Introduzione

HTML
rappresentazione standardizzata delle informazioni per il web

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

Marcatori non standard per introdurre nuove funzionalità

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)

SGML

Metalinguaggio (padre di HTML)

PREGI

Potente e flessibile

(standard iso, espandibile, fortemente strutturato, non proprietario, indipendente dalla piattaforma)

DIFETTI

Struttura pesante

Sono obbligatori un DTD e uno StyleSheet

Obbligatoria la VALIDAZIONE del documento

Le istanze SGML sono troppo pesanti e non robuste per applicazioni WEB

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 informazione Sulla semantica. L'elaborazione dei dati è demandata a livello Java, JavaScript…
Html è diventato un linguaggio di presentazione invece di rimanere un linguaggio di descrizione	

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!

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)

obiettivi progettuali di XML

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

obiettivi progettuali di XML

- Chiari e Leggibili
- Progettazione rapida
- Linguaggio Conciso
- Facilità di creazione
- I markup sono gratis. Non risparmiamo

Sintassi XML

Simile all'HTML ma maggiormente rigida.

Aumenta la leggibilità del documento

Sintassi XML

Definizione di elementi e attributi personalizzati.

DTD

La creazione di un DTD personale è OPZIONALE!

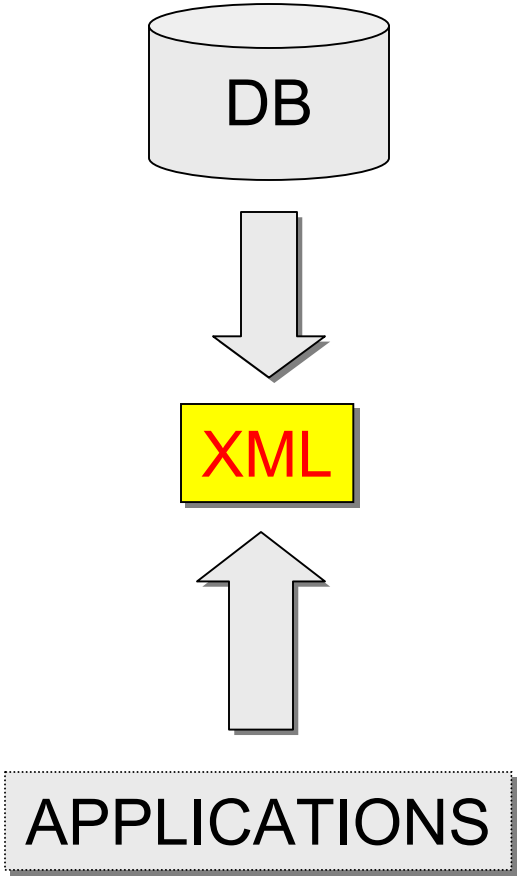
Sintassi XML

I TAG sono personalizzabili

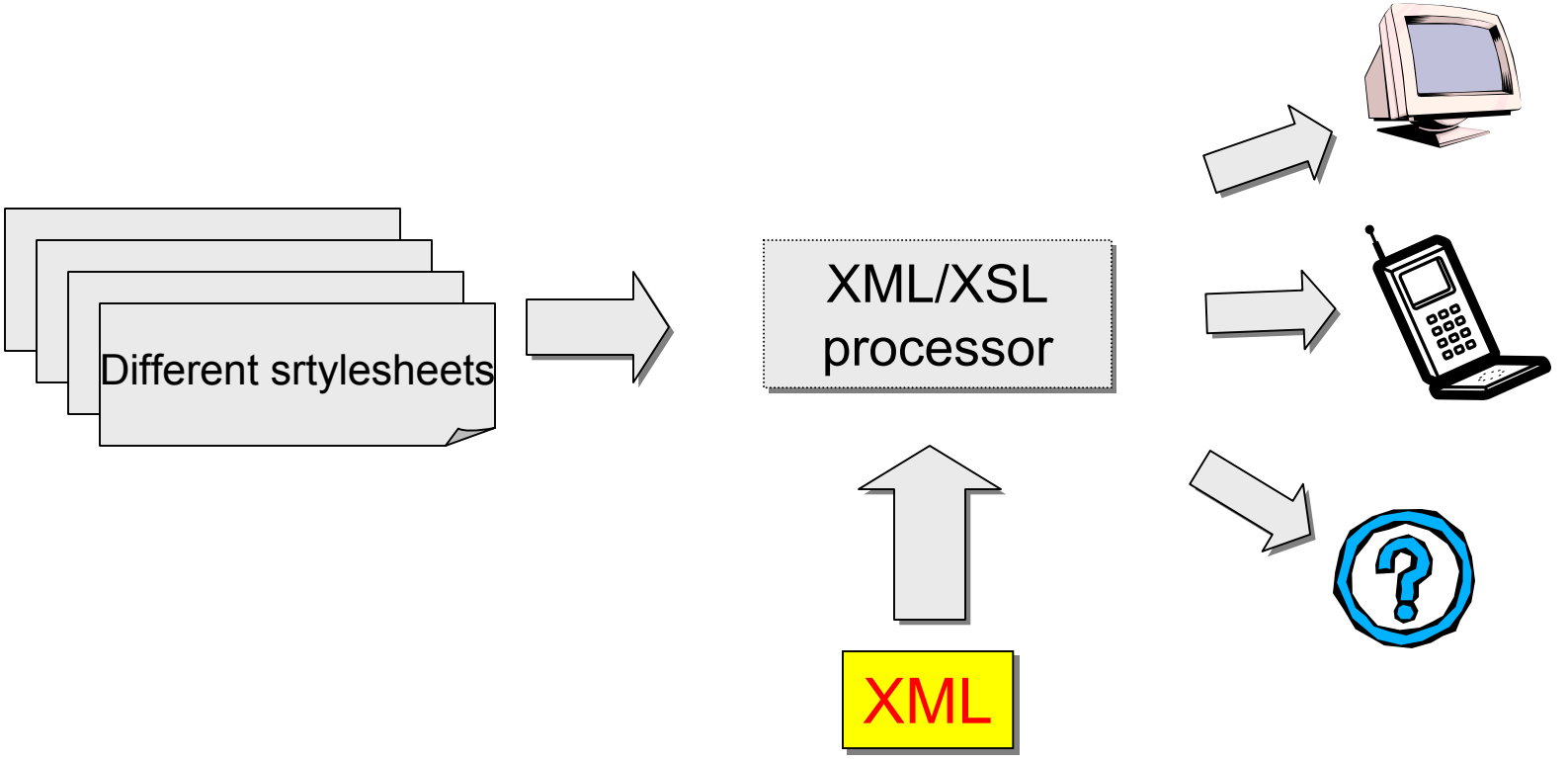
Diversi documenti possono utilizzare i medesimi tag

Necessità di un NameSpace

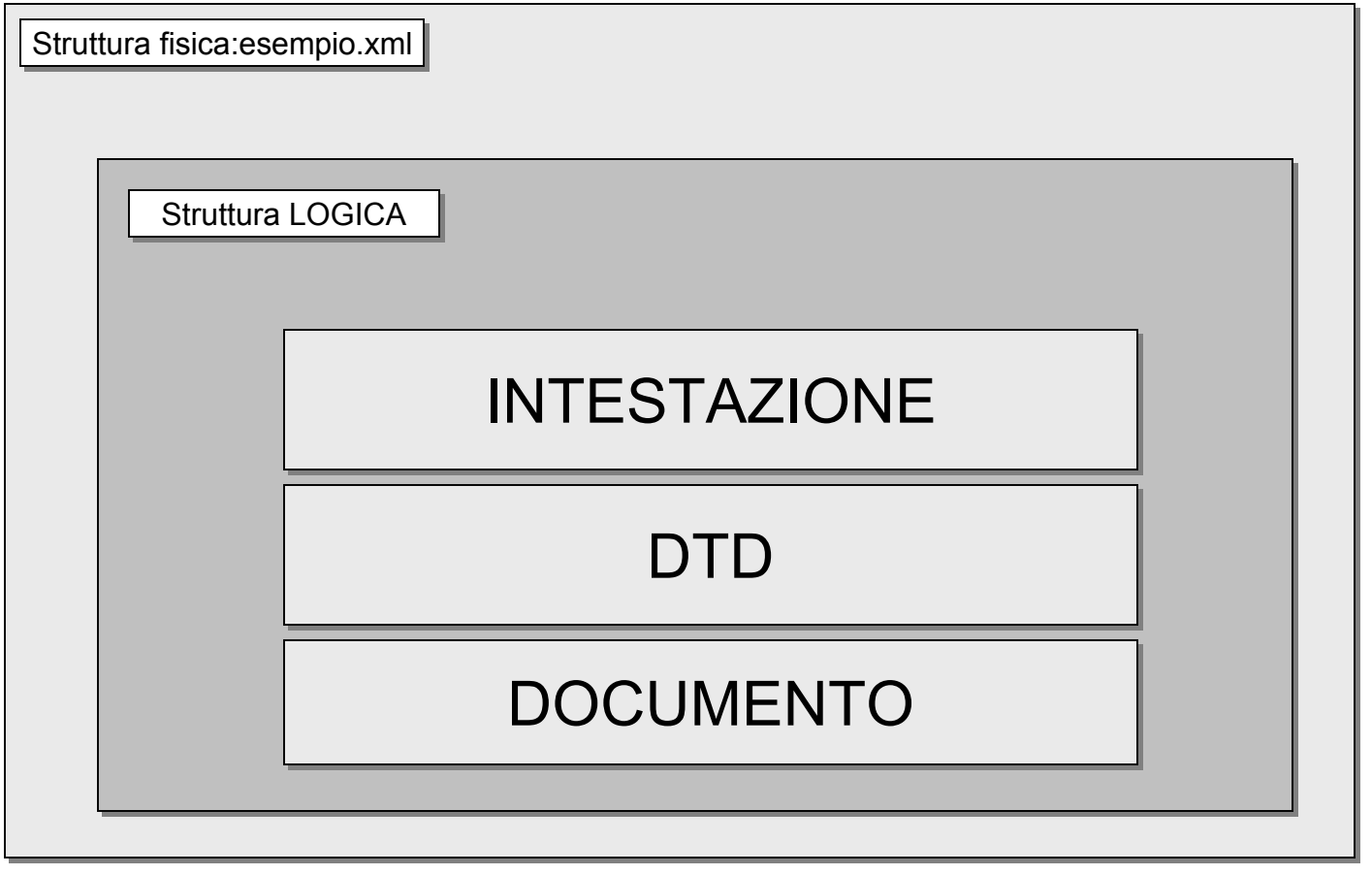
XML per lo scambio di informazioni



La Rappresentazione dei Dati (delivery context)



Struttura XML



Struttura Logica

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

Dichiarazione XML

```
<!DOCTYPE Wildflowers SYSTEM "Wldflr.dtd">
```

Dichiarazione tipo documento (DTD)

```
<PLANT>
      <COMMON>Columbine</COMMON>
      <BOTANICAL>Aquilegia canadensis</BOTANICAL>
</PLANT>
```

DOCUMENTO

Le entità in XML

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

Le entità in XML

entità analizzabili (*parsed)*

in grado di essere letta dall'elaboratore che ne consente l'astrazione e la sostituzione con il corrispondente oggetto richiamato

```
<!ENTITY myName "Alessio Ravani">
```

entità NON analizzabili (*unparsed)*

dati binari o testi non conformi alle regole XML. L'oggetto di riferimento viene letto da un'applicazione in grado di farlo

```
<!ENTITY MyImage SYSTEM "Image.gif" NDATA GIF>

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

Le entità in XML

entità predefinite

< < (parentesi angolare di apertura)

> > (parentesi angolare di chiusura)

&amp ; & (e commerciale)

' ‘ (apostrofo)

" " (virgolette doppie)

Le entità in XML

entità interne

(dichiarata all'interno del documento)

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

Entità esterne

(fa riferimento ad un file esterno al documento)

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

La Sintassi XML

Tutti i tag DEVONO essere chiusi

(HTML meno rigido al riguardo)

TAG di elemento Vuoto

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

```
<EMPTY/>
```

attributi

```
<A HREF = "http://java.sun.com">
java Home Page
</A>
```

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


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

La Sintassi XML

Documento VALIDO

Deve rispettare le regole imposte dal DTD

Documento BEN FORMATO

1. La dichiarazione XML (se c'è) deve comparire come primo elemento del documento

2. Elementi che contengono dati devono avere sia il tag di apertura che quello di chiusura.

3. Elementi che non contengono dati e utilizzano un tag singolo devono terminare con />

4. il documento deve contenere solamente un elemento che contiene completamente tutti gli altri

5. Gli elementi possono essere innestati ma non sovrapposti

6. I valori degli attributi devono essere messi tra apici

7. i caratteri < and & possono essere utilizzati solamente come inizio di un tag e per le entity.

8. Gli unici riferimenti a entità sono &, <, >, ' ".

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

Solo un elemento di root è ammesso
???
<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>
```

```
<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>
```

CDATA: sezioni di testo che il parser XML non tenta di interpretare

```
<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

esercitazione

Evoluzione di un documento XML

```
modulo3/xml/prodsa.txt
modulo3/xml/prodsb.xml
modulo3/xml/prodsc.xml
modulo3/xml/prodsd.xml
modulo3/xml/prodse.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>
```

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

Attributi e elementi: scelte stilistiche

Visibilità
elemento: dati che devono essere visualizzati (nome di un prodotto)
a*ttributo*: dati interni (codice 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 avrà scopi applicativi (attributo)

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

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 diversi posti (vedi costanti)

Riferimento allo stesso oggetto in diversi documenti

(possono essere normalizzati anche stylesheets e DTDs)

Progettazione di una struttura dati XML

Quando si crea un documento XML ci si deve chiedere se è stato modellato nel modo più accessibile ed efficiente.

Il design di strutture XML deve tenere in considerazione i seguenti aspetti:

- **Audience**
- **Performance**
- **Data Modelling vs Representation Modelling**

Progettazione di una struttura dati XML

Audience

Il documento deve essere human-readable?

Scopo principale del documento: display o processing

Quanti utilizzatori?

Deve rispettare uno standard?

Progettazione di una struttura dati XML

performance

```
<Invoice customerName="Kevin Williams">
  <LineItem
      productName="Grommets (2 inch)"
      quantity="17"
      price="0.10" />
</Invoice>
```

Questo documento è chiaro e leggibile ma questo potrebbe non essere un fattore di interesse.

```
<I c="c17"><L p="p22" q="17" pr=".1" /></I>
```

Stesso contenuto informativo per processi tra sistemi.

Quale rappresentazione utilizzare dipende dai casi.

Progettazione di una struttura dati XML

Modellazione sui dati o sulla rappresentazione ?

