a1 a2 a3 a4 a5 a6</stringArray>
  <stringArray builtin="elementType">C O H H H H</stringArray>
  <floatArray builtin="x3" units="pm">
    -0.748 0.558 ...
  </floatArray>
  <floatArray builtin="y3" units="pm">
    -0.015 0.420 ...
  </floatArray>
  <floatArray builtin="z3" units="pm">
    0.024 -0.278 ...
  </floatArray>
 </atomArray>
</molecule>
```


ebXML: Transazioni Finanziarie

```
<MultiPartyCollaboration name="DropShip">
 <BusinessPartnerRole name="Customer">
  <Performs initiatingRole='//binaryCollaboration[@name="Firm Order"]/
                    InitiatingRole[@name="buyer"]' />
 </BusinessPartnerRole>
 <BusinessPartnerRole name="Retailer">
  <Performs respondingRole='//binaryCollaboration[@name="Firm Order"]/
                    RespondingRole[@name="seller"]' />
  <Performs initiatingRole='//binaryCollaboration[...]/
                    InitiatingRole[@name="buyer"]' />
 </BusinessPartnerRole>
 <BusinessPartnerRole name="DropShip Vendor">
  ...
 </BusinessPartnerRole>
</MultiPartyCollaboration>
```

ThML: Addirittura la Teologia

```
<h3 class="s05" id="One.2.p0.2">Having a Humble Opinion of Self</h3>
<p class="First" id="One.2.p0.3">EVERY man naturally desires knowledge
 <note place="foot" id="One.2.p0.4">
  <p class="Footnote" id="One.2.p0.5"><added id="One.2.p0.6">
   <name id="One.2.p0.7">Aristotle</name>, Metaphysics, i. 1.
  </added></p>
 </note>;
 but what good is knowledge without fear of God? Indeed a humble
 rustic who serves God is better than a proud intellectual who
 neglects his soul to study the course of the stars.
 <added id="One.2.p0.8"><note place="foot" id="One.2.p0.9">
  <p class="Footnote" id="One.2.p0.10">
   Augustine, Confessions V. 4.
  </p>
 </note></added>
</p>
```