



CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

XML

a.a. 2020-2021

Che cos'è XML?

- XML: Extensible Markup Language:
 - è un linguaggio che consente la rappresentazione di documenti e dati strutturati su supporto digitale
 - è uno strumento potente e versatile per la creazione, memorizzazione e distribuzione di documenti digitali
 - la sua **sintassi rigorosa** e al contempo **flessibile** consente di utilizzarlo nella rappresentazione di dati strutturati anche molto complessi

Le origini

- XML è stato sviluppato dal **World Wide Web Consortium (W3C)**
- Nel **1996** è stato formato un gruppo di lavoro con l'incarico di definire un linguaggio a markup estensibile di uso generale
- Le specifiche sono state rilasciate come **W3C Recommendation** nel 1998 e aggiornate nel 2004
- XML deriva da **SGML (Standard Generalized Markup Language)**, un linguaggio di mark-up dichiarativo sviluppato dalla International Standardization Organization (ISO), e pubblicato ufficialmente nel 1986 con la sigla ISO 8879
- XML nasce come un sottoinsieme semplificato di SGML orientato all'utilizzo su World Wide Web
- Ha assunto ormai un ruolo autonomo e una diffusione ben maggiore del suo progenitore

XML come linguaggio di markup

- Un linguaggio di markup è composto da istruzioni, **definite tag** o **marcatori**, che descrivono la struttura e la forma di un documento
 - Ogni marcitore (o coppia di marcatori) identifica **un elemento** o componente del documento
- I marcatori vengono inseriti all'interno del documento
 - Un documento XML è “leggibile” da un utente umano senza la mediazione di software specifico

Esempio

- Un documento XML è leggibile , chiaro, intuibile:

```
<documento>           ← Marcatore di inizio  
    <corpo>  
        Testo del mio primo documento  
    </corpo >  
</documento>           ← Marcatore di fine
```

- **Attenzione:** XML è **case sensitive**
 - nei **nomi dei tag** distingue fra maiuscole e minuscole

Altro esempio

<prenotazione>

 <idVolo>PA321</idVolo>

 <idCliente>PP2305</idCliente>

 <data>22-10-2001</data>

 <prezzo valuta="Euro">245</prezzo>

</prenotazione>

Esempio 3

```
<utenti>
  <utente>
    <nome>Luca</nome>
    <cognome>Cicci</cognome>
    <indirizzo>Milano</indirizzo>
  </utente>
  <utente>
    <nome>Max</nome>
    <cognome>Rossi</cognome>
    <indirizzo>Roma</indirizzo>
  </utente>
</utenti>
```

Come può essere usato XML?

- XML separa i dati dalla presentazione
 - Non fornisce nessuna informazione su come i dati debbano essere visualizzati
 - Lo stesso XML può essere usato i differenti scenari di presentazione
- XML è spesso un complemento di HTML
 - Spesso XML viene usato per memorizzare o trasportare i dati, mentre HTML è utilizzato per formattare e visualizzare i dati
- XML separa i dati da HTML
 - Con XML, quando si visualizzano i dati non è necessario editare il file HTML se i dati cambiano
 - Con XML i dati sono memorizzati in file separati
 - Utilizzando JavaScript è possibile leggere un file XML e aggiornare i dati di una pagina HTML

Dati per le transazioni

- Esistono migliaia di formati XML, in molte industrie differenti, per descrivere le transazioni di ogni giorno:
 - Stocks and Shares (titoli e azioni)
 - Financial transactions
 - Medical data
 - Mathematical data
 - Scientific measurements
 - News information
 - Weather services
 - ...

Esempio: XML News

- È una specifica per scambiare news

```
<?xml version="1.0" encoding="UTF-8"?>
<nitf>
  <head>
    <title>Colombia Earthquake</title>
  </head>
  <body>
    <headline>
      <h1>143 Dead in Colombia Earthquake</h1>
    </headline>
    <byline>
      <bytag>By Jared Kotler, Associated Press Writer</bytag>
    </byline>
    <dateline>
      <location>Bogota, Colombia</location>
      <date>Monday January 25 1999 7:28 ET</date>
    </dateline>
  </body>
</nitf>
```

XML: caratteristiche

- XML è **indipendente dal tipo di piattaforma** hardware e software su cui viene utilizzato
- Permette la rappresentazione di qualsiasi tipo di documento (e di struttura) indipendentemente dalle finalità applicative
- È **indipendente** dai dispositivi di archiviazione e visualizzazione:
 - può essere archiviato su qualsiasi tipo di supporto digitale
 - può essere visualizzato su qualsiasi dispositivo di output
 - può essere facilmente trasmesso via Internet tramite i protocolli HTTP, SMTP, FTP

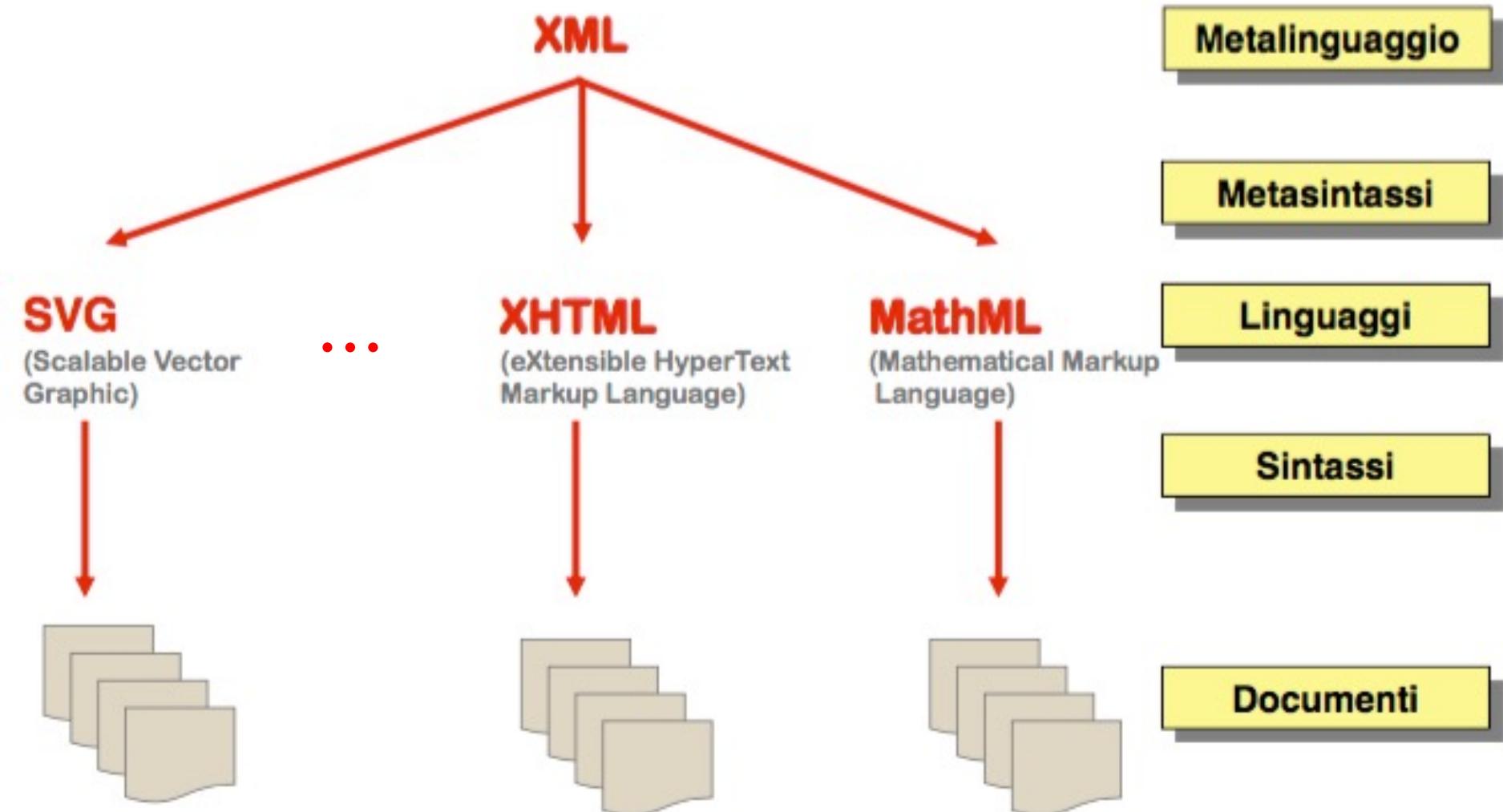
XML: caratteristiche

- XML è uno **standard di pubblico dominio**
- Ogni software “conforme a XML” è in grado di gestire dati in formato XML
- Sono disponibili numerose applicazioni e librerie open source per la manipolazione di dati in formato XML basate su diversi linguaggi di programmazione (Java, C, C#, Python, Perl, PHP...)
- Una applicazione in grado di elaborare dati in formato XML viene definita **elaboratore XML**

