Introduzione

HTML

rappresentazione standardizzata delle informazioni per il web

L'hypertext markup language assieme al protocollo HTTP ha permesso di distribuire sul web un numero sempre più elevato di informazioni in modo che potessero essere fruibili ad un numero di utenti sempre maggiore.

Rivolto alla *rappresentazione* dei dati
Nessuna informazione sul tipo o la struttura

Si tratta di un linguaggio il cui scopo principale è quello di RAPPRESENTARE l'informazione e non da informazioni sulla struttura della stessa.

Marcatori non standard per introdurre nuove funzionalità

L'html ha dovuto soddisfare esigenze sempre più complesse. Questo ha portato alla realizzazione di nuovi markup il cui comportamento non è sempre standard.

Introduzione

eXtensible Markup Language

- Linguaggio di markup aperto
- Basato su testo
- Fornisce informazioni di tipo strutturale e semantico
- Derivato dal SGML (Standard Generalized Markup Language)

•XML sviluppato dall'XML Working group costituito nel nel 1996 (supervisione di W3C)

Introduzione

SGML

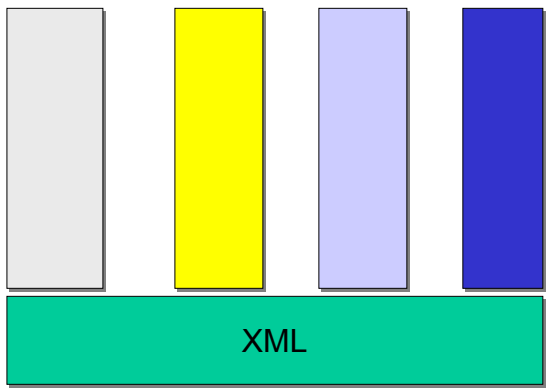
Metalinguaggio (padre di HTML)

PREGI
Potente e flessibile

DIFETTI
Struttura pesante
Sono obbligatori un DTD e uno StyleSheet
Obbligatoria la VALIDAZIONE del documento

Introduzione

Tramite XML è possibile sviluppare i propri linguaggi di mark-up



Introduzione

HTML:pregi e difetti

Pregi	Difetti
Struttura semplice	Non estensibile.
Rapida Visualizzazione dei dati su browser.	Poco controllo sul risultato finale (risultati diversi su browser diversi). Supporti di visualizzazione differenti dal browser non contemplati.
E' sufficiente che la sintassi sia corretta	Nessuna infromazione Sulla semantica. L'elaborazione dei dati è demandata a livello Java, JavaScript…
Html è diventato un linguaggio di rappresentazione invece di rimanere un linguaggio di descrizione	

Introduzione

Limiti di HTML

- Nasce con lo scopo di condividere documentazione su sistemi differenti (testo, immagini, link)
- Le informazioni sul Web sono diventate troppo complesse (suoni, video, audio)
- Applicazioni complesse su WEB
- Diversi produttori implementano differenti estensioni del linguaggio

XML

Permette la creazione di un linguaggio di mark up specifico in base all'informazione da trattare!

Introduzione

XML vs HTML

- Definizione di differenti layout (per la rappresentazione) per differenti dispositivi.
- Definizione di nuovi TAG
- Strutturazione gerarchica delle informazioni
- Descrizione della grammatica per la validazione dei dati (DTD opzionale)

Introduzione

obiettivi progettuali di XML

- Orientato al Web
- Iteroperabilità
- Compatibile con SGML
- Semplicità
- Caratteristiche opzionali ridotte

Introduzione

obiettivi progettuali di XML

- Chiari e Leggibili
- Progettazione rapida
- Conciso
- Facilità di creazione
- markup

Introduzione

Sintassi XML

Simile all'HTML ma maggiormente rigida.

Aumenta la leggibilità del documento

Introduzione

Sintassi XML

Definizione di elementi e attributi personalizzati.

DTD

La creazione di un DTD personale è OPZIONALE!

Introduzione

Sintassi XML

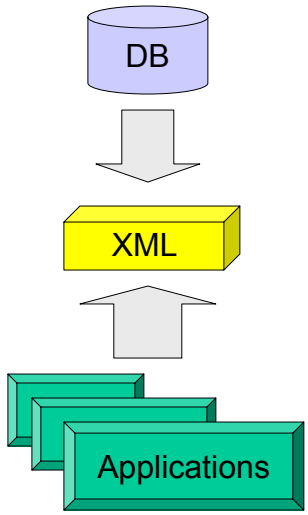
I TAG sono personalizzabili

Diversi documenti possono utilizzare i medesimi tag

Necessità di un NameSpace

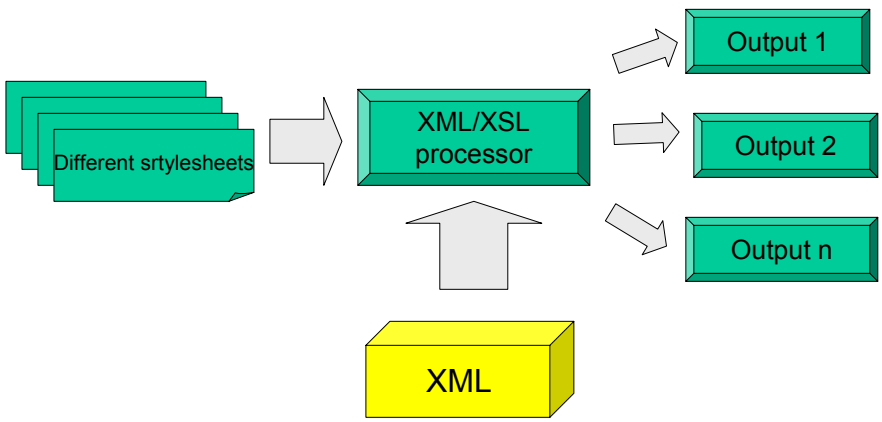
Introduzione

XML per lo scambio di informazioni

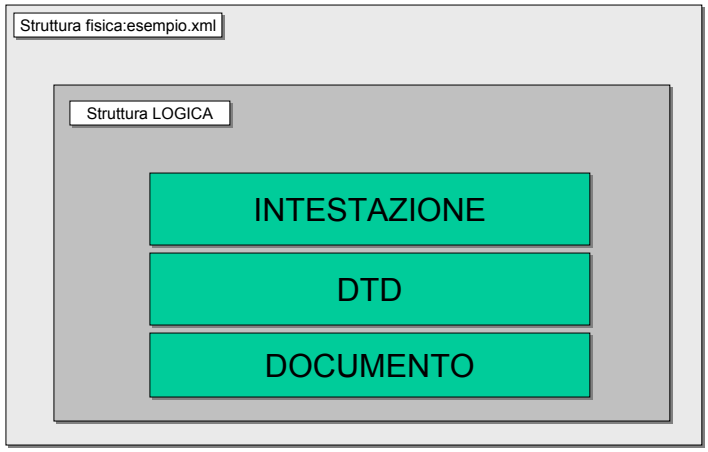


Introduzione

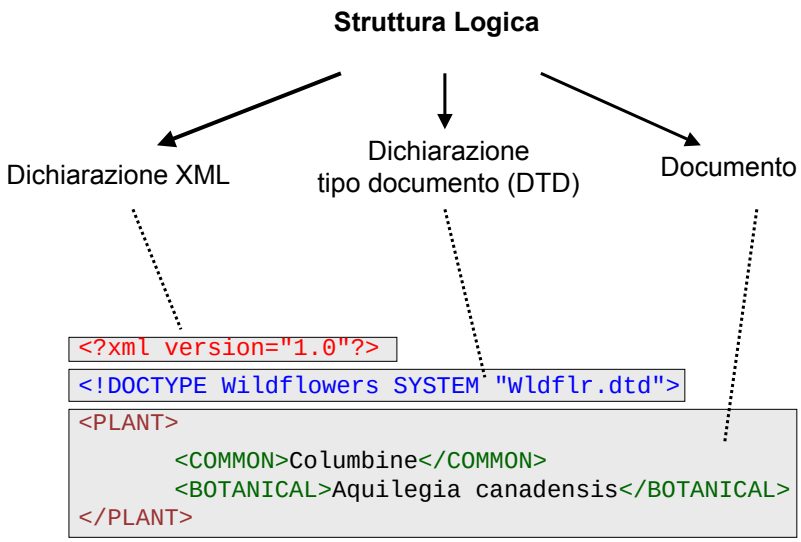
La Rappresentazione dei Dati



Struttura XML



Struttura XML



Struttura XML

Le entità in XML

- Rappresentano unità di memorizzazione
- Necessitano un nome univoco
- Esterne o interne
- Riferimenti ad una entità indicano di recuperarne il valore.

Struttura XML

Le entità in XML

Entità analizzabili

```
<!ENTITY LR1 "light requirement: mostly shade">
```

Entità NON analizzabili

```
<!ENTITY MyImage SYSTEM "Image001.gif" NDATA GIF>
<!NOTATION GIF SYSTEM "/Utils/Gifview.exe">
```

Entità predefinite

< < (parentesi angolare di apertura)

> > (parentesi angolare di chiusura)

&amp ; & (e commerciale)

' ‘ (apostrofo)

" " (virgolette doppie)

Entità interne e esterne

```
<!ENTITY LR1 "light requirement: mostly shade">
```

```
<!ENTITY MyImage SYSTEM "http://www.wildflowers.com/Image001.gif"
                        NDATA GIF>
```

Sintassi

Tutti i tag DEVONO essere chiusi

(HTML meno rigido al riguardo)

TAG di elemento Vuoto

```
<EMPTY></EMPTY>
```

```
<EMPTY/>
```


Sintassi

attributi

```
<A HREF = "http://www.microsoft.com">
microsoft Home Page
</A>
```

```
<?xml version="1.0"?>
<!DOCTYPE Wildflowers SYSTEM "Wldflr.dtd">

<PLANT ZONE=3>
      <COMMON>Columbine</COMMON>
      <BOTANICAL>Aquilegia canadensis</BOTANICAL>
</PLANT>
```

Sintassi

Documenti Validi e Ben Formati

Documento VALIDO

Deve rispettare le regole imposte dal DTD

Documento BEN FORMATO

- Tutti i tag di apertura e di chiusura corrispondono.
- I tag vuoti utilizzano una sintassi XML speciale.
- Tutti i valori degli attributi sono racchiusi tra virgolette.
- Tutte le entità sono dichiarate

Sintassi

Esempi

```
<list>
<item>Item 1</item>
<item>Item 2</item>
<item>Item 3</item>
</list>

Only one root element is
permitted:
???
<item>Item 1</item>
<item>Item 2</item>
<item>Item 3</item>
???
```

```
<list>
<item>Car</item>
<ITEM>Plane</ITEM>
<Item>Train</Item>
</list>


<list>
 <item>Car</itm>
<item>Plane</ITEM>
<item>Train</item>
<list>
```

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

```
<text>
  <bold><italic>XML</bold></italic>
</text>
```

Sintassi

Esempi

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

```
<!-- doc A -->
<example>
  <!-- <HEAD>  -->
  <!-- Characters <&<  -->
</example>
```

```
<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>
```


Sintassi

Esempio CDATA

La sezione CDATA è utilizzata per bypassare una porzione di testo contenete caratteri che potrebbero essere riconosciuti come markup.

```
<example>
  <![CDATA[ <aaa>bb&cc<<<]]>
</example>
```

Errori Tipici

- Dimenticare il tag di chiusura
- XML è case sensitive
- Blank nei nomi dei tag
- Dimenticare i quote negli attributi

Progettazione di una struttura dati XML

Attributi e elementi

Come decidere se un dato debba essere inserito come attributo di uno già esistente o come nuovo elemento?

```
<slide>
  <title>This is the title</title>
</slide>

<slide title="This is the title">...</slide>
```

Progettazione di una struttura dati XML

Attributi e elementi: scelte forzate

Nuovo elemento se:

Il dato è strutturato
linee multiple
cambia spesso

Attributo se:

Il dato è una stringa semplice (che non cambia spesso)

È un numero o appartiene a un set di possibilità

Progettazione di una struttura dati XML

Attributi e elementi: scelte stilistiche

Visibilità
elemento: dati che devono essere visualizzati (nome di un prodotto)
a*ttributo*: dati interni (codice a barre del prodotto)

Consumer / Provider
Ci si può chiedere chi fornisce / utilizza l'informazione in questione. Il nome di un prodotto è utilizzato dal cliente in un catalogo (elemento) mentre il codice a barre avrà scopi applicativi (attributo)

Container / Contents
Il contenuto di un contenitore (acqua, latte) → elemento
caratteristica del contenitore (blu, rosso, piccolo, grande) → attributo

Progettazione di una struttura dati XML

Normalizzazione

Processo di eliminazione delle ridondanze

HTML → tramite i Link → documento frammentato

XML → entities esterne → documento completo + riferimenti *&nome;*

Quando realizzare un file di entities esterno?

Ripetizione dello stesso oggetto

informazione che può cambiare. Usata in dversi posti (vedi costanti)

Riferimento allo stesso oggetto in diversi documenti

(medesimo discorso per gli stylesheets e DTDs)

Benefici nell'utilizzo di XML

Strutturato
Tramite editor è possibile definire sia i dati che le relazioni tra esse. Un parser XML è in grado di veridicare la correttezza di queste relazioni. Non è necessario scrivere codice aggiuntivo per validare i dati.

Si può rendere facilmente persistente.
Un file XML può essere salvato su file o su Db. Nel primo caso si dispone di un file di testo che può essere inviato su altre macchine , piattaforme o programmi.

Platform Independent
Vantaggi nell'utilizzo di file di testo per il salvataggio di informazioni XML:
- maggior facilità nella realizzazione dei parser su piattaforme differenti
- facilità l'interoperabilità
- programmi differenti su piattaforme differenti comunicano facilmente

Standard Aperto
Essendo la W3C la proprietaria dello standard, garantisce che nessun venditore sia in grado di causare problemi di interoperabilità. Mantenendo XML sia per i dati sia per i protocolli di comunicazione le industrie massimizzano il tempo di vita dei loro investimenti in prodotti e soluzioni.

Benefici nell'utilizzo di XML

DOM-SAX sono open e language-independent
La gestione di documenti XML a livello programmatico è reso semplice da una serie di interfaccie, indipendenti dal linguaggio, definite da W3C. SAX rappresenta API di basso livello, mentre DOM è di alto livello e mette a disposizione un modello ad oggetti per ogni documento XML.

XML è Web-Enabled
XML deriva da SGML come HTML. Questo significa che le infrastrutture disponibili oggi per comunicare con contenuti HTML possono essere riutilizzati per lavorare con XML.. Software e infrastrutture di rete presenti oggi possono essere utilizzate per inviare documenti XML.. Il fatto che non tutti i client non supportino XML a livello nativo viene superato server side tramite Java e Servlet.

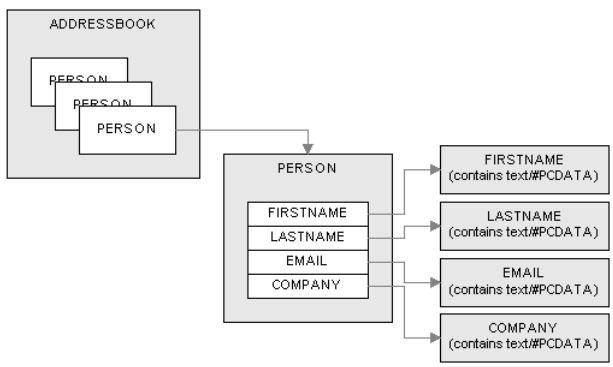
Totalmente Estensibile
Nessun tags è predefinito, questo significa avere il pieno controllo da parte degli sviluppatori dei propri dati.

DTD
L'utilizzo del DTD per definire la struttura di un documento rende agevole lo scambio di documenti XML tra sistemi differenti che condividano il medesimo DTD. Il DTD rappresenta un metodo per essere certi che due o più documenti XML siano dello stesso "tipo".

Interoperabilità
Tutti i vantaggi visti precedentemente rendono l'interoperabilità possibile. La capacità di poter condividere informazioni tra sistemi diffrerentgi rappresenta una delle caratteristiche principali di XML.

