Linguaggi di Schema

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

DTD (Document Type Definition)

- Definisce la grammatica che descrive la composizione degli elementi costituenti una certa classe di documenti XML
  → definisce un linguaggio
  → fornisce uno strumento per la validazione semantica dei documenti XML

Non è un linguaggio XML!!!

Perché è Importante Usare un Linguaggio di Schema?

- XML ha supporti standard per la validazione dei documenti
- Se volessimo farne a meno ci troveremmo nella situazione per cui almeno il 60% del codice che scriviamo sarebbe orientato alla validazione di documenti. Il problema sarebbe complesso
- Usando XML ed i linguaggi di schema possiamo aumentare la produttività e possiamo sviluppare sistemi aperti ed interoperabili

Dichiarazione

<!DOCTYPE root-element SYSTEM "filename">

- root-element: nome dell'elemento radice

- **SYSTEM**: definisce documenti di utilizzo locale; in alternativa: **PUBLIC** (altre regole di utilizzo) utilizzato per definire documenti di utilizzo pubblico come XHTML, XSLT ecc.

→ La dichiarazione va posta sotto l'*XML Declaration:*

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "message.dtd">
<message> … </message>
```

Esempio: Messaggio (1)

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM “message.dtd”>
<message>
  <to>Bob</to>
  <from>Janet</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</message>
```

Cosa deve specificare il DTD?

L’elemento message è composto di:

1. Elemento to contenente testo
2. Elemento from contenente testo
3. Elemento heading contenente testo
4. Elemento body contenente testo

Esempio: Messaggio (2)

```
<!ELEMENT message (to,from,heading,body)>
<!ELEMENT to       (#PCDATA)>
<!ELEMENT from     (#PCDATA)>
<!ELEMENT heading  (#PCDATA)>
<!ELEMENT body     (#PCDATA)>
```

- L’elemento **message** è **vincolato** a **contenere** gli **elementi** specificati **nell’ordine di apparizione**
- **PCDATA** = Parsed Character Data: rappresenta **l’unico tipo di dato possibile ovvero il testo**

→ **Non è possibile vincolare il testo in alcun modo!!!**

Elementi (1)

<!ELEMENT element-name content-model>

Il contenuto (**content-model**) può essere:

→ Keyword **EMPTY**: **elemento vuoto**

→ Keyword **ANY**: **testo** + **elementi** qualsiasi purché **dichiarati** nel **DTD**

→ Content model ***Children***: **elementi figli** specifici con **ordine determinato**

→ Content model ***Mixed***: **testo** + **elementi figli** specificati **senza un ordine** particolare

Elementi (2)

- **Children** e **Mixed** si **differenziano** non tramite keyword, ma **tramite la notazione** (v. slide successive)
- **Tutte le dichiarazioni sono "globali"**
- **Vincolo: un elemento deve essere dichiarato una ed una sola volta**

Content Model

EMPTY

<!ELEMENT ElementoVuoto EMPTY>

Corrisponde a:

<ElementoVuoto />

ANY

<!ELEMENT Elemento ANY>
<!ELEMENT Child EMPTY>
<!ELEMENT Child1 EMPTY>

Corrisponde a:

```
<Elemento>
    <Child/>
    <Child1/>
    …qualcosa…
    <Child/>
</Elemento>
```


Content Model – Children (1)

E' possibile specificare:

- **SEQUENZA**: la **lista ordinata** degli **elementi figli** che **devono** comparire **nell'ordine specificato**: $(E_1,E_2,\ldots,E_n)$
  - Esempio:

    (A,B,(C,D))

  - Nota: fra (A,B,(C,D)) e (A,B,C,D) non c'è differenza nel content model ma c'è differenza semantica:
    - (A,B,(C,D)) : l'elemento dichiarando deve essere composto dalla sequenza A, B e dalla sequenza C, D.
    - (A,B,C,D) : l'elemento dichiarando deve essere composto dalla sequenza A, B, C, D

Content Model – Children (2)

E' possibile specificare:

- **SCELTA**: la **lista** degli **elementi figli** che **possono** comparire in **alternativa** l'uno all'altro: **($E_1|E_2|\ldots|E_n$)**
  - Esempio:

    (A|B|(C|D))

  - Nota: stesse considerazioni valide per la sequenza.
- **Occorrenza** di **ogni elemento** tramite gli **operatori ?, +, ***

→ I caratteri (,),,,|,?, +, * sono meta-simboli e non simboli dell'alfabeto terminale (sono meta-notazione DTD).