```
Widgets, Inc.
Invoice

Customer:   Kevin Williams
            742 Evergreen Terrace
            Springfield, KY 12345

Ship to:    Kevin Williams
            742 Evergreen Terrace
            Springfield, KY 12345
Shipper:    FedEx

Item Code   Description          Quantity       Price         Total
1A2A3AB     Widget (3 inch)         17          $0.10         $1.70
2BC3DCB     Grommet (2 inch)        22          $0.05         $1.10

Total                                                         $2.80
```

Progettazione di una struttura dati XML

```
<Invoice>
Widgets, Inc.
Invoice

Customer:    <customerName>Kevin Williams</customerName>
             <orderAddress>742 Evergreen Terrace</orderAddress>
             <orderCity>Springfield</orderCity>,
             <orderState>KY</orderState>
             <orderPostalCode>12345</orderPostalCode>
Ship to:     <shipName>Kevin Williams</shipName>
             <shipAddress>742 Evergreen Terrace</shipAddress>
             <shipCity>Springfield</shipCity>,
             <shipState>KY</shipState>
             <shipPostalCode>12345</shipPostalCode>
Shipper:     <shippingCompany>FedEx</shippingCompany>

Item Code    Description          Quantity       Price          Total
   <LineItem>
      <itemCode>1A2A3AB</itemCode>
      <itemDescription>Widget (3 inch)</itemDescription>
      <quantity>17</quantity>
      <price>$0.10</price>
      <linePrice>$1.70</linePrice>
   </LineItem>
   <LineItem>
      <itemCode>2BC3DCB</itemCode>
      <itemDescription>Grommet (0.5 inch)</itemDescription>
      <quantity>22</quantity>
      <price>$0.05</price>
      <linePrice>$1.10</linePrice>
   </LineItem>

Total
   <totalPrice>$2.80</totalPrice>
</Invoice>
```

Questa modellazione ha diversi problemi:

- Contiene informazioni di formattazione
- Contiene informazioni di *riassunto*
- Contiene informazioni sulla formattazione dei campi

Progettazione di una struttura dati XML

eliminiamo tutte le informazioni di formattazione

```
<Invoice
   customerName="Kevin Williams">
   <Address
     addressType="billing" ⇦
     street="742 Evergreen Terrace"
     city="Springfield"
     state="KY"
     postalCode="12345" />
   <Address
     addressType="shipping" ⇦
     street="742 Evergreen Terrace"
     city="Springfield"
     state="KY"
     postalCode="12345" />
   <LineItem
      itemCode="1A2A3AB"
      itemDescription="Widget (3 inch)"
      quantity="17"
      price="0.10"
      currency="USD" /> ⇦
   <LineItem
      itemCode="2BC3DCB"
      itemDescription="Grommet (0.5 inch)"
      quantity="22"
      price="0.05"
      currency="USD" /> ⇦
</Invoice>
```

Progettazione di una struttura dati XML

Approccio a *Elements*

```
<Invoice>
   <customerName>Kevin Williams</customerName>
   <billingAddress>742 Evergreen Terrace</billingAddress>
   <billingCity>Springfield</billingCity>
   <billingPostalCode>12345</billingPostalCode>
   <shippingAddress>742 Evergreen Terrace</shippingAddress>
   <shippingCity>Springfield</shippingCity>
   <shippingState>KY</shippingState>
   <shippingPostalCode>12345</shippingPostalCode>
   <shippingCompany>FedEx</shippingCompany>
   <LineItem>
      <itemCode>1A2A3AB</itemCode>
      <itemDescription>Widget (3 inch)</itemDescription>
      <quantity>17</quantity>
      <price>0.10</price>
   </LineItem>
   <LineItem>
      <itemCode>2BC3DCB</itemCode>
      <itemDescription>Grommet (0.5 inch)</itemDescription>
      <quantity>22</quantity>
      <price>0.05</price>
   </LineItem>
</Invoice>
```

Progettazione di una struttura dati XML

Approccio a *Attributes*

```
<Invoice
   customerName="Kevin Williams"
   billingAddress="742 Evergreen Terrace"
   billingCity="Springfield"
   billingState="KY"
   billingPostalCode="12345"
   shippingAddress="742 Evergreen Terrace"
   shippingCity="Springfield"
   shippingState="KY"
   shippingPostalCode="12345"
   shippingCompany="FedEx">
   <LineItem
      itemCode="1A2A3AB"
      itemDescription="Widget (3 inch)"
      quantity="17"
      price="0.10" />
   <LineItem
      itemCode="2BC3DCB"
      itemDescription="Grommet (0.5 inch)"
      quantity="22"
      price="0.05" />
</Invoice>
```

Progettazione di una struttura dati XML

confronto

	ELEMENTS	**ATTRIBUTES**
leggibilità	ALTA	ALTA
Compatibilità database (*)	ambiguo	Struttura e contenuti sono ben distinti
Data typing	Con i DTD pochi tipi di dato	Possibilità di definire delle liste di valori ammissibili
complessità	DOM: meno passi con gli attributi ma sostanzialmente stesse performance. SAX: meno passi con gli attributi ma sostanzialmente stesse performance.	
Dimensioni del documento	Presenza di più tag di chiusura	Meno tag
Un corretto utilizzo degli attributi è sicuramente conveniente per il risultato finale della modellazione del documento.		

(*) utilizzando solamente *elementi* diventa più difficile capire quando si descrive un contenuto o una relazione. Gli attributi possono essere messi in qualunque ordine, mentre l'ordinamento degli elementi a volte aggiunge complessità.

Progettazione di una struttura dati XML

Confronto dei passi necessari con le tecnologie DOM e SAX per accedere alla quantità del primo item ordinato di un ordine

ELEMENTS	ATTRIBUTES
1. Get the root element (the `Invoice` element). 2. Go to the first `LineItem` child of the `Invoice` element. 3. Go to the `quantity` child of the `LineItem` element. 4. Go to the text node child of the `quantity` element and return its value.	1. Get the root element (the `Invoice` element). 2. Get the first `LineItem` child of the `Invoice` element. 3. Iterate through the `LineItem` element for the attribute name-value pair list of the `quantity` attribute, and return its value.

La soluzione ad attributi richiede meno step. Essendo l'albero DOM in memoria le performance non cambiano in modo significativo.

Progettazione di una struttura dati XML

ELEMENTS	ATTRIBUTES
1. Before starting, set the Booleans `bInInvoice`, `bInLineItem`, and `bInQuantity` to false, and also, set the value of the counter `iLineItem` to zero and the string `sQuantity` to a blank string. 2. In the `startElement` event, when the `Invoice` start tag is encountered, set the value of the Boolean variable `bInInvoice` to `true`. 3. In the `startElement` event, when the `LineItem` start tag is encountered, set the value of the Boolean variable `bInLineItem` to `true` if `bInInvoice` is `true`, indicating that the processor window is inside a `LineItem` tag. Also increment the `iLineItem` counter to indicate which `LineItem` is being read. 4. In the `startElement` event, when the `quantity` start tag is encountered, if the Boolean `bInLineItem` is `true` and the `iLineItem` counter is `1`, set the value of the `bInQuantity` Boolean variable to `true`. 5. In the `characters` event, if `bInQuantity` is `true`, append the received text to the `sQuantity` string. 6. In the `endElement` event, if the `quantity` end tag is encountered, set the `bInQuantity` Boolean to `false`. The value for the quantity is now available for use. 7. In the `endElement` event, if the `LineItem` end tag is encountered, set the `bInLineItem` Boolean to `false`. In addition, if the `Invoice` end tag is encountered, set the `bInInvoice` Boolean to `false`.	1. Before starting, set the Boolean `bInInvoice` to `false`, and also set the value of the counter `iLineItem` to zero. 2. In the `startElement` event, when the `Invoice` start tag is encountered, set the value of the Boolean variable `bInInvoice` to `true`. 3. In the `startElement` event, when the `LineItem` start tag is encountered, if `bInInvoice` is `true`, increment the value of the `iLineItem` counter. If this value is `1`, pull the value of the `quantity` attribute from the attribute name-value pair set provided as a parameter to the `startElement` event. The value for the quantity is now available for use. 4. In the `endElement` event, if the `LineItem` end tag is encountered, set the `bInLineItem` Boolean to `false`. 5. In the `endElement` event, if the `Invoice` end tag is encountered, set the `bInInvoice` Boolean to `false`.

La soluzione ad attributi richiede meno step

Progettazione di una struttura dati XML

SCOPE

Il primo passo è quello di comprendere correttamente gli scopi e l'utilizzo del documento

Deve rispettare uno standard?
Chi utilizzerà il documento?
A cosa serve? (archivio, presentazione…)

STANDARD nel DESIGN di una struttura XML

•*System Internal Standards*
utilizzo ristretto ad un team di persone. Il più semplice da sciluppare perché tutti hanno gli stessi scopi

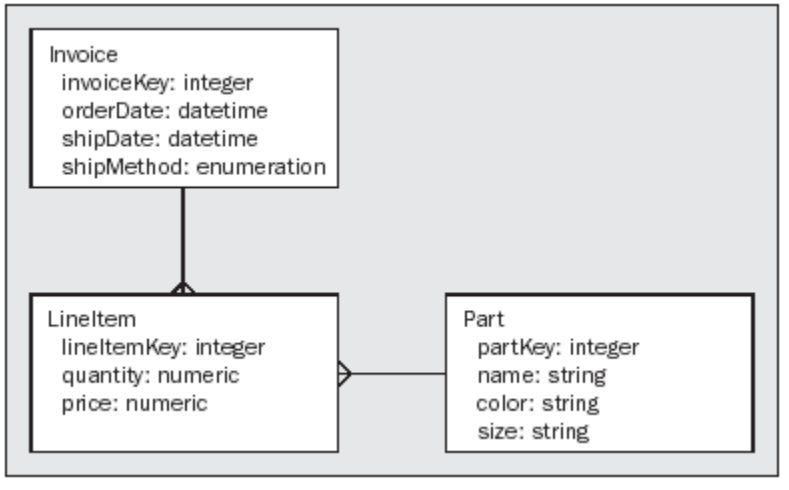
•*Cross-System Standards*
diversi team devono concordare su uno standard per la comunicazione di due sistemi anche molto diversi tra loro

•*Industry-level Standards*
il più complesso. La struttura sviluppata coinvolge molte e differenti persone accomunate da un medesimo scopo comune. Si necessità una *visione* che tenga conto anche di esigenze future.

Progettazione di una struttura dati XML

Utilizzo del Documento

ARCHIVIO di INFORMAZIONI




```
<!ELEMENT Invoice (Customer, LineItem+)>
<!ATTLIST Invoice
   orderDate CDATA #REQUIRED
   shipDate CDATA #REQUIRED
   shipMethod CDATA #REQUIRED>
<!ELEMENT Customer EMPTY>
<!ATTLIST Customer
   name CDATA #REQUIRED
   address CDATA #REQUIRED
   city CDATA #REQUIRED
   state CDATA #REQUIRED
   zip CDATA #REQUIRED>
<!ELEMENT LineItem EMPTY>
<!ATTLIST LineItem
   partKey CDATA #REQUIRED
   quantity CDATA #REQUIRED
   price CDATA #REQUIRED>
```

Questa soluzione lega i dettagli ad una chiave del sistema. Nel tempo il sistema originale potrebbe cambiare o cambiare le informazioni

```
<!ELEMENT Invoice (Customer, LineItem+)>
<!ATTLIST Invoice
   orderDate CDATA #REQUIRED
   shipDate CDATA #REQUIRED
   shipMethod CDATA #REQUIRED>
<!ELEMENT Customer EMPTY>
<!ATTLIST Customer
   name CDATA #REQUIRED
   address CDATA #REQUIRED
   city CDATA #REQUIRED
   state CDATA #REQUIRED
   zip CDATA #REQUIRED>
<!ELEMENT LineItem EMPTY>
<!ATTLIST LineItem
   partNumber CDATA #REQUIRED
   name CDATA #REQUIRED
   color CDATA #REQUIRED
   size CDATA #REQUIRED
   quantity CDATA #REQUIRED
   price CDATA #REQUIRED>
```

Il documento deve essere, per quanto possibile *self-contained* in modo da assicurare consistenza dei dati nel tempo.

Progettazione di una struttura dati XML

Utilizzo del Documento
TRASMISSIONE di INFORMAZIONI

Documenti il cui scopo è quello trasmettere informazioni attraverso due sistemi. Possono contenere delle referenze interne che non hanno significato nel contesto del documento stesso. Se si ipotizza di realizzare differenti transazioni serve un meccanismo di *envelope* che descriva il comportamento che ci aspettiamo dal sistema ricevente.

```
<!ELEMENT PartRequest (Part)>
<!ATTLIST PartRequest
   requestType (UpdatePart | GetCurrentPrice) #REQUIRED
   requestKey CDATA #REQUIRED>
<!ELEMENT Part EMPTY>
<!ATTLIST Part
   partKey CDATA #REQUIRED
   partNumber CDATA #IMPLIED
   name CDATA #IMPLIED
   color CDATA #IMPLIED
   size CDATA #IMPLIED>
```

```
<PartRequest
   requestType="UpdatePart"
   requestKey="1028">
   <Part
      partKey="17"
      partNumber="1A2A3AB"
      name="Sprocket"
      color="Blue"
      size="2 in." />
</PartRequest>
```

REQUEST

```
<!ELEMENT PartResponse EMPTY>
<!ATTLIST PartResponse
   requestKey CDATA #REQUIRED
   status (Success | Failure) #REQUIRED
   price CDATA #IMPLIED>
```

```
<PartResponse
   requestKey="1028"
   status="Failure" />
```

```
<PartResponse
   requestKey = "1028"
   status="Success" />
```

RESPONSE

Ridurre le dimensioni del documento XML significa ridurre l'utilizzo di memoria in caso si usi DOM, ridurre l'occupazione di banda ma ridurre anche la leggibilità. Trade-off.

Progettazione di una struttura dati XML

Utilizzo del Documento
PRESENTAZIONE

Invoice
Order date: 12/1/2000
Ship date: 12/4/2000
Ship method: UPS

Part	Quantity	Unit Price	Price
2 in. blue grommet	17	0.10	1.70
3 in. silver widget	22	0.20	4.40
Total			6.10

Invoice
Order date: 12/2/2000
Ship date: 12/5/2000
Ship method: USPS

Part	Quantity	Unit Price	Price
1 in. red sprocket	13	0.30	3.90
2 in. blue grommet	11	0.10	1.10
Total			5.00

Il documento XML dovrebbe essere tale da rendere minime le trasformazioni XSLT per la rappresentazione finale. La prima soluzione richiede una navigazione attraverso puntatori. La seconda soluzione è più semplice e più performante.

```
<InvoiceData>
   <Invoice
      orderDate="12/1/2000"
      shipDate="12/4/2000"
      shipMethod="UPS">
      <LineItem
         partIDREF="p1"
         quantity="17"
         price="0.10" />
      <LineItem
         partIDREF="p2"
         quantity="22"
         price="0.20" />
   </Invoice>
   <Invoice
      orderDate="12/2/2000"
      shipDate="12/5/2000"
      shipMethod="USPS">
      <LineItem
         partIDREF="p3"
         quantity="13"
         price="0.30" />
      <LineItem
         partIDREF="p1"
         quantity="11"
         price="0.10" />
   </Invoice>
   <Part
      partID="p1"
      name="qrommet"
      size="2 in."
      color="blue" />
   <Part
      partID="p2"
      name="widget"
      size="3 in."
      color="silver" />
   <Part
      partID="p3"
      name="sprocket"
      size="1 in."
      color="red" />
</InvoiceData>
```

```
<InvoiceData>
   <Invoice
      orderDate="12/1/2000"
      shipDate="12/4/2000"
      shipMethod="UPS"
      total="6.10">
      <LineItem
         partDescription="2 in. blue grommet"
         quantity="17"
         price="0.10"
         linePrice="1.70" />
      <LineItem
         partDescription="3 in. silver widget"
         quantity="22"
         price="0.20"
         linePrice="2.20" />
   </Invoice>
   <Invoice
      orderDate="12/2/2000"
      shipDate="12/5/2000"
      shipMethod="USPS"
      total="5.00">
      <LineItem
         partDescription="1 in. red sprocket"
         quantity="13"
         price="0.30"
         linePrice="3.90" />
      <LineItem
         partDescription="2 in. blue grommet"
         quantity="11"
         price="0.10"
         linePrice="1.10" />
   </Invoice>
</InvoiceData>
```

Progettazione di una struttura dati XML

Le conseguenze di una determinata decisione di design
PERFORMANCE

Dimensioni del documento

Le dimensioni del documento si ripercuotono sul consumo di memoria nel caso si utilizzi DOM.

Le soluzioni possono essere:

- Ridurre le dimensioni dei tag
- Agire sull'hardware
- Ridurre lo scope della struttura
- Utilizzare SAX

Puntatori

L'uso di puntatori può essere problematico in alcune implementazioni dei processori. Utilizzare una struttura self-conteined.

Profondità dell'albero

Un albero troppo profondo può essere più lento da navigare.

Progettazione di una struttura dati XML

Le conseguenze di una determinata decisione di design
CODING TIME

Livello di astrazione

In un documento XML, aumentare il livello di astrazione può sembrare un vantaggio…

puntatori

Troppi puntatori diventano problematici da gestire nel codice.

Complessità del documento

Maggiore sarà la complessità del documento e maggiore sarà la dimensione del codice necessario per gestirlo!