XML come metalinguaggio

- XML è **un metalinguaggio**
 - Definisce un insieme **regole** (meta-)sintattiche, attraverso le quali è possibile **descrivere formalmente un linguaggio di markup**, detto applicazione XML
- Ogni applicazione XML:
 - eredita un insieme di caratteristiche sintattiche comuni
 - definisce una sua sintassi formale
 - è dotata di una semantica

Metalinguaggio e linguaggi



Documenti ben formati e documenti validi

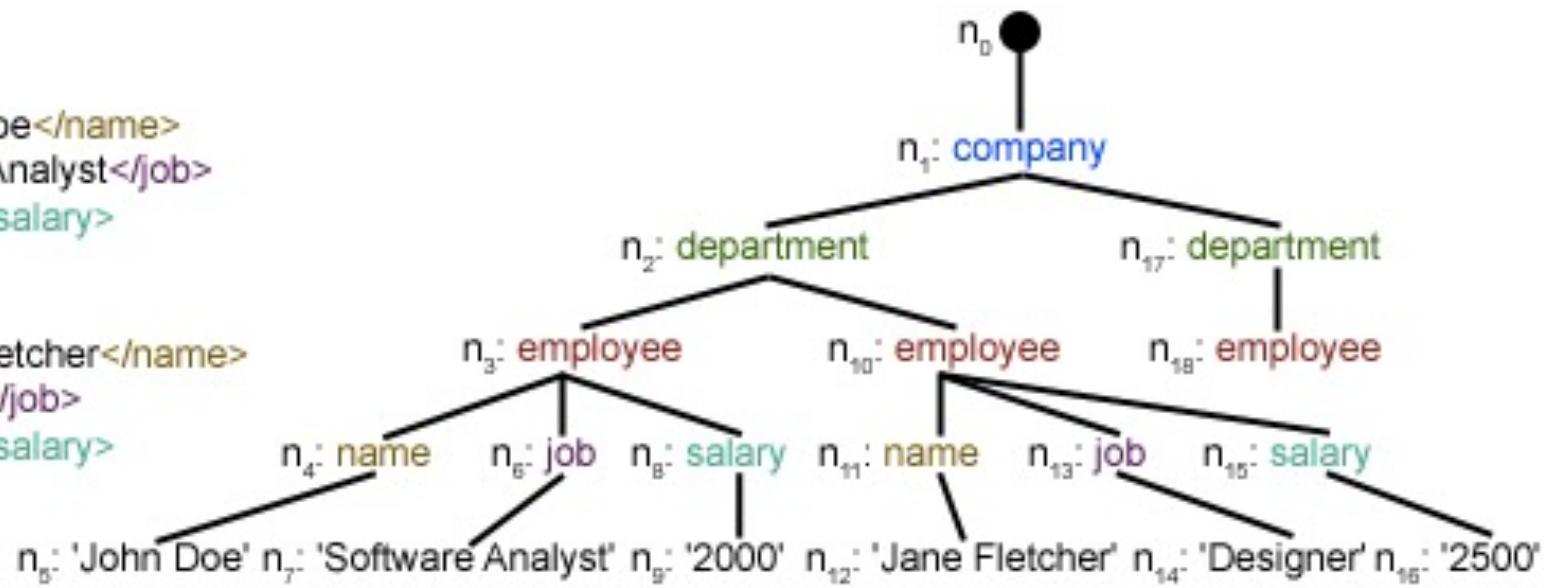
- In XML ci sono regole sintattiche (o meglio meta-sintattiche)
 - come dobbiamo scrivere le informazioni all'interno dei documenti
- Ci possono essere (ma non è obbligatorio) regole semantiche
 - cosa possiamo scrivere in un documento XML
- Un documento XML che rispetta le regole sintattiche si dice **ben formato (well-formed)**
- Un documento XML che rispetta le regole sintattiche e le regole semantiche si dice **valido**

Struttura **logica** di un documento XML

- Un **documento XML**
 - è strutturato in modo gerarchico
 - è composto da elementi
- Un **elemento**
 - rappresenta un componente logico del documento
 - può contenere un frammento di testo oppure altri elementi (sotto-elementi)
- Ad un elemento possono essere associate informazioni descrittive chiamate **attributi**
- Gli elementi sono organizzati ad albero con radice **root**
- Ogni documento XML può essere rappresentato come un albero
 - **document-tree**

XML Document-Tree

```
<company>
  <department>
    <employee>
      <name>John Doe</name>
      <job>Software Analyst</job>
      <salary>2000</salary>
    </employee>
    <employee>
      <name>Jane Fletcher</name>
      <job>Designer</job>
      <salary>2500</salary>
    </employee>
  </department>
  <department>
    <employee>
    </employee>
  </department>
</company>
```



Ogni documento XML deve avere una radice

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

- Esempio con radice `<note>`

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

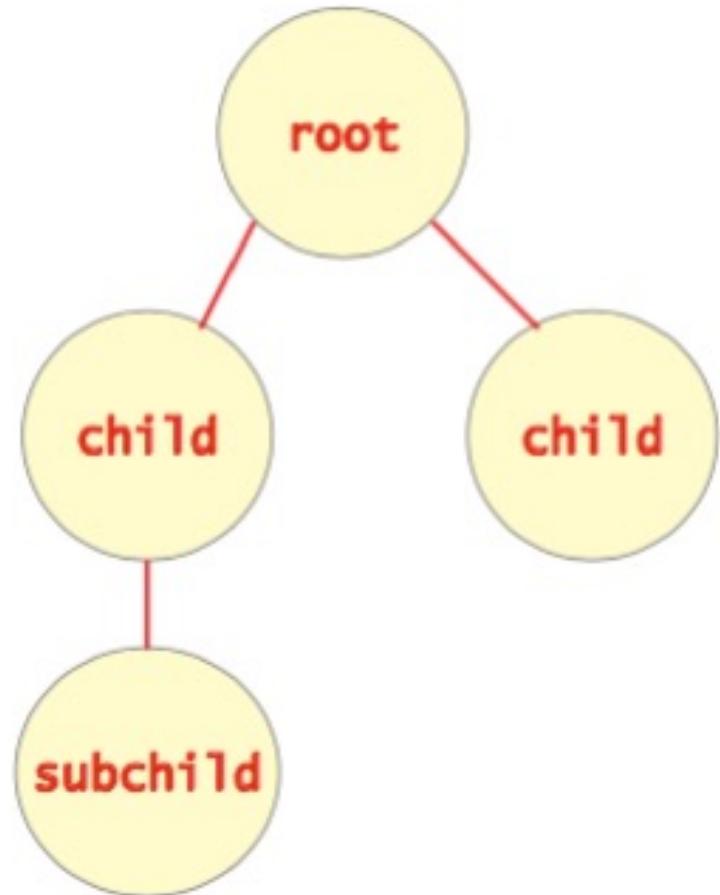
Struttura **fisica** di un documento XML