Definizione Tipo Documento (DTD)

```
<?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)> ]>
```



Definizione Tipo Documento (DTD)

Esempio 1

DTD

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

A document can contain only the root element **tutorial** which can contain some text

DOCUMENTO CORRETTO

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<tutorial>This is an XML document</tutorial>
```

DOCUMENTO ERRATO

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<text>This is an XML document</text>
```

Definizione Tipo Documento (DTD)

Esempio 2: DTD

```
<!ELEMENT XXX (AAA , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

The root element **XXX** must contain precisely one element **AAA** followed by one element **BBB**. Elements **AAA** and **BBB** can contain some text but no other elements

Definizione Tipo Documento (DTD)

Esempio 2: DOCUMENTO CORRETTO

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

        <XXX>
        <AAA>Start</AAA>
        <BBB>End</BBB>
        </XXX>


<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

        <XXX> <AAA/> <BBB/> </XXX>
```


Definizione Tipo Documento (DTD)

Esempio 2: DOCUMENTO ERRATO

Element **BBB** is missing:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <AAA/> ___ </XXX>
```

Element **BBB** must follow element **AAA**:

```
<XXX> <BBB/> <AAA/> </XXX>
```

Root element **XXX** can contain only one element **BBB**:

```
<XXX> <AAA/> <BBB/> <BBB/> </XXX>
```

Root element **XXX** must not contain any text.:

```
<XXX> Elements: <AAA/> <BBB/>  </XXX>
```

Definizione Tipo Documento (DTD)

Esempio 3

[*]

```
<!ELEMENT XXX (AAA* , BBB)>
        <!ELEMENT AAA (#PCDATA)>
        <!ELEMENT BBB (#PCDATA)>


The root element XXX can contain one or
several elements AAA followed by precisely
one element BBB. Element BBB must be always
present.
```


Definizione Tipo Documento (DTD)

Esempio 3: DOCUMENTO CORRETTO

A valid document:

```
        <!DOCTYPE tutorial SYSTEM "tutorial.dtd">

        <XXX> <AAA/> <BBB/> </XXX>
```

Another valid document. The element **AAA** is not mandatory.:

```
        <!DOCTYPE tutorial SYSTEM "tutorial.dtd">

        <XXX> <BBB/> </XXX>
```

Several **AAA** elements can occur inside the document:

```
        <!DOCTYPE tutorial SYSTEM "tutorial.dtd">

        <XXX> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <BB
B/> </XXX>
```

Definizione Tipo Documento (DTD)

Esempio 3: DOCUMENTO ERRATO

Element **BBB** is missing:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> ___ </XXX>
```

Element **BBB** must follow element **AAA**:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <BBB/> <AAA/> </XXX>
```

Element **AAA** must not follow element **BBB**:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX>
<AAA/> <AAA/> <AAA/> <AAA/> <BBB/> <AAA/> <AAA/>
</XXX>
```


Definizione Tipo Documento (DTD)

Esempio 4

[+]

The root element **XXX** must contain one or several elements **AAA** followed by precisely one element **BBB**. Element **BBB** must be always present.:

```
<!ELEMENT XXX (AAA+ , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

Definizione Tipo Documento (DTD)

Esempio 4: DOCUMENTO CORRETTO

A valid document:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <AAA/> <BBB/> </XXX>
```

Several **AAA** elements can occur inside the document:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/>
 <BBB/> </XXX>
```


Definizione Tipo Documento (DTD)

Esempio 4: DOCUMENTO ERRATO

Elements **AAA** and **BBB** are missing:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> ___ ___ </XXX>
```

at least one element **AAA** must be present.:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> ___<BBB/> </XXX>
```

Element **BBB** must follow element **AAA**:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <BBB/> <AAA/> </XXX>
```

Element **AAA** must not follow element **BBB**:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <AAA/> <AAA/> <AAA/> <AAA/> <BBB/> <AAA/> <AAA/> </XXX>
```

Definizione Tipo Documento (DTD)

Esempio 5

[?]

The root element **XXX** can contain one element **AAA** followed by precisely one element **BBB**. Element **BBB** must be always present.:

```
<!ELEMENT XXX (AAA? , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```


Definizione Tipo Documento (DTD)

Esempio 5: DOCUMENTO CORRETTO

A valid document:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <AAA/> <BBB/> </XXX>
```

Element **AAA** is not mandatory:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <BBB/> </XXX>
```

Definizione Tipo Documento (DTD)

Esempio 5: DOCUMENTO ERRATO

Element **BBB** is missing:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> ___ </XXX>
```

Maximaly one **AAA** element can occur inside the document:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <AAA/> <BBB/> </XXX>
```

Element **BBB** must follow element **AAA**:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX> <BBB/> <AAA/> </XXX>
```


Definizione Tipo Documento (DTD)

Esempio 6

[*,?,+]

The root element **XXX** can contain one element **AAA** followed by one or more elements **BBB**. Element **AAA** can contain one element **CCC** and several elements **DDD**. Element **BBB** must contain precisely one element **CCC** and one element **DDD**:

```
<!ELEMENT XXX (AAA? , BBB+)>
<!ELEMENT AAA (CCC? , DDD*)>
<!ELEMENT BBB (CCC , DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

Definizione Tipo Documento (DTD)

Esempio 6: DOCUMENTO CORRETTO

A valid document:

```
<XXX>
   <AAA>
      <CCC/><DDD/>
   </AAA>
   <BBB>
      <CCC/><DDD/>
   </BBB>
</XXX>
```

Elements in **AAA** are not mandatory:

```
<XXX>
   <AAA/>
   <BBB>
      <CCC/><DDD/>
   </BBB>
</XXX>
```


Definizione Tipo Documento (DTD)

Esempio 6: DOCUMENTO ERRATO

Element **BBB** must contain elements **CCC** and **DDD**:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX>
   <AAA/>
   <BBB/>
</XXX>
```

Element **AAA** can contain maximaly one element **CCC**:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX>
   <AAA>
      <CCC/><CCC/>
      <DDD/><DDD/>
   </AAA>
   <BBB>
      <CCC/><DDD/>
   </BBB>
</XXX>
```

Definizione Tipo Documento (DTD)

Esempio 7

[|]

The root element **XXX** must contain one element **AAA** followed by one element **BBB**. Element **AAA** must contain one element **CCC** followed by element **DDD**. Element **BBB** must contain either one element **CCC** or one element **DDD**:

```
<!ELEMENT XXX (AAA , BBB)>
<!ELEMENT AAA (CCC , DDD)>
<!ELEMENT BBB (CCC | DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```


Definizione Tipo Documento (DTD)

Esempio 7: DOCUMENTO CORRETTO

A valid document:

```
<XXX>
   <AAA>
      <CCC/> <DDD/>
   </AAA>
   <BBB>
      <CCC/>
   </BBB>
</XXX>
```

Another valid document:

```
<XXX>
   <AAA>
      <CCC/> <DDD/>
   </AAA>
   <BBB>
      <DDD/>
   </BBB>
</XXX>
```

Definizione Tipo Documento (DTD)

Esempio 6: DOCUMENTO ERRATO

The element **BBB** can contain either element **CCC** or **DDD** but not both:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX>
   <AAA>
      <CCC/> <DDD/>
   </AAA>
   <BBB>
      <CCC/> <DDD/>
   </BBB>
</XXX>
```

The element **BBB** can contain either element **CCC** or **DDD** but not both:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<XXX>
   <AAA>
      <CCC/> <DDD/>
   </AAA>
   <BBB>
      <DDD/> <CCC/>
   </BBB>
</XXX>
```


Definizione Tipo Documento (DTD)

Esempio 8
[ATTLIST]

An attribute of CDATA type can contain any character if it conforms to well formedness constraints. The **required** attribute must be always present, the **implied** attribute is optional:

```
<!ELEMENT attributes (#PCDATA)>
<!ATTLIST attributes
      aaa CDATA #REQUIRED
      bbb CDATA #IMPLIED>
```

Definizione Tipo Documento (DTD)

Esempio 8: DOCUMENTO CORRETTO

CDATA attribute can contain any character conforming to well-formedness constraints:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<attributes aaa="#d1" bbb="*~*">
   Text
</attributes>
```

The order of attributes is not important:

```
<attributes bbb="$25" aaa="13%">
   Text
</attributes>
```

The **bbb** attribute can be omitted as it is implied:

```
<attributes aaa="#d1" />
```


Definizione Tipo Documento (DTD)

Esempio 8: DOCUMENTO ERRATO

The **aaa** attribute is required. Therefore it must be always present.:

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">

<attributes ___ bbb="X24"/>
```

Definizione Tipo Documento (DTD)

Esempio 9
[NMTOKEN, NMTOKENS]

The attributes **bbb** and **ccc** must be always present, the attribute **aaa** is optional:

```
<!ELEMENT attributes (#PCDATA)>
<!ATTLIST attributes
      aaa CDATA #IMPLIED
      bbb NMTOKEN #REQUIRED
      ccc NMTOKENS #REQUIRED>
```


Definizione Tipo Documento (DTD)

Esempio 9: DOCUMENTO CORRETTO

The attributes **id**, **code** and **X** uniquely determine their element:

```
<!ELEMENT XXX (AAA+ , BBB+ , CCC+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ELEMENT CCC (#PCDATA)>
<!ATTLIST AAA
      id ID #REQUIRED>
<!ATTLIST BBB
      code ID #IMPLIED
      list NMTOKEN #IMPLIED>
<!ATTLIST CCC
      X ID #REQUIRED
      Y NMTOKEN #IMPLIED>
```

Definizione Tipo Documento (DTD)

Esempio 9: DOCUMENTO ERRATO

The ID attribute must not start with a number or contain a character not permitted in NMTOKEN:

```
<XXX>
   <AAA id="L12"/>
   <BBB code="#QW" list="L12"/>
   <CCC X="12" Y="QW" />
</XXX>
```

The ID attribute must have a unique value:

```
<XXX>
   <AAA id="L12"/>
   <BBB code="QW" list="L12"/>
   <CCC X="ZA" Y="QW" />
   <CCC X="ZA" Y="QW" />
</XXX>
```

The ID attribute must have a unique value. Both **id** and **X** are of type ID:

```
<XXX>
   <AAA id="L12"/>
   <BBB code="QW" list="L12"/>
   <CCC X="L12" Y="QW" />
</XXX>
```

Definizione Tipo Documento (DTD)

Esempio 10
[ID, IDREF, IDREFS]

The attributes **id** and **mark** uniquely determine their element. The attributes **ref** refer to these elements:

```
<!ELEMENT XXX (AAA+ , BBB+, CCC+, DDD+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
<!ATTLIST AAA
      mark ID #REQUIRED>
<!ATTLIST BBB
      id ID #REQUIRED>
<!ATTLIST CCC
      ref IDREF #REQUIRED>
<!ATTLIST DDD
      ref IDREFS #REQUIRED>
```

Definizione Tipo Documento (DTD)

All ID values are unique and all IDREF and IDREFS point to elements with relevant IDs:

```
<XXX>
   <AAA mark="a1"/>
   <AAA mark="a2"/>
   <AAA mark="a3"/>
   <BBB id="b001" />
   <CCC ref="a3" />
   <DDD ref="a1 b001 a2" />
</XXX>
```

There are no ID attributes with value a3 or b001:

```
<XXX>
   <AAA mark="a1"/>
   <AAA mark="a2"/>
   <BBB id="b01" />
   <CCC ref="a3" />
   <DDD ref="a1 b001 a2" />
</XXX>
```


Definizione Tipo Documento (DTD)

Esempio 11

This DTD precisely states permitted values:

```
<!ELEMENT XXX (AAA+, BBB+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ATTLIST AAA
          true ( yes | no ) #REQUIRED>
<!ATTLIST BBB
   month (1|2|3|4|5|6|7|8|9|10|11|12) #IMPLIED>
```

Definizione Tipo Documento (DTD)

Esempio 12

Both attributes are implied. Their default value is given.:

```
<!ELEMENT XXX (AAA+, BBB+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ATTLIST AAA
          true ( yes | no ) "yes">
<!ATTLIST BBB
   month NMTOKEN "1">
```


Definizione Tipo Documento (DTD)

Esempio 13
[EMPTY]

The **AAA** elements can contain only attributes but no text:

```
<!ELEMENT XXX (AAA+)>
<!ELEMENT AAA EMPTY>
<!ATTLIST AAA
      true ( yes | no ) "yes">
```

Definizione Tipo Documento (DTD)

Le entità vanno definite nel DTD

```
<!ENTITY EntityName EntityDefinition>
```

Entità

- interne
- esterne

```
<!ENTITY SIGNATURE "Bill">
&SIGNATURE;
```

```
<!ENTITY IMAGE1 SYSTEM "Xmlquot.gif" NDATA GIF>
```

```
<!ENTITY IMAGE1 SYSTEM “http://XMLCo.com/Images/Xmlquot.gif” NDATA GIF>
```

```
<!ENTITY IMAGE1 PUBLIC "-//XMLCo//TEXT Standard Images//EN" "http://XMLCo.com/Images/Xmlquot.gif" NDATA GIF>
```


Definizione Tipo Documento (DTD)

annotazioni (NOTATION)
per entità non analizzabili (NDATA)

```
<!ENTITY IMAGE1 SYSTEM "Xmlquot.gif" NDATA GIF>
```

Un riferimento a tale enitià produrrebbe un errore:
Declaration ‘IMAGE1’ contains reference to undefined notation ‘GIF’.

```
<!NOTATION GIF SYSTEM "Iexplore.exe">
```

Definizione Tipo Documento (DTD)

Entità di parametro

```
<!ENTITY % ENCRYPTION "40bit CDATA #IMPLIED 128bit CDATA #IMPLIED">
```

```
<!ELEMENT EMAIL (TO+, FROM, CC*, BCC*, SUBJECT?, BODY?)>
<!ATTLIST EMAIL LANGUAGE(Western|Greek|Latin|Universal) "Western"
ENCRYPTED %ENCRYPTION;
PRIORITY (NORMAL|LOW|HIGH) "NORMAL">
```


Definizione Tipo Documento (DTD)

Istruzioni di elaborazione

```
<?AVI CODEC="VIDEO1" COLORS="256"?>
<?WAV COMPRESSOR="ADPCM" BITS="8" RESOLUTION="16"?>
```

Definizione Tipo Documento (DTD)

namespace

spazio dei nomi: raccolta di nomi identificata da un URI e può essere qualificato o non qualificato.

```
<?xml version="1.0"?>
<?xml:namespace ns=http://inventory/schema/ns prefix="inv"?>
<?xml:namespace ns=http://wildflowers/schema/ns prefix="wf"?>
<PRODUCT>
  <PNAME>Test1</PNAME>
  <inv:quantity>1</inv:quantity>
  <wf:price>323</wf:price>
  <DATE>6/1</DATE>
</PRODUCT>
```

Nomi qualificati

```
<CATALOG>
<INDEX>
<ITEM>Trees</ITEM>
</INDEX>
<PRODUCT xmlns:wf="urn:shemas-wildflowers-com”>
<NAME>Bloodroot</NAME>
<QUANTITY>10</QUNTITY>
</PRODUCT>
</CATALOG>
```

Namespace predefinito

Definizione Tipo Documento (DTD)

DICHIARAZIONE DELLO SPAZIO DEI NOMI COME URL O URN

```
<wf:product xmlns:wf="urn:shemas-wildflowers-com”>
<wf:name>Bloodroot>/wf:name>
<QUANTITY>10</QUNTITY> <PRICE>$2.44</PRICE>
</wf:product>
```

XML Schema

Limitazione dei DTD

i DTD nascono con lo scopo di trattare una tipologia ben spefifica di documenti come libri, brochures, manuali, pagine web. E' facile infatti imporre regole che definiscano ad esempio che in un libro esistano uno o più autori, oppure che una canzone abbia un solo titolo. I DTD dimostrano la loro inefficienza in tutte quelle applicazioni di XML che si differenziano da questi ambienti (scambio di dati computer-to-computer).

I DTD non dispongono di Tipi di dato

I DTD non permettono di definire come debba essere composto il contenuto di un elemento

I DTD non hanno una sintassi XML

Non è possibile definire il numero di figli di un nodo senza imporne l'ordine

XML Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="GREETINGS" type="xsd:string"/>
</xsd:schema>
```

L'elemento di root di ogni schem è rappresentato da *schema* che deve essere all'interno del namespace specificato. Il prefisso xsd può essere cambiato se rimane lo stesso URI.

```
<?xml version="1.0" encoding="UTF-8"?>
<GREETINGS
 xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance
 xsi:noNamespaceSchemaLocation="C:\unife\greetings.xsd">HELLO!
</GREETINGS>
```

XML VALIDO

<GREETING> various random text but no markup</GREETING>

<GREETING>Hello!</GREETING>

<GREETING></GREETING>

XML NON VALIDO

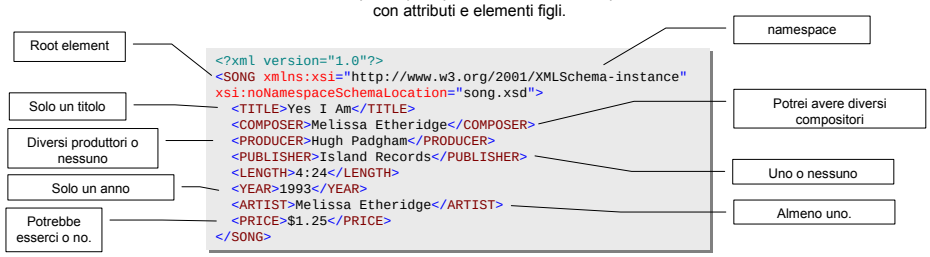
<GREETING> <SOME_TAG>various random text</SOME_TAG>
<SOME_EMPTY_TAG/> </GREETING>

<GREETING> <GREETING>various random text</GREETING> </GREETING>

XML Schema

Complex Types

A differenza dei tipi semplici, permettono di definire tipi di elementi con attributi e elementi figli.




```
<?xml version="1.0"?>
<SONG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="song.xsd">
  <TITLE>Yes I Am</TITLE>
  <COMPOSER>Melissa Etheridge</COMPOSER>
  <PRODUCER>Hugh Padgham</PRODUCER>
  <PUBLISHER>Island Records</PUBLISHER>
  <LENGTH>4:24</LENGTH>
  <YEAR>1993</YEAR>
  <ARTIST>Melissa Etheridge</ARTIST>
  <PRICE>$1.25</PRICE>
</SONG>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="SongType">
  <xsd:sequence>
    <xsd:element name="TITLE" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="COMPOSER" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element name="PRODUCER" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="PUBLISHER" type="xsd:string" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="LENGTH" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="YEAR" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="ARTIST" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element name="PRICE" type="xsd:string" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```


XML Schema

Complex Types

Aumentiamo la profondità della struttura