```
<!ELEMENT Invoice (Date+)>
<!ELEMENT Date EMPTY>
<!ATTLIST Date
   dateType (orderDate | shipDate) #REQUIRED
   dateValue CDATA #REQUIRED>
```

```
<Invoice>
   <Date dateType="orderDate" dateValue="12/1/2000" />
   <Date dateType="shipDate" dateValue="12/4/2000" />
</Invoice>
```

Per aggiungere una tipologia di data, basta aggiungere nella lista di enumerazione ma…

```
<Invoice>
   <Date dateType="orderDate" dateValue="12/1/2000" />
   <Date dateType="orderDate" dateValue="12/4/2000" />
</Invoice>
```

Serve codice addizionale per verificarne la correttezza

```
<Invoice
   orderDate="12/1/2000"
   shipDate="12/4/2000" />
```

Progettazione di una struttura dati XML

Le conseguenze di una determinata decisione di design

DEVELOPER RAMP-UP TIME

Quanto costerà ad uno sviluppatore capire la vostra struttura XML?

EXTENSIBILITY

Ricordarsi che presumibilmente la struttura cambierà nel tempo

La Trasmissione dei Dati

Lo scambio di informazioni tra sistemi differenti rappresenta uno degli utilizzi maggiori di XML

L'esecuzione di una trasmissione dati implica dover risolvere problematiche relative a:

Un formato comune
se non esiste uno standard al quale adattarsi deve essere creato.

Trasporto
mail, http,ftp…

Routing
nel caso il flusso di trasmissione richieda l'invio delle informazioni a diversi sistemi. Il costo dell'interoperabilità rischia di diventare molto costoso

Request-response
necessità di mantenere traccia delle transazioni richiede un protocollo di request response.

STRATEGIE CLASSICHE
LA SCELTA DEL FORMATO

Campi delimitati

```
Kevin Williams,744 Evergreen
Terrace,Springfield,KY,12345,12/01/2000,12/04/2000
Homer Simpson,742 Evergreen Terrace,Springfield,KY,,12/02/2000,12/05/2000
```

Campi a lunghezza fissa

```
Kevin Williams 744 Evergreen Terrace  Springfield   KY12345
12/01/200012/04/2000
Homer Simpson  742 Evergreen Terrace  Springfield   KY12345
12/02/200012/05/2000
```

Utilizzo di TAG

```
I,12/01/2000,12/04/2000
C,Kevin Williams,744 Evergreen Terrace,Springfield,KY,12345
L,blue,2 in. grommet,0001700000.10
L,silver,3 in. widget,0002200000.20
I,12/02/2000,12/05/2000
C,Homer Simpson,742 Evergreen Terrace,Springfield,KY,12345
L,red,1 in. sprocket,0001300000.30
L,blue,2 in. grommet,0001100000.10
```

STRATEGIE CLASSICHE

PROBLEMI CON LE STRUTTURE CLASSICHE

NON AUTO-ESPLICATIVO

Senza documentazione non è possibile capire il senso dei dati trasmessi

NON NORMALIZZATO

FRAGILE

In un file a campi fissi cambiare un formato diventa costoso sia per chi crea il file e sia per chi lo legge.

ROUTING e REQUESTING

IL TRASPORTO DELL'INFORMAZIONE

SUPPORTO FISICO

Necessita di intervento umano
processo lento
diversi hardware per diversi supporti

eMail

Informazioni come attachment

FTP

Necessita di un ftp server in modo da depositare i file. Batch di caricamento. Problemi di sicurezza

Socket

Via TCP si crea una connessione su una particolare porta. Streaming. Politiche di sicurezza spesso bloccano l'uso di porte diverse dalla 80.

VPN

Più sicuro dei precedenti ma sempre vulnerabile.

Linea Dedicata

I vantaggi di XML

Auto Esplicativo

La comprensione dei contenuti non necessita di documentazione aggiuntiva grazie ai tag.

FLESSIBILITA'

Si possono aggiungere informazioni mantenendo compatibilità con i documenti precedenti.

NORMALIZZATO

TOOL ESTERNI di MANAGEMENT

ROUTING e REQUESTING

Permette di gestire tecniche di *envelope* in maniera semplice grazie alla struttura ad albero

SOAP

(Simple Object Access Protocol)

Istanziare Componenti in maniera Platform-independent

RPC

Fornisce meta-informazioni sul documento mediante un meccanismo di envelope

XML over HTTP

La Trasmissione dei Dati

envelope

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Permette di includere informazioni aggiuntive su come trattare i dati

header

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
      <invoice:MessageStatus
         xmlns:invoice="http://www.invoicesystem.com/soap">
        <invoice:status>Resend</invoice:status>
      </invoice:MessageStatus>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

E' possibile rendere obbligatoria la comprensione dei parametri dell'header

```
<invoice:MessageStatus
   xmlns:invoice="http://www.invoicesystem.com/soap"
   SOAP-ENV:mustUnderstand="1">
  <invoice:status>Resend</invoice:status>
</invoice:MessageStatus>
```

body

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
     <invoice:MessageStatus
        xmlns:invoice="http://www.invoicesystem.com/soap"
        SOAP-ENV:mustUnderstand="1">
       <invoice:status>Resend</invoice:status>
     </invoice:MessageStatus>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
     <Invoice
       customerIDREF="c1"
       orderDate="10/17/2000"
       shipDate="10/20/2000">
       <LineItem
         partIDREF="p1"
         quantity="17"
         price="0.15" />
       <LineItem
         partIDREF="p2"
         quantity="13"
         price="0.25" />
     </Invoice>
     <Customer
       CustomerID="c1"
       name="Homer J. Simpson"
       address="742 Evergreen Terrace"
       city="Springfield"
       state="KY"
       postalCode="12345" />
     <Part
       PartID="p1"
       name="grommets"
       size="3 in."
       color="blue" />
     <Part
       PartID="p2"
       name="sprockets"
       size="2 in."
       color="red" />
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP over HTTP

la trasmissione su HTTP permette di evitare problemi relativi a restrizioni di accesso alle reti imposte dai firewall che tipicamente permettono il traffico sulla porta 80 (443 per HTTPS).

si utilizza un meccanismo di request-response

http soap request

segue le regole di una richiesta HTTP con l'aggiunta di un header soap: SOAPAction. Il valore rappresenta la procedura o il processo sul server in attesa del messaggio

```
SOAPAction: "http://www.invoiceserver.com/soap/handler.exe#Invoice"
SOAPAction: ""
SOAPAction:
```

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
     <invoice:MessageStatus
        xmlns:invoice="http://www.invoicesystem.com/soap"
        SOAP-ENV:mustUnderstand="1">
       <invoice:status>Resend</invoice:status>
     </invoice:MessageStatus>
     ...
```

http soap response

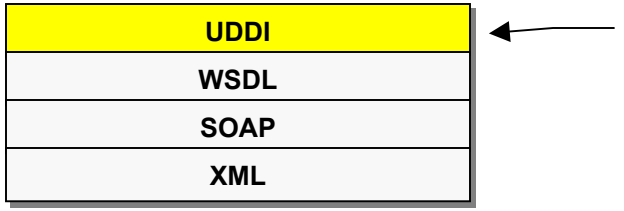
utilizza gli stessi codici di risposta di una response HTTP.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
   <SOAP-ENV:Body />
</SOAP-ENV:Envelope>
```

I Web Services

interfaccia che descrive una collezione di operazioni, accessibili attraverso una rete mediante messaggistica XML

i servizi web vengono descritti mediante dei descrittori di servizio che informano su come debbano essere utilizzati
questo ne permette un utilizzo automatico (ad esempio tramite degli agenti software)



Universal Description. Discovery and Integration

framework aperto ed indipendente dalla piattaforma per descrivere servizi, individuare società e integrare servizi di business utilizzando Internet

Funzionamento

1. Popolazione del registro con le *descrizioni* dei differenti *tipi di servizi*
2. Società erogatrici di servizi registrano i propri secondo i tipi presenti.
3. Il registry assegna un indicativo univoco per ogni tipo di servizio
4. Interrogazione dei registry per trovare i servizi e le società erogatrici
5. Integrazione via Web col servizio trovato

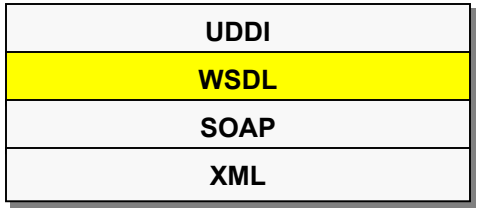
I tre elementi principali del framework UDDI sono:

- Descrizione (società,categoria…)
- Individuazione
- Integrazione (all'interno delle proprie applicazioni)

Tipologia di informazioni:

- Pagine bianche: nome,inidirzzo,num. Tel della società erogatrice di servizi
- Pagine gialle: categorizzazione delle società
- Pagine verdi: info tecniche

documento XML che descrive i messaggi di I/O di un determinato servizio, le relazioni tra questi messaggi e il collegamento fisico al servizio rappresenta un livello di indirezione in modo da disaccoppiare il client dal fornitore fisico del servizio.



Web Services Description Language

grammatica di definizione di un servizio di network generico indipendente da SOAP e UDDI

Permette di definire:

- *tipi*
- *messaggi* di input e output di un determinato servizio
- *Operazioni (*relazioni tra messaggi)
- *Binding*
- Definizione del servizio

Sezioni di un documento WSDL

- *Types* – tipi utilizzati dai messaggi
- *Message* – messaggi di scambio con i servizi
- *portType* – sono i metodi esposti dal servizio remoto
- *Binding* – come raggiungere, con che protocollo il servizio
- *Service* – descrizione del servizio (URL)

Esempio WSDL

```
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="TradePriceRequest">
          <complexType>
            <all>
              <element name="tickerSymbol" type="string"/>
            </all>
          </complexType>
        </element>
        <element name="TradePrice">
          <complexType>
            <all>
              <element name="price" type="float"/>
            </all>
          </complexType>
        </element>
      </schema>
    </types>

    <message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
      <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
      </operation>
    </portType>

    <binding name="StockQuoteSoapBinding"
        type="tns:StockQuotePortType">
      <soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetLastTradePrice">
        <soap:operation
            soapAction="http://example.com/GetLastTradePrice"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
      </operation>
    </binding>

    <service name="StockQuoteService">
      <documentation>My first service</documentation>
      <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://example.com/stockquote"/>
      </port>
    </service>

</definitions>
```

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 verificare 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 interfacce, 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

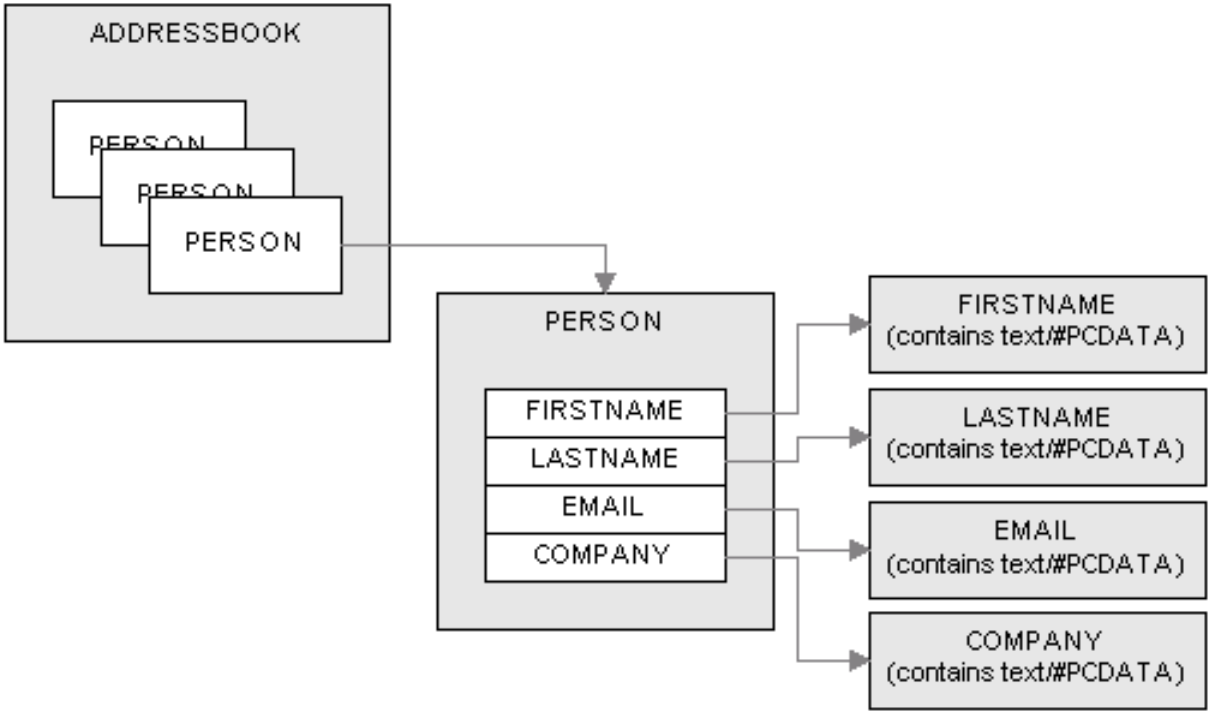
L'utilizzo del DTD per definire la struttura di un documento rende agevole lo scambio di documenti XML tra sistemi differenti che condividano il medesimo DTD. Il DTD rappresenta un metodo per essere certi che due o più documenti XML siano dello stesso "tipo".

Interoperabilità

Tutti i vantaggi visti precedentemente rendono l'interoperabilità possibile. La capacità di poter condividere informazioni tra sistemi differenti 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)

DTD

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

il documento può contenere solamente l'elemento di root **tutorial** che a sua volta può contenere del testo

corretto

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

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

errato

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

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

Definizione Tipo Documento (DTD)

DTD

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

l'elemento di root **XXX** deve contenere esattamente un elemento **AAA** seguito da un elemento **BBB**. **AAA** e **BBB** possono contenere testo ma non altri elementi

corretto

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">
        <XXX>
        <AAA>Start</AAA>
        <BBB>End</BBB>
        </XXX>

 <!DOCTYPE tutorial SYSTEM "tutorial.dtd">
        <XXX> <AAA/> <BBB/> </XXX>
```

errato

BBB mancante

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

BBB deve seguire **AAA**:

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

XXX può contenere solo un elemento **BBB**:

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

XXX non può contenere testo

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

Definizione Tipo Documento (DTD)

DTD

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

L'elemento **XXX** *può* contenere zero, uno o più elementi **AAA** seguiti da un elemento **BBB**. **BBB** deve essere sempre presente

corretto

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

AAA non è obbligatorio

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

diverse ricorrenze di **AAA** sono ammesse

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

errato

BBB mancante

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

BBB deve seguire **AAA**

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

AAA non può venire dopo **BBB**:

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

Definizione Tipo Documento (DTD)

DTD

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

XXX *deve* contenere uno o più elementi **AAA** seguiti da esattamente un elemento **BBB**. **BBB** deve essere sempre presente

corretto

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

errato

AAA e **BBB** mancanti

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

almeno un elemento **AAA** deve essere presente

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

BBB deve essere dopo **AAA**:

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

AAA non può essere dopo **BBB**:

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

Definizione Tipo Documento (DTD)

DTD

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

XXX *può* contenere un elemento **AAA**

corretto

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

AAA non è obbligatorio

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

errato

BBB mancante

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

al massimo un elemento **AAA:**

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

BBB deve seguire **AAA**:

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

Definizione Tipo Documento (DTD)

DTD

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

XXX può contenere un elemento **AAA** seguito da uno o più elementi **BBB**. **AAA** può contenere un **CCC** e diversi **DDD**. **BBB** deve contenere un elemento **CCC** e un elemento **DDD**

corretto

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

AAA non è obbligatorio

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

errato

BBB deve contenere **CCC** e **DDD**:

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

AAA può contenere al massimo un elemento **CCC**:

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

Definizione Tipo Documento (DTD)

DTD

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

BBB deve contenere un elemento **CCC** oppure un elemento **DDD**

corretto

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

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

errato

Non possono essere presenti entrambi:

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

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

Definizione Tipo Documento (DTD)

DTD

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

Un attributo di tipo CDATA può contenere qualsiasi stringa ben formata.. Un attributo **required** deve essere sempre presente, mentre uno **implied** è opzionale.

corretto

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

L'ordine degli attributi non è importante

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

l'attributo **bbb** può essere omesso essendo *implied*

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

errato

L'attributo **aaa** è *required*. Deve essere SEMPRE presente.