- Un documento XML è un semplice file di testo (.xml)
- La struttura del documento viene rappresentata mediante marcatori (**markup**)
- Gli elementi sono rappresentati mediante **tag**
 - coppie di marcatori che racchiudono il contenuto dell'elemento
- I **sottoelementi** sono tag contenuti all'interno di un altro tag
- Gli **attributi** vengono rappresentati sotto forma di coppie *nome-valore* all'interno dei tag
- La **radice** è un tag che racchiude tutto il resto del documento (e quindi tutti gli altri tag)
- Un documento può inoltre contenere *spazi bianchi*, a capo e **commenti**

Struttura logica e fisica

- Esiste una corrispondenza diretta fra struttura fisica (**markup**) e struttura logica (**tree**)

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



Aspetti di sintassi

- Un documento XML è una stringa di caratteri **ASCII o Unicode**
- Nomi di elementi, attributi e entità sono **case-sensitive**
- Il mark-up è separato dal contenuto testuale mediante caratteri speciali:
 - < > &** (parentesi angolari e ampersand)
 - " " '** (doppi apici e apici)
- I caratteri speciali non possono comparire come contenuto testuale e devono essere eventualmente sostituiti mediante i riferimenti a entità
 - <** (<), **>** (>), **&** (&), **"** ("'), and **'** ('')

Struttura formale di un documento XML

- Un documento è costituito da due parti:
 - **Prologo:** contiene una dichiarazione XML ed il riferimento (opzionale) ad altri documenti che ne definiscono la struttura o direttive di elaborazione
 - **Corpo:** è il documento XML vero e proprio

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="gree.css"?>

<root>
  <!-- Questo è un commento -->
  <child>
    ...
  </child>
  <child>
    ...
  </child>
</root>
```

Prologo

Corpo

Prologo: XML Declaration

- Ogni documento XML inizia con un prologo che contiene una **XML declaration**
- Forme di XML declaration:

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

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

- Contiene informazioni su:
 - Versione: per ora solo 1.0
 - Set di caratteri (opzionale)

Prologo: riferimenti a documenti esterni

- Il prologo può contenere riferimenti a documenti esterni utili per il trattamento del documento
- **Processing instruction:** istruzioni di elaborazione

Esempio. Rappresentazione mediante CSS:

```
<?xmlstylesheet type="text/css" href="gree.css"?>
```

- **Doctype declaration:** grammatica da utilizzare per la validazione del documento
 - grammatica contenuta in un **file locale**

```
<!DOCTYPE book SYSTEM "book.dtd">
```
 - grammatica accessibile ad un **URL pubblico**

```
<!DOCTYPE book PUBLIC "http://www.books.org/book.dtd">
```

Commenti

- I commenti possono apparire ovunque in un documento XML (sia nel prologo che nel corpo)
- I commenti sono utili per:
 - spiegare la struttura del documento XML
 - commentare parti del documento durante le fasi di sviluppo e di test del nostro software
- I commenti non vengono mostrati dai browser ma sono visibili da parte di chi guarda il codice sorgente del documento XML

`<!-- Questo è un commento -->`

Element e Tag

- Un **elemento** è un frammento di testo racchiuso fra uno start tag e un end tag
- Uno **start tag** è costituito da un nome più eventuali attributi racchiusi dai simboli '<', '>'

<TagName attribute-list>

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

</TagName>

- Un tag vuoto è rappresentabile come:

<TagName attribute-list />

- Equivale a

<TagName attribute-list></TagName>

- *Attenzione: I tag non possono avere nomi che iniziano per XML, XMI, Xml, xml...*

I nomi degli elementi

- I nomi dei tag possono essere inventati a nostro piacere, a patto che rispettiamo alcune regole di definizione:
 - Devono iniziare con un carattere o con un underscore (_)
 - Non possono iniziare con numeri
 - Non possono contenere spazi
 - Possono contenere un qualsiasi numero di lettere, numeri, trattini, punti, underscore
 - Evitare di utilizzare le tre lettere **xml** nei nomi dei tag perchè spesso queste corrispondono a nomi utilizzati anche da tecnologie elaborate dai gruppi di lavoro sull'XML

Elementi

- Un elemento può contenere testo, attributi, altri elementi, un mix delle voci precedenti. Esempio:
- <title>, <author>, <year>, e <price> hanno del **text content** in quanto essi contengono testo (e.g., Harry Potter e 29.99)
- <bookstore> e <book> hanno degli **element content**, poiché contengono altri elementi
- <book> ha un **attribute** (category="children")

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J. K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Attributi

- A ogni elemento possono essere associati uno o più **attributi** che ne specificano ulteriori caratteristiche o **proprietà** non strutturali
- Ad esempio:
 - la lingua del suo contenuto testuale
 - un identificatore univoco
 - un numero di ordine
 - ...
- Gli attributi XML sono caratterizzati da
 - un **nome** che li identifica
 - un **valore**

Esempio di documento con attributi

```
<?xml version="1.0" ?>
<articolo titolo="Titolo dell'articolo">
    <paragrafo titolo="Titolo del primo paragrafo">
        <testo>Blocco di testo del primo
        paragrafo</testo>
        <immagine file="immagine1.jpg"></immagine>
    </paragrafo>
    <paragrafo titolo="Titolo del secondo paragrafo">
        <testo>Blocco di testo del secondo
        paragrafo</testo>
        <codice>Esempio di codice</codice>
        <testo>Altro blocco di testo</testo>
    </paragrafo>
    <paragrafo tipo="bibliografia">
        <testo>Riferimento ad un articolo</testo>
    </paragrafo>
</articolo>
```

Elementi o attributi?

- Qualche regola per decidere:
 - Un **elemento è estendibile** in termini di contenuto (con elementi figli) e di attributi
 - Un **attributo non è estendibile**: può solo modellare una proprietà di un elemento in termini di valore
 - Un elemento è un'entità a se stante (un oggetto?)
 - Un attributo è strettamente legato ad un elemento
 - Un attributo può solamente contenere un valore “atomico”
- *In pratica non c'è una regola valida in assoluto*
- La scelta dipende da diversi fattori: leggibilità, semantica, tipo di applicazione, efficienza...

Elementi o attributi: esempio

- Vediamo tre varianti dello stesso pezzo di documento che usano in modo diverso elementi e attributi

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

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

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

A Simple XML Document

```
<Article>

    <Author>Gerhard Weikum</author>

    <Title>the web in ten years</title>

    <Text>

        <Abstract>in order to evolve...</Abstract>

        <Section number="1" title="introduction">
            The <index>web</index> provides the
            universal...