```
<?xml version="1.0"?>
<SONG xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="24-
10.xsd">
  <TITLE>Hot Cop</TITLE>
  <COMPOSER><NAME>Jacques Morali</NAME></COMPOSER>
  <COMPOSER><NAME>Henri Belolo</NAME></COMPOSER>
  <COMPOSER><NAME>Victor Willis</NAME></COMPOSER>
  <PRODUCER><NAME>Jacques Morali</NAME></PRODUCER>
  <PUBLISHER>PolyGram Records</PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="ComposerType">
    <xsd:sequence>
      <xsd:element name="NAME" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ProducerType">
    <xsd:sequence>
      <xsd:element name="NAME" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE"     type="xsd:string"/>
      <xsd:element name="COMPOSER"  type="ComposerType"  maxOccurs="unbounded"/>
      <xsd:element name="PRODUCER"  type="ProducerType" minOccurs="0" MaxOccurs="unbounded"/>
      <xsd:element name="PUBLISHER" type="xsd:string" minOccurs="0"/>
      <xsd:element name="LENGTH" type="xsd:string"/>
      <xsd:element name="YEAR"   type="xsd:string"/>
      <xsd:element name="ARTIST" type="xsd:string"  maxOccurs="unbounded"/>
      <xsd:element name="PRICE" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

XML Schema

Nell'esempio precedente i due tipi definiti sono identici. E' possible creare un solo tipo e condividerlo.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="NAME" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE"     type="xsd:string"/>
      <xsd:element name="COMPOSER"  type="PersonType"  maxOccurs="unbounded"/>
      <xsd:element name="PRODUCER"  type="PersonType"  minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="PUBLISHER" type="xsd:string" minOccurs="0"/>
      <xsd:element name="LENGTH" type="xsd:string"/>
      <xsd:element name="YEAR"   type="xsd:string"/>
      <xsd:element name="ARTIST" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="PRICE" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```


XML Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="SONG" type="SongType"/>

 <xsd:complexType name="NameType">
  <xsd:sequence>
     <xsd:element name="GIVEN"  type="xsd:string"/>
      <xsd:element name="FAMILY" type="xsd:string"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="PersonType">
   <xsd:sequence>
     <xsd:element name="NAME" type="NameType"/>
   </xsd:sequence>
 </xsd:complexType>


 <xsd:complexType name="SongType">
   <xsd:sequence>
     …
   </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

In questo modo posso utilizzare lo stesso elemento in posizioni diverse con tipi diversi. Per esempio è possibile che NAME di PERSON contenga i figli GIVEN e FAMILY, mentre NAME di MOVIE contenga una stringa.

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="NAME">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="GIVEN"
type="xsd:string"/>
            <xsd:element name="FAMILY"
type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

XML Schema

XML Schema mette a disposizione tre construtti che permettono di specificare come e se l'ordine degli elementi è importante

```
<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="NAME">
      <xsd:complexType>
        <xsd:all>
          <xsd:element name="GIVEN" type="xsd:string"
                       minOccurs="1" maxOccurs="1"/>
          <xsd:element name="FAMILY" type="xsd:string"
                       minOccurs="1" maxOccurs="1"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

xsd:all
Specifica che tutti gli elementi devono essere presenti in qualunque ordine.

```
<xsd:complexType name="SongType">
  <xsd:sequence>
    <xsd:element name="TITLE" type="xsd:string"/>
    <xsd:choice>
      <xsd:element name="COMPOSER" type="PersonType"/>
      <xsd:element name="PRODUCER" type="PersonType"/>
    </xsd:choice>
    <xsd:element name="PUBLISHER" type="xsd:string"
                 minOccurs="0"/>
    <xsd:element name="LENGTH" type="xsd:string"/>
    <xsd:element name="YEAR"   type="xsd:string"/>
    <xsd:element name="ARTIST" type="xsd:string"
                 maxOccurs="unbounded"/>
    <xsd:element name="PRICE" type="xsd:string"
minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

xsd:choice
Corrisponde ad un OR logico. Nell'esempio signifca che devono essere presenti o l'elemento COMPOSER o PRODUCER ma non contemporanemante.

xsd:sequence
indica l'ordine nel quale devono comparire gli elementi. Il numero di volte che questi si possono ripetere viene controllato con gli attributi minOccours e maxOccours.

XML Schema

Simple Types

Numeric data types

Name:	Type:	Examples:
xsd:float	IEEE 754 32-bit floating point number, or as close as you can get using a base 10 representation; same as Java's float type	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
xsd:double	IEEE 754 64-bit floating point number, or as close as you can get using a base 10 representation; same as Java's double type	-INF, 1.401E-90, -1E4, -0, 0, 12.78E-2, 12, INF, NaN, 3.4E42
xsd:decimal	Arbitrary precision, decimal numbers; same as java.math.BigDecimal	-2.7E400, 5.7E-444, -3.1415292, 0, 7.8, 90200.76, 3.4E1024
xsd:integer	An arbitrarily large or small integer; same as java.math.BigInteger	-5000000000000000000000000, -9223372036854775809, -126789, -1, 0, 1, 5, 23, 42, 126789, 9223372036854775808, 456734987324983264987624958
xsd:nonPositiveInteger	An integer less than or equal to zero	0, -1, -2, -3, -4, -5, -6, -7, -8, -9, . . .
xsd:negativeInteger	An integer strictly less than zero	-1, -2, -3, -4, -5, -6, -7, -8, -9, . . .
xsd:long	An eight-byte two's complement integer such as Java's long type	-9223372036854775808, -9223372036854775807, . . . -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . ., 2147483645, 2147483646, 2147483647, 2147483648, . . . 9223372036854775806, 9223372036854775807
xsd:int	An integer that can be represented as a four-byte, two's complement number such as Java's int type	-2147483648, -2147483647, -2147483646, 2147483645, . . . -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . ., 2147483645, 2147483646, 2147483647
xsd:short	An integer that can be represented as a two-byte, two's complement number such as Java's short type	-32768, -32767, -32766, . . ., -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, . . . 32765, 32766, 32767
xsd:byte	An integer that can be represented as a one-byte, two's complement number such as Java's byte type	-128, -127, -126, -125, . . ., -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, . . . 121, 122, 123, 124, 125, 126, 127
xsd:nonNegativeInteger	An integer greater than or equal to zero	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, . . ., . . .
xsd:unsignedLong	An eight-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, . . . 18446744073709551614, 18446744073709551615
xsd:unsignedInt	A four-byte unsigned integer	0, 1, 2, 3, 4, 5, . . . 4294967294, 4294967295
xsd:unsignedShort	A two-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . . 65533, 65534, 65535
xsd:unsignedByte	A one-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . . 252, 253, 254, 255
xsd:positiveInteger	An integer strictly greater than zero	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . .

XML Schema

Simple Types

Time data types

Name:	Type:	Examples:
xsd:float	IEEE 754 32-bit floating point number, or as close as you can get using a base 10 representation; same as Java's float type	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
xsd:double	IEEE 754 64-bit floating point number, or as close as you can get using a base 10 representation; same as Java's double type	-INF, 1.401E-90, -1E4, -0, 0, 12.78E-2, 12, INF, NaN, 3.4E42
xsd:decimal	Arbitrary precision, decimal numbers; same as java.math.BigDecimal	-2.7E400, 5.7E-444, -3.1415292, 0, 7.8, 90200.76, 3.4E1024
xsd:integer	An arbitrarily large or small integer; same as java.math.BigInteger	-5000000000000000000000000, -922337203685477580 9, -126789, -1, 0, 1, 5, 23, 42, 126789, 922337203685477580 8, 45673498732498326498 73624958
xsd:nonPositiveInteger	An integer less than or equal to zero	0, -1, -2, -3, -4, -5, -6, -7, -8, -9, . . .
xsd:negativeInteger	An integer strictly less than zero	-1, -2, -3, -4, -5, -6, -7, -8, -9, . . .
xsd:long	An eight-byte two's complement integer such as Java's long type	-9223372036854775808, -9223372036854775807, . . . -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . ., 2147483645, 2147483646, 2147483647, 2147483648, . . .9223372036854775806, 9223372036854775807
xsd:int	An integer that can be represented as a four-byte, two's complement number such as Java's int type	-2147483648, -2147483647, -2147483646, 2147483645, . . . -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . ., 2147483645, 2147483646, 2147483647
xsd:short	An integer that can be represented as a two-byte, two's complement number such as Java's short type	-32768, -32767, -32766, . . ., -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, . . . 32765, 32766, 32767
xsd:byte	An integer that can be represented as a one-byte, two's complement number such as Java's byte type	-128, -127, -126, -125, . . ., -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, . . .121, 122, 123, 124, 125, 126, 127
xsd:nonNegativeInteger	An integer greater than or equal to zero	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, . . ., . .
xsd:unsignedLong	An eight-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, . . .18446744073709551614, 18446744073709551615
xsd:unsignedInt	A four-byte unsigned integer	0, 1, 2, 3, 4, 5, . . .4294967294, 4294967295
xsd:unsignedShort	A two-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . .65533, 65534, 65535
xsd:unsignedByte	A one-byte unsigned integer	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . .252, 253, 254, 255
xsd:positiveInteger	An integer strictly greater than zero	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, . . .

XML Schema

Simple Types

XML data types

Name:	Type:	Examples:
xsd:ID	XML 1.0 ID attribute type; any XML name that's unique among ID type attributes and elements	p1, p2, ss124-45-6789, _92, red, green, NT-Decl, seventeen
xsd:IDREF	XML 1.0 IDREF attribute type; any XML name that's used as the value of an ID type attribute or element elsewhere in the document	p1, p2, ss124-45-6789, _92, p1, p2, red, green, NT-Decl, seventeen
xsd:ENTITY	XML 1.0 ENTITY attribute type; any XML name that's declared as an unparsed entity in the DTD	PIC1, PIC2, PIC3, cow_movie, MonaLisa, Warhol
xsd:NOTATION	XML 1.0 NOTATION attribute type; any XML name that's declared as a notation name in the schema using xsd:notation	GIF, jpeg, TIF, pdf, TeX
xsd:IDREFS	XML 1.0 IDREFS attribute type; a white space-separated list of XML names that are used as values of ID type attributes or elements elsewhere in the document	p1 p2, ss124-45-6789 _92, red green NT-Decl seventeen
xsd:ENTITIES	XML 1.0 ENTITIES attribute type; a white space-separated list of ENTITY names	PIC1 PIC2 PIC3
xsd:NMTOKEN	XML 1.0 NMTOKEN attribute type	12 are you ready 199
xsd:NMTOKENS	XML 1.0 NMTOKENS attribute type, a white space-separated list of name tokens	MI NY LA CA p1 p2 p3 p4 p5 p6 1 2 3 4 5 6
xsd:language	Valid values for xml:lang as defined in XML 1.0	en, en-GB, en-US, fr, i-lux, ama, ara, ara-EG, x-choctaw
xsd:Name	An XML 1.0 Name, with or without colons	set, title, rdf, math, math123, xlink:href, song:title
xsd:QName	a prefixed name	song:title, math:set, xsd:element
xsd:NCName	a local name without any colons	set, title, rdf, math, tei.2, href

XML Schema

Simple Types

String data types

Name:	Type:	Examples:
xsd:string	A sequence of zero or more Unicode characters that are allowed in an XML document; essentially the only forbidden characters are most of the C0 controls, surrogates, and the byte-order mark	p1, p2, 123 45 6789, ^*&^*&_92, red green blue, NT-Decl, seventeen; Mary had a little lamb, The love of money is the root of all Evil., Would you paint the lily? Would you gild gold?
xsd:normalizedString	A string that does not contain any tabs, carriage returns, or linefeeds	PIC1, PIC2, PIC3, cow_movie, MonaLisa, Hello World , Warhol, red green
xsd:token	A string with no leading or trailing white space, no tabs, no linefeeds, and not more than one consecutive space	p1 p2, ss123 45 6789, _92, red, green, NT Decl, seventeenp1, p2, 123 45 6789, ^*&^*&_92, red green blue, NT-Decl, seventeen; Mary had a little lamb, The love of money is the root of all Evil.

XML Schema

Derivazione di tipi semplici

A partire dai tipi semplici visti possono essere derivati altri tipi di dato.

- **xsd:restriction** per selezionare un subset dei valori ammessi dal tipo base
- **xsd:union** combinazione di tipi
- **xsd:list** lista di elementi di un tipo semplice di base

esempio

```
<xsd:simpleType name="phonoYear">
  <xsd:restriction base="xsd:gYear">
    <xsd:minInclusive value="1877"/>
    <xsd:maxInclusive value="2100"/>
  </xsd:restriction>
</xsd:simpleType>
```

FACETS

xsd:minExclusive: valore minimo del quale devono essere strettamente maggiori (>).

xsd:minInclusive: valore minimo del quale devono essere maggiori o uguali (>=).

xsd:maxInclusive: valore massimo del quale devono essere minori o uguali (<=).

xsd:maxExclusive: valore massimo del quale devono essere stretamente minori (<).

xsd:enumeration: lista di valori

xsd:whiteSpace: come vengono trattati i whitespace

xsd:pattern: expressione di confronto

xsd:length: num di caratteri ammessi

xsd:minLength: numero min di caratteri

xsd:maxLength: max num di caratteri

xsd:totalDigits: max numero di cifre

xsd:fractionDigits: max numero di decimali

XML Schema

xsd:union permette di derivare un tipo semplice dall'unione di di altri tipi semplici: nell'esempio il tipo MoneyOrDecimal può contenere o un numerico decimale o un tipo "valuta" definito tramite un pattern.

```
<xsd:simpleType name="MoneyOrDecimal">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

xsd:list permette di derivare un tipo semplice come lista di valori. Nell'esempio il tipo YearList può contenere una lista di anni tipo:<YEAR>1992 1997 1980 1971</YEAR>.

E' possibile restringere il numero degli elementi ammessi nella lista.

```
<xsd:simpleType name="YearList">
  <xsd:list itemType="xsd:gYear"/>
</xsd:simpleType>

<xsd:simpleType name="DoubleYear">
  <xsd:restriction base="YearList">
    <xsd:length value="2"/>
  </xsd:restriction>
</xsd:simpleType>
```


XML Schema

Anche con gli *schema* è possibile definire gli atrributi. Rispetto ai DTD possiamo utilizzare anche per essi i vari data types.

```
<xsd:complexType name="PhotoType">
   <xsd:attribute name="SRC"    type="xsd:anyURI"/>
   <xsd:attribute name="WIDTH"  type="xsd:positiveInteger"/>
   <xsd:attribute name="HEIGHT" type="xsd:positiveInteger"/>
   <xsd:attribute name="ALT"    type="xsd:string"/>
 </xsd:complexType>
```

```
<TITLE ID="test">Yes I Am</TITLE>
```

```
<xsd:complexType name="StringWithID">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="ID" type="xsd:ID" use="required" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:element name="TITLE" type="StringWithID"/>
```

XSL

XSL = XSLT + FO

XSLT
linguaggio di
TRASFORMAZIONE

FO
linguaggio di
FORMATTAZIONE

XSL

Ogni XML WF è un ALBERO (tree)

Albero: struttura composta da nodi connessi il cui nodo principale è chiamato ROOT. Nodi che non hanno figli sono chiamati FOGLIE.

Per gli scopi di XSLT tutto viene trasformato in un NODO

(elementi, attributi, namespaces, pi, commenti)

Un processore XSLT modella un documento XML come albero contenente 7 tipi di nodi:

- Root
- Elements
- Text
- Attributes
- Namespaces
- PI
- Commenti

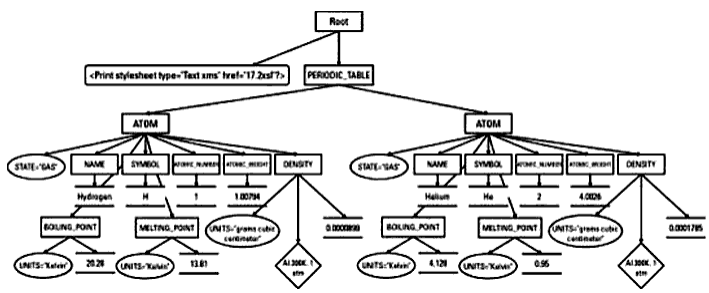
XSL

Esempio: tavola periodica

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xml" href="esempio.xsl"?>
<PERIODIC_TABLE>
  <ATOM STATE="GAS">
    <NAME>Hydrogen</NAME>
    <SYMBOL>H</SYMBOL>
    <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
    <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
    <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter">
    <!-- At 300K, 1 atm --> 0.0000899
    </DENSITY>
  </ATOM>

  <ATOM STATE="GAS">
    <NAME>Helium</NAME>
    <SYMBOL>He</SYMBOL>
    <ATOMIC_NUMBER>2</ATOMIC_NUMBER>
    <ATOMIC_WEIGHT>4.0026</ATOMIC_WEIGHT>
    <BOILING_POINT UNITS="Kelvin">4.216</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">0.95</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->
    0.0001785
    </DENSITY>
  </ATOM>
</PERIODIC_TABLE>
```


XSL

Rappresentazione interna



XSLT style sheet documents

I documenti XSLT sono costituiti da *TEMPLATE*

Ogni template ha un *PATTERN* che indica con quale nodo del documento XML debba fare matching

Il processore XSLT per ogni nodo verifica i pattern ed eventualmente esegue l'output del template corrispondente.