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

Definizione Tipo Documento (DTD)

bbb e **ccc** devono essere sempre presenti,
aaa è opzionale

DTD

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

Gli attributi **id**, **code** e **X** devono essere unici.

DTD

```
<!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>
```

CDATA – qualsiasi stringa ben formata

NMTOKEN – lettere, numeri e i caratteri **. - _ :**

NMTOKENS – NMTOKEN e whitespaces

errato

Un attributo ID non può partire con un numero o contenere caratteri non ammessi da NMTOKEN:

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

L'attributo ID deve avere valore UNICO

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

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

Definizione Tipo Documento (DTD)

DTD

```
<!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>
```

Il valore di un attributo **IDREF** deve corrispondere con il valore di qualche attributo **ID.**

Con **IDREFS** è possibile specificare diversi valori separati da spazi.

Gli attributi **id** e **mark** determinano unicamente i loro elementi l'attributo **ref** referenzia una di questi elementi

corretto

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

errato

Non ci sono attributi di tipo ID con valore a3 o b001:

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

Definizione Tipo Documento (DTD)

Possibilità di definire un insieme finito di elementi

```
<!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>
```

Attributi facoltativi con valore di default

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

L'elemento **AAA** può contenere attributi ma non un valore.

```
<!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 interna

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

entità esterna

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

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">
```

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 = prefix : localPart

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

Namespace predefinito

Definizione Tipo Documento (DTD)

DICHIARAZIONE DELLO SPAZIO DEI NOMI COME **URI**

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

L’URI che definisce il namespace è puramente formale: non c’è nessuna garanzia che il documento all’URI specificato contenga la descrizione della sintassi utilizzata o che esista effettivamente un documento.
Per un processore XML un documento che utilizzi namespace o no non fa differenza e deve essere sempre well-formed.

NON E’ POSSIBILE APPLICARE LO STESSO DTD SE QUALIFICHIAMO I NOMI

`<!ELEMENT DIVISION (DIVISION_NAME, TEAM+)>` → `<!ELEMENT bb:DIVISION (bb:DIVISION_NAME, bb:TEAM+)>`

URI

Gli URI (Universal Resource Identifier) sono una sintassi usata in WWW per definire i nomi e gli indirizzi di oggetti (risorse) su Internet. Questi oggetti sono considerati accessibili tramite l’utilizzo di protocolli esistenti, inventati appositamente, o ancora da inventare.

Criteri di design degli URI

Gli *Universal Resource Identifier* (URI) sono, per definizione:
Universal Resource Names (URN)
Universal Resource Locator (URL).

Gli URL sono un indirizzo della risorsa che possa essere immediatamente utilizzato da un programma per accedere alla risorsa. Contengono tutte le informazioni necessarie per accedere all’informazione, ma sono fragili a modifiche non sostanziali del meccanismo di accesso (es. cambio del nome di una directory).

Gli URN sono un nome stabile e definitivo di una risorsa, che possa fornire un informazione certa ed affidabile sulla sua esistenza ed accessibilità. Debbono essere trasformati da un apposito servizio, negli URL attualmente associati alla risorsa. Inoltre la mappa deve essere aggiornata ogni volta che la risorsa viene spostata.

esercitazione

Esempio di utilizzo del namespace

modulo3/xml/namespace.xml
modulo3/xml/namespace.xsl

Come utilizzare entità esterne in un dtd

modulo3/dtd/externalentities.xml
modulo3/dtd/externalentities.dtd
modulo3/dtd/companies.dtd
modulo3/dtd/states.dtd

XML Schema

Limitazione dei DTD

i DTD nascono con lo scopo di trattare una tipologia ben specifica 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 schema 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

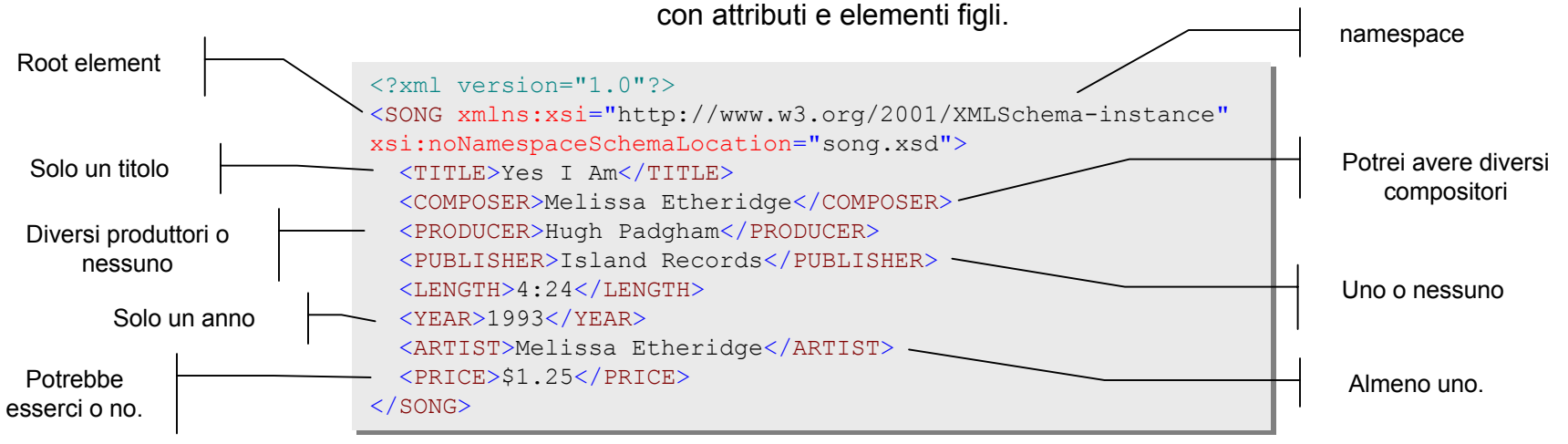
```
<GREETING> various random text but no markup</GREETING>

<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>
```

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>
```

Complex Types

Aumentiamo la profondità della struttura

```
<?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 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>
```

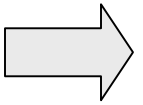
Nell'esempio precedente i due tipi definiti sono identici. E' possibile creare un solo tipo e condividerlo.

```
<?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 significa che devono essere presenti o l'elemento COMPOSER o PRODUCER ma non contemporaneamente.

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.

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, 4567349873249832649873624958
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, . . .

Simple Types: Time data types

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

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.

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

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 attributi. 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"/>
```

esercitazione

Evoluzione di uno *schema* per la definizione di una lista di dischi musicali

```
modulo3/schema/songs.xml
modulo3/schema/songs.xsd

modulo3/schema/songs2.xml
modulo3/schema/songs2.xsd

modulo3/schema/songs3.xml
modulo3/schema/songs3.xsd
```

XSL = XSLT + FO

XSLT
linguaggio di
TRASFORMAZIONE

FO
linguaggio di
FORMATTAZIONE

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

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

```
root
elements
text
attributes
namespaces
PI
commenti
```

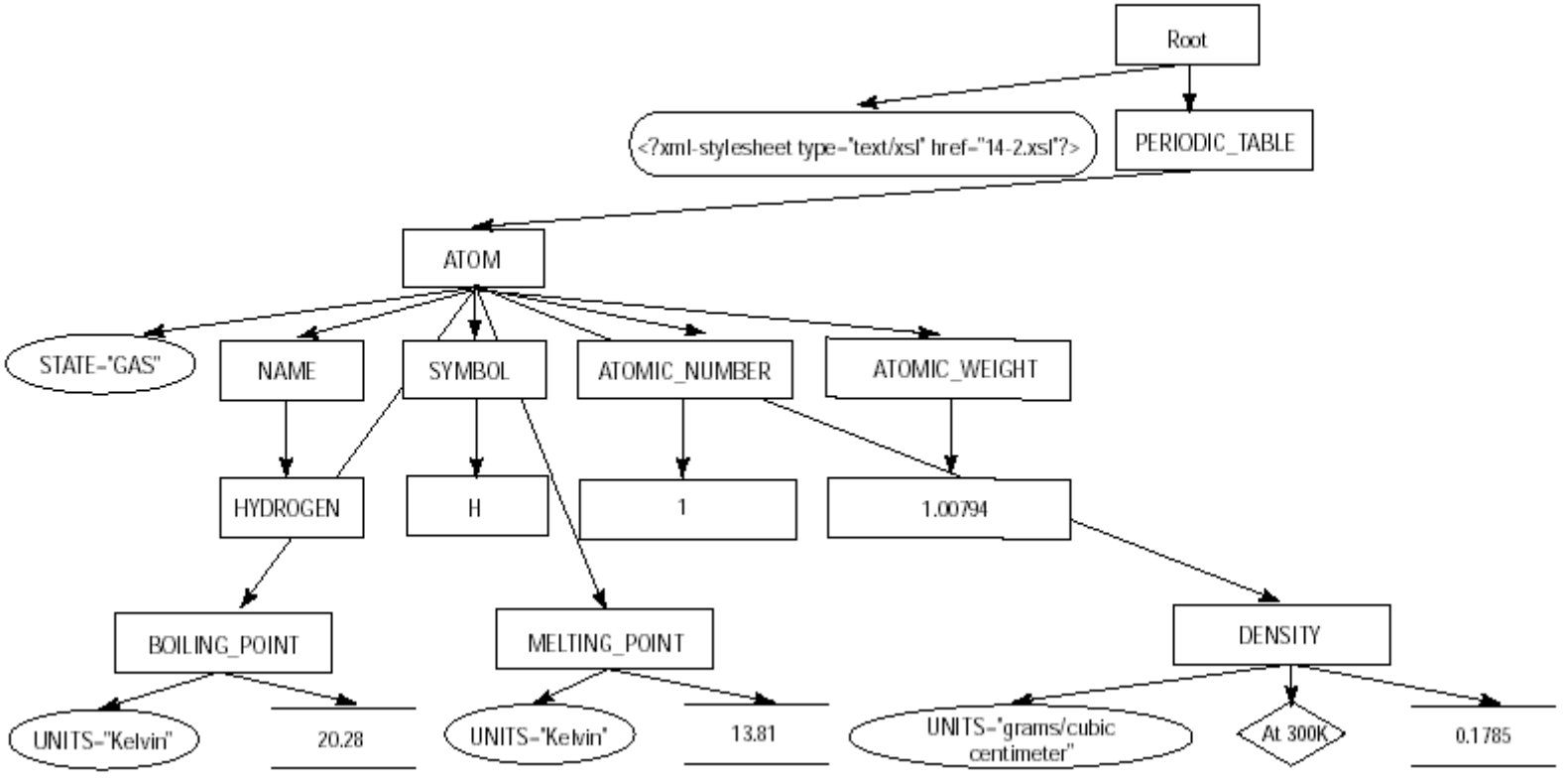
XSLT agisce trasformando un albero XML in un altro albero XML; più precisamente un processore XSLT accetta in ingresso un albero rappresentato da un documento XML e produce come output un nuovo albero rappresentato da un documento XML.

Una trasformazione XSLT NON può produrre un documento XML mal formato

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>
```

Rappresentazione interna



XSLT style sheet documents

I documenti XSLT sono costituiti da *TEMPLATE*

Ogni template ha un *PATTERN* che indica la condizione per la quale il template debba essere istanziato.

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

Stylesheet esempio: TEMPLATE

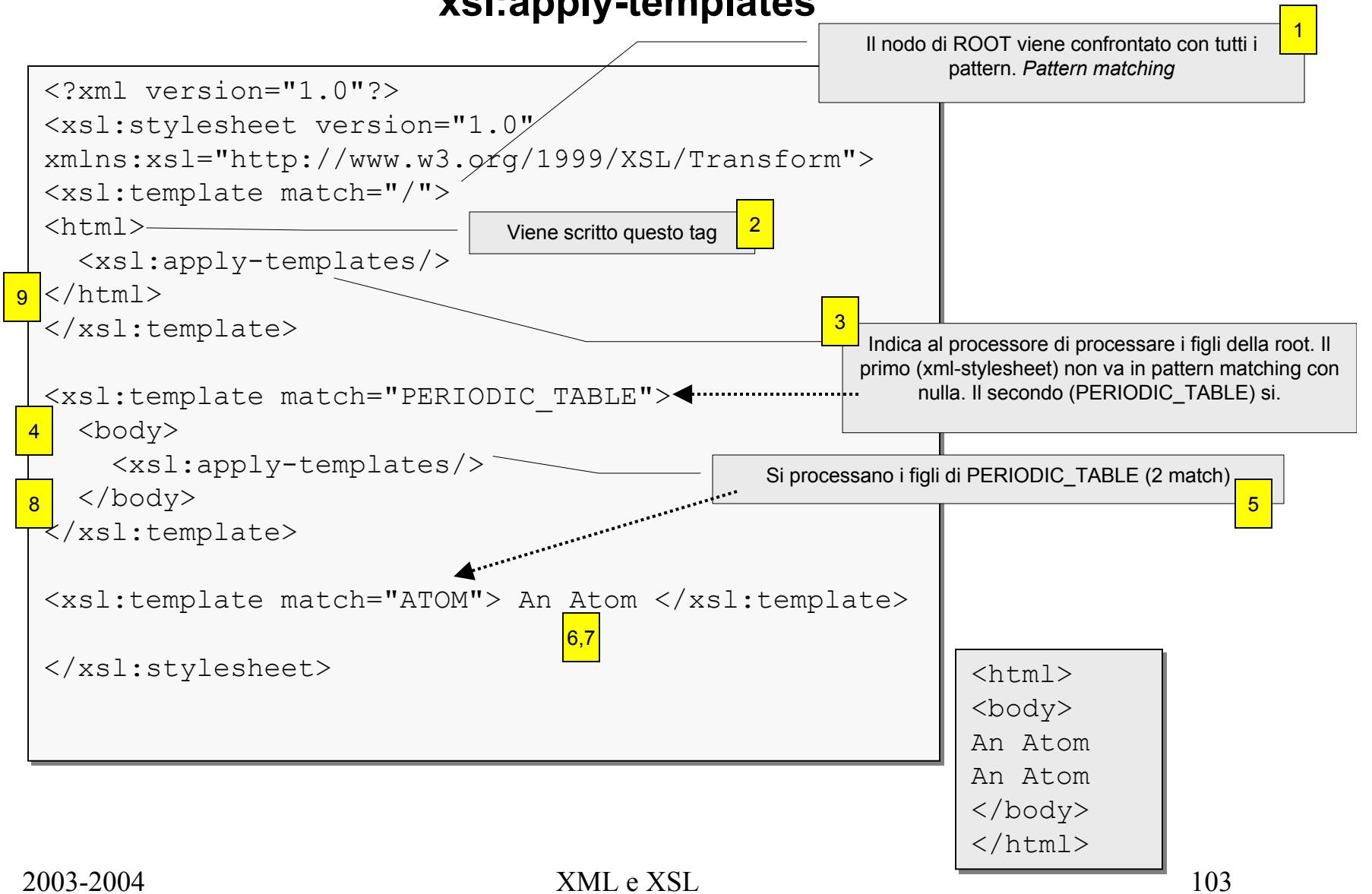
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<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>
```

Un elemento xsl:template associa un output ad un particolare output. Ognuno di questi elementi ha un attributo match che specifica per quali nodi del documento questo template debba essere istanziato.

Un template può contenere
testo o
istruzioni xslt
(la distinzione avviene tramite namespace)

xsl:apply-templates



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>
```

Rispetto all'esempio precedente si utilizza anche l'attributo select che utilizza le stesse tipologie di pattern utilizzate per l'attributo match.