        </Section>

    </Text>

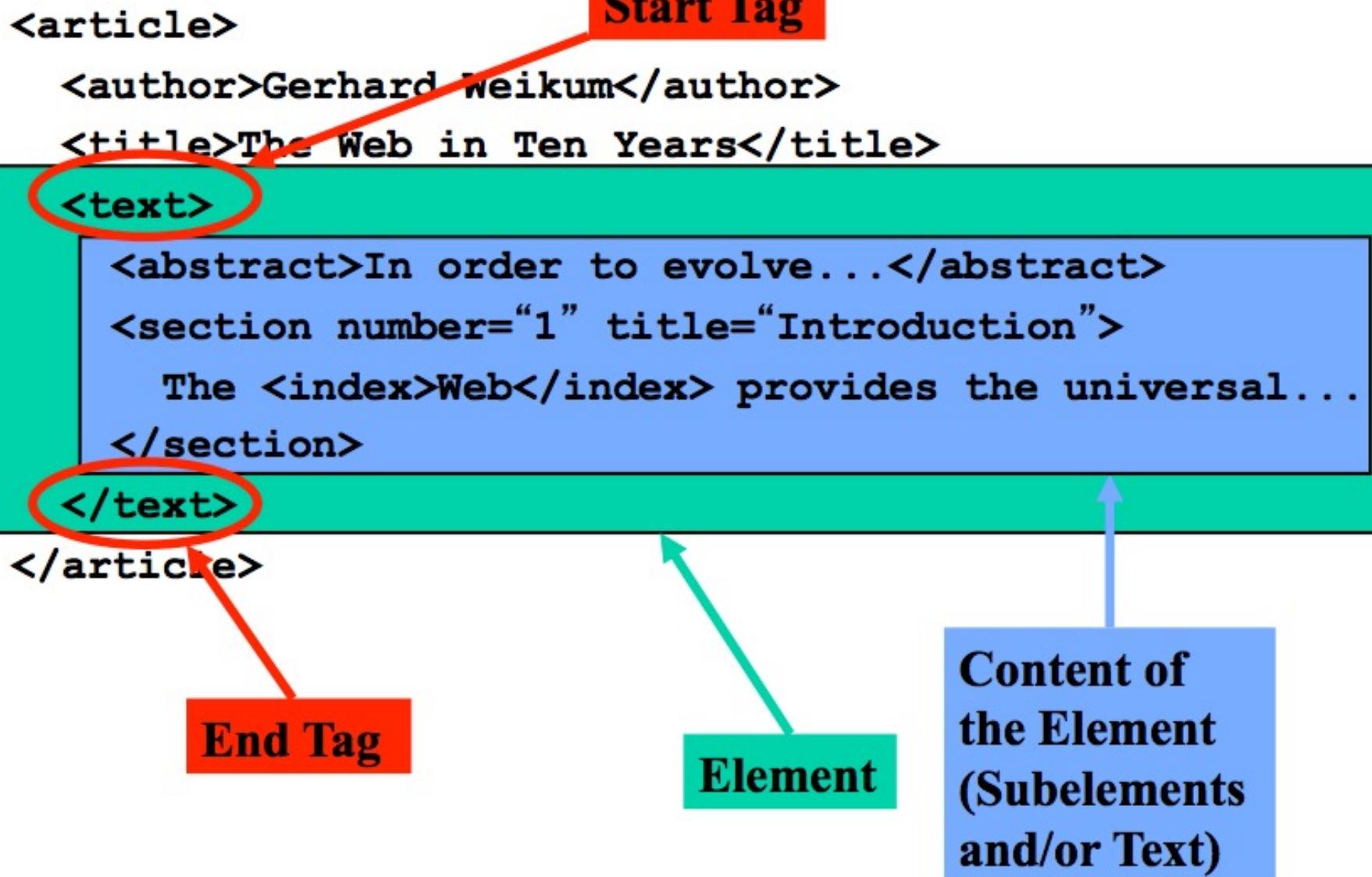
</Article>
```

A Simple XML Document

```
<Article>
  <Author>Gerhard Weikum</author>
  <Title>the web in ten years</title>
  <Text>
    <Abstract>in order to evolve...</Abstract>
    <Section number="1" title="introduction">
      The <index>web</index> provides the
      universal...
    </Section>
  </Text>
</Article>
```

Freely definable tags

Example of XML document

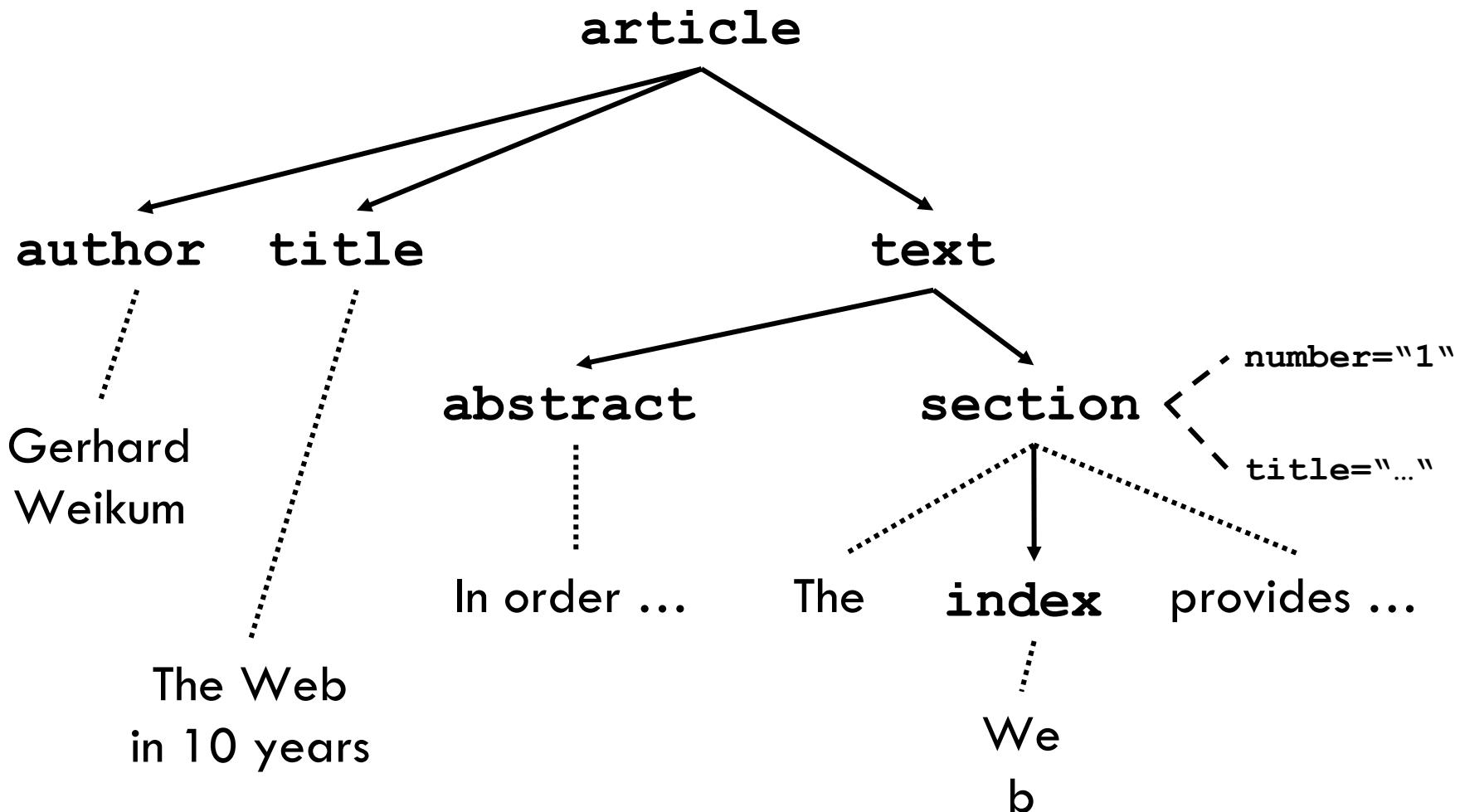


Example of an XML document

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

Attributes with
name and value

XML Documents as Ordered Trees



Riferimenti ad entità

I riferimenti ad entità servono per rappresentare caratteri riservati (per esempio, < > o &)

Nome entità	Riferimento	Carattere
lt	<	<
gt	>	>
amp	&	&
apos	'	'
quot	"	"

Oppure per rappresentare caratteri UNICODE mediante la notazione &#XXXX:

- ½ → ½

Sezione CDATA

- Per poter inserire brani di testo (porzioni di codice XML o XHTML) senza preoccuparsi di sostituire i caratteri speciali si possono utilizzare le sezioni **CDATA (Character Data)**
- Il testo contenuto in una sezione CDATA **NON viene analizzato dal parser**
- Una sezione CDATA può contenere caratteri “normalmente” proibiti
- Si utilizza la seguente sintassi:
<! [CDATA[Contenuto della sezione]]>
- L'unica sequenza non ammessa è **]]** (chiusura)
- Esempi:

```
<E1> <! [CDATA[ <<'' !] && ]]> </E1>
<E> <! [CDATA [<Elemento/><A>Ciao</A>]]> </E>
```

Conflitti sui nomi

- Capita abbastanza comunemente, soprattutto in documenti complessi, di dare nomi uguali ed elementi (o attributi) con significati diversi
- Ad esempio:

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

Esempi di conflitti sui nomi

- In XML i nomi degli elementi sono definiti dagli sviluppatori. Possono capitare dei conflitti quando si provano ad integrare documenti XML da applicazioni diverse
- Questo XML contiene informazioni di una tabella di dati (`table`):

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

- Questo XML su di un tavolo (mobile):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Namespace

- Per risolvere il problema si ricorre al concetto di “**spazio dei nomi**” (**namespace**)
- Si usano **prefissi che identificano il vocabolario di appartenenza** di elementi ed attributi
- Ogni **prefisso** è associato ad un **URI** (Uniform Resource Identifier) ed è un alias per l’URI stesso
- L’URI in questione è normalmente un URL: si ha quindi la certezza di univocità
- È un meccanismo simile ai nomi lunghi delle classi in Java (i package definiscono un sistema di namespace):
 - Nome breve: **JButton**
 - Nome lungo: **javax.swing.JButton**

Esempio di utilizzo di namespace

Prefisso

Dichiarazione del prefisso
e associazione all'URI

URI