Content Model – Children (3)

- Occorrenza:

Content Particle	Significato
?	Zero o Uno
+	Uno o Più
*	Zero o Più

Content Model – Children (4)

- È possibile **innestare liste** di ***content model*** a piacimento:
  Esempio: **(A?,(B|(C,D)*))**

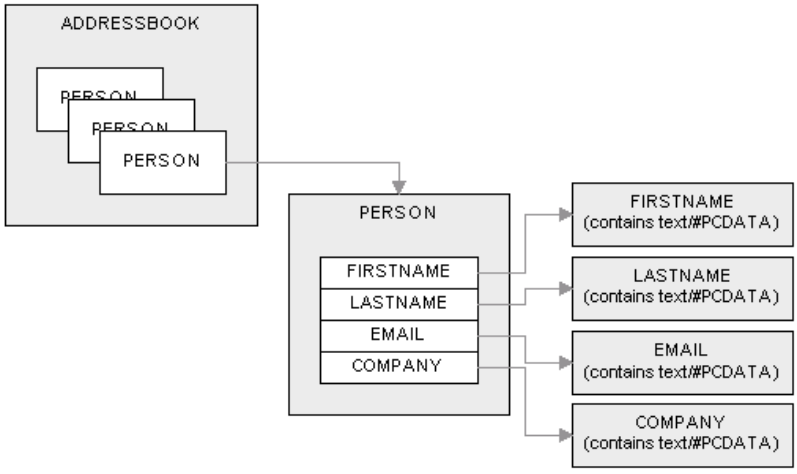
- L'elemento **#PCDATA** (Parsed Character Data) **specifica** che il **contenuto** dell'elemento **è solo testo**
  Esempio: <!ELEMENT Elemento (#PCDATA)>

→ Il **testo** è **considerato** nodo figlio **dell'elemento** che si sta dichiarando

Esempio

```
<?xml version="1.0"?>

<!DOCTYPE ADDRESSBOOK [
<!ELEMENT ADDRESSBOOK (PERSON)*>
<!ELEMENT PERSON (LASTNAME, FIRSTNAME, COMPANY, EMAIL)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT COMPANY (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)> ]>
```



Esempio: espressioni (1.1)

Progettare un DTD che consenta di validare espressioni aritmetiche in cui:

- Ogni operazione è rappresentata da un elemento diverso: Plus, Minus, Mul, Div
- L'elemento Num contiene i valori numerici
- L'elemento radice Exp può contenere un'operazione oppure un numero
- Ogni elemento operazione deve contenere due elementi figli a scelta fra un numero e un'operazione

Esempio: espressioni (1.2)

Possibile documento istanza:

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

Esempio: espressioni (1.3)

- L'elemento Num contiene i valori numerici

<!ELEMENT Num (#PCDATA)>

- Ogni operazione è rappresentata da un elemento diverso: Plus, Minus, Mul, Div

  +

- Ogni elemento operazione **deve contenere due elementi** figli **a scelta** fra un **numero** e un'**operazione**

<!ELEMENT Plus
((Num|Plus|Minus|Mul|Div), (Num|Plus|Minus|Mul|Div))

…continua

Esempio: espressioni (1.4)

<!ELEMENT Minus
((Num|Plus|Minus|Mul|Div), (Num|Plus|Minus|Mul|Div))>

<!ELEMENT Mul
((Num|Plus|Minus|Mul|Div), (Num|Plus|Minus|Mul|Div))>

Esempio: espressioni (1.5)

<!ELEMENT Div
((Num|Plus|Minus|Mul|Div), (Num|Plus|Minus|Mul|Div))>

- L'elemento radice Exp può contenere un'operazione oppure un numero

<!ELEMENT Exp (Num|Plus|Minus|Mul|Div)>

Content Model – Mixed

- **Simile ad ANY** ma **consente** di **specificare quali elementi possono comparire**
- Il ***content-model Mixed*** **non** è **specificato** tramite una **parola chiave** ma, come *Children*, tramite una **notazione particolare**:

  (#PCDATA|E_1|E_2|…|E_n)*
- #PCDATA deve essere sempre il primo elemento della lista di scelta
- La lista di scelta deve poter comparire zero o più volte (modificatore *)

Content Model: note

- Il modello ANY è troppo poco vincolante e, di fatto, **poco usato.**
- Il modello mixed è anch'esso **poco vincolante** e **non allineato** con la "**filosofia**" d'uso di **XML** → descrivere dati strutturati.

  Usando il modello **mixed** si **perde** inevitabilmente in **strutturazione**.

  → *E' buona norma marcare tutti i dati con appositi tag*

XHTML fa largo uso del modello mixed… infatti non è un linguaggio di descrizione ma di presentazione.