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

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: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.
L'attributo href indica l'URI della risorsa da importare

Le regole del documento che importa sono prioritarie rispetto alle regole importate

apply-imports è una variante di apply-templates che permette di accedere alle regole importate.

xsl:include copia il contenuto dello stylesheet remoto nello stylesheet corrente nel punto in cui compare. Le regole incluse hanno la stessa precedenza delle regole del documento che importa.

Cosa accade se ho due template nello stesso documento con lo stesso pattern?

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 – /,//,*

/AAA/CCC

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

Seleziona tutti gli elementi CCC che sono figli dell'elemento di root AAA

//DDD/BBB

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

Tutti BBB figli di DDD

/AAA/CCC/DDD/*

```
<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>
```

/*/*/*/BBB

```
<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>
```

Seleziona BBB che hanno tre antenati

XPATH – [ ], @, count()

/AAA/BBB[1]

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

/AAA/BBB[last()]

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

//@id

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

//BBB[@id]

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

//BBB[@id='b1']

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

//*[count(BBB)=2]

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

Seleziono gli elementi che hanno due filgi BBB

//*[count(*)=2]

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

Seleziono gli elementi che hanno due filgi

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 – stringhe e operatore or |

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

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

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

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

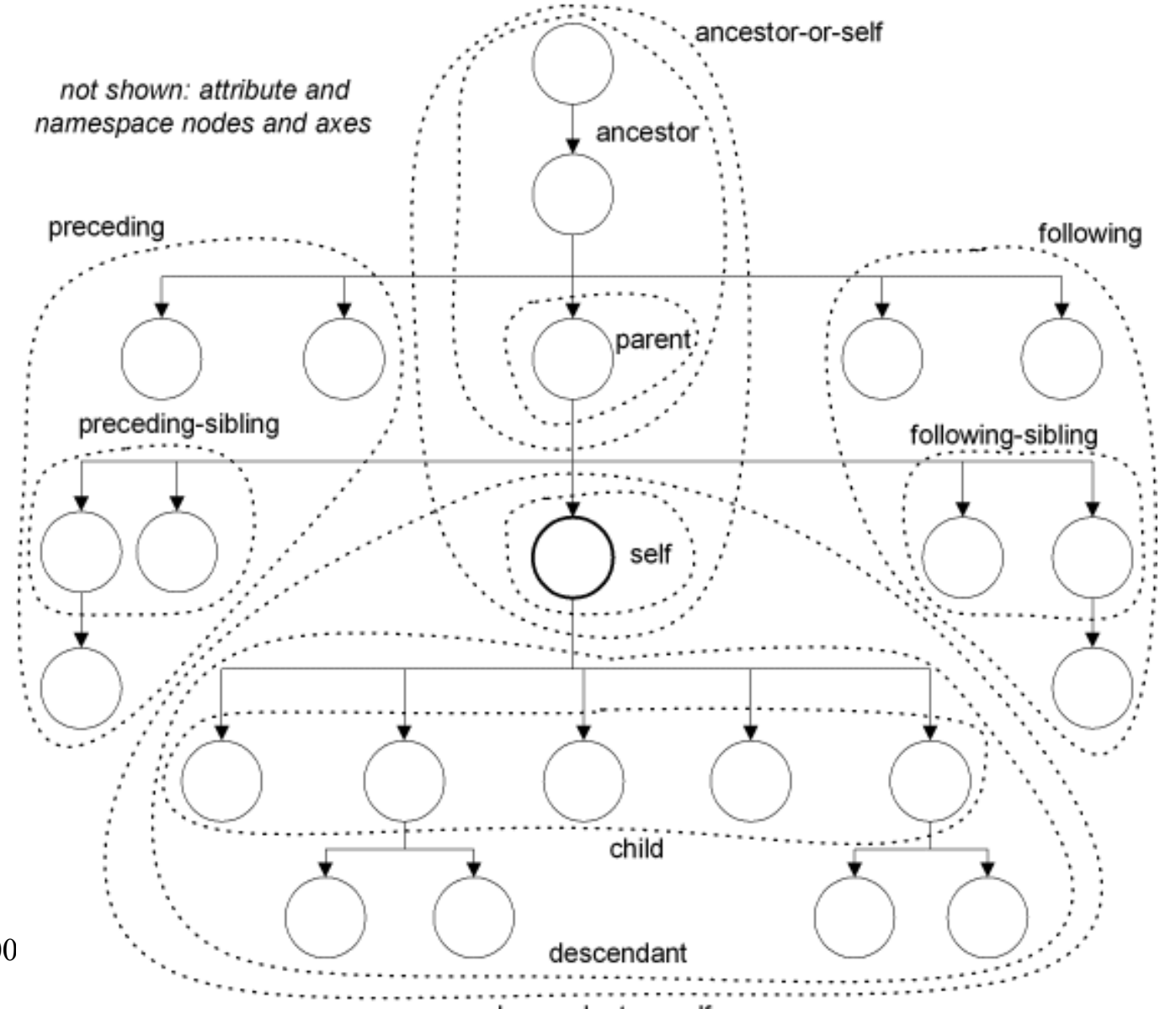
//*[contains(name(),'C')]

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

/AAA/EEE | //BBB

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

Axis Specifiers



XPATH

/child::AAA

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

/AAA/BBB/descendant::*

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

/AAA/BBB/following-sibling::*

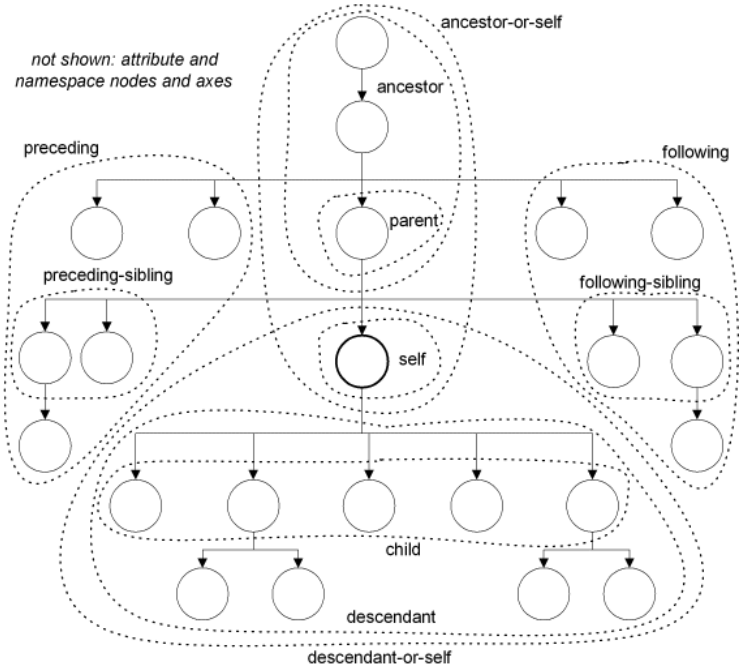
```
<AAA>
    <BBB>
        <CCC/>
        <DDD/>
    </BBB>
    <XXX>
        <DDD>
            <EEE/>
            <DDD/>
            <CCC/>
            <FFF/>
            <FFF>
                <GGG/>
            </FFF>
        </DDD>
    </XXX>
    <CCC>
        <DDD/>
    </CCC>
</AAA>
```

//DDD/parent::*

```
<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::*

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



unabbreviated syntax

•child::para selects the para element children of the context node

•child::* selects all element children of the context node

•child::text() selects all text node children of the context node

•child::node() selects all the children of the context node, whatever their node type

•attribute::name selects the name attribute of the context node

•attribute::* selects all the attributes of the context node

•descendant::para selects the para element descendants of the context node

•ancestor::div selects all div ancestors of the context node

•ancestor-or-self::div selects the div ancestors of the context node and, if the context node is a div element, the context node as well

•descendant-or-self::para selects the para element descendants of the context node and, if the context node is a para element, the context node as well

•self::para selects the context node if it is a para element, and otherwise selects nothing

•child::chapter/descendant::para selects the para element descendants of the chapter element children of the context node

•child::*/child::para selects all para grandchildren of the context node

•/ selects the document root (which is always the parent of the document element)

•/descendant::para selects all the para elements in the same document as the context node

•/descendant::olist/child::item selects all the item elements that have an olist parent and that are in the same document as the context node

•child::para[position()=1] selects the first para child of the context node

•child::para[position()=last()-1] selects the last but one para child of the context node

•child::para[position()>1] selects all the para children of the context node other than the first para child of the context node

•following-sibling::chapter[position()=1] selects the next chapter sibling of the context node

•preceding-sibling::chapter[position()=1] selects the previous chapter sibling of the context node

•/descendant::figure[position()=42] selects the forty-second figure element in the document

•/child::doc/child::chapter[position()=5]/child::section[position()=2] selects the second section of the fifth chapter of the doc document element

•child::para[attribute::type="warning"] selects all para children of the context node that have a type attribute with value warning

•child::para[attribute::type='warning'][position()=5] selects the fifth para child of the context node that has a type attribute with value warning

•child::para[position()=5][attribute::type="warning"] selects the fifth para child of the context node if that child has a type attribute with value warning

•child::chapter[child::title] selects the chapter children of the context node that have one or more title children

•child::*[self::chapter or self::appendix] selects the chapter and appendix children of the context node

esercitazione

xsl:value-of e l'utilizzo di {}

modulo3/xslt/01_discussionForumHome.xml
modulo3/xslt/01_discussionForumHome.xsl

escaping e template nidificati

modulo3/xslt/02_schedule.xml
modulo3/xslt/02_schedule.xsl

XSL – xsl:sort

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

```
<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>
```

```
<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 – xsl:element

```
<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>
```

```
<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>
```

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

XSL – xsl:attribute

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

```
<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>
```

```
<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 – xsl:with-param

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

```
<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>
```

```
<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 – xsl:variable

```
<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>
```

```
<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>
```

```
<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>
```

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 – position(), last(),count()

```
<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: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)"/>
```

XSL – stringhe

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

```
<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)"/>
```

```
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
```

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

```
<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>
```

```
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 – id, generate-id

```
<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>
```

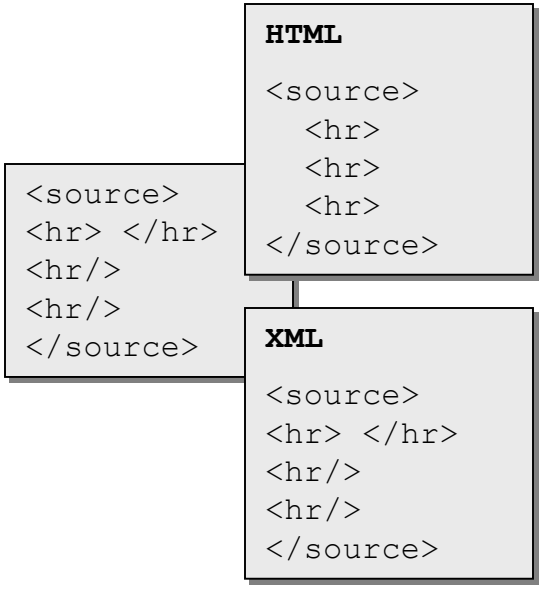
```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="*">
<xsl:copy>
  <xsl:attribute name="id">
    <xsl:value-of select="generate-id()"/>
  </xsl:attribute>
  <xsl:for-each select="@*">
    <xsl:attribute name="{name()}">
      <xsl:value-of select="."/>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

```
<source 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 – output()



<xsl:output method="html|xml|text"/>

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

XSL – 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/Transform'>
<xsl:template match="p">
  <xsl:copy-of select="."/>
  <xsl:copy/>
</xsl:template>
</xsl:stylesheet>
```

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

Utilizzato per il nodes copying.
`xsl:copy` copia solamente il nodo corrente senza figli e attributi, mentre `xsl:copy-of` copia tutto

esercitazione

ricorsione e utilizzo di template parametrizzati

```
modulo3/xslt/03_familyTree.xml
modulo3/xslt/03_familyTree.xsl
```

Formattazione numerica

```
modulo3/xslt/04_numberFormatting.xml
modulo3/xslt/04_numberFormatting.xslt
```

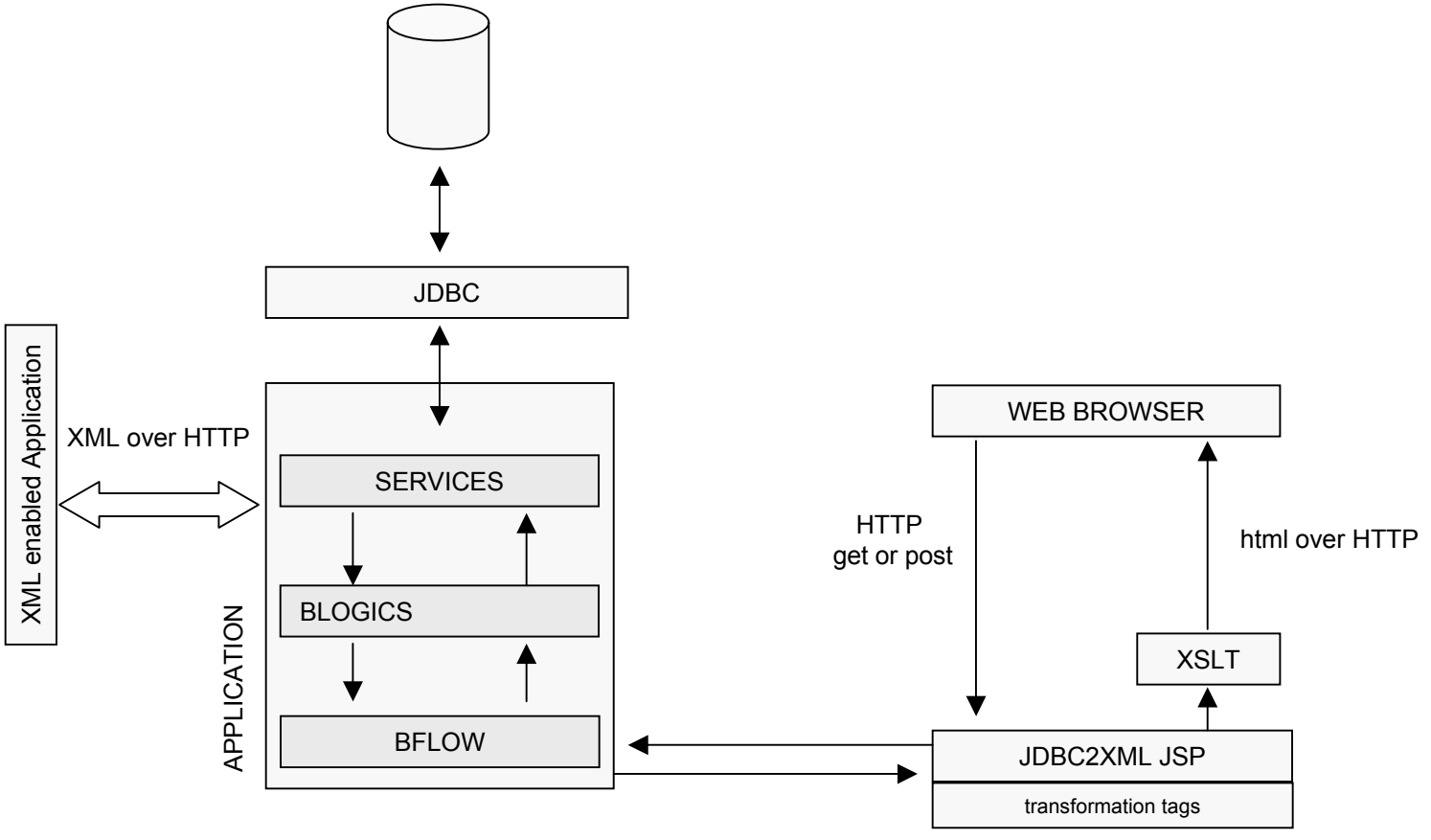
stringhe

```
modulo3/xslt/05_stringFunctions.xml
modulo3/xslt/05_stringFunctions.xsl
```

xsl:call-template

```
modulo3/xslt/06_team.xml
modulo3/xslt/06_namedTemplate.xslt
```

web application schema



Progetto rubrica XML

separazione del livello di contenuto da quello di presentazione

utilizzo delle **tag libraries**

delivery dei contenuti in funzione del *contesto*
(multilingua)