```
<lb:libro xmlns:lb="mysite.com/libri">  
  <au:autore xmlns:au="mysite.com/autori">  
    <au:titolo>Sir</au:titolo>  
    <au:nome>William Shakespeare</au:nome>  
  </au:autore>  
  <lb:titolo>Romeo and Juliet</lb:titolo>  
</lb:libro>
```

Definizione di namespace

- Per **definire** un namespace si usa la seguente sintassi:
xmlns : NamespacePrefix="NamespaceURI"
- La definizione è un attributo di un elemento e può essere messa ovunque all'interno del documento
- Lo **scope** del namespace è l'elemento all'interno del quale è stato dichiarato
 - Si estende a tutti i sottoelementi
 - Se si dichiara un namespace nell'elemento radice, il suo scope è l'intero documento
- L'URI può essere qualsiasi (il parser non ne controlla l'univocità) ma dovrebbe essere scelto in modo da essere effettivamente univoco

Esempio

```
<DC:Docenti xmlns:DC="www.unisa.it/docenti" >  
  <DC:Docente codAteneo="112233">  
    <DC:Nome>Rita</DC:Nome>  
    <DC:Cognome>Francese</DC:Cognome>  
    <CR:Corso id="123" xmlns:CR="www.unisa.it/corsi">  
      <CR:Nome>Programmazione web</CR:Nome>  
    </CR:Corso >  
    <CO:Corso id="124" xmlns:CO="www.unisa.it/corsi">  
      <CO:Nome>Programmazione I</CO:Nome>  
    </CO:Corso >  
  </DC:Docente>  
</DC:Docenti>
```

- **CR** e **CO** sono prefissi “collegati” allo stesso namespace
- Nel secondo elemento Corso è necessario ripetere la dichiarazione di namespace poiché ricade fuori dallo scope della prima dichiarazione
- Per evitare la seconda dichiarazione basta dichiarare il namespace in un elemento più in alto nella gerarchia

Namespace di default

- È possibile definire un namespace di default associato al prefisso nullo
- Tutti gli elementi non qualificati da prefisso appartengono al namespace di default
- *Attenzione: riduce la leggibilità di un documento*

```
<Docenti xmlns="www.unisa.it/docenti">  
    <Docente codAteneo="112233">  
        <Nome>Rita</Nome>  
        <Cognome>Francese</Cognome>  
        <CR:Corso id="123" xmlns:CR="www.unisa.it/corsi">  
            <CR:Nome>Programmazione web</CR:Nome>  
        </CR:Corso >  
    </Docente>  
</Docenti>
```

Ridefinizioni di prefissi

- Un **prefisso** di namespace (anche quello vuoto di default) può essere associato a diversi namespace all'interno di uno stesso documento
- È però preferibile evitare le ridefinizioni: *riducono la leggibilità del documento*

```
<PR:Docenti xmlns:PR="www.unisa.it/docenti">  
    <PR:Docente codAteneo="112233">  
        <PR:Nome>Rita</PR:Nome>  
        <PR:Cognome>Francese</PR:Cognome>  
        <PR:Corso id="123" xmlns:PR="www.unisa.it/corsi">  
            <PR:Nome>Programmazione web</PR:Nome>  
        </PR:Corso >  
    </PR:Docente>  
</PR:Docenti>
```

Vincoli di buona formazione

- Affinché un documento XML sia **ben formato**:
 - Deve contenere una dichiarazione (XML Declaration) corretta
 - Il corpo deve avere un unico elemento radice
 - Ogni elemento deve avere un tag di apertura e uno di chiusura
 - se l'elemento è vuoto si può utilizzare la forma abbreviata (<nometag/>)
 - Gli elementi devono essere opportunamente nidificati, cioè i tag di chiusura devono seguire l'ordine inverso dei rispettivi tag di apertura (<**tag**><**subtag**>...</**subtag**></**tag**>)
 - I nomi dei tag di apertura e chiusura devono coincidere
 - anche in termini di maiuscole e minuscole
 - I valori degli attributi devono sempre essere racchiusi tra singoli o doppi apici

Documenti ben formati e documenti validi

- In XML ci sono **regole sintattiche**
 - come dobbiamo scrivere le informazioni all'interno dei documenti
- Ci possono essere **regole semantiche**
 - cosa possiamo scrivere in un documento XML
- Un documento XML che rispetta le regole sintattiche si dice **ben formato**
- Un documento XML che rispetta le regole sintattiche e le regole semantiche si dice **valido**

Validazione: Document Type Definition (DTD)

- Un DTD è costituito da un elenco di dichiarazioni (**markup declaration**) che descrivono la struttura del documento
- Le dichiarazioni di un DTD definiscono:
 - gli elementi strutturali (**element**) di un documento mediante un identificatore generico
 - il modello di contenuto di ogni elemento (**content model**), ovvero gli elementi che contiene e le loro relazioni (un elemento può essere vuoto)
 - la lista degli **attributi** associati a ciascun elemento e il loro tipo

Esempio di file XML e DTD (DTD-Elements)

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

message.dtd

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

- **PCDATA** (Parsed Character Data) rappresenta il tipo di dato carattere parsabile
- **CDATA** (Character Data) rappresenta il tipo di dato di tipo carattere (immune al parsing)

DTD-Elements (2)

- Si può utilizzare anche le clausole **empty** o **any**:

- **<!ELEMENT br EMPTY>**

- **<!ELEMENT note ANY>** *<!-- può contenere qualsiasi combinazione di dati parsabili -->*

```
<!ELEMENT element-name (child-name)>
```

```
<!-- una sola occorrenza -->
```

```
<!ELEMENT element-name (child-name+)>
```

```
<!-- minimo una occorrenza -->
```

```
<!ELEMENT element-name (child-name*)>
```

```
<!-- zero o più occorrenze -->
```

```
<!ELEMENT element-name (child-name?)>
```

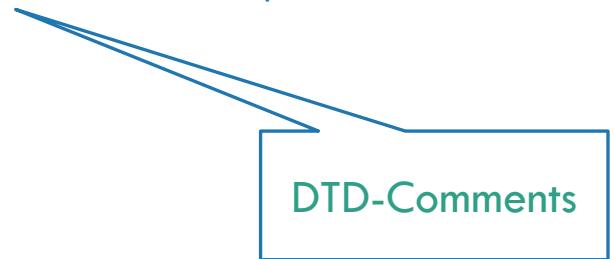
```
<!-- zero o una occorrenza -->
```

```
<!ELEMENT element-name (to | from)>
```

```
<!-- occorrenza esclusiva -->
```

```
<!ELEMENT element-name (#PCDATA | to | from)>
```

```
<!-- contenuto misto esclusivo -->
```



DTD-Comments

DTD-Elements (3)

<!ATTLIST payment type CDATA "check"> <!-- valore di default-->

Valid XML: <payment type="check" />

<!ATTLIST person number CDATA #REQUIRED>

Valid XML: <person number="5677" />

Invalid XML: <person />

<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML: <contact fax="123-456789" />

Valid XML: <contact />

DTD-Elements (4)

<!-- valori degli attributi enumerati-->

<!ATTLIST payment type (check | cash) "cash">

Valid XML: <payment type="check" />

<!ATTLIST sender company CDATA #FIXED "Apple">

Valid XML: <sender company="Apple" />

Invalid XML: <sender company="Samsung" />

DTD-Elements (5)