Mixed o non Mixed?

```
<html>
 <head><title>Tesi di laurea</title>
 <body>
    <P>Questa è la mia <B>tesi di laurea</B></P>
    <P>Bella, eh?</P>
    <H1>Ora…<I>datemi la laurea!!</I></H1>
 </body>
</html>
```

La riscrittura del documento di cui sopra senza content model mixed produrrebbe un documento illeggibile!

Attributi (1)

<!ATTLIST ElementName
$AttrName_1$ $AttrType_1$ $Value_1$
$AttrName_2$ $AttrType_2$ $Value_2$
…>

Definisce una **lista di attributi** per **l'elemento** ElementName in cui:

- $AttrName_n$ → nome dell'attributo n-esimo
- $AttrType_n$ → tipo dell'attributo n-esimo
- $Value_n$ → valore di default dell'attributo n-esimo o modificatore di presenza

Attribute Type

Tipo	Significato
CDATA	Testo
$(en_1\|en_2\|\ldots\|en_n)$	Valore scelto da una lista di enumerazione
ID	Identificatore univoco a livello di documento
Altre possibilità	…

Attribute Value

Valore	Significato
“VALUE”	L’attributo ha valore di default pari a VALUE
#REQUIRED	L’attributo deve essere presente
#IMPLIED	L’attributo è opzionale
#FIXED “VALUE”	L’attributo deve avere un valore fisso pari a VALUE

Attributi – Valore di Default

DTD:

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

XML:

```
<square width="100" />
```

Se all’attributo **NON viene assegnato** un **valore esplicito**, il suo **valore** di **default è 0. L’autore** del **documento non** è **obbligato** a **specificare** un **valore** per un **attributo** cui è stato **associato** un **default**:

<square /> → Il valore di width non è nullo, ma “0”!

Attributi – Valore Implied

DTD:

```
<!ELEMENT contact EMPTY>
<!ATTLIST contact fax CDATA #IMPLIED>
```

XML:

```
<contact fax="555-667788" />
```

Si utilizza un attributo di tipo ***implied*** quando tale **attributo non** è **obbligatorio** e **non** è **possibile stabilire** un **valore** di **default**.

<contact /> → Il valore dell'attributo fax è nullo.

Attributi – Valore Required

DTD:

```
<!ELEMENT person EMPTY>
<!ATTLIST person number CDATA #REQUIRED>
```

XML:

```
<person number="5677" />
```

Utilizzare un attributo di tipo ***required*** quando **non** è **possibile specificare** un **valore** di **default** e occorre **forzare** la **presenza** di tale **attributo**.

<person /> → Errore!

Attributi – Valore Fixed

DTD:

<!ELEMENT sender EMPTY>

<!ATTLIST sender person CDATA #FIXED “Gabriele">

XML:

<sender person=“Gabriele" />

Utilizzare un attributo di tipo ***fixed*** quando occorre che tale **attributo** abbia un **valore prefissato**. Il parser riporta un **errore** nel caso in cui venga incontrato un **valore diverso** da **quello previsto**. Se **l’attributo non è presente**, ne **viene inserito** uno col **valore *fixed*.**

<sender person=”Giuseppe” /> → Errore!

Attributi – Tipo (1)

- **CDATA**: valore di tipo testo; v. esempi precedenti
- **($en_1|en_2|\ldots|en_n$):** valore scelto da una lista:

esempio

<!ELEMENT payment EMPTY>

<!ATTLIST payment type (check|cash) “cash”>

<payment type=“check” />

<payment type=“cash” />

<payment type=“creditcard” /> → Errore!!

Attributi – Tipo (2)

ID: valore di tipo **identificatore**: il **valore dell'attributo** deve essere **univoco a livello di documento**.

ID viene **normalmente** utilizzato con **#REQUIRED**

DTD

```
<!ELEMENT orders (order+)>
<!ELEMENT order EMPTY>
<!ATTLIST order code ID #REQUIRED>
```

XML

```
<orders>
 <order code="a101"/>
 <order code="a102"/>
 …
</orders>
```

Il valore di un attributo di tipo ID deve essere un nome XML valido → non può iniziare con un numero

Esempio: espressioni (2.1)

- L'elemento Num contiene i valori numerici
- Un'espressione (Exp) può contenere o un numero o un'espressione moltiplicativa o un'espressione additiva
- Le espressioni moltiplicative (elemento MulExp) possono essere di tipo moltiplicazione o divisione. L'operazione è discriminata dall'attributo op che può contenere i valori times o div

Esempio: espressioni (2.2)

