



DTD Document Type Definition

1

Che cos'è DTD?

- **DTD: Document Type Definition**
- E' un linguaggio per definire la grammatica che descrive la composizione degli elementi costituenti una certa classe di documenti XML
- Definisce un linguaggio XML
- Fornisce uno strumento per la **validazione** dei documenti XML
- Non è un linguaggio XML (non rispetta la sintassi XML)

2

Perché 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
 - sviluppare sistemi aperti ed interoperabili
- Inoltre uno schema definisce un contratto fra chi produce il dato e chi lo utilizza
 - In qualunque momento è possibile applicare lo schema per verificare il rispetto del contratto

3

Dichiarazione del DTD

- Per applicare un DTD ad un documento XML nel suo prologo dobbiamo inserire una dichiarazione con questa sintassi:
`<!DOCTYPE root-element SYSTEM "filename">`
- Dove:
 - `root-element` è il nome dell'elemento radice
 - `SYSTEM` definisce documenti di utilizzo locale
 - `filename` è il file che contiene il DTD
- In alternativa a `SYSTEM` si può usare la parola chiave `PUBLIC` che serve 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">`

4

Esempio: il file XML

```
<?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 da:
 - Un elemento **to** contenente testo
 - Un elemento **from** contenente testo
 - Un elemento **heading** contenente testo
 - Un elemento **body** contenente testo

5

Esempio: il DTD

```
<!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!!!

6

Struttura di un DTD

- Un DTD è costituita da un elenco di **dichiarazioni** (**markup declarations**) che descrivono la struttura del documento
- Le dichiarazioni di una DTD definiscono:
 - gli elementi (**element**) di un documento XML
 - il modello di contenuto di ogni elemento (**content model**) ovvero gli elementi che contiene ed i loro rapporti (un elemento può essere vuoto)
 - la lista degli **attributi** associati a ciascun elemento, il loro tipo e il loro valore

7

Elementi

- Per dichiarare un elemento si usa la sintassi:
<!ELEMENT *element-name* *content-model*>
 - Il contenuto (*content-model*) può essere di 4 tipi:
 - **EMPTY**: parola chiave che indica l'elemento vuoto
 - **ANY**: indica che si può inserire testo o elementi qualsiasi (purché dichiarati nel DTD)
 - Elenco di elementi figli specifici con ordine determinato (content-model Children)
 - Testo più elenco di elementi figli senza ordine specifico (Content model Mixed)
 - I due ultimi tipi non sono indicati da una parola chiave ma attraverso la notazione usata nella definizione
 - Tutte le dichiarazioni sono globali
- 💡 **Un elemento può essere indicato una sola volta**

8

Esempi: EMPTY e ANY

```
<!ELEMENT ElementoVuoto EMPTY>
```



```
<ElementoVuoto />
```

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



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

9

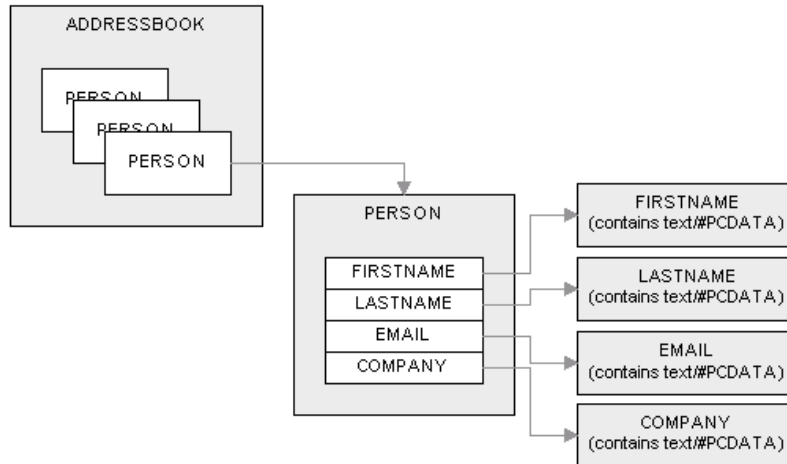
Content model - Children

- Si può specificare una **sequenza** di elementi figli che devono comparire nell'ordine specificato: $(E_1, E_2, \dots, E_n)$
- Oppure una **scelta**: lista di elementi figli che possono comparire in alternativa: $(E_1 | E_2 | \dots | E_n)$
- La differenza è data dal separatore: virgola per le sequenze, | per le scelte.
- E' anche possibile stabilire l'occorrenza di ogni elemento tramite gli operatori ?, +, *:
 - ? = zero o 1
 - + = 1 o più
 - * = zero o più
- È possibile innestare liste e operatori: $(A?, (B | (C, D) *))$
- **#PCDATA** indica che il contenuto dell'elemento è solo testo:

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

Esempio

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



11

Esempio: espressioni - 1

- Progettiamo 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

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

12

Esempio: espressioni - 2

- 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))>
<!ELEMENT Minus   ((Num|Plus|Minus|Mul|Div) ,
                    (Num|Plus|Minus|Mul|Div))>
<!ELEMENT Mul     ((Num|Plus|Minus|Mul|Div) ,
                    (Num|Plus|Minus|Mul|Div))>
<!ELEMENT Div     ((Num|Plus|Minus|Mul|Div) ,
                    (Num|Plus|Minus|Mul|Div))>
```
 - La radice Exp può contenere un'operazione oppure un numero
 - ➔ `<!ELEMENT Exp (Num|Plus|Minus|Mul|Div)>`
-

13

Content mode - Mixed

- E' 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:
$$(\#PCDATA | E1 | E2 | \dots | E_n) ^*$$
 - #PCDATA deve essere sempre il primo elemento della lista di scelta
 - La lista di scelta deve poter comparire zero o più volte (si usa quindi il modificatore *)
 - Il modello mixed è poco vincolante e non allineato con la "filosofia" di XML: descrivere dati strutturati.
-

14

Attributi

- Per definire una lista di possibili attributi per un elemento si usa la sintassi:

```
<!ATTLIST ElementName  
    AttrName1 AttrType1 Value1  
    AttrName2 AttrType2 Value2  
...>
```

- Dove il significato dei vari termini è il seguente:
 - ElementName*: nome dell'elemento
 - 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

15

Tipi e valori degli attributi

Tipo	Significato
CDATA	Testo
(<i>en</i> ₁ <i>en</i> ₂ ... <i>en</i> _n)	Valore scelto da una lista di enumerazione
ID	Identificatore univoco a livello di documento
Altre possibilità	...

Valore	Significato
" <i>VALUE</i> "	L'attributo ha valore di default pari a <i>VALUE</i>
#REQUIRED	L'attributo deve essere presente
#IMPLIED	L'attributo è opzionale
#FIXED " <i>VALUE</i> "	L'attributo deve avere un valore fisso pari a <i>VALUE</i>

16

Attributi: tipi CDATA ed enumerati

- I valori ammessi per il tipo sono:
 - **CDATA**: valore di tipo testo
 - $(en_1 | en_2 | \dots | en_n)$: valore scelto da una lista
- **DTD**:

```
<!ELEMENT payment EMPTY>
```

```
<!ATTLIST payment mode (check|cash) "cash">
```
- **XML**:

```
<payment mode="check" />
```

 Ok

```
<payment mode="cash" />
```

 Ok

```
<payment mode="creditcard" />
```

 Errore!!

17

Attributi: tipo ID

- **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

18

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:
- Quindi se scriviamo:
 - `<square />`
- Il valore di `width` non è nullo, ma "0"

19

Attributi: valore #implied

- DTD:

- `<!ELEMENT contact EMPTY>`

- `<!ATTLIST contact fax CDATA #IMPLIED>`

- XML:

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

- Si utilizza il valore **#implied** quando
 - un attributo non è obbligatorio
 - non è possibile stabilire un valore di default.
- Se scriviamo
 - `<contact />`
- Il valore dell'attributo `fax` è nullo.

20

Attributi: valore #required

- DTD:
 - `<!ELEMENT person EMPTY>`
 - `<!ATTLIST person number CDATA #REQUIRED>`
- XML:
 - `<person number="5677" />`
- Si utilizza il valore #required quando
 - non è possibile specificare un valore di default
 - occorre forzare la presenza di tale attributo.
- Quindi se scriviamo:
 - `<person />`
- Otteniamo un errore!

21

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.
- Quindi se scriviamo:
 - `<sender person="Giuseppe" />` errore!
 - `<sender />` ok, e l'attributo person vale "Gabriele"

22

Esempio: catalogo di film

- 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).

23

Catalogo film - 1

- 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**

```
<!ELEMENT Catalogo (Film*)>
```

```
<!ELEMENT Film      (Titolo,Regista
```

```
+,Attore*,Genere?)>
```

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

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

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

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

24

Catalogo film - 2

- 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 >
```

25

Catalogo film - 3

- Ecco il DTD completo
- Lo salviamo nel file catalogo.dtd

```
<!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>
```

26

Catalogo film - 4

- Ecco un documento XML valido secondo il DTD appena definito:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE Catalogo SYSTEM "catalogo.dtd"
<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>
```

27

Limiti dei 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 estensibilità

28