```
<!ELEMENT artist EMPTY>
```

```
<!ATTLIST artist name CDATA #REQUIRED>
```

```
<!ATTLIST artist artistId ID #REQUIRED>
```

Valid XML: <artist name="Pink Floyd" artistId="PF" />

```
<!ELEMENT albumEMPTY>
```

```
<!ATTLIST album name CDATA #REQUIRED>
```

```
<!ATTLIST album albumArtistId IDREF #IMPLIED>
```

Valid XML: <album name="The Division Bell" albumArtistId="PF" />

Invalid XML: <album name="The Wall" albumArtistId="Pink Floyd" />

Limiti e problemi del DTD

- I DTD sono difficili da comprendere
- Sono scritti in un linguaggio diverso da quello usato per descrivere le informazioni
 - Il formato non è XML
- Soffrono di alcune limitazioni:
 - Non permettono di definire il tipo dei dati (interi, reali, date, ...)
 - Non consentono di specificare il numero minimo o massimo di occorrenze di un tag in un documento

Validazione: XML Schema (XSD)

- *Dato che XML può descrivere tutto perché non usarlo per descrivere anche lo schema di un documento?*
 - È stato quindi definito lo standard XSD (**XML Schema Definition**)
- XSD nasce dall'idea di utilizzare XML per descrivere la struttura di XML:
 - Descrive le regole di validazione di un documento
 - Permette di tipizzare i dati (intero, stringa, ora, data, ecc.)
 - È estensibile ed aperto alla possibilità di supportare modifiche

Elementi di XSD

- Un documento XML Schema (XSD) comprende:
- **Namespace** di riferimento:

<http://www.w3.org/2001/XMLSchema>

- **Dichiarazione di:**
 - Elementi
 - Attributi
- **Definizione di tipi**
 - Semplici
 - Complessi
 - Estesi

Esempio di file XML e XSD

```
<?xml version="1.0"?>
<message
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://mysite.it/msg.xsd">
    <to>Bob</to>
    <from>Janet</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend</body>
</message>
```



- È un documento XML

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="message" type="messageType"/>
    <xs:complexType name="messageType">
        <xs:sequence>
            <xs:element name="to" type="xs:string"/>
            <xs:element name="from" type="xs:string"/>
            <xs:element name="heading" type="xs:string"/>
            <xs:element name="body" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Esempio di file XML e XSD (2)

- Descrive la struttura del documento XML note

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Esempio di file XML e XSD (3)

```
<xs:attribute name="lang" type="xs:string"/>  
<xs:attribute name="lang" type="xs:string" default="EN"/>  
<xs:attribute name="lang" type="xs:string" use="required"/>
```



```
<lastname>Smith</lastname>  
<lastname lang="EN">Smith</lastname>
```

```
<xs:element name="book">  
  <xs:complexType>  
    <xs:attribute name="year" type="xs:positiveInteger">  
      <xs:restriction base="xs:positiveInteger">  
        <xs:minInclusive value="1900"/>  
        <xs:maxInclusive value="2017"/>  
      </xs:restriction>  
    <xs:attribute>  
  </xs:complexType>  
</xs:element>
```

HTML e XML: XHTML

- HTML può essere descritto da uno schema XML?
- Quasi, però sono ammessi “pasticci” che XML non prevede:
 - Tag “non chiusi”:

(in XML
</br> o
)
 - Tag “incrociati” <u>Ciao</u>
(in XML <u>Ciao</u>)
- È stata definita una versione di HTML “corretta” in modo da rispettare la sintassi XML: XHTML
- ***Un documento XHTML è un documento XML ben formato che può essere validato su uno schema definito dal W3C***

DOM

- Il **DOM (Document Object Model)** è un modello ad oggetti definito dal W3C per navigare e creare contenuti XML
- Rappresenta il contenuto di un documento XML tramite **un albero** in memoria
- Permette di navigare l'albero ragionando per gradi di parentela (nodi figli, nodo padre, ecc.)
- Esistono 3 interfacce base
 - **Node** (è praticamente la base di tutto)
 - **NodeList** (collezione di nodi)
 - **NamedNodeMap** (collezione di attributi)
- Un **parser DOM** è un'applicazione in grado di leggere un file XML e creare un DOM e viceversa

Presentazione di documenti XML

- Un documento XML definisce il contenuto informativo e non come deve essere rappresentato tale contenuto
- ***La presentazione di un documento XML viene controllata da uno o più fogli di stile***
- Cascading Style Sheet (CSS) è usato con HTML
 - HTML usa tag predefiniti. È ben noto il significato di ogni tag e come deve essere visualizzato
- Extensible Stylesheet Language (XSL) usato con XML
 - XML non usa tag predefiniti ed il significato di ogni tag non è facile da comprendere
 - Es: <table> in HTML indica una tabella, mentre in XML potrebbe indicare un mobile o qualche altra cosa ed il browser non sa come visualizzarlo

XSL

- **XSL** = eXtensible Stylesheet Language
- Si occupa della trasformazione e della impaginazione di contenuti XML
- Si basa principalmente su:
 - **XSLT (XSL for Transformations)**: gestisce le trasformazioni e non l'impaginazione dei contenuti
 - **XSL-FO (XSL Formatting Objects)**: orientato alla visualizzazione ed impaginazione dei contenuti (es. in PDF)
 - **XPath (XML Path Language)**: serve per costruire percorsi di ricerca di informazioni all'interno di documenti XML

XSLT

- XSLT è un linguaggio di programmazione a tutti gli effetti
- Permette di gestire **variabili, parametri, cicli, condizioni, funzioni**
- Lavora sulla **struttura** del documento
 - Costruisce l'albero del documento
 - Lo attraversa cercando le informazioni indicate
 - Produce un nuovo documento - di solito XHTML - applicando le regole definite

<https://www.w3schools.com/xml/tryslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog>

XML Code:

Edit and Click Me »

XSLT Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
    <cd>
        <title>Empire Burlesque</title>
        <artist>Bob Dylan</artist>
        <country>USA</country>
        <company>Columbia</company>
        <price>10.90</price>
        <year>1985</year>
    </cd>
    <cd>
        <title>Hide your heart</title>
        <artist>Bonnie Tyler</artist>
        <country>UK</country>
        <company>CBS Records</company>
        <price>9.90</price>
        <year>1988</year>
    </cd>
```