- Le espressioni additive (elemento AddExp) possono essere di tipo somma o sottrazione. L'operazione è discriminata dall'attributo op che può contenere i valori plus o minus
- Le espressioni moltiplicative o additive possono contenere due elementi figli a scelta fra numeri e espressioni moltiplicative o additive

Esempio: espressioni (2.3)

```
<!ELEMENT EXP ( NUM | ADDEXP | MULEXP ) >

<!ELEMENT ADDEXP  ( ( NUM | ADDEXP | MULEXP ),
    (NUM|ADDEXP|MULEXP) ) >

<!ATTLIST ADDEXP op (plus|minus) #REQUIRED>

<!ELEMENT MULEXP  ( ( NUM | ADDEXP | MULEXP ),
    ( NUM | ADDEXP | MULEXP ) ) >

<!ATTLIST MULEXP op (times|div) #REQUIRED>

<!ELEMENT NUM (#PCDATA) >
```

Esempio: espressioni (2.4)

- Possibile documento istanza:

```
<?xml version="1.0”>
<!DOCTYPE EXP SYSTEM "EsprApt.dtd">
<EXP>
 <ADDEXP op=“plus">
  <ADDEXP op=“minus">
   <NUM>4</NUM>
   <NUM>2</NUM>
  </ADDEXP>
  <MULEXP op=“div">
   <NUM>10</NUM>
   <NUM>2</NUM>
  </MULEXP>
 </ADDEXP>
</EXP>
```


Esempio: catalogo Film (1)

Si modelli un documento XML di catalogazione Film e relativo DTD di validazione in cui:

- Un Catalogo può contenere zero o più Film
- Un Film è descritto da un Titolo, almeno un Regista, zero o più Attore ed eventualmente un Genere
- Un Film è dotato di proprietà quali un codice identificativo univoco (cod), un’indicazione di “originalità” del supporto (originale sì – no) in cui si assume di default l’acquisto legale, un’indicazione del tipo di formato (obbligatoria) che può essere VHS, DVD, DIVX, un voto (opzionale).

Esempio: catalogo Film (2)

- Un Catalogo può contenere zero o più Film

<!ELEMENT Catalogo (Film*)>

- Un Film è descritto da un Titolo, almeno un Regista, zero o più Attore ed eventualmente un Genere

<!ELEMENT Film (Titolo,Regista+,Attore*,Genere?)>
<!ELEMENT Titolo (#PCDATA)>
<!ELEMENT Regista (#PCDATA)>
<!ELEMENT Attore (#PCDATA)>
<!ELEMENT Genere (#PCDATA)>

Esempio: catalogo Film (3)

- Un Film è dotato di proprietà quali un codice identificativo univoco (cod), un'indicazione di "originalità" del supporto (originale sì – no) in cui si assume di default l'acquisto legale, un'indicazione del tipo di formato (obbligatorio) che può essere VHS, DVD, DIVX, un voto (opzionale).

<!ATTLIST Film
cod ID #REQUIRED
originale (si|no) "si"
formato (VHS|DVD|DIVX) #REQUIRED
voto CDATA #IMPLIED>

Esempio: catalogo Film (4)

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE Catalogo [
 <!ELEMENT Catalogo (Film*)>
 <!ELEMENT Titolo (#PCDATA)>
 <!ELEMENT Regista (#PCDATA)>
 <!ELEMENT Attore (#PCDATA)>
 <!ELEMENT Genere (#PCDATA)>
 <!ELEMENT Film (Titolo,Regista+,Attore*,Genere?)>
 <!ATTLIST Film cod ID #REQUIRED
                originale (si|no) 'si'
                formato (VHS|DVD|DIVX) #REQUIRED
                voto CDATA #IMPLIED>
]>
```


Esempio: catalogo Film (5)

```
<Catalogo>
   <Film cod="f1" formato="DVD" voto="10">
        <Titolo>Blade Runner</Titolo>
        <Regista>Ridley Scott</Regista>
        <Attore>Harrison Ford</Attore>
        <Attore>Rutger Hauer</Attore>
        <Genere>Fantascienza</Genere>
   </Film>
   <Film cod="f2" formato="DIVX">
        <Titolo>Fantozzi</Titolo>
        <Regista>Luciano Salce</Regista>
   </Film>
</Catalogo>
```

Limiti DTD

- **Nessun supporto** per i ***namespace***
- Non è possibile vincolare i **dati** oltre la **stringa** generica: niente interi, reali, date…
- **Non** è **possibile creare tipi** di **dato**
- Gli **identificatori univoci** hanno ***scope* pari** al **documento**
- **Non** è **possibile** creare **chiavi** con scope **limitato**
- Il formato NON è XML
- Bassa estendibilità