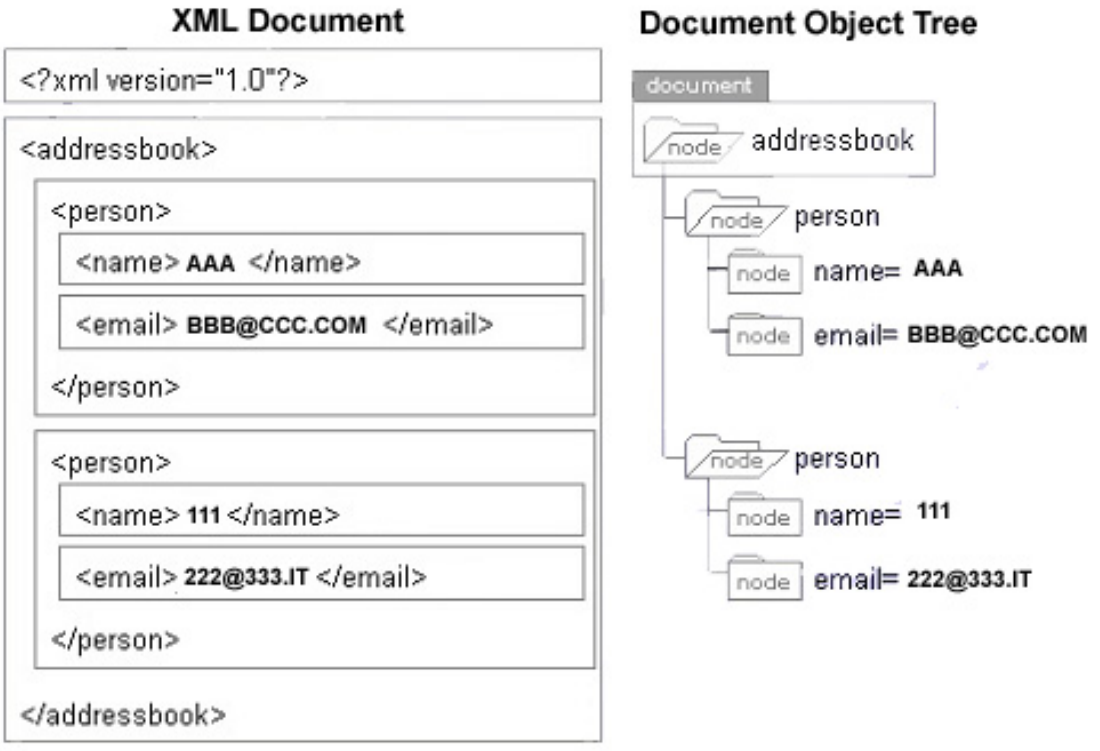
DOM & SAX - introduzione

SAX (Simple API for XML) e DOM (Document Object Model) sono stati entrambi realizzati per permettere l'accesso a documenti XML. 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 elementi 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 riferimento 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?

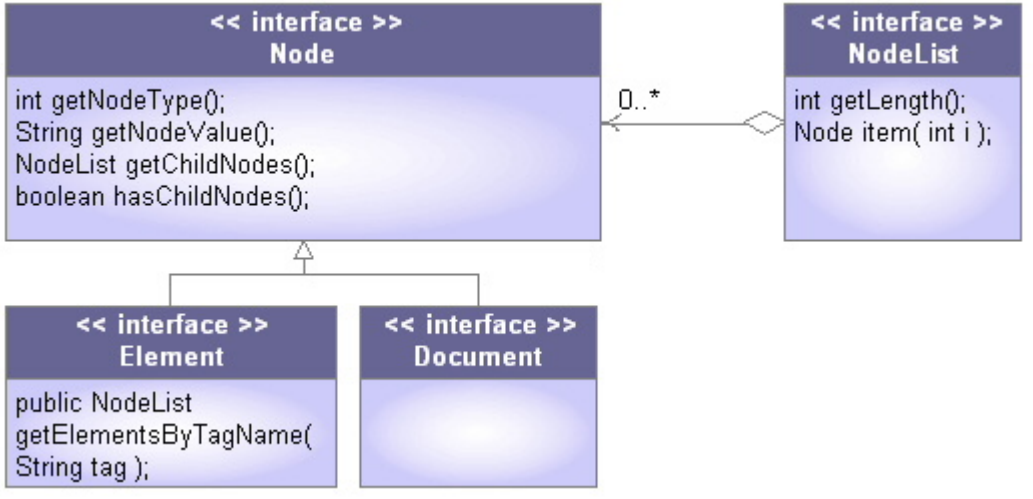
Quando si ha a che fare con dati strutturati generati (resultset) e che tipicamente devono essere strutturati secondo un modello ad oggetti personale.
Problemi di prestazioni. Documento troppo grande.

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

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.

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.

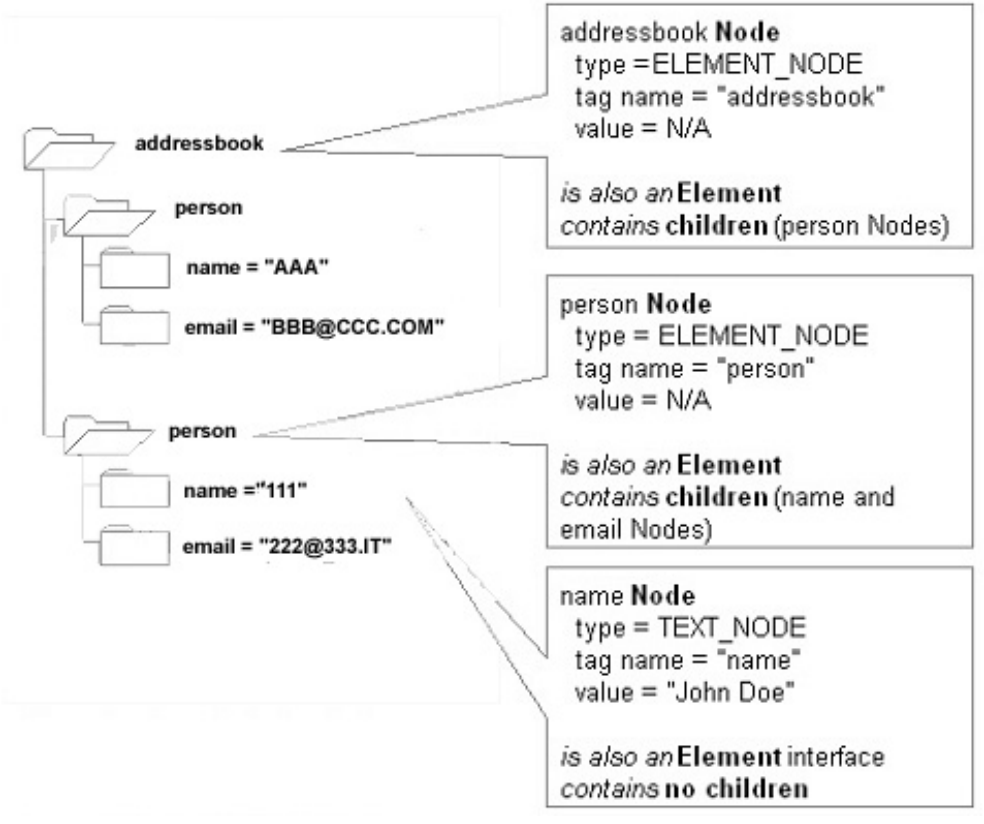
DOM

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?



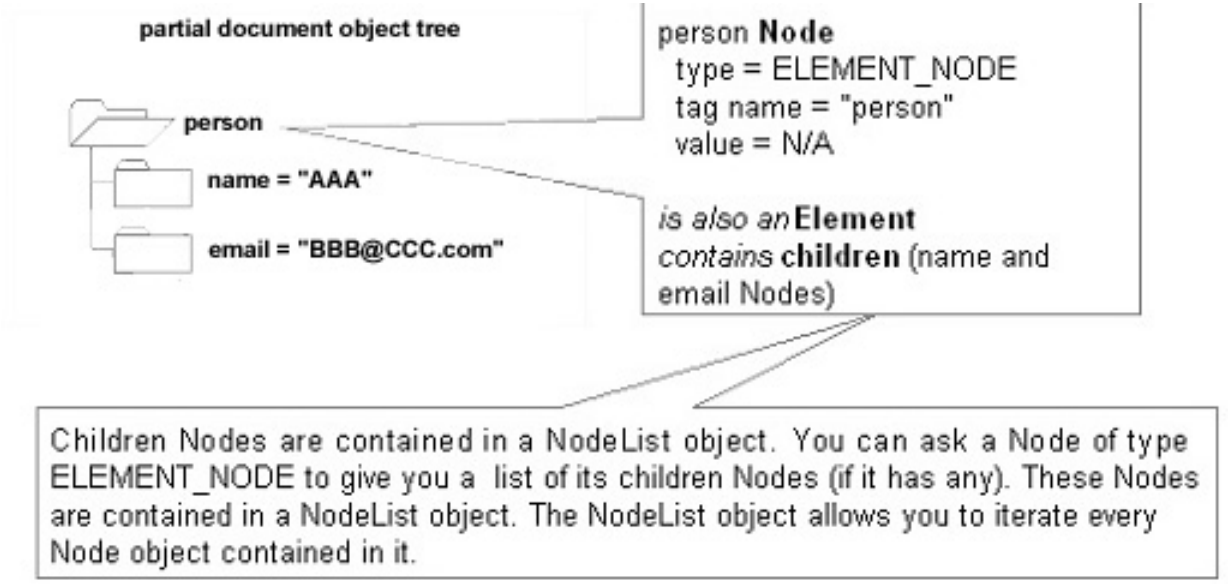
Notiamo che:

• 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)

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 );
```

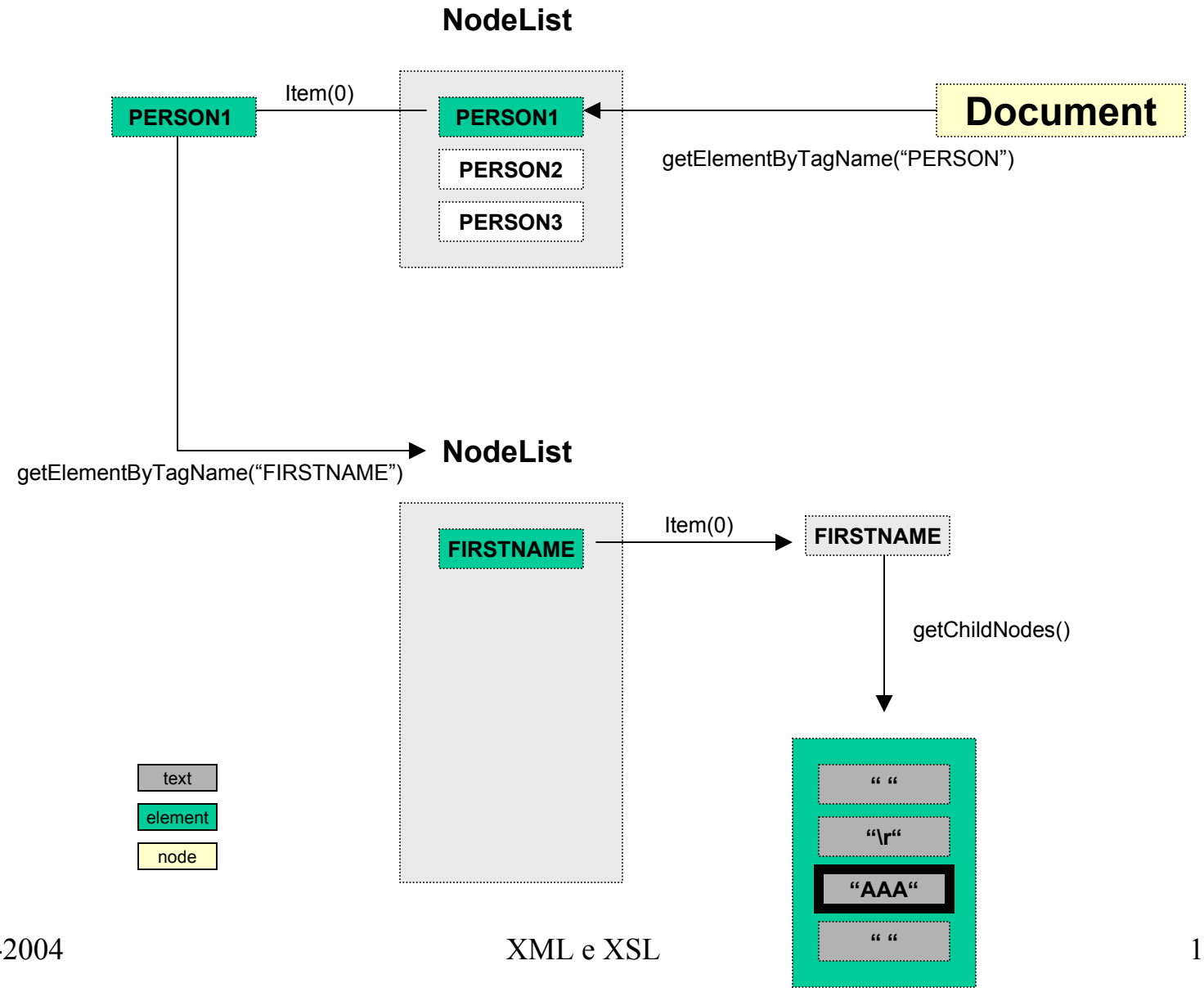
firstNameElement contiene una lista di TEXT_NODE uno dei quali rapprseneta il valore di FIRSTNAME.

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

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

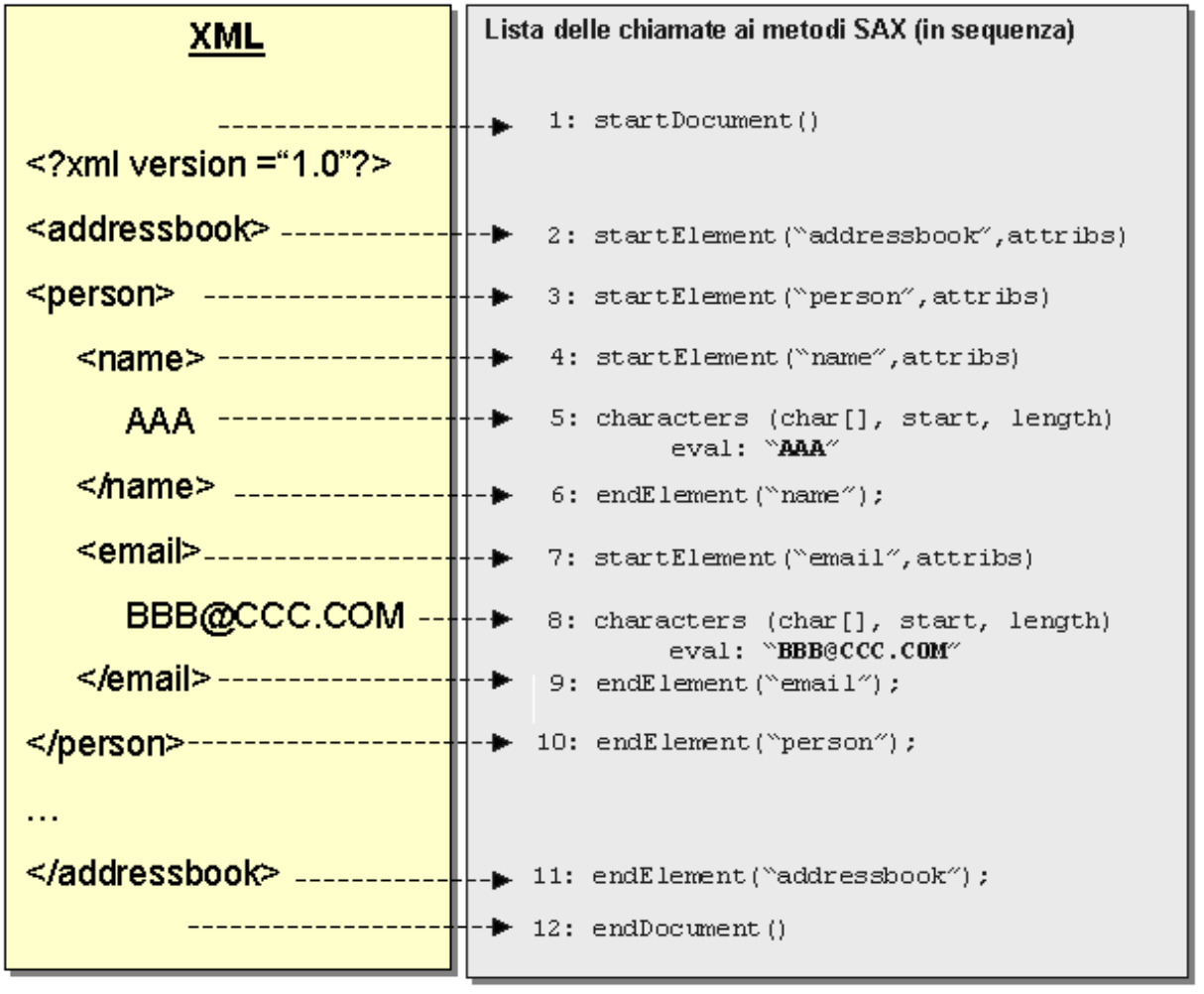

NodeList
PERSON1
Item(0)
PERSON1
PERSON2
PERSON3
Document
getElementByTagName("PERSON")
getElementByTagName("FIRSTNAME")
NodeList
FIRSTNAME
Item(0)
FIRSTNAME
getChildNodes()
text
element
node
" "
"\r"
"AAA"
" "

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>
```