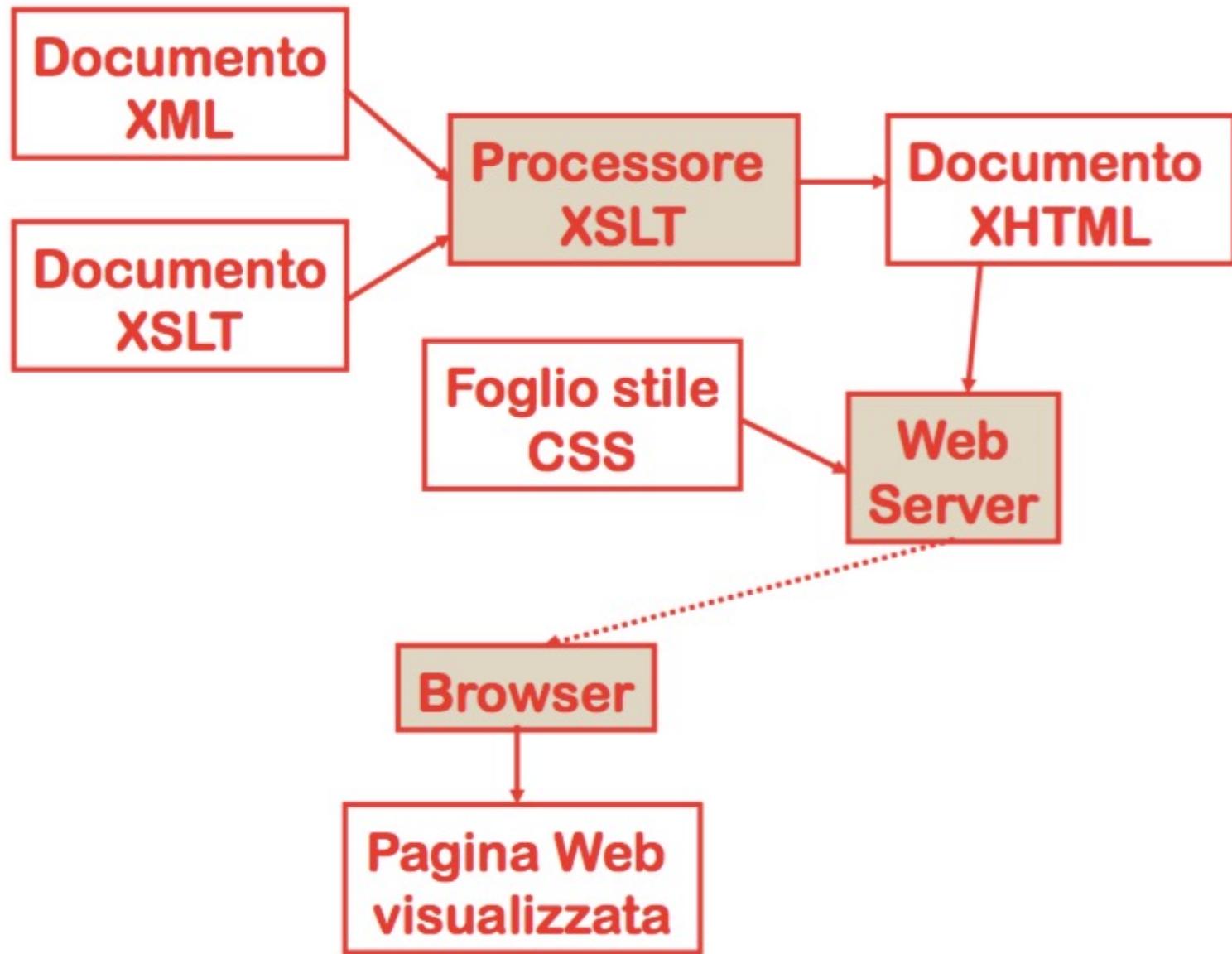
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
        <body>
            <h2>My CD Collection</h2>
            <table border="1">
                <tr bgcolor="#9acd32">
                    <th>Title</th>
                    <th>Artist</th>
                </tr>
                <xsl:for-each select="catalog/cd">
                    <xsl:if test="price>10">
                        <tr>
                            <td><xsl:value-of select="title"/></td>
                            <td><xsl:value-of select="artist"/></td>
                        </tr>
                    </xsl:if>
                </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:stylesheet>
```

Your Result:

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Still got the blues	Gary Moore
One night only	Bee Gees
Romanza	Andrea Bocelli
Black angel	Savage Rose
1999 Grammy Nominees	Many

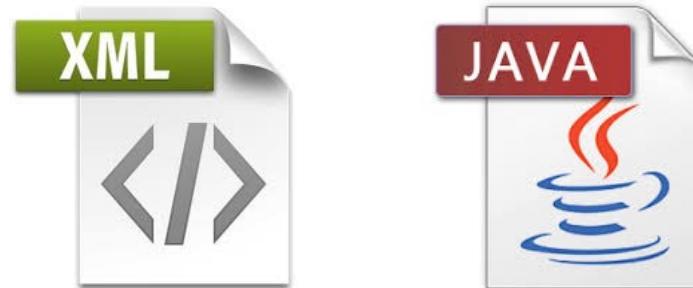
Un esempio tipico completo



Riferimenti

- XML Specification: <http://www.w3.org/XML/>
- XSL Specification: <http://www.w3.org/Style/XSL/>
- Guida XML: <http://www.w3schools.com/xml/default.asp>
- DTD Specification:
<http://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm>
- Guida DTD: <http://www.w3schools.com/dtd/default.asp>
- XSD Specification: <http://www.w3.org/2001/XMLSchema>
- Guida XSD: <http://www.w3schools.com/schema/default.asp>

XML e Java



Progetto: XML.zip

Elaborazione di documenti XML

- Un elemento importante che ha consentito la diffusione dei linguaggi e delle tecnologie XML è il supporto fornito da strumenti per il parsing
 - analisi sintattica
- Ogni applicazione che vuole fruire di XML deve includere un supporto per il parsing
- I parser XML sono diventati strumenti standard nello sviluppo delle applicazioni
 - per esempio, JDK, a partire dalla versione 1.4, integra le sue API di base con API specifiche per il supporto al parsing
 - specifiche JAXP (Java API for XML Processing)

Compiti del parser

- Decomporre i documenti XML (istanza) nei loro elementi costitutivi
 - elementi, attributi, testo, ecc.
- Eseguire controlli sul documento
 - Controllare che il documento sia ben formato
 - Controllare eventualmente che il documento sia valido (**DTD, XML Schema**)
- Non tutti i parser consentono la validazione dei documenti:
 - si parla quindi di parser **validanti** e **non validanti**
 - La validazione è una operazione computazionalmente costosa e quindi è opportuno potere scegliere se attivarla o meno

Modelli di parsing per XML

- Dal punto di vista dell'interfacciamento tra applicazioni e parser esistono due grandi categorie di API:

1. Interfacce basate su eventi

- Interfacce **SAX** (Simple API for XML)
- sfruttano un modello a callback

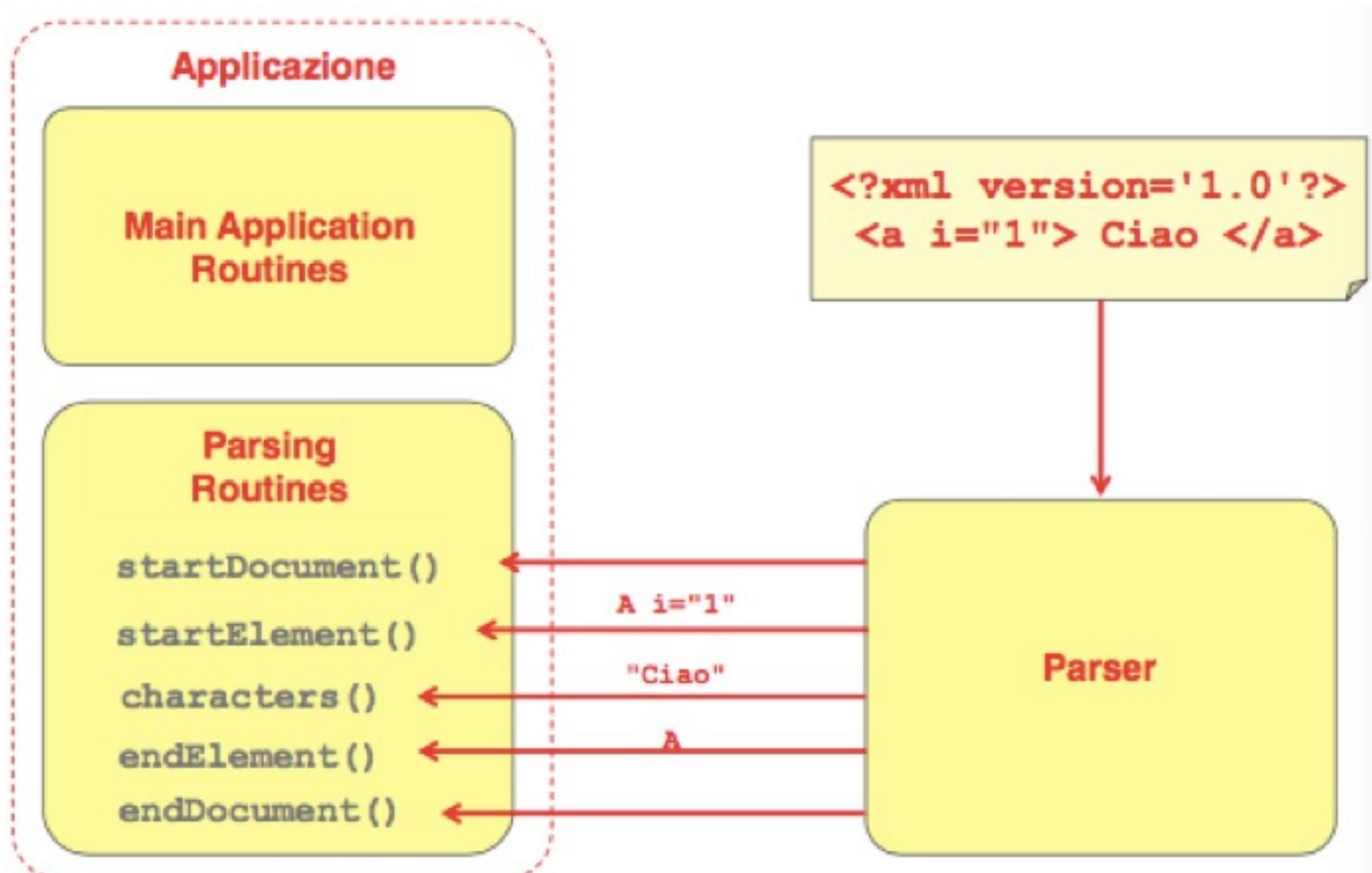
2. Interfacce Object Model

- W3C **DOM** Recommendation

1. Interfacce ad eventi