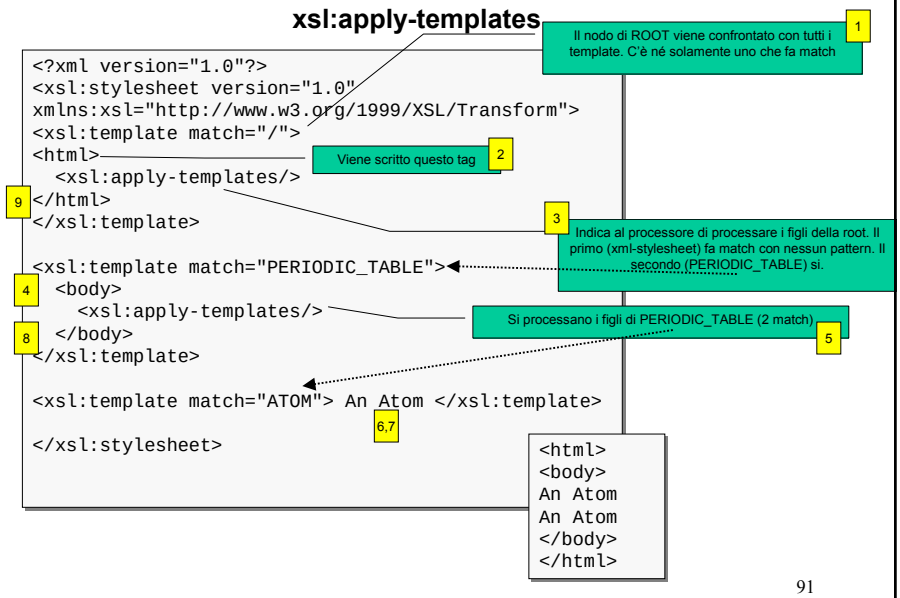
Stylesheet esempio: TEMPLATE

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transf
orm">

<xsl:template match="PERIODIC_TABLE">
<html>
<xsl:apply-templates/>
</html>
</xsl:template>

<xsl:template match="ATOM">
<P> <xsl:apply-templates/> </P>
</xsl:template>

</xsl:stylesheet>
```

```
<html>
<P>
Hydrogen H 1 1.00794 20.28 13.81 0.0000899 </P>
<P> Helium He 2 4.0026 4.216 0.95 0.0001785 </P>
</html>
```

XSL

xsl:apply-templates




```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
  <xsl:apply-templates/>
</html>
</xsl:template>

<xsl:template match="PERIODIC_TABLE">
  <body>
    <xsl:apply-templates/>
  </body>
</xsl:template>

<xsl:template match="ATOM"> An Atom </xsl:template>

</xsl:stylesheet>
```

```
<html>
<body>
An Atom
An Atom
</body>
</html>
```


XSL

xsl:apply-templates

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
  <xsl:apply-templates/>
/html>
</xsl:template>

<xsl:template match="PERIODIC_TABLE">
  <body>
    <xsl:apply-templates/>
  </body>
</xsl:template>

<xsl:template match="ATOM">
<xsl:apply-templates select="NAME"/>
</xsl:template>

</xsl:stylesheet>
```

```
<html>
<body>
Hydrogen
Helium
</body>
</html>
```

XSL

xsl:value-of

Computa il valore dell'attributo *SELECT* e ne restituisce l'output

```
<xsl:template match="ATOM">
  <xsl:value-of select="NAME"/>
</xsl:template>
```

Locale all'atomo processato

Il valore di un nodo dipende dal tipo di elemento che si sta processando

XSL

xsl:value-of

Node Type:	**Value:**
Root	The value of the root element
Element	The concatenation of all parsed character data contained in the element, including character data in any of the descendants of the element
Text	The text of the node; essentially the node itself
Attribute	The normalized attribute value as specified by Section 3.3.3 of the XML 1.0 recommendation; basically the attribute value after entities are resolved and leading and trailing white space is stripped; does not include the name of the attribute, the equals sign, or the quotation marks
Namespace	The URI of the namespace
Processing instruction	The data in the processing instruction; does not include the processing instruction , <? or ?>
Comment	The text of the comment, <!-- and --> not included

XSL

xsl:for-each

Per processare elementi multipli possiamo:

```
<xsl:template match="PERIODIC_TABLE">
<xsl:apply-templates select="ATOM"/>
</xsl:template>
<xsl:template match="ATOM">
<xsl:value-of select="."/>
</xsl:template>
```

Oppure:

```
<xsl:template match="PERIODIC_TABLE">
<xsl:for-each select="ATOM">
<xsl:value-of select="."/>
</xsl:for-each>
</xsl:template>
```


XSL

xsl:if, xsl:choose

```
<xsl:template match="ATOM">
<xsl:value-of select="NAME"/>
<xsl:if test="position()!=last()">, </xsl:if>
</xsl:template>
```

```
<xsl:template match="ATOM">
<xsl:choose>
<xsl:when test="@STATE='SOLID'">
<P style="color: black"><xsl:value-of select="."/></P>
</xsl:when>
<xsl:when test="@STATE='LIQUID'">
<P style="color: blue"> <xsl:value-of select="."/> </P>
</xsl:when>

<xsl:when test="@STATE='GAS'">
<P style="color: red"> <xsl:value-of select="."/> </P>
</xsl:when>

<xsl:otherwise>
<P style="color: green"> <xsl:value-of select="."/> </P>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
```

XSL

xsl:import, xsl:include

```
<xsl:stylesheet version="1.0”
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="genealogy.xsl"/>
<xsl:import href="standards.xsl"/>
<!-- other child elements follow -->
</xsl:stylesheet>
```

xsl:import deve apparire come primo elemento del documento.
Href indica cosa importare.

Le regole del documento sono prioriatrie rispetto alle regole importate.

apply-imports ←→ apply-templates

xsl:include copia il contenuto dello stylesheet remoto

XPATH

Sintassi che permette di esprimere esattamente quale nodo si vuole selezionare.

Da utilizzare nell’attributo *select* di

```
xsl:apply-templates
    xsl:value-of
    xsl:for-each
     xsl:copy-of
       xsl:sort
```

XPATH

Esempi /,//,*

/AAA/CCC

Select all elements CCC which are children of the root element AAA

```
<AAA>
  <BBB />
  <CCC />
  <BBB />
  <BBB />
  <DDD>
    <BBB/>
  </DDD>
  <CCC />
</AAA>
```

/AAA/DDD/BBB

Select all elements BBB which are children of DDD which are children of the root element AAA

```
<AAA>
  <BBB />
  <CCC />
  <BBB />
  <BBB />
  <DDD>
    <BBB/>
  </DDD>
  <CCC />
</AAA>
```

//DDD/BBB

Select all elements BBB which are children of DDD

```
<AAA>
  <BBB />
  <CCC />
  <BBB />
  <DDD>
    <BBB />
  </DDD>
  <CCC>
    <DDD>
      <BBB />
      <BBB />
    </DDD>
  </CCC>
</AAA>
```

/AAA/CCC/DDD/*

Select all elements CCC which are children of the root element AAA

```
<AAA>
  <XXX>
    <DDD>
      <BBB />
      <BBB />
      <EEE />
      <FFF/>
    </DDD>
  </XXX>
  <CCC>
    <DDD>
      <BBB />
      <BBB />
      <EEE />
      <FFF/>
    </DDD>
  </CCC>
  <CCC>
    <BBB>
      <BBB />
      <BBB />
    </BBB>
  </CCC>
</AAA>
```

/*/*/BBB

Select all elements BBB which have 3 ancestors

```
<AAA>
  <XXX>
    <DDD>
      <BBB/>
      <BBB/>
      <EEE />
      <FFF />
    </DDD>
  </XXX>
  <CCC>
    <DDD>
      <BBB/>
      <BBB/>
      <EEE />
      <FFF />
    </DDD>
  </CCC>
  <CCC>
    <BBB>
      <BBB>
        <BBB />
      </BBB>
    </BBB>
  </CCC>
</AAA>
```


XPATH

Esempi [], @, count()

/AAA/BBB[1]

Select the first BBB child of element AAA

```
<AAA>
  <BBB />
  <BBB />
  <BBB />
  <BBB />
</AAA>
```

/AAA/BBB[last()]

Select the last BBB child of element AAA

```
<AAA>
  <BBB />
  <BBB />
  <BBB />
  <BBB />
</AAA>
```

//@id

Select all attributes @id

```
<AAA>
  <BBB id = "b1"/>
  <BBB id = "b2"/>
  <BBB name = "bbb" />
  <BBB />
</AAA>
```

//BBB[@id]

Select BBB elements which have attribute id

```
<AAA>
  <BBB id = "b1"/>
  <BBB id = "b2" />
  <BBB name = "bbb" />
  <BBB />
</AAA>
```

//BBB[@id='b1']

Select BBB elements which have attribute id with value b1

```
<AAA>
  <BBB id = "b1" />
  <BBB name = " bbb " />
  <BBB name = "bbb" />
</AAA>
```

//*[count(BBB)=2]

Select elements which have two children BBB

```
<AAA>
  <CCC>
    <BBB />
    <BBB />
    <BBB />
  </CCC>
  <DDD>
    <BBB />
    <BBB />
  </DDD>
  <EEE>
    <CCC />
    <DDD />
  </EEE>
</AAA>
```

//*[count(*)=2]

Select elements which have 2 children

```
<AAA>
  <CCC>
    <BBB />
    <BBB />
    <BBB />
  </CCC>
  <DDD>
    <BBB />
    <BBB />
  </DDD>
  <EEE>
    <CCC />
    <DDD />
  </EEE>
</AAA>
```

XPATH

Manipolazione di stringhe

Function:	Return Type:	Returns:
starts-with(main_string, prefix_string)	Boolean	True if main_string starts with prefix_string; false otherwise
contains(containing_string, contained_string)	Boolean	True if the contained_string is part of the containing_string; false otherwise
substring(string, offset, length)	String	length characters from the specified offset in string; or all characters from the offset to the end of the string if length is omitted; length and offset are rounded to the nearest integer if necessary
substring-before(string, marker-string)	String	The part of the string from the first character up to (but not including) the first occurrence of marker-string
substring-after(string, marker-string)	String	The part of the string from the end of the first occurrence of marker-string to the end of string; the first character in the string is at offset 1
string-length(string)	Number	The number of characters in string
normalize-space(string)	String	The string after leading and trailing white space is stripped and runs of white space are replaced with a single space; if the argument is omitted the string value of the context node is normalized
translate(string, replaced_text, replacement_text)	String	Returns string with occurrences of characters in replaced_text replaced by the corresponding characters from replacement_text
concat(string1, string2, . . .)	String	Returns the concatenation of as many strings as are passed as arguments in the order they were passed
format-number(number, format-string, locale-string)	String	Returns the string form of number formatted according to the specified format-string as if by Java 1.1's java.text.DecimalFormat class (see http://java.sun.com/products/jdk/1.1/docs/api/java.text.DecimalFormat.html); the locale-string is an optional argument that provides the name of the xsl:decimal-format element used to interpret the format-string

XPATH

Esempi operatore or |

//*[string-length(name()) = 3]

Select elements with three-letter name

```
<AAA>
  <Q />
  <SSSS />
  <BB />
  <CCC/>
  <DDDDDDDD />
  <EEEE />
</AAA>
```

//*[starts-with(name(),'B')]

Select all elements name of which starts with letter B

```
<AAA>
  <BCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </BCC>
  <DDB>
    <BBB/>
    <BBB/>
  </DDB>
  <BEC>
    <CCC/>
    <DBD/>
  </BEC>
</AAA>
```

//*[contains(name(),'C')]

Select all elements name of which contain letter C

```
<AAA>
  <BCC>
    <BBB />
    <BBB />
    <BBB />
  </BCC>
  <DDB>
    <BBB />
    <BBB />
  </DDB>
  <BEC>
    <CCC />
    <DBD />
  </BEC>
</AAA>
```

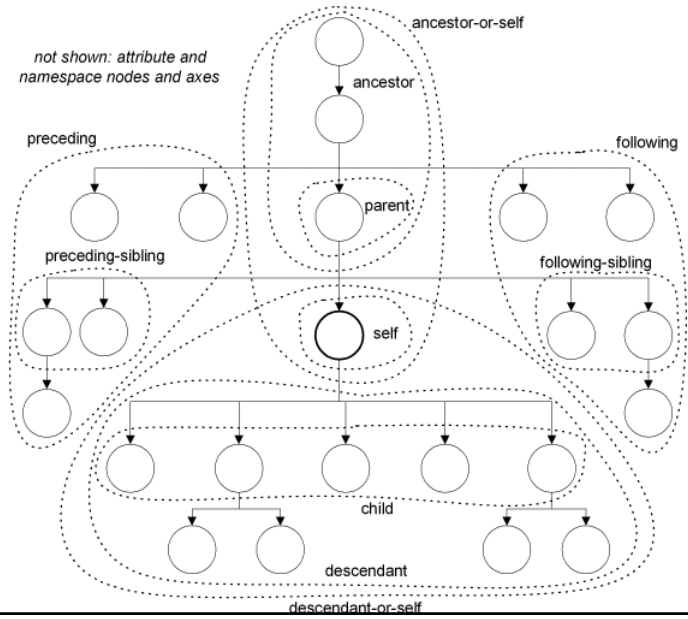
/AAA/EEE | //BBB

Select all elements BBB and elements EEE which are children of root element AAA

```
<AAA>
  <BBB />
  <CCC />
  <DDD>
    <CCC/>
  </DDD>
  <EEE/>
</AAA>
```

XPATH

Axis Specifiers



XPATH

esempi

/child::AAA

Equivalent of /AAA

```
<AAA>
  <BBB />
  <CCC />
</AAA>
```

/AAA/BBB/descendant::*

Select all descendats of /AAA/BBB

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD />
        <EEE />
      </CCC>
    </DDD>
  </BBB>
  <CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF />
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

//DDD/parent::*

Select all parents of DDD element

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD />
        <EEE />
      </CCC>
    </DDD>
  </BBB>
  <CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF />
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

/AAA/BBB/DDD/CCC/EEE/ancestor::*

Select all elements given in this absolute path

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD />
        <EEE />
      </CCC>
    </DDD>
  </BBB>
  <CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF />
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

/AAA/BBB/following-sibling::*

```
<AAA>
  <BBB>
    <CCC />
    <DDD />
  </BBB>
  <XXX>
    <DDD>
      <EEE />
      <DDD />
      <CCC />
      <FFF />
      <FFF>
        <GGG />
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD />
  </CCC>
  <CCC />
</AAA>
```

XSL

Esempio xsl:sort

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:template match="/">
<TABLE>
<xsl:for-each select="//name">
<xsl:sort order="ascending" select="."/>
<TR>
<TH>
<xsl:value-of select="."/>
</TH>
</TR>
</xsl:for-each>
</TABLE>
</xsl:template>

</xsl:stylesheet>
```

```
<source>
<name>John</name>
<name>Josua</name>
<name>Charles</name>
<name>Alice</name>
<name>Martha</name>
<name>George</name>
</source>
```

```
<TABLE>
  <TR><TH>Alice</TH>
  </TR>
  <TR>
     <TH>Charles</TH>
  </TR>
  <TR>
     <TH>George</TH>
  </TR>
  <TR>
     <TH>John</TH>
  </TR>
  <TR>
     <TH>Josua</TH>
  </TR>
  <TR>
     <TH>Martha</TH>
  </TR>
</TABLE>
```


XSL

Esempio: xsl:element

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transfor
m'>
<xsl:template match="/">
     <xsl:for-each select="//text">
          <xsl:element name="{@size}">
               <xsl:value-of select="."/>
          </xsl:element>
     </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

```
<source>
<text size="H1">Header1</text>
<text size="H3">Header3</text>
<text size="b">Bold text</text>
<text size="sub">Subscript</text>
<text size="sup">Superscript</text>
</source>
```

```
<H1>Header1</H1>
<H3>Header3</H3>
<b>Bold text</b>
<sub>Subscript</sub>
<sup>Superscript</sup>
```

XSL

Esempio xsl:attribute

```
<xsl:stylesheet version = '1.0'

xmlns:xsl='http://www.w3.org/1999/XSL/
Transform'>
<xsl:template match="color">

<TABLE>
<TR>
  <TD>
    <xsl:attribute name="style">
    <xsl:text>color:</xsl:text>
    <xsl:value-of select="."/>
      </xsl:attribute>
    <xsl:value-of select="."/>
  </TD>
</TR>
</TABLE>

</xsl:template>
</xsl:stylesheet>
```

```
<source>
<color>blue</color>
<color>navy</color>
<color>green</color>
<color>lime</color>
<color>red</color>
</source>
```

```
<TABLE>
  <TR><TD style="color:blue">blue</TD></TR>
</TABLE>
<TABLE>
  <TR><TD style="color:navy">navy</TD></TR>
</TABLE>
<TABLE>
<TR><TD style="color:green">green</TD></TR>
</TABLE>
<TABLE>
<TR><TD style="color:lime">lime</TD></TR>
</TABLE>
<TABLE>
  <TR><TD style="color:red">red</TD></TR>
</TABLE>
```


XSL

Esempio xsl:with-param

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
<TABLE>
<xsl:for-each select="//number">
<TR>
<TH>
<xsl:choose>
  <xsl:when test="text() mod 2">
    <xsl:apply-templates select=".">
      <xsl:with-param name="type">odd</xsl:with-param>
    </xsl:apply-templates>
  </xsl:when>
  <xsl:otherwise>
    <xsl:apply-templates select="."/>
  </xsl:otherwise>
</xsl:choose>
</TH>
</TR>
</xsl:for-each>
</TABLE>
</xsl:template>
<xsl:template match="number">
  <xsl:param name="type">even</xsl:param>
  <xsl:value-of select="."/>
  <xsl:text> (</xsl:text>
  <xsl:value-of select="$type"/>
  <xsl:text>)</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

```
<source>
<number>1</number>
<number>3</number>
<number>4</number>
<number>17</number>
<number>8</number>
</source>
```

```
<TABLE>
  <TR>
     <TH>1 (odd)</TH>
  </TR>
  <TR>
     <TH>3 (odd)</TH>
  </TR>
  <TR>
     <TH>4 (even)</TH>
  </TR>
  <TR>
     <TH>17 (odd)</TH>
  </TR>
  <TR>
     <TH>8 (even)</TH>
  </TR>
</TABLE>
```

XSL

Esempio xsl:variable

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:variable name="A1">
  <xsl:copy-of select="//TABLE[1]"/>
</xsl:variable>

<xsl:variable name="A2">
  <xsl:copy-of select="//TABLE[2]"/>
</xsl:variable>

<xsl:template match="/">
     <xsl:copy-of select="$A2"/>
     <xsl:copy-of select="$A1"/>
     <xsl:copy-of select="$A2"/>
</xsl:template>


</xsl:stylesheet>
```

```
<source>
<TABLE border="1">
<TR>
  <TD>AAA</TD>
  <TD>BBB</TD>
</TR>
<TR>
  <TD>aaa</TD>
  <TD>bbb</TD>
</TR>
</TABLE>
<TABLE border="1">
<TR>
<TD>1111111</TD>
</TR>
<TR>
<TD>22222222</TD>
</TR>
</TABLE>
</source>
```

```
<TABLE border="1">
<TR>
<TD>1111111</TD>
</TR>
<TR>
<TD>22222222</TD>
</TR>
</TABLE>
<TABLE border="1">
<TR>
<TD>AAA</TD>
<TD>BBB</TD>
</TR>
<TR>
<TD>aaa</TD>
<TD>bbb</TD>
</TR>
</TABLE>
<TABLE border="1">
<TR>
<TD>1111111</TD>
</TR>
<TR>
<TD>22222222</TD>
</TR>
</TABLE>
```


XSL

Numeri e funzioni matematiche

La rappresentazione dei numeri con Xpath è:
64-bit IEEE 754 floating-point doubles

Valori non numerici (stringhe, boolean) vengono covertiti automaticamente se necessario (test) o tramite la funzione *number()*

Operatori aritmetici (+, -, *, div)

floor() il più grande intero <= n

Ceiling() il più piccolo intero >= n

Round() arrotonda all'intero più prossimo a n

Sum() somma degli argomenti

XSL

floor(), ceiling(), round(), number()

```
<source>
<number>1</number>
<number>3</number>
<number>4</number>
<number>17</number>
<number>8</number>
<number>11</number>
</source>
```

```
<xsl:value-of select="sum(//number)"/>
```

```
<source>
<text>124</text>
<text>1 2 4</text>
<text>-16</text>
<text>- 16</text>
<text>125.258</text>
<text>125.</text>
<text>ASDF</text>
<text>A123</text>
<text>true</text>
<text>false()</text>
</source>
```

```
<xsl:for-each select="//text">
<xsl:value-of select="."/>
<xsl:value-of select="number()"/>
</xsl:for-each>
```

text	number
124	124
1 2 4	NaN
-16	-16
- 16	NaN
125.258	125.258
125.	125
ASDF	NaN
A123	NaN
true	NaN
false()	NaN

```
<source>
<number>6</number>
<number>3.8</number>
<number>1.234</number>
<number>-6</number>
<number>-3.8</number>
<number>-1.234</number>
</source>
```

```
<xsl:for-each select="//number">
  <xsl:value-of select="."/>
  <xsl:value-of select="floor(.)"/>
  <xsl:value-of select="ceiling(.)"/>
  <xsl:value-of select="round(.)"/>
</xsl:for-each>
```

number	floor	ceiling	round
6	6	6	6
3.8	3	4	4
1.234	1	2	1
-6	-6	-6	-6
-3.8	-4	-3	-4
-1.234	-2	-1	-1

XSL

Esempio position(), last(),count()

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
<DIV>
  <xsl:for-each select="//BBB">
    <xsl:call-template name="printout"/>
  </xsl:for-each>
</DIV>
<DIV>
  <xsl:apply-templates select="//CCC"/>
</DIV>
<DIV>
  <xsl:apply-templates select="//AAA[last()]//CCC"/>
</DIV>
</xsl:template>

<xsl:template match="CCC">
  <xsl:call-template name="printout"/>
</xsl:template>

<xsl:template name="printout">
  <xsl:if test="position()=1">
    <xsl:value-of select="name()"/>
  </xsl:if>
  <xsl:text>(</xsl:text>
  <xsl:value-of select="position()"/>
  <xsl:text>/</xsl:text>
  <xsl:value-of select="last()"/>
  <xsl:text>)</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

```
<source>
<chapter>Chapter A</chapter>
<chapter>Chapter B</chapter>
<chapter>Chapter C</chapter>
<chapter>Chapter D</chapter>
</source>
```

```
<xsl:value-of
select="count(//chapter)"/>
```

```
<source>
<AAA>
  <BBB>
    <CCC>Carl</CCC>
  </BBB>
  <BBB/>
  <BBB/>
</AAA>
<AAA>
  <BBB/>
  <BBB>
    <CCC>John</CCC>
    <CCC>Charles</CCC>
    <CCC>Robert</CCC>
    <CCC>Anthony</CCC>
  </BBB>
</AAA>
</source>
```

```
<DIV>
BBB(1/5)(2/5)(3/5)(4/5)(5/5)
</DIV>

<DIV>
CCC(1/5)(2/5)(3/5)(4/5)(5/5)
</DIV>

<DIV>
CCC(1/4)(2/4)(3/4)(4/4)
</DIV>
```

XSL

Esempio stringhe

```
<source>
  <text>Welcome to XSL world.</text>
  <string>XSL</string>
  <start>4</start>
  <end>10</end>
</source>
```

```
Text: Welcome to XSL world.
Text before XSL: Welcome to
Text after XSL: world.
Text from position 4: come to XSL world.
Text from position 4 of length 10: come to XS
```

```
<xsl:value-of select="substring-before(//text,//string)"/>
<xsl:value-of select="substring-after(//text,//string)"/>
<xsl:value-of select="substring(//text,//start)"/>
<xsl:value-of select="substring(//text,//start,//end)"/>
```

```
<source>
<P>
<text>Normalized text</text>
<text>Sequences of whitespace
characters</text>
<text> Leading and trailing whitespace.
</text>
</P>
</source>
```

```
Normalized text
Starting length:15
Normalized length:15
Sequences of whitespace characters
Starting length:41
Normalized length:34
Leading and trailing whitespace.
Starting length:40
Normalized length:32
```

```
<xsl:for-each select="//text">
  <xsl:value-of select="."/>
  <xsl:value-of select="string-length(.)"/>
  <xsl:value-of select="string-length(normalize-space(.))"/>
</xsl:for-each>
```


XSL

Esempio id, generate-id

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Tra
nsform'>
<xsl:template match="*">
<xsl:copy>
  <xsl:attribute name="id">
    <xsl:value-of select="generate-
id()"/>
  </xsl:attribute>
  <xsl:for-each select="@*">
    <xsl:attribute name="{name()}">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:for-each>
  <xsl:apply-templates/>
</xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

```
<source>
<AAA name="top">
     <BBB pos="1"
val="bbb">11111</BBB>
     <BBB>22222</BBB>
</AAA>
<AAA name="bottom">
     <BBB>33333</BBB>
     <BBB>44444</BBB>
</AAA>
</source>
```

```
<source id="d0e1">
  <AAA id="d0e3" name="top">
    <BBB id="d0e5" pos="1“
al="bbb">11111</BBB>
     <BBB id="d0e8">22222</BBB>
  </AAA>
  <AAA id="d0e12" name="bottom">
  <BBB id="d0e14">33333</BBB>
  <BBB id="d0e17">44444</BBB>
  </AAA>
</source>
```

```
<!DOCTYPE source [
<!ELEMENT chapter ANY>
<!ATTLIST chapter id ID #REQUIRED>
<!ELEMENT title ANY>
<!ATTLIST title id CDATA #REQUIRED>
<!ELEMENT text ANY>
<!ATTLIST text value ID #REQUIRED>
]>
<source>
<chapter id="intro">Introduction</chapter>
<chapter id="body">
<title id="t1">BODY</title>
<text value="text1">text text text</text>
</chapter>
<chapter id="end">THE END</chapter>
</source>
```

```
<xsl:value-of select="id('intro')"/>
<xsl:value-of select="id('body')/text"/>
<xsl:value-of select="id('text1')"/>
```

XSL

Esempio output()

```
<source>
<hr> </hr>
<hr/>
<hr/>
</source>
```

HTML

```
<source>
  <hr>
  <hr>
  <hr>
</source>
```

XML

```
<source>
<hr> </hr>
<hr/>
<hr/>
</source>
```

- Specifica come formattare l'output
- Default: xml
- Output = html → nessun escaping, no empty tag
- encoding

XSL

Esempio copy, copy-of

```
<source>
<p id="a12"> Compare
     <B>these constructs</B>.
</p>
</source>
```

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Tr
ansform'>
<xsl:template match="p">
  <xsl:copy-of select="."/>
  <xsl:copy/>
</xsl:template>
</xsl:stylesheet>
```

```
<p id="a12"> Compare
<B>these constructs</B>. </p>
<p/>
```

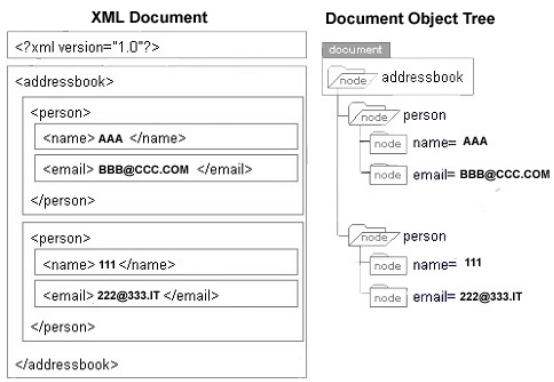
DOM & SAX - introduzione

SAX (Simple API for XML) e DOM (Document Object Model) sono stati entrambi realizzati per permettere l'accesso a documenti XML in modo che gli sviluppatori non dovessero realizzare i propri parser. Sono disponibili differenti parser sul mercato ma mantenendo i dati in XML e utilizzando le suddette API è possibile utilizzare indifferentemente uno dall'altro.

Malgrado gli scopi siano gli stessi, l'approccio SAX e DOM sono completamente differenti.

DOM & SAX - introduzione

DOM permette l'accesso ai dati di un documento XML tramite un modello ad oggetti gerarchico. Viene creato un albero di *nodi* basato sulla struttura dell'informazione presente nel documento.



Il modello DOM realizza questa struttura ad albero. La scelta di questa rappresentazione è naturale essendo XML gerarchico di natura. L'esempio è semplificato: infatti ad ogni nodo può corrispondere non solo testo (valore del nodo), ma anche un ulteriore lista di nodi. L'accesso al valore di un nodo richiede per questo qualche sforzo in più. In un documento la lista degli elemeni è importante mentre non lo è ad esempio per una tabella di un database. DOM mantiene l'ordine degli elementi letti dal documento XML perché tratta tutto come fosse appunto un documento, da qui deriva il nome DOCUMENT object model..

DOM & SAX - introduzione

SAX da accesso al documento XML tramite una *sequenza di eventi*. Non viene creato un modello ad oggetti di default come in DOM. Questo rende SAX molto più veloce ed inoltre permette di realizzare il proprio modello ad oggetti e le classi che intercettano gli eventi lanciati da SAX.

Mentre DOM fa quasi tutto per noi (legge il documento, crea un modello ad oggetti e da un riferfimento a quest'oggetto) SAX risulta molto più semplice (*simple*!):

•*Legge un documento XML*

•*Lancia un evento che dipende dal tipo di tag incontrato (tag aperto, tag chiuso, #PCDATA, CDATA…)*

Noi dobbiamo

•*Creare un modello ad oggetti per le informazioni XML*

•*Creare un handler che stia in ascolto degli eventi SAX (generati dal parser SAX nella lettura del documento) e dare un senso a questi eventi creato oggetti nel modello che abbiamo definito.*

Questo approccio risulta molto più veloce di quello di DOM (non essendoci la fase di generazione del document object).

DOM & SAX - introduzione

La scelta di quale dei due approcci utilizzare dipende, ovviamente dai casi. E' possibile comunque definire alcune regoli generali.

Quando usare DOM?

Se si sta trattando con manipolazione di documenti, document management
Se non si hanno problemi di prestazioni e il documento è di dimensioni tali da poter essere rappresentato in-memory.

Quando usare SAX?

Quando si ha a che fare con dati strutturati generati (resultset) e che tipicamente devono essere strutturati secondo un modello ad aoggetti personale.
Problemi di prestazioni. Documento troppo grande.

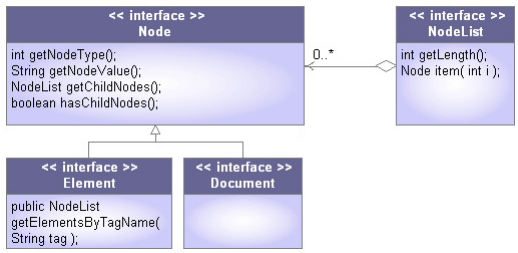
Per fare un esempio un documento tipo WordProcessor si presta ad essere manipolato in DOM, mentre puri dati tipo un agenda predilige un approccio SAX.

DOM

API Package names

Le interfacce DOM esistono nel package org.w3c.dom. Quindi è necessario includere sempre: `import org.w3c.dom.*;` mentre le implementazioni di tali interfacce vengono realizzate dal parser XML che si è deciso di utilizzare.Una volta istanziato l'oggetto parser si deve accedere a tale oggetto solamente attraverso le interfacce DOM in modo da rendere il codice completamente portabile (tranne per la creazione del parser).

Interfacce

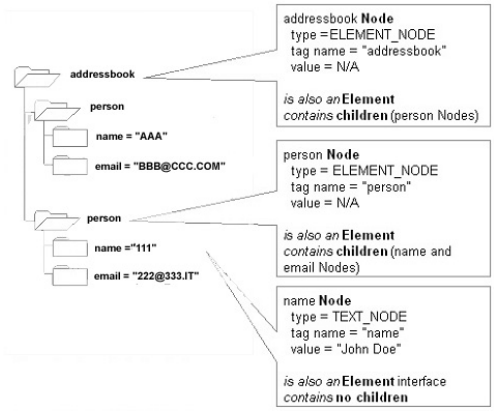


Un *Document* Object contiene un albero di oggetti *Node*. In DOM tutto è un Node. Il Node di root è anche *Document*. L'interfaccia *Element* permette di accedere agli elementi dell'oggetto Document.

DOM

L'interfaccia Node permette di ottenere diverse informazioni su un nodo:

- Tipo nodo / nome tag / valore
- Quali sono i figli (Nodes) di un nodo (Node) ?
- Un nodo ha figli?

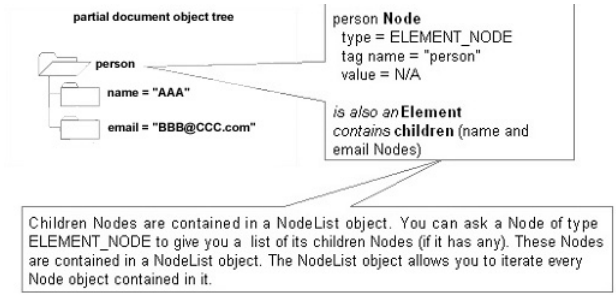


Notiamo che:

- se un oggetto Node ha un valore non è detto che abbia figli
- Se ha figli può o non può avere un valore
- Node può sempre non avere figli o valore (tag vuoto)

DOM

Con getChildNodes() si ottengono i nodi all'interno di un nodo.
Restituisce un container di Node objects.



DOM

Come primo passo si deve ottenere la lista degli oggetti Element con con tag name "PERSON".

```
Document doc = ... //create DOM from AddressBook.xml
NodeList listOfPersons =      doc.getElementsByTagName( "PERSON" );
int numberOfPersons = listOfPersons.getLength();
```

Ottenuta la NodeList estraiamo il primo nodo PERSON. Il metodo item(int index) applicato su una NodeList restituisce un Node.

```
if (numberOfPersons > 0 ){
  Node firstPersonNode = listOfPersons.item( 0 );
  if( firstPersonNode.getNodeType() == Node.ELEMENT_NODE ) {
          Element firstPersonElement = (Element)firstPersonNode;         }
 }
```

Ottenuto il riferimento a firstPersonElement, dobbiamo estrarre FIRSTNAME, LASTNAME, cioè le informazioni associate all'elemento PERSON. Essendo firstPersonElement un Element possiamo utilizzare nuovamente getElementsByTagName

```
NodeList firstNameList =      firstPersonElement.getElementsByTagName( "FIRSTNAME" );
Element firstNameElement =      firstNameList.item( 0 );
```

firstNameElement contiene una lista di TEXT_NODE uno dei quali rapprseneta il valore di FIRSTNAME.

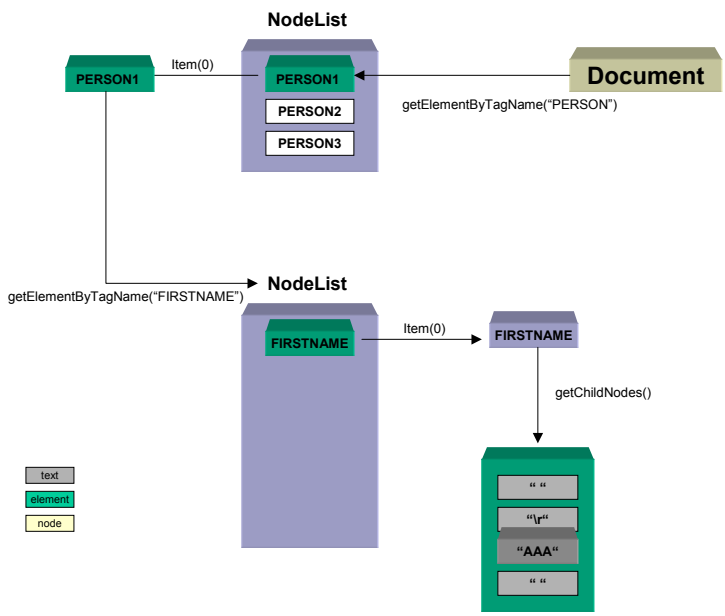
```
NodeList list = firstNameElement.getChildNodes();
```

DOM

Questa lista di TEXT_NODE può contenere dei dati che non sono utili per noi (whitespaces, cr, lf). Eliminati questi nodi otteniamo il testo che ci interessa realmente. Si deve iterare la lista e verificarne il contenuto con getNodeValue()

```
String firstName = null;
for (int i = 0 ; i < list.getLength() ; i ++ ){
          String value =          ((Node)list.item( i )).getNodeValue().trim();
          if( value.equals("") || value.equals("\r") ){
                    continue; //keep iterating     }     else{         firstName =
value;         break; //found the firstName!     } }
```


DOM



DOM

```
import org.w3c.dom.*;

public class XmlUtils{

    /**
     Return an Element given an XML document, tag name, and index
     @param     doc     XML document
     @param     tagName a tag name
     @param     index   a index
     @return    an Element
     */

    public static Element getElement( Document doc , String
tagName ,int index ){

        //given an XML document and a tag
        //return an Element at a given index
        NodeList rows =
doc.getDocumentElement().getElementsByTagName(
                tagName );
        return (Element)rows.item( index );
    }
/**
     Return the number of person in an XML document
     @param     doc     XML document
     @param     tagName a tag name
     @return    the number of person in an XML document
     */
      public static int getSize( Document doc , String
tagName ){
        //given an XML document and a tag name
        //return the number of ocurances
        NodeList rows =
doc.getDocumentElement().getElementsByTagName(
                tagName );
        return rows.getLength();
    }
```

```
/**
      Given a person element, must get the element specified
      by the tagName, then must traverse that Node to get the
      value.
      Step1) get Element of name tagName from e
      Step2) cast element to Node and then traverse it for its
               non-whitespace, cr/lf value.
      Step3) return it!
      NOTE: Element is a subclass of Node
      @param    e   an Element
      @param    tagName a tag name
      @return   s   the value of a Node
    */
    public static String getValue( Element e , String tagName
){
    try{
        //get node lists of a tag name from a Element
        NodeList elements = e.getElementsByTagName( tagName );
        Node node = elements.item( 0 );
        NodeList nodes = node.getChildNodes();
        //find a value whose value is non-whitespace
        String s;
        for( int i=0; i<nodes.getLength(); i++){
            s = ((Node)nodes.item( i )).getNodeValue().trim();
            if(s.equals("") || s.equals("\r")) {
                continue;
            }
            else {
              return s;}
        }
    }
    catch(Exception ex){
        System.out.println( ex );
        ex.printStackTrace();
    }
    return null;
    }
```


DOM

```
/**
     For testing purpose, it print out Node list
     @param     rows    a Nodelist
     */
    public static void printNodeTypes( NodeList rows ){
        System.out.println( "\tenumerating NodeList (of Elements):");
        System.out.println( "\tClass\tNT\tNV" );
        //iterate a given Node list
        for( int ri = 0 ; ri < rows.getLength() ; ri++){
            Node n = (Node)rows.item( ri );
            if( n instanceof Element) {
                System.out.print( "\tElement" );
            }
            else System.out.print( "\tNode" );
            //print out Node type and Node value
            System.out.println(
                "\t"+
                n.getNodeType() + "\t" +
                n.getNodeValue()
                );
        }
        System.out.println();
    }
}//end class
```

DOM

```
import oracle.xml.parser.v2.*;
import com.sun.xml.tree.*;
import org.xml.sax.*;
import org.w3c.dom.*;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AddressBookServlet extends HttpServlet {
    //CONSTANTS
    public static final String ROOT_ELEMENT_TAG = "person";
    public static final String[] colNames ={
        "lastname",
        "firstname",
        "company",
        "email"
    };
    //DATA
    protected Document doc;

/**
     When this method receives a request from a browser, it returns
a  XML document in table format.
     @param     req     http servlet request
     @param     res     http servlet response
     @exception ServletException
     @exception IOException
     */
    protected void doGet(HttpServletRequest req, HttpServletResponse
res)
    throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = new PrintWriter(res.getOutputStream());
        out.print("<html>");
        out.print("<title>DOM example: Oracle Parser</title>");
        out.print( "<center>" );
        out.print( "<head><pre>addr.xml</pre></head><hr>" );

        //format a table
        out.print( "<table BORDER=0 CELLSPACING=2 CELLPADDING=10
BGCOLOR=\"#CFCFFF\" >" );
        out.print( "<tr>");
        //display table column
        for(int i=0; i<colNames.length; i++){
            out.print(
"<td><b><center>"+colNames[i]+"</center></b></td>" );
        }
        out.print( "</tr>" );
```

```
//need to iterate the doc to get the fields in it

        int rowCount = XmlUtils.getSize( doc , ROOT_ELEMENT_TAG );
        for(int r=0; r<rowCount; r++) {
            out.print( "<tr>" );
            Element row = XmlUtils.getElement( doc , ROOT_ELEMENT_TAG ,
r );
            int colCount = colNames.length;

            for(int c=0; c < colCount; c++) {
                out.print( "<td>" );
                out.print( XmlUtils.getValue( row , colNames[c] ) );
                out.print( "</td>" );
            }
            out.print( "</tr>" );
        }//endfor

        out.print( "</table>" );
        out.print( "</center>" );
        out.println("</body>");
        out.println("</html>");
        out.flush();
        out.close();
    }//end method

    /**
     Create a DOM from an XML document when the servlet starts up.
     @param     config  servlet configuration
     @exception ServletException
     */
    public void init( ServletConfig config ) throws ServletException{
        super.init( config );
        //load the Document
        try{
            //create xml document
            DOMParser parser = new DOMParser();
            parser.parse(new BufferedReader( new
FileReader("/PROD/addr.xml")));
            doc = parser.getDocument();
        }
        catch( Exception e ) {
            System.out.println( e );
        }
    }
    /**
     Return servlet information
     @return    message about this servlet
     */
    public String getServletInfo(){
        return "-";
    }
}//end class
```

SAX

SAX non fornisce un modello ad oggetti di default per i dati XML per questo motivo il primo passo sarà proprio quello di realizzarne uno (AddressBook). Secondo passo la creazione di un document handler che crei istanze di questi oggetti in base alle informazioni presenti nel documento XML. Questo handler è un listener degli eventi che sono lanciati dal parser SAX in base al contenuto del documento XML.. L'esempio che segue realizza modello ad oggetti e listener per la realizzazione di un address book

```
<?xmlversion="1.0"?>
<addressbook>
  <person>
    <lastname>aaa</lastname>
    <firstname>bbb</firstname>
    <company>zzz</company>
    <email>ccc@ddd.com</email>
  </person>
</addressbook>
```

SAX

Creazione del proprio modello ad oggetti

```
import java.util.*;

public class AddressBook{
// Data Members
List persons = new java.util.ArrayList();

// mutator method
public void addPerson( Person p ){persons.add( p );}

// accessor methods
public int getSize(){ return persons.size();}
public Person getPerson( int i ){
    return (Person)persons.get( i );}

// toXML method
public String toXML(){
    StringBuffer sb = new StringBuffer();
    sb.append( "<?xml version=\"1.0\"?>\n" );
    sb.append( "<addressbook>\n\n" );
    for(int i=0; i<persons.size(); i++) {
        sb.append( getPerson(i).toXML() );
        sb.append( "\n" );
    }
    sb.append( "</addressbook>" );
    return sb.toString();
}}
```

```
public class Person{
// Data Members
String fname, lname, company, email;

// accessor methods
public String getCompany(){return company;}
public String getEmail(){return email;}
public String getFirstName(){return fname;}
public String getLastName(){return lname;}

// mutator methods
public void setLastName( String s ){lname = s;}
public void setFirstName( String s ){fname = s;}
public void setCompany( String s ){company = s;}
public void setEmail( String s ){email = s;}

// toXML() method
public String toXML(){
    StringBuffer sb = new StringBuffer();
    sb.append( "<person>\n" );
    sb.append( "\t<lastname>"+lname+"</lastname>\n" );
    sb.append( "\t<firstname>"+fname+"</firstname>\n" );
    sb.append( "\t<company>"+company+"</company>\n" );
    sb.append( "\t<email>"+email+"</email>\n" );
    sb.append( "</person>\n" );
    return sb.toString();
}}
```

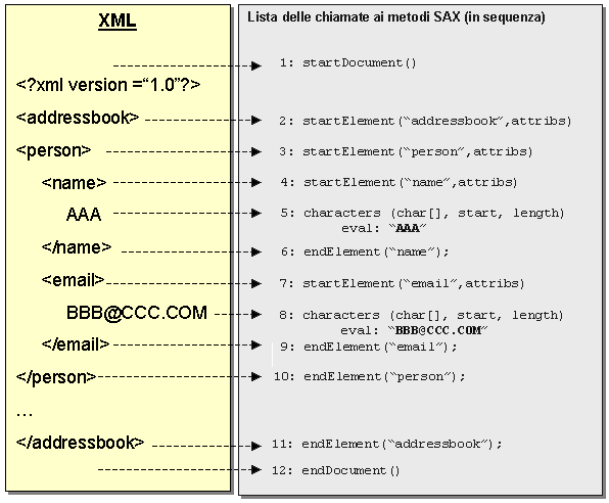
SAX

Creazione di un parser SAX

```
import org.xml.sax.Parser.*;
import org.xml.sax.helpers.ParserFactory;
public class SaxDemo
{
public static void main(String args[])
{
try{
    //create an InputSource from the XML document source
    BufferedReader isr = new BufferedReader(
        new FileReader( "/PROD/addr.xml" )
    );
    InputSource is = new InputSource( isr );

    //create an documenthandler to create obj model
    DocumentHandler handler = new SaxAddressBookHandler();

    //create a SAX parser using SAX interfaces and classes
    String parserClassName = "oracle.xml.parser.v2.SAXParser";
    org.xml.sax.Parser parser = org.xml.sax.helpers.ParserFactory.
        makeParser( parserClassName );

    //create document handler to do something useful
    //with the XML document being parsed by the parser.
    parser.setDocumentHandler( handler );
    parser.parse( is );
    AddressBook ab = ((SaxAddressBookHandler)handler).getAddressBook();
    System.out.println(ab.toXML());
    }
catch(Throwable t){
    System.out.println( t );
    t.printStackTrace();
}
}
}
```

SAX

Creazione di un DocumentHandler



arser SAX che abbiamo creato,
cia eventi per ogni tag aperto,
uso, CDATA, #PCDATA… Questi
nti provengono dal documento
L dall'alto verso il basso, uno alla
a. Il Parser SAX per notificare
esti eventi ad un oggetto utilizza
erfaccia DocumentHandler.
sieme alla interfaccie
ityResolver, DTDHandler,
orHandler si coprono tutti i possibili
nti generabili durante la lettura di
documento XML.

SAX

ErrorHandler interface

Condizioni di errore nel sorgetnte XML vengono gestite implementando i metodi di tale interfaccia. Non è obbligatorio comunque farlo. Implementando l'interfaccia HanderBase viene fornita una implementazione di default che lancia un eccezione di tipo SAXException in caso di errore. I metodi di questa interfaccia sono;

```
error(SAXParseException e)
fatalError(SAXParseException e)
warning(SAXParseException e)
org.xml.sax.Parser.parser = //create a SAX
parserErrorHandler handler = //instantiate your implementation
parser.setErrorHandler( handler );
```

DTDHandler interface

Questa interfaccia comunica con il DTD del documento XML che si sta leggendo.

```
notationDecl( String name , String publicId, String systemId )
unparsedEntityDecl( (String name, String publicId, String systemId, String notationName).
org.xml.sax.Parser.parser = //create a SAX
parserDTDHandler handler = //instantiate your DTDHandler implementation
parser.setDTDHandler( handler );
```

EntityResolver interface

Serve per gestire entità esterne come ad esempio un DTD localizzabile tramite URI.

```
InputSource resolveEntity (String publicId, String systemId).
org.xml.sax.Parser.parser = //create a SAX
parserEntityResolver handler = //instantiate your implementation
parser.setDTDHandler( handler );
```

HandlerBase class

Invece di implementare tutte e 4 queste interfaccie è possibile esteendere la HandlerBase class che mette a disposzione un'implementazione di default (empty) per ognuna delle 4. In questo modo è possibile fare l'override delle sole implementazioni che necessitiamo.

SAX

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.ParserFactory;
import oracle.xml.parser.v2.*;

public class SaxAddressBookHandler extends HandlerBase{
// data members
private AddressBook ab = new AddressBook();
private Person p = null;              //temp Person ref
private String currentElement = null; //current element name


// AddressBook accessor method
public AddressBook getAddressBook(){    return ab; }

// HandlerBase method overrides. This is SAX.
/*
This method is called when the SAX parser encounters an open
element tag. Must remember which element tag was just opened (so
that thecharacters(..) method can do something useful with the data
that isread by the parser.
*/

public void startElement( String name , AttributeList atts ){
    if( name.equalsIgnoreCase("LASTNAME") ) {
        currentElement = "LASTNAME";
    }
    else if( name.equalsIgnoreCase("FIRSTNAME") ) {
        currentElement = "FIRSTNAME";
    }
    else if( name.equalsIgnoreCase("COMPANY") ) {
        currentElement = "COMPANY";
    }
    else if( name.equalsIgnoreCase("EMAIL") ) {
        currentElement = "EMAIL";
    }
    else if( name.equalsIgnoreCase("PERSON") ) {
        p = new Person();
    }
}
```

```
/*
This method is called when the SAX parser encounters a close
element tag. If the person tag is closed, then the person objec
must be added to the AddressBook (ab).
*/

public void endElement( String name ){
    if( name.equalsIgnoreCase("PERSON") ) {
        ab.addPerson( p );
        p = null;
    }
}


/*
This method is called when the SAX parser encounters #PCDATA or
CDATA.It is important to remember which element tag was just
opened so that this data can be put in the right object. I had to
trim() the textual data and make sure that empty data is just
ignored.Also the start index and length integer must be used to
retrieve only a portion of the data stored in the char[].
*/

public void characters( char ch[], int start, int length ){
    //dont try to read ch[] as it will go on forever, must use the
    //range provided by the SAX parser.
    String value = new String( ch , start , length );
    if(!value.trim().equals("")) {
        if( currentElement.equalsIgnoreCase("FIRSTNAME") ) {
            p.setFirstName( value );
        }
        else if( currentElement.equalsIgnoreCase("LASTNAME") ) {
            p.setLastName( value );
        }
        else if( currentElement.equalsIgnoreCase("COMPANY") ) {
            p.setCompany( value );
        }
        else if( currentElement.equalsIgnoreCase("EMAIL") ) {
            p.setEmail( value );
        }
    }
}
}
```

Applicazioni Java - XML

introduzione

La realizzazione di un'applicazione XML coinvolge l'utilizzo di differenti componenti che vanno dalle API all'integrazione con soluzioni di terze parti.

- W3C DOM and SAX
- Servlet API
- Swing API
- JDBC API
- RMI API
- Java core API

E ovviamente si deve tenere conto di:

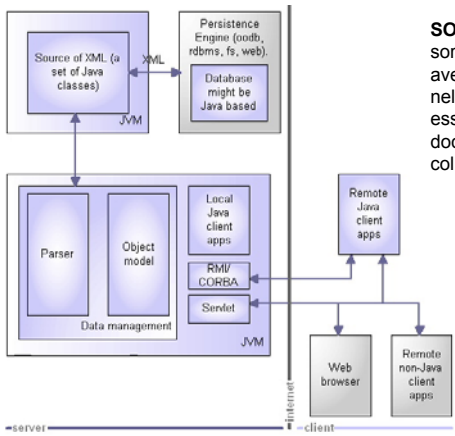
- Databases
- FileSystems
- WebServers

Introduzione
componenti e criticità

Persistence Layer
Responsabile dell'immagazzinamento dei dati in modo che non vengano persi. Ad esempio Db, o file system in caso di file XML. Non è escluso che si debba aver a che fare con differenti layer. Nel caso di Db relazionali tramite si può contare su JDBC e SQL. E' importante definire delle interfacce tali da poter realizzare differenti implementazioni in base al layer sul quale si vuole lavorare.

Firewall
Serve a proteggere intranet aziendali dal mondo esterno. Ogni pc all'interno di una LAN utilizza una proxy server installato nel firewall per accedere al web. Questo significa che non è sempre possibile utilizzare RMI per la comunicazione tra client e server. Web browser dietro a un firewall possono comunicare con un webserver fuori dal firewall perché tipicamente la porta 80 viene lasciata libera proprio per questi scopi. Per questo motivo è desiderabile effettuare tali comunicazioni tramite un firewall tunneling tramite HTTP. E' possibile fare questo su un client java dietro firewall che utilizzi la classe URLConnection.

Parser
Permette di accedere ad un documento XML. Tali dati devono essere rappresentati tramite un modello ad oggetti. Una volta modificato si accede al persistence layer per salvarlo nuovamente in formato XML. SAX non fornisce un modello di default come invece fa DOM. E' consigliabile di salavare i dati XML in una forma "pura" e non salvare direttamente gli oggetti (serializzazione)



SOURCE – set di interfacce che virtualizzano la sorgente dei documenti XML (db, fs, web). Si possono avere differenti sorgenti senza dover cambiare codice nella restante parte del sistema. Ad esempio potrebbe essere presente un metodo che dato un URI recupera il documento (come Stringa). Un altro metodo comunica col Persistence Layer per memorizzare tale documento.

OBJECT-MODEL – rappresenta il modello ad oggetti del documento XML che è stato recuperato. Anche se un parser DOM fornisce un modello ad oggetti di default potrebbe risultare necessario convertirlo in un altro. Devono essere scritte una serie di interfacce per agire su questi dati, presentarli, e renderli persistenti.

PRESENTATION LAYER – tranne il caso in cui non vi sia nessuna interazione umana sarà necessario provvedere ad una interfaccia utrente per visualizzare i dati e editarli. Tale rappresentazione può essere Web Based o Java Based.

Applicazioni Java - XML

Vantaggi

• *Flessibilità:* Possibilità di modificare un sottosistema se necessario senza impattare sul resto del sistema. Ad esempio se serve un nuovo persistence layer basta implementare un set di interfacce che avevamo definito in origine.

•*Integrazione* della nostra soluzione con sistemi pre-esistenti. Anche se tale sistema non fosse network/web enabled sarebbe possibile risolvere questo realizzando un layer al di sopra di questo che si occupa del networking e della trasmissione e ricezione dei dati XML.

•*Web – Enabled.* Essendo XML tale, risulta facile ed economico rendere disponibili dei servizi sul Web tramite l'utilizzo delle Servlets.

Svantaggi

•*Attenzione alla fase di design.* Un design non corretto potrebbe rendere il sistema non estensibile anche se scalabile e performante.

•*Tecnologie innovative.* Essendo tecnologie nuove ci sono meno sistemi reali sui quali confrontarsi.

Applicazioni Java – XML Tipologie

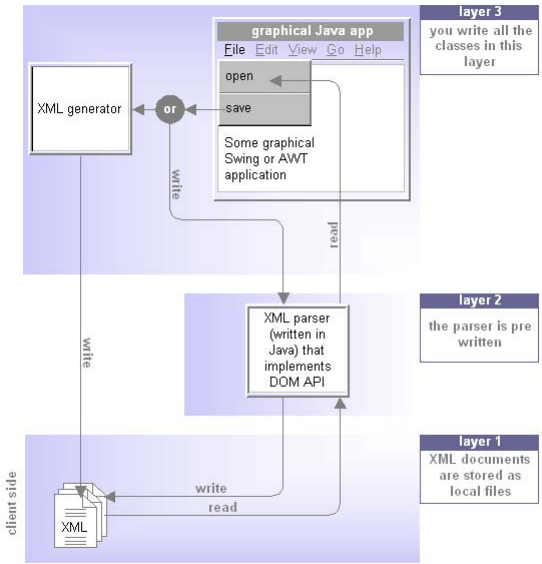
Le tre categorie di applicazioni che seguiranno non sono esaustive ma rappresentano delle situazioni standard che ben si prestano all'utilizzo delle tecnologie Java e XML.

•*Client side - Graphical Java Applications*

•*Client and Server side - Application Servers*

•*Web-based Applications*

Applicazioni Java – XML *Client side - Graphical Java Applications*

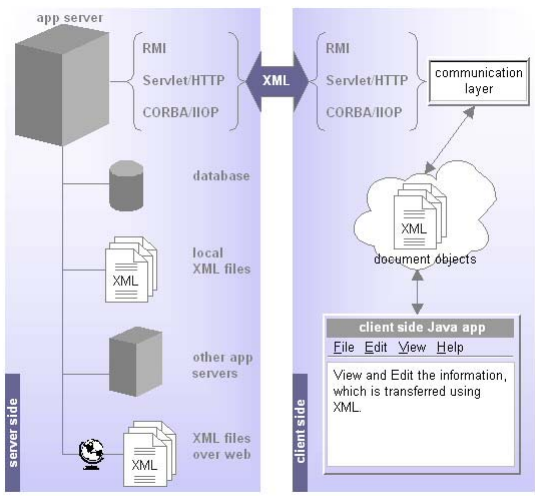


In questo tipo di applicazione le informazioni sono memorizzate in un documento XML (file). Non è necessario utilizzare un formato binario proprietario essendo possibile definire i propri tag. Andranno realizzate classi che permettano di:
- importare/esportare un formato XML
- validare tramite un DTD
- user interface: save/load/edit

L'importazione/esportazione Può seguire diverse strategie:
- utilizzare DOM per manipolare i dati
- utilizzare un proprio modello ad oggetti
- utilizzare un adapter su DOM

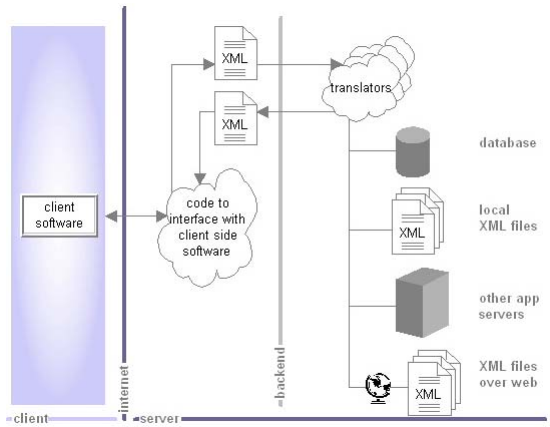
il salvataggio delle informazioni in base al modello ad oggetti che si è utilizzato può seguire due strade:
- generazione personale dell'XML (dal prorpio modello ad oggetti)
- utilizzare il parser DOM per la generazione automatica dell'XML.

Applicazioni Java – XML *Client and Server side - Application Servers*



Un application server lega differenti networked software components assieme in modo da poter provvedere informazioni da differenti sorgenti a differenti clients. Realizzare una application server consiste dunque nel realizzare differenti sistemi interconnessi tra loro che accettano e distribuiscono informazioni da/verso diverse sorgetni/destinazioni. In questo scenario XML rappresenta il substrato comune col quale sistemi diversi possono condividere informazioni. Utilizzando file di testo non servono convertitori per i diversi formati binari possibili. Essendo HTTP in grado di inviare plain text risulta evidente l'utilità di questo protocollo per inviare documenti XML su diverse reti superando così anche problemi legati a eventguali firewall.

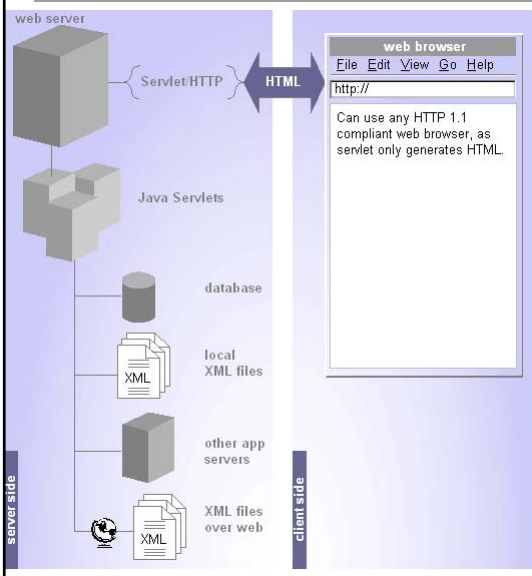
Applicazioni Java – XML *Client and Server side - Application Servers*



Qualunque protocollo può essere utilizzato (non solo HTTP). L'importante è ricordare che utilizzando XML come flusso dei dati questi potranno essere processati da ogni parte del sistema. La serializzazione di Java non permette questo perché serializza in un formato binario che dipende da molti fattori. App servers permette l'accesso ai propri clients a informazioni presenti su db, fs remoti, risorse web, o altri app servers. Utilizzando tecnologie come Java,XML,JDBC,RMI, CORBA è possibile rendere disponibili tutte queste informazioni ai propri clients.

Rendendo pubblici e standardizzando DTD per industrie verticali sarà possibile rendere condivisibili dati di diverse compagnie indipendentemente dal software che ognuna utilizza per immagazzinare i dati tramite i corrispondenti app server.

Applicazioni Java – XML *Web-based Applications*



Simili alle app server si differenziano per il fatto che viene utiloizzato un web-browser su lato cliente invece di una applicazione. Il front-end viene generato dinamicamente dall'applicazione server in HTML. JSP e Servlets rappresentano lo strumento migliore (in ambiente Java) per ralizzare ciò. Una Web-based apps potrebbe legarsi ad una app server per ottenere informazioni da presentare al web browser client. Informazioni possono essere prese attraverso servlets da un db, o da altre risorse locali o remote. La distribuzione di un'applicazione simile diventa molto semplice non essendoci nessuna necessità a livello client e se l'HTML risulta suffciente per gli scopi preposti.

Tramite web-based apps e app server è possibile rendere fruibili le informazioni ovunque e su qualunque dispositivo (web-browser, web-tv, cell. 3G). Questo è possible manenendo i dati in XML puro e ralizzando differenti stylesheet per i diversi dispositivi.

Java e XML permettono di realizzare un'applicazione realmente platform e device independant.

Tecnologie XML based

XLINK

XPOINTER

XSL:FO

XLINK

XLINK

Limiti del link html (<A>)

- È possibile puntare un solo documento
- Per aumentare la granularità si deve agire sul documento destinazione (anchor)
- Non esiste relazione tra i documenti linkati
- Un documento sa cosa sta linkando, ma non sa da chi è linkato

XLINK realizza tutto ciò che è possibile fare in HTML e in più:

- Link multidirezionali
- Ogni elemento può diventare un Link (non solo gli elementi marcati da <A>)
- I link non devono necessariamente essere messi nel medesimo file come i documenti che connettono.

L'utilizzo dell'elemento XLINK necessita la dichiarazione del namespace di XLINK

URI = `http://www.w3.org/1999/xlink`

Un elemento XLINK viene identificato dal valore dell'attributo *type* che può valere:

- `simple`
- `extended`
- `locator`
- `arc`
- `resource`
- `title`
- `none`

XLINK

Link Semplici

```
Gli attributi di un elemento XLINK ne definiscono
il comportamento:

<FOOTNOTE
xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="simple"
xlink:href="footnote7.xml">7</FOOTNOTE>

<IMAGE xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="simple"
xlink:href="logo.gif"
xlink:actuate="onLoad"
xlink:show="embed"/>
```

Link semplice

target

Attivazione automatica

L'elemento viene sostituito dall'oggetto linkato.

Se il documento che stiamo creando ha un DTD gli attributi XLINK dovrebbero essere dichiarati.

Un elemento XLINK non ha restrizioni su altri attributi o figli.

```
<!ELEMENT COMPOSER (#PCDATA)>
<!ATTLIST COMPOSER
  xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
  xlink:type  CDATA #FIXED "simple"
  xlink:href  CDATA #REQUIRED >
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE
  xmlns:xlink CDATA #FIXED  "http://www.w3.org/1999/xlink"
  xlink:type  CDATA #FIXED "simple"
  xlink:href  CDATA #REQUIRED
  xlink:show  CDATA #FIXED "onLoad"
  xlink:actuate CDATA #FIXED "embed" >
```

```
<IMAGE xlink:href="logo.gif"/>
```

XLINK

Altri attributi di XLINK

Descrizione della risorsa remota

•`xlink:`**`title`**`="Search with Google"`
Descrizione della risorsa (testo)

•`xlink:`**`role`**`="http://www.google.com/help.html`
Uri che descrive in maniera completa la risorsa remota

Comportamento dei Link

•`xlink:`**`show`**`="replace|new|embed|other|none"`
Definisce come il contenuto debba essere visualizzato quando il link è attivo.
- •Replace: apre il link nello stesso documento
- •New:apre in una nuova finestra
- •Embed: inserisce il contenuto nella posizione dell'elemento xlink
- •Other: cerca in altri markup la descrizione del comportamento
- •None: nessuna informazione

•`xlink:`**`actuate`**`="onRequest|onLoad|other|none"`
`Definisce come il link debba essere attivato.`
- •onRequest: su richiesta dell'utente
- •onLoad: quando il documento è stato letto
- •Other: altro markup
- •None: nessuna informazione

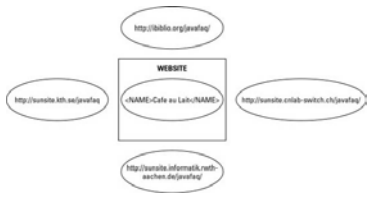
```
<!ELEMENT SIMPLE ANY>
<!ATTLIST SIMPLE
XML-LINK CDATA #FIXED "SIMPLE"
ROLE CDATA #IMPLIED
HREF CDATA #REQUIRED
TITLE CDATA #IMPLIED
INLINE (TRUE|FALSE) "TRUE"
CONTENT-ROLE CDATA #IMPLIED
CONTENT-TITLE CDATA #IMPLIED
SHOW (EMBED|REPLACE|NEW) "REPLACE"
ACTUATE (onRequest|onLoad|) "onRequest"
BEHAVIOR CDATA #IMPLIED
>
```

XLINK

Link Estesi

- Sono rappresentati da un insieme di risorse e connessioni tra esse.
- Le risorse possono essere locali (parte di un elemento XLINK EXT) o remote
  - *Risorsa locale:* contenuta nell'elemento XLINK, con `xlink:xtype="resource"`
  - *Risorsa Remota:* esiste all'esterno dell'elemento XLINK, verosimilmente in un altro documento. Nell'elemento XLINK locale si inseriscono figli con `link:xtype="locator"` e un URI che referenzia la risorsa remota.
- Ogni risorsa può essere sia source che target di link (dipende in quale senso si segue il link).
- Un link out-of-line è caratterizzato dal non avere risorse locali

```
<WEBSITE xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended">
  <NAME xlink:type="resource">Cafe au Lait</NAME>
  <HOMESITE xlink:type="locator" xlink:href="http://ibiblio.org/javafaq/"/>
  <MIRROR xlink:type="locator" xlink:href="http://sunsite.kth.se/javafaq"/>
  <MIRROR xlink:type="locator" xlink:href="http://sunsite.informatik.rwth-aachen.de/javafaq/"/>
  <MIRROR xlink:type="locator" xlink:href="http://sunsite.cnlab-switch.ch/javafaq/"/>
</WEBSITE>
```



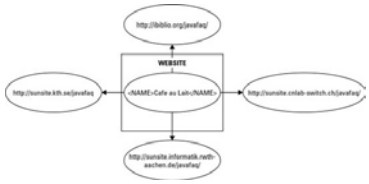
L'elemento WEBSITE contiene una risorsa che ne referenzia altre 4 tramite un URL.

XLINK

ARCS

Nei link estesi è possibile definire comportamenti più complessi di quelli visti nei link semplici per quanto riguarda l'attiivazione di un link. Definendo un elemento `xlink:type="arc"`, si introducono due proprietà `from` e `to` che permettono di indicare come collegare le risorse.

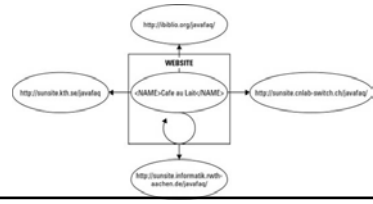
```
<WEBSITE xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended" xlink:title="Cafe au Lait">
<NAME xlink:type="resource" xlink:label="source"> Cafe au Lait </NAME>
<HOMESITE xlink:type="locator" xlink:href="http://ibiblio.org/javafaq/" xlink:label="us"/>
<MIRROR xlink:type="locator" xlink:title="Cafe au Lait Swedish Mirror" xlink:label="se"
xlink:href="http://sunsite.kth.se/javafaq"/>
<MIRROR xlink:type="locator" xlink:title="Cafe au Lait German Mirror" xlink:label="sk"
  xlink:href="http://sunsite.informatik.rwth-aachen.de/javafaq/"/>
<MIRROR xlink:type="locator" xlink:title="Cafe au Lait Swiss Mirror" xlink:label="ch"
  xlink:href="http://sunsite.cnlab-switch.ch/javafaq/"/>
<CONNECTION xlink:type="arc" xlink:from="source" xlink:to="ch" xlink:show="replace" xlink:actuate="onRequest"/>
<CONNECTION xlink:type="arc" xlink:from="source" xlink:to="us" xlink:show="replace" xlink:actuate="onRequest"/>
<CONNECTION xlink:type="arc" xlink:from="source" xlink:to="se" xlink:show="replace" xlink:actuate="onRequest"/>
<CONNECTION xlink:type="arc" xlink:from="source" xlink:to="sk" xlink:show="replace" xlink:actuate="onRequest"/>
</WEBSITE>
```



La definizione di archi multipli è possibile definendo differento elementi con la medesima label

E' possibile omettere le proprietà from, to, o entrambe. In questo caso tutte le risorse sostituiscono l'attributo mancante

```
<CONNECTION xlink:type="arc" xlink:from="source"
xlink:show="replace" xlink:actuate="onRequest"/>
```



XLINK

Out-of-line links

Un link “out-of-line” non contiene nulla delle risorse che connette. I links sono memorizzati in un documento separato chiamato *linkbase*.

Ad esempio la sequenza di una serie di pagine che prevedono una funzione next/previous, può essere memorizzata in un linkbase. L’ordine delle pagine potrà essere cambiato agendo solo sulla linkbase.

Links tra indice e pagine e viceversa:

```
<COURSE xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended">
<TOC xlink:type="locator" xlink:href="index.xml" xlink:label="index"/>
<CLASS xlink:type="locator" xlink:href="week1.xml" xlink:label="class"/>
<CLASS xlink:type="locator" xlink:href="week2.xml" xlink:label="class"/>
<CLASS xlink:type="locator" xlink:href="week3.xml" xlink:label="class"/>
<CLASS xlink:type="locator" xlink:href="week4.xml" xlink:label="class"/>
<CLASS xlink:type="locator" xlink:href="week5.xml" xlink:label="class"/>
<CONNECTION xlink:type="arc" from="index" to="class"/>
<CONNECTION xlink:type="arc" from="class" to="index"/>
</COURSE>
```

Links tra indice e pagine e viceversa:

```
<COURSE xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended">
<CLASS xlink:type="locator" xlink:href="week1.xml" xlink:label="1"/>
<CLASS xlink:type="locator" xlink:href="week2.xml" xlink:label="2"/>
<CLASS xlink:type="locator" xlink:href="week3.xml" xlink:label="3"/>
<CLASS xlink:type="locator" xlink:href="week4.xml" xlink:label="4"/>
<!-- Previous Links -->
<CONNECTION xlink:type="arc" xlink:from="2" xlink:to="1"/>
<CONNECTION xlink:type="arc" xlink:from="3" xlink:to="2"/>
<CONNECTION xlink:type="arc" xlink:from="4" xlink:to="3"/>
<!-- Next Links -->
<CONNECTION xlink:type="arc" xlink:from="1" xlink:to="2"/>
<CONNECTION xlink:type="arc" xlink:from="2" xlink:to="3"/>
<CONNECTION xlink:type="arc" xlink:from="3" xlink:to="4"/>
</COURSE>
```

XPOINTER

Deriva dalla necessità di linkare un elemento all’interno di un documento senza dover intervenire su esso (<A>). Se si dispone di un documento di dimensioni elevate, sarebbe utile poter accedere solamente ad una sezione di questo. tale problematica può essere scomposta in tre parti:

I) *Indirizzamento di un singolo elemento*
Con XPointer è possibile puntare un singolo elemento del documento

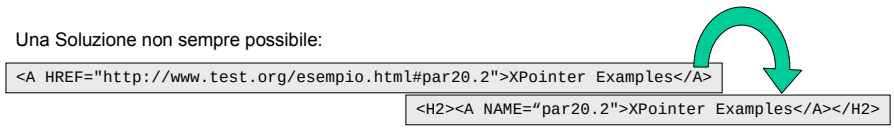
II) *Protocollo*
Deve essere tale per cui il web server sia in grado di restituire solamente una parte del documento.(ricerca)

III) *Integrità dell’informazione*
Come essere certi che l’informazione restituita, tolta dal suo contesto, abbia ancora significato? (ricerca)

Cosa può fare ora Xpointer?
Può essere utilizzato come indice all’interno di un documento che viene letto completamente e posizionato nel punto indicato da Xpointer.

XPOINTER

Una Soluzione non sempre possibile:



Inoltre non è XML-oriented. Infatti un ancora nel documento non dice nulla sul documento stesso, ma serve all'esterno per le referenze. *Xpointer permette di linkare non un punto del documento, ma un intero elemento oppure un range di testo tra due elementi.*

Esempi:

`xpointer(id("ebnf")) oppure ebnf`
elemento con ID uguale a ebnf
`xpointer(descendant::language[position()=2])`
secondo elemento di nome language
`xpointer(/child::spec/child::body/child::*/child::language[2])`
`xpointer(/spec/body/*/language[2])`
secondo elemento language di qualunque filgio di body che discende da spec.
`/1/14/2`
secondo figlio del quattordicesimo figlio della root.
`xpointer(id("ebnf"))xpointer(id("EBNF"))`
seleziona ebnf. Se non c'è cerca EBNF

XPOINTER

Il documento viene specificato da un URI presente in un elemento Xlink. A questo elemento succesivamente si applica Xpointer,

```
http://test.org/test.xml#xpointer(id("ebnf"))
http://test.org/test.xml#xpointer(descendant::language[position()=2])
http://test.org/test.xml#ebnf
http://test.org/test.xml#xpointer(/child::spec/child::body/child::*/child::language[2])
http://test.org/test.xml#xpointer(/spec/body/*/language[2])
http://test.org/test.xml#/1/14/2
http://test.org/test.xml#xpointer(id("ebnf"))xpointer(id("EBNF"))
```

```
<SPECIFICATION xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="simple"
xlink:href="http://test.org/test.xml#xpointer(id('ebnf'))"
xlink:actuate="onRequest"
xlink:show="replace">
Extensible Markup Language (XML) 1.0
</SPECIFICATION>
```

XPOINTER

Ranges e points

Indirizzamento di NODI. (come con Xpath)

```
<AAA>
   <BBB bbb="111">Text in the first element BBB.</BBB>
   <BBB bbb="222">Text in another element BBB.
     <DDD ddd="999">Text in more nested element.</DDD>
   </BBB>
   <CCC ccc="123" xxx="321">Again some text in some element.</CCC>
 </AAA>
```

X = point
container node
locations
X = collapsed range

Indirizzamento di RANGES. (non possibile con Xpath)

```
<AAA>
   <BBB bbb="111">Text in the first element BBB.</BBB>
   <BBB bbb="222">Text in another element BBB.
     <DDD ddd="999">Text in more nested element.</DDD>
   </BBB>
   <CCC ccc="123" xxx="321">Again some text in some element.</CCC>
 </AAA>
```

Indirizzamento di POINTS. (non possibile con Xpath)

```
<AAA>
   <BBB bbb="111">Text in the first element BBB.</BBB>
   <BBB bbb="222">Text in another element BBB.
     <DDD ddd="999">Text in more nested element.</DDD>
   </BBB>
   <CCC ccc="123" xxx="321">Again someX text in some
element.</CCC>
 </AAA>
```


XPOINTER

Definizione di node location:NODE-POINT
Una location di tipo point è definita da un nodo (container node) che contiene il point, e da un indice.

Un indice pari a 0, indicail punto prima di ogni figlio.
XPointer: **xpointer(start-point(//AAA))**
XPointer (alternative): **xpointer(start-point(range(//AAA/BBB[1])))**

```
<AAA>X
   <BBB bbb="111">Text.</BBB>
   <BBB bbb="222">Text.
      <DDD ddd="999">Text.</DDD>
   </BBB>
   <CCC ccc="123" xxx="321">Text.</CCC>
 </AAA>
```

X = point
container node
locations
X = collapsed range

Un indice n diverso da zero indica il punto subito dopo l'n-esimo figlio.
XPointer: **xpointer(end-point(range(//AAA/BBB[2])))**
XPointer (alternative): **xpointer(start-point(range(//AAA/CCC)))**

```
<AAA>
   <BBB bbb="111">Text.</BBB>
   <BBB bbb="222">Text.
      <DDD ddd="999">Text.</DDD>
   </BBB>X
   <CCC ccc="123" xxx="321">Text.</CCC>
 </AAA>
```

XPOINTER

Definizione di node location:CHARACTER-POINT
Quando il container node non può avere figli, come un nodo testo, l'indice diventa un indice all'interno della stringa e il punto in questione è di tipo character.

Un indice pari a 0, indicail punto prima di ogni figlio.
XPointer: **xpointer(start-point(string-range(//*,'another', 3, 0)))**

```
<AAA>
  <BBB bbb="111">
    Text in the first element BBB.</BBB>
  <BBB bbb="222"> Text in anXother element BBB.<DDD
ddd="999">
      Text in more nested element.</DDD>
  </BBB>
  <CCC ccc="123" xxx="321">
    Again some text in some element.</CCC>
</AAA>
```

X = point
container node
locations
X = collapsed range

XPOINTER

Location sets: node-type locations
Location set is an ordered list of document nodes, points, and/or ranges.
XPointer: **xpointer(//AAA/BBB)**

```
<AAA>
   <BBB bbb="111">Text in the first element BBB.</BBB>
   <BBB bbb="222">Text in another element BBB.
     <DDD ddd="999">Text in more nested element.</DDD>
   </BBB>
   <CCC ccc="123" xxx="321">Again some text in some element.</CCC>
</AAA>
```

Location sets: range-type locations
Location set is an ordered list of document nodes, points, and/or ranges.
XPointer: **xpointer(string-range(//*,'element'))**

```
<AAA>
   <BBB bbb="111">Text in the first element BBB.</BBB>
   <BBB bbb="222">Text in another element BBB.
     <DDD ddd="999">Text in more nested element.</DDD>
   </BBB>
   <CCC ccc="123" xxx="321">Again some text in some element.</CCC>
</AAA>
```

Location sets: point-type locations
Location set is an ordered list of document nodes, points, and/or ranges.
XPointer: **xpointer(start-point(string-range(//*,'element')))**

```
<AAA>
   <BBB bbb="111">Text in the first Xelement BBB.</BBB>
   <BBB bbb="222">Text in another Xelement BBB.
     <DDD ddd="999">Text in more nested Xelement.</DDD>
   </BBB>
   <CCC ccc="123" xxx="321">Again some text in some Xelement.</CCC>
</AAA>
```

X = point
container node
locations
X = collapsed range

XSL:FO

•XSL Formatting Objects descrive come le pagine debbano essere visualizzate ad un lettore.

•Attualmente i browser non supportano XSL-FO possiamo però rappresentarli in formati alternativi (PDF)

•Mette a disposizione funzionalità molto più complesse di quanto non faccia HTML+CSS

•XSL è costantemente in revisione

•Esistono numerosi formatting objects presenti nel namespace http://www.w3.org/1999/XSL/Format

XSL:FO

Il modello di XSL-FO è basato su boxes rettangolari chiamati **AREA** che possono contenere testo, spazi vuoti, immagini o altri FO. Un XSL formatter legge i FO's per determinare quale area mettere in quale punto della pagina. Ogni pagina può contenere differenti AREE:

REGIONS
rappresenta il conteiner di livello superiore. Una pagina di un libro può essere vista come composta da tre regioni: header, body, footer.
`(fo:region-body, fo:region-before, fo:region-after, fo:region-start, and fo:region-end)`

BLOCK AREAS
rappresenta un paragrafo o una lista di elementi. Viene posizionata sequenzialmente nell'area che lo contiene. Se un blocco prima, viene eliminato e se ne si aggiunge uno le posizioni shiftano di conseguenza.
`( fo:block, fo:table-and-caption, and fo:list-block)`

LINE AREAS
rappresenta una linea di testo all'interno di un blocco. È il formatter che decide come scomporre il teso (linea area) in base alle dimensioni del blocco.

INLINE AREAS
rappresenta singoli caratteri, note a piè di pagina, equazioni matematiche.
`fo:character, fo:external-graphic, fo:inline, fo:instream-foreign-object, fo:leader, and fo:page-number.`

XSL:FO

XSL-FO è un vocabolario XML completo per la rappresentazione di testo su su una pagina. Per convenzione i file che contengono formatting objects devono avere l'estensione .**fob** oppure .**fo**
Nell'esempio abbiamo che :

-la root del documento è rappresentata da *fo:root*
che contiene
-*fo:layout-master-set* contiene template per la pagina che verrà creata: contiene l'oggetto:
-*fo:simple-page-master*: descrive il tipo di pagina che conterrà il content
-*fo:page-sequence*: defiinisce le pagine nelle quali verrà inserito il content. L'attributo master-name definisce quale pagina dovrà essere utilizzata
-*fo:flow* definisce all'interno della pagina quale è il contenuto.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:output indent="yes"/>
<xsl:template match="/">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master master-name="only">
        <fo:region-body/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-name="only">
      <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates select="//ATOM"/>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>
<xsl:template match="ATOM">
  <fo:block font-size="20pt" font-family="serif" line-height="30pt">
    <xsl:value-of select="NAME"/>
  </fo:block>
</xsl:template>
</xsl:stylesheet>
```

Definizione del layout

Master page container

content

formattazione

XSL:FO

Root Element

Serve per la definizione del namespace. Non ha effetto sulla pagina

Simple page masters

I template di pagina sono chiamati *page masters*. Definiscono il layout in termini di margini, dimensione dell'header, footer. *Fo:simple-page-master* definisce una pagina rettangoplare. *Fo:layout-master-set* contiene diverse master oages.
Attributi:
master-name / page-height / page-width
margin-bottom / margin-left / margin-top / writing-mode / reference-orientation

Regions

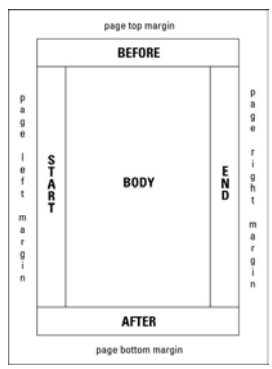
Fo:region-before extent="1.oin"
fo:region-after extent="1.0in"
fo:region-body margin-top="1.0in" margin-bottom="1.0in"
fo:region-start
fo:region-end

Page sequences

Ogni pagina nella sequenza ha una page master (layout) associata. Si possono avere diverse master page per differenti sequenze di pagine. Una sequence può avere tre figli:
- *fo:title* titolo del documento
- *fo:static-content (*tipo titolo del capitolo: si ripete in diverse pagine)
- *fo:flow* t (non si ripete in altre pagine)

Flows

Fo:flow indica il contenuto che verrà messo nelle istanze delle master pages. I contenuti vengono specificati dagli elementi *fo:block, fo:block-container, fo:table-and-caption, fo:table, and fo:list-block elements*. L'attributo *flow-name* di fo:flow specifica in quale regione debba essere messo il contenuto.
xsl-region-body,xsl-region-before ,xsl-region-after ,xsl-region-start ,xsl-region-end



XSL:FO

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0“
  xmlns:xsl=http://www.w3.org/1999/XSL/Transform
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master master-name="A4“
           page-width="297mm"  page-height="210mm“
           margin-top="0.5in"  margin-bottom="0.5in“
           margin-left="0.5in" margin-right="0.5in">
          <fo:region-before extent="1.0in"/>
          <fo:region-body margin-top="1.0in“
                          margin-bottom="1.0in"/>
          <fo:region-after  extent="1.0in"/>
        </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-name="A4“
        initial-page-number="1" language="en" country="us">
        <fo:static-content flow-name="xsl-region-before">
          <fo:block>The Periodic Table</fo:block>
        </fo:static-content>
        <fo:static-content flow-name="xsl-region-after">
          <fo:block><fo:leader leader-pattern="rule“
                               leader-length="18cm" />
          </fo:block>
          <fo:block>p. <fo:page-number/></fo:block>
        </fo:static-content>
        <fo:flow flow-name="xsl-region-body">
          <xsl:apply-templates select="//ATOM"/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
  <xsl:template match="ATOM">
    <fo:block><xsl:value-of select="NAME"/></fo:block>
  </xsl:template>
</xsl:stylesheet>
```

In questo esempio, si introducono i concetti di:

- Page numbering
- Static content

XSL:FO

CONTENT

XSL:FO mette a disposizione differenti elementi di rappresentazione dei contenuti.

BLOCK-LEVEL Formatting Objects
realizza un'area rettangolare separata da un line break dalle aree che lo precedeno o lo seguono
fo:block, fo:block-container , fo:table-and-caption , fo:table ,fo:list-block

InLine Formatting Objects
mette una linea di testo, una dopo l'altra in base allo stile di scrittura impostato (sx->dx oppure dx->sx)

Table Formatting Objects
per la realizzazione di tabelle

Out-Of-Line Formatting Objects