Creazione di un DocumentHandler



Il Parser SAX che abbiamo creato, lancia eventi per ogni tag aperto, chiuso, CDATA, #PCDATA… Questi eventi provengono dal documento XML dall'alto verso il basso, uno alla volta. Il Parser SAX per notificare questi eventi ad un oggetto utilizza l'interfaccia DocumentHandler. Assieme alla interfaccie EntityResolver, DTDHandler, ErrorHandler si coprono tutti i possibili eventi generabili durante la lettura di cun documento XML.

SAX

ErrorHandler interface

Condizioni di errore nel sorgente 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 interfacce è possibile estendere 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.

esercitazione

SAX

```
modulo3/dom,sax/sax/AddressBook.java
modulo3/dom,sax/sax/Person.java
modulo3/dom,sax/sax/SaxAddressBookHandler.java
modulo3/dom,sax/sax/SaxDemo.java
```

DOM

```
modulo3/dom,sax/dom/AddressBook.jsp
```

Java API for XML Parsing 1.1.1
JAXP 1.1.1

http://java.sun.com/xml/jaxp/dist/1.1/docs/api/index.html

xml application example – RDF

RDF per la rappresentazione della
conoscenza *machine-understandable*

METADATI
informazioni, comprensibili dalla macchina, relative ad una qualunque risorsa

I metadati relativi ad un documento possono essere estratti
dalla risorsa stessa (HEAD di HTML),
da un'altra risorsa,
o possono essere trasferiti con la risorsa stessa (POST)

I metatadi sono rappresentati da un insieme di asserzioni indipendenti

Un'asserzione è rappresentata da un nome di asserzione e un insieme di parametri

RDF

RDF – rappresenta lo strumento base per la codifica, lo scambio e il riutilizzo di metadati strutturati, e consente l' interoperabilità tra applicazioni che si scambiano sul Web informazioni machine-understandable

Descrizione del contenuto di un sito Web

Intelligent software agent

Classificazione del contenuto

Criteri di proprietà intellettuale delle singole pagine

RDF Model and Syntax

RDF Schema

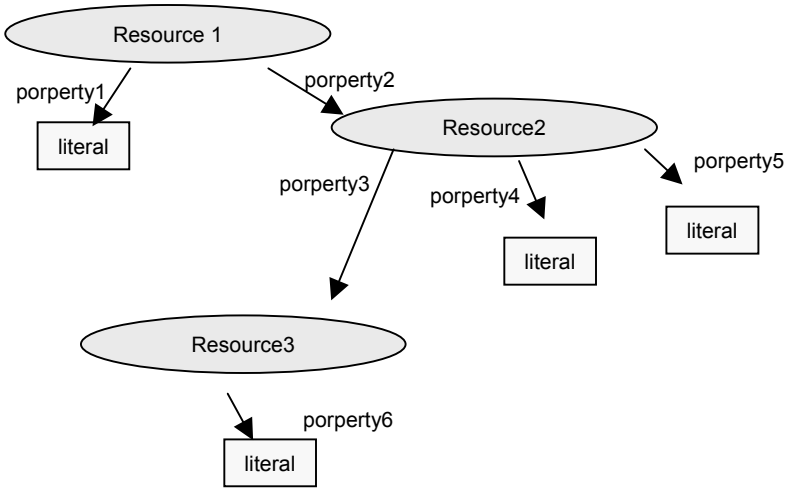
RDF Data Model

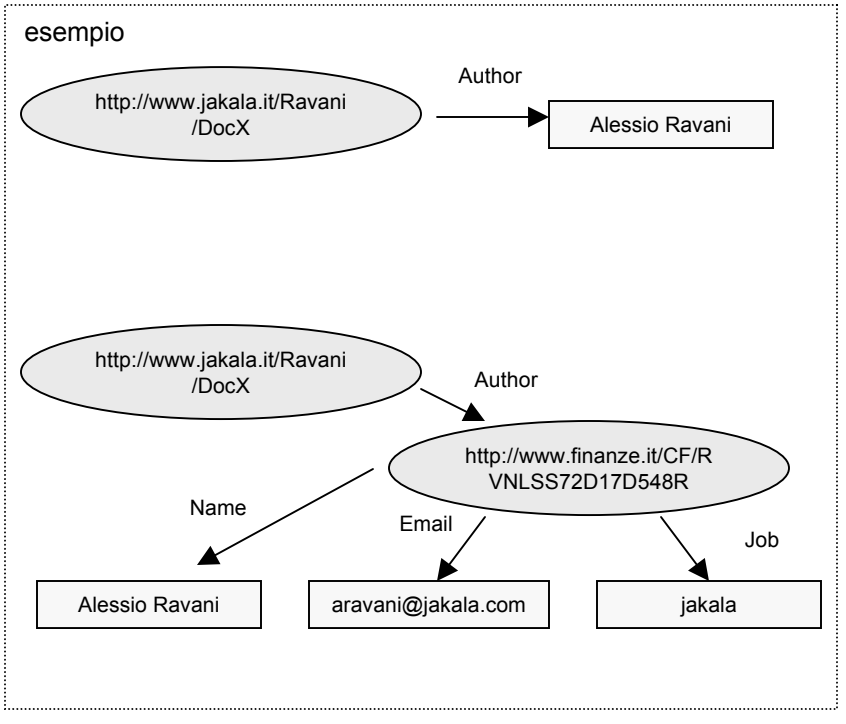
Basato su tre tipi di oggetti

Resources – qualunque oggetto che sia identificabile univocamente tramite un URI

Properties – coppie nome-valore che identificano una caratteristica della risorsa

Statements – insieme della risorsa con una proprietà identificata da un nome e relativo valore.
Soggetto (risorsa) + Predicato (proprietà) + Oggetto (valore).





The Dublin Core

The Dublin Core (http://purl.org/dc/) is a collection of elements designed to help researchers find electronic resources in a manner similar to using a library card catalog. Dublin Core elements include basic cataloging information, in particular:

- TITLE: The name given to the resource.
- CREATOR: The person or organization that created most of the resource (the author of a novel or the photographer who took a picture).
- SUBJECT: The topic of the resource.
- DESCRIPTION: A brief description of the resource, such as an abstract.
- PUBLISHER: The person or organization making the resource available (for example, IDG Books, Claremont University, or Apple Computer).
- CONTRIBUTOR: A non-CREATOR who contributed to the resource (the illustrator or editor of a novel).
- DATE: The date the resource was made available in its present form, generally in the format YYYY-MM-DD, such as 1999-12-31.
- TYPE: The category of the resource for example Web page, short story, poem, article, or photograph. Work is ongoing to produce a definitive list of acceptable resource types.
- FORMAT: The format of the resource, such as PDF, HTML, or JPEG. Work is ongoing to produce a definitive list of acceptable resource formats.
- IDENTIFIER: A unique string or number for the resource (as with a URL, a social security number, or an ISBN).
- SOURCE: A string or number that uniquely identifies the work from which the resource was derived. For instance, a Web page with the text of Jerome K. Jerome's 19th century novel *Three Men in a Boat* might use this to specify the specific edition from which text was scanned.
- LANGUAGE: The primary language in which the resource is written as ISO 639 language code.
- RIGHTS: Copyright and other intellectual property notices specifying the conditions under which the resource may or may not be used.

Several other possible Dublin Core elements are in the experimental stage including RELATION and COVERAGE. The Dublin Core is used throughout the examples in this chapter. However, you are by no means limited to using only these elements. You are free to use different vocabularies and namespaces for properties as long as you put them in a namespace.

NAMESPACES

I namespace XML forniscono un metodo per identificare in maniera non ambigua la semantica e le convenzioni che regolano l' utilizzo delle proprietà identificando l' authority che gestisce il vocabolario

CONTAINER

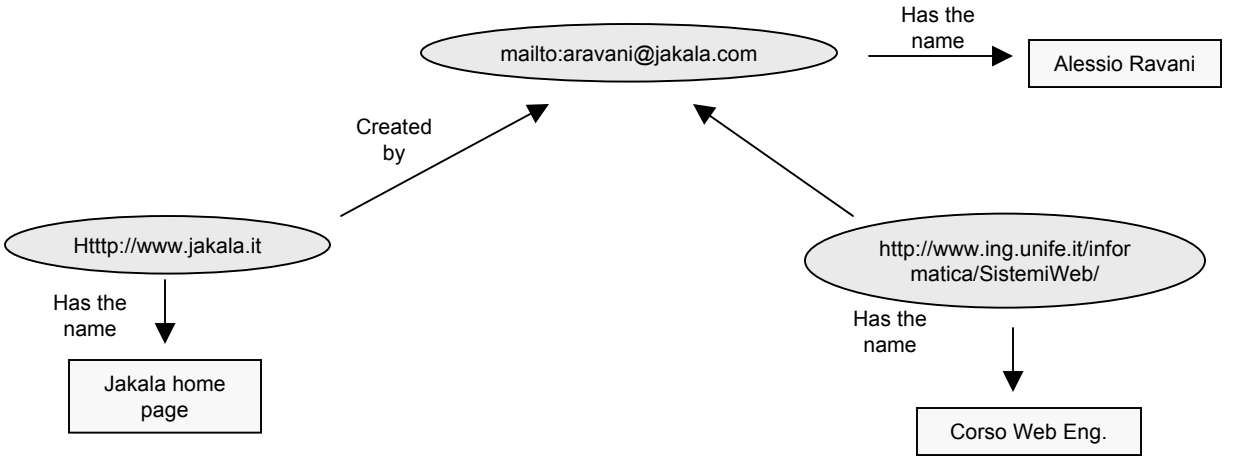
per far fronte alla necessità di riferirsi a più di una risorsa (un libro è scritto da più autori)

- **BAG –** lista non ordinata
- **SEQUENCE –** Lista ordinata
- **ALTERNATIVE –** Per definire delle alternative

RDF Schema

permette di definire l' insieme delle proprietà semantiche individuata da una particolare comunità

RDF




```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://www.purl.org/DC/">
        <rdf:Description about="http://www.jakala.it">
                <dc:TITLE>Home Page Jakala</dc:TITLE>
                <dc:CREATOR rdf:resource="mailto:aravani@jakala.com"/>
        </rdf:Description>
        <rdf:Description about="http://www.ing.unife.it/informatica/SistemiWeb/">
                <dc:TITLE>Corso Web Eng</dc:TITLE>
                <dc:CREATOR rdf:resource="mailto:aravani@jakala.com"/>
        </rdf:Description>
        <rdf:Description about="mailto:aravani@jakala.com">
                <dc:TITLE>Alessio Ravani</dc:TITLE>
        </rdf:Description>
</rdf:RDF>
```

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.

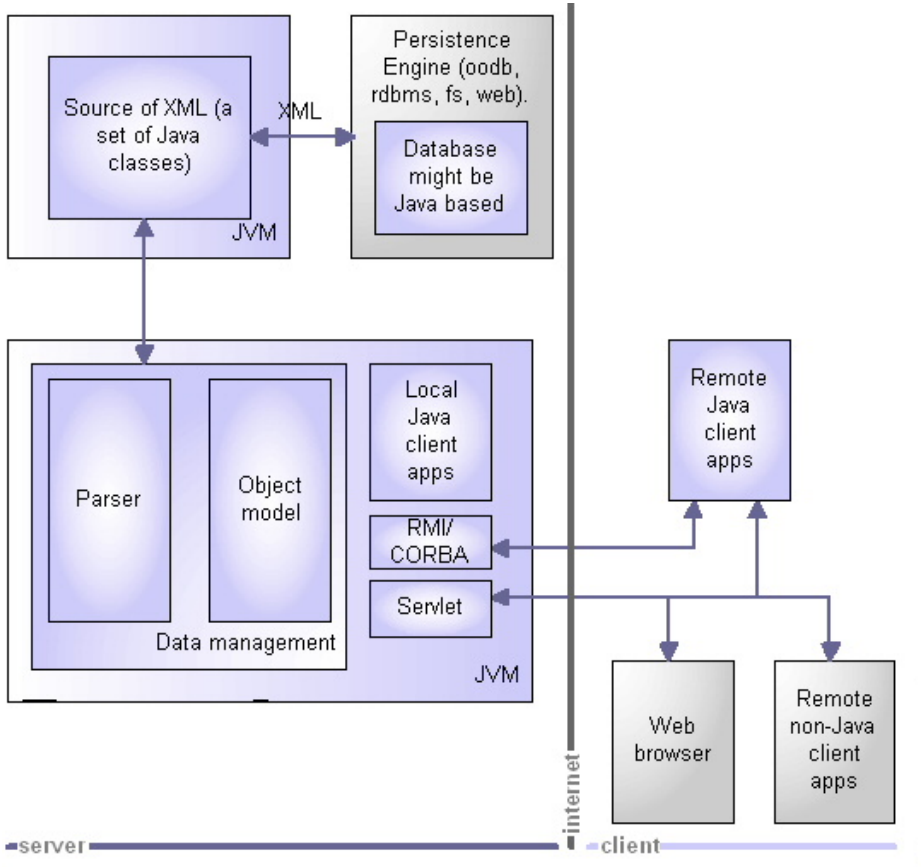
Firewall

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)

Applicazioni Java - XML



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.