- L'applicazione implementa un insieme di **metodi di callback** che vengono invocati dal parser mentre elabora il documento
 - Le callback sono invocate al verificarsi di specifici **eventi** (es. start-tag, end-tag, ...)
- I parametri passati al metodo di callback forniscono ulteriori informazioni sull'evento
 - Nome dell'elemento
 - Nome e valore assunto dagli attributi
 - Valore delle stringhe contenute ...
- È una modalità semplice ed efficiente
 - non mantiene in memoria una rappresentazione del documento
 - usato per grossi file XML o per accessi veloci

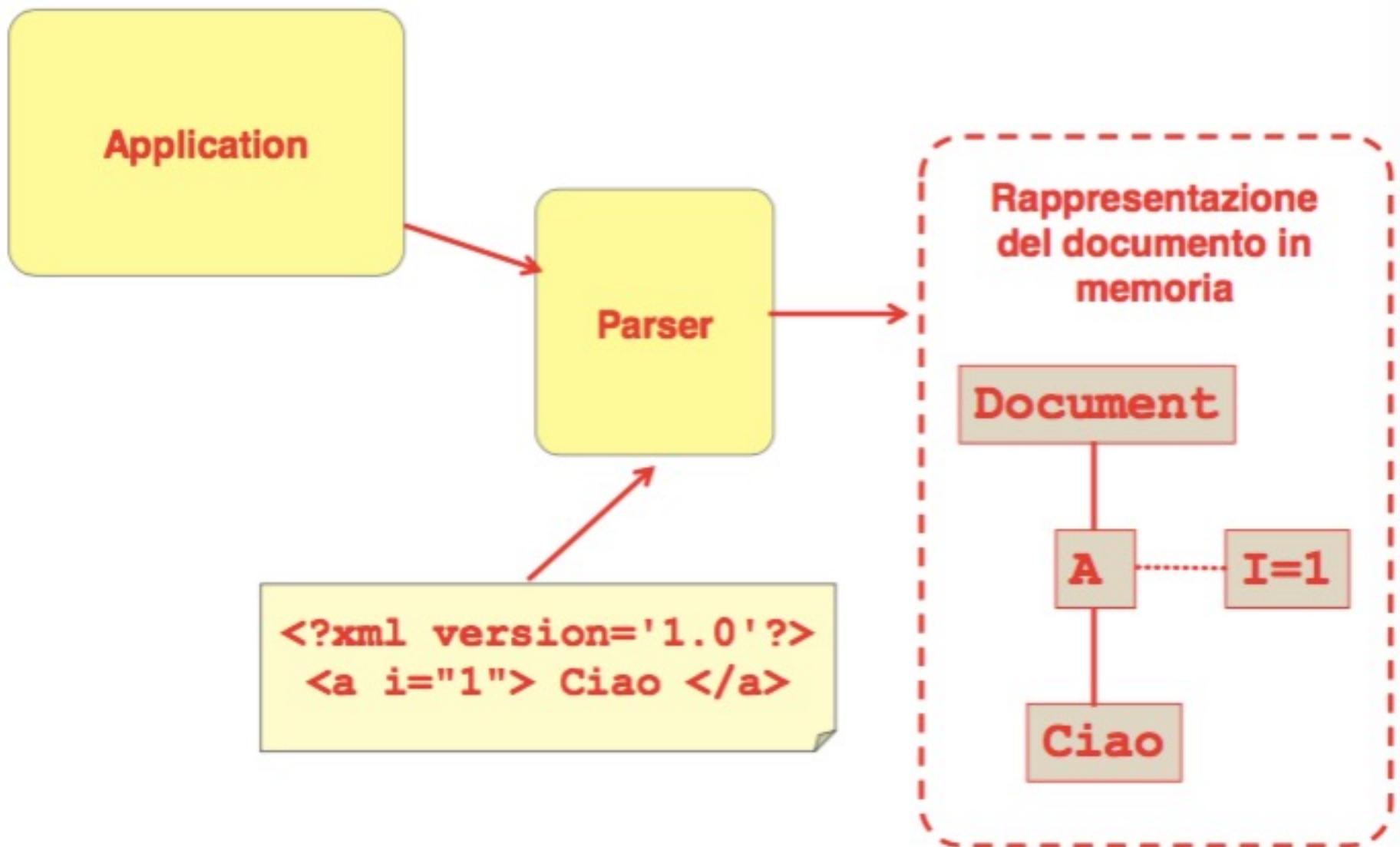
Schema di parsing ad eventi



2. Interfacce Object Model

- L'applicazione interagisce con una rappresentazione object-oriented del documento
- Il documento è rappresentato da un **albero** (**parse-tree**) costituito da vari oggetti quali document, element, attribute, text, ...
- *Il livello di astrazione che le interfacce Object Model forniscono al programmatore è maggiore rispetto a quello fornito dalle interfacce basate su eventi*
- Facile accedere ai figli, ai fratelli, ecc. di un elemento e/o aggiungere, rimuovere dinamicamente nodi
- **Problema:** mantiene in memoria l'intero parse-tree; dunque richiede la disponibilità di memoria
 - adatto per documenti di ridotta dimensione

Schema di parsing object model



Ricapitolando

- I parser XML rendono disponibile alle applicazioni la struttura ed il contenuto dei documenti XML e si interfacciano mediante due tipologie di API
- **Event-based API (es. SAX)**
 - Notificano alle applicazioni eventi generati nel parsing dei documenti
 - Usano poche risorse ma non sono sempre comodissime da usare
- **Object-model based API (es. DOM)**
 - Forniscono accesso al parse-tree che rappresenta il documento XML; molto comode ed eleganti
 - Richiedono però maggiori risorse in termini di memoria
- *I parser più diffusi supportano sia SAX sia DOM*
 - spesso i parser DOM sono sviluppati su parser SAX

Parser in Java: JAXP

- **JAXP** (Java API for XML Processing) è un framework che ci consente di istanziare ed utilizzare parser XML (sia SAX che DOM) nelle applicazioni Java
- Fornisce vari package:
 - `javax.xml.parsers`: inizializzazione ed uso dei parser
 - `org.xml.sax`: interfacce SAX
 - `org.w3c.dom`: interfacce DOM
- JAXP permette di operare indipendentemente dall'effettiva implementazione del parser utilizzata

JAXP: factory e parser

- `javax.xml.parsers` espone due classi factory, una per SAX e una per DOM:
 - **SAXParserFactory**
 - **DocumentBuilderFactory**
 - Sono classi astratte ed espongono il metodo statico **newInstance()** che consente di ottenere un'istanza di una classe discendente concreta:
SAXParserFactory `spf = SAXParserFactory.newInstance();`
DocumentBuilderFactory `dbf = DocumentBuilderFactory.newInstance();`
 - Una volta ottenuta una factory possiamo invocarla per creare un parser con interfaccia SAX o DOM:

```
SAXParser saxParser = spf