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.

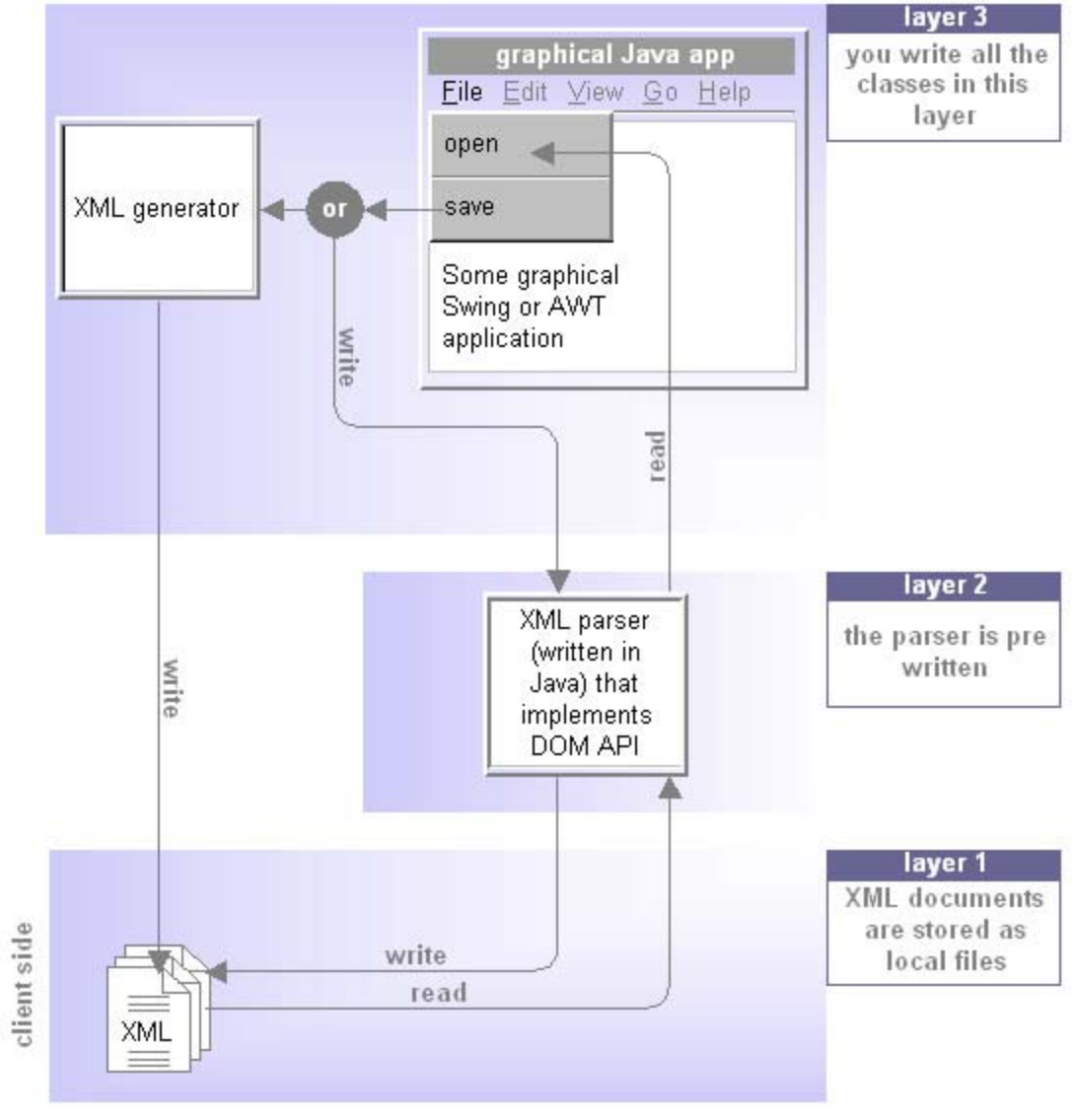
•*Tecnologie innovative.* Essendo tecnologie nuove ci sono meno sistemi reali sui quali confrontarsi.

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*

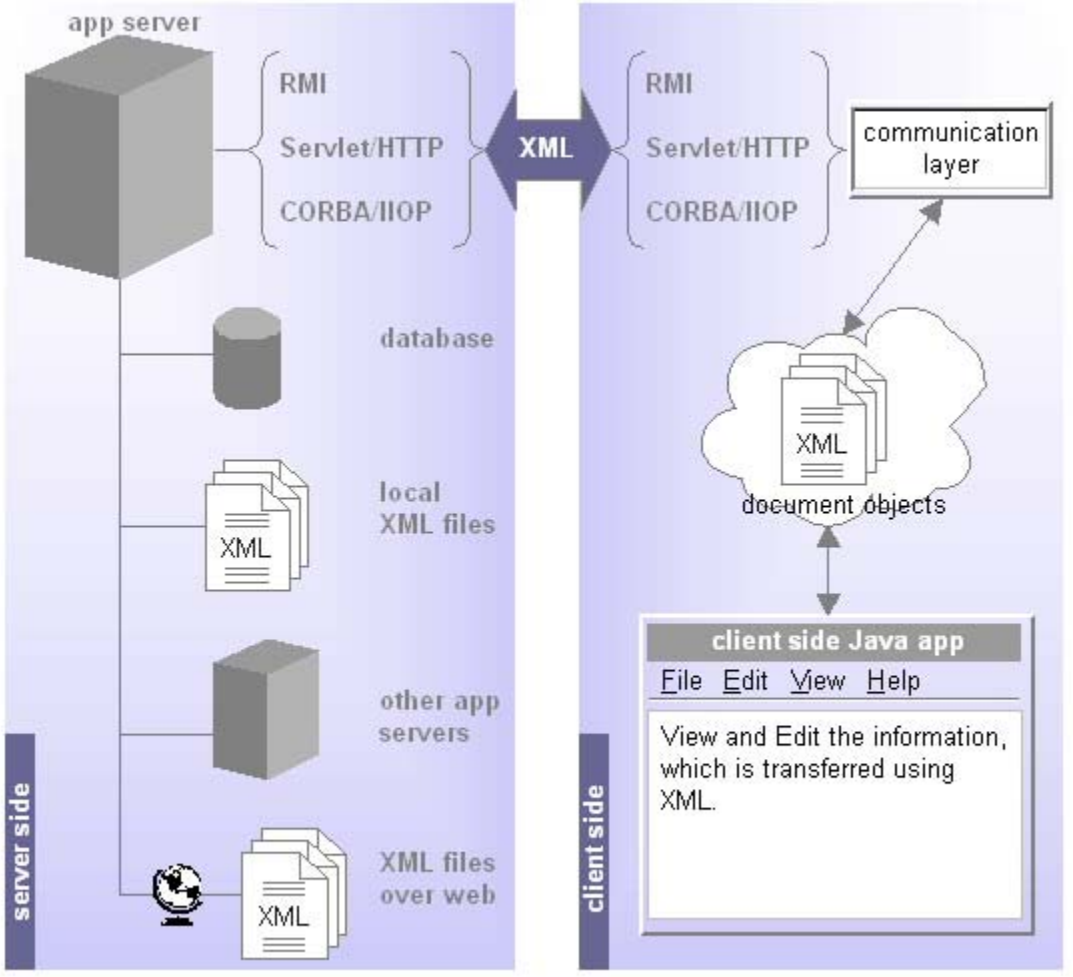
Applicazioni Java – XML *Client side - Graphical Java Applications*



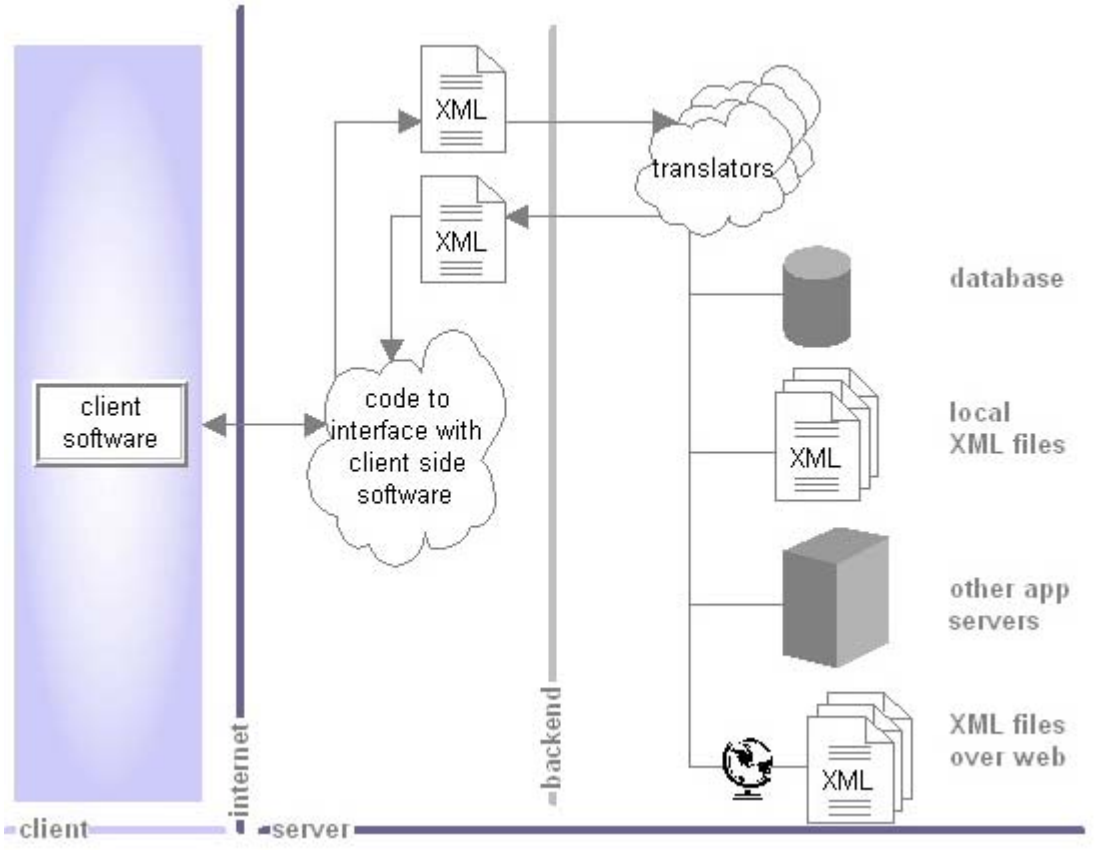
In questo tipo di applicazione le informazioni sono memorizzate in un documento XML (file). Non è necessario utilizzare un formato binario proprietario essendo possibile definire i propri tag.
Andranno realizzate classi che permettano di:
- importare/esportare un formato XML
- validare tramite un DTD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



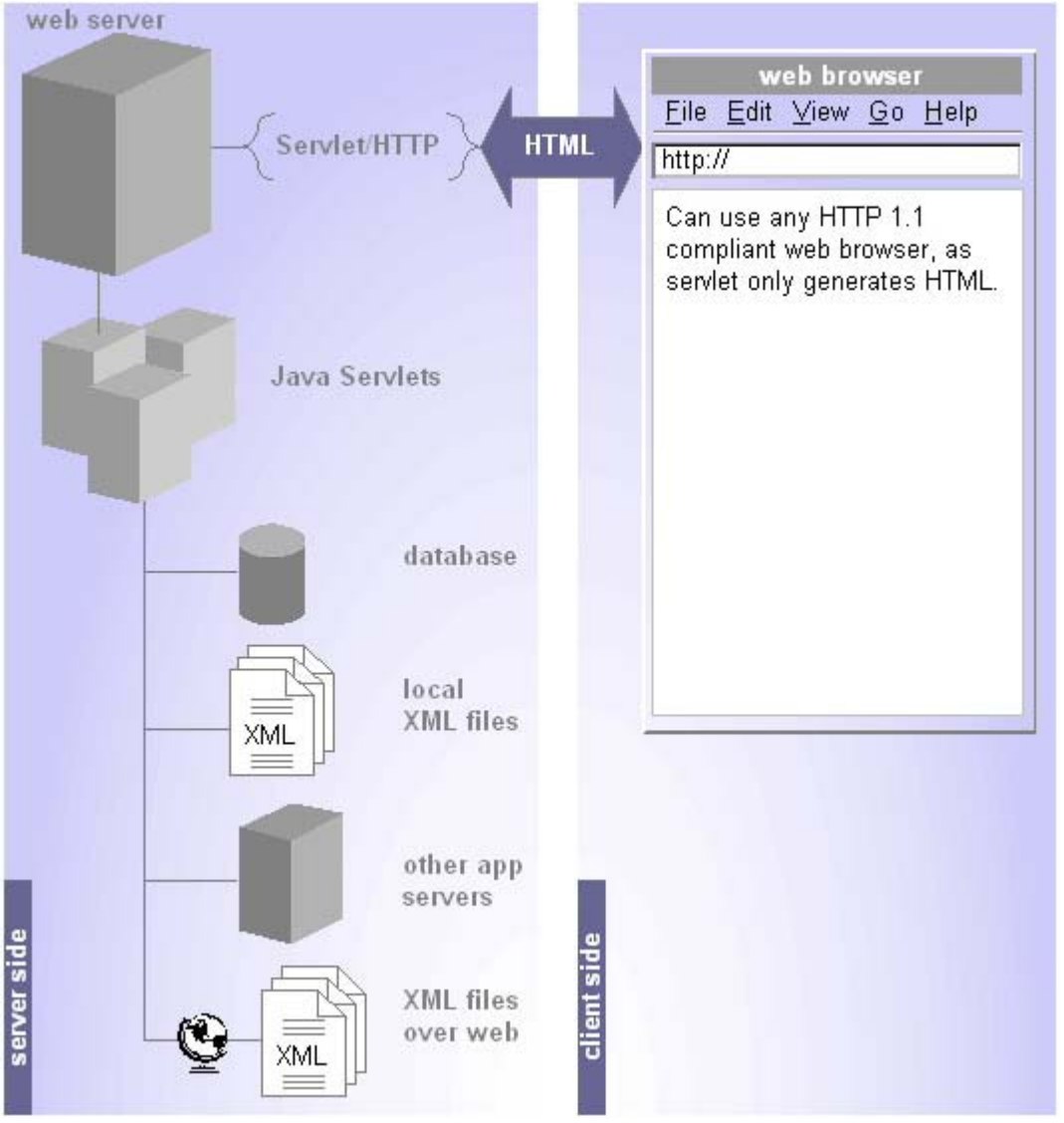
Un application server lega differenti networked software components assieme in modo da poter provvedere informazioni da differenti sorgenti a differenti clients. Realizzare una application server consiste dunque nel realizzare differenti sistemi interconnessi tra loro che accettano e distribuiscono informazioni 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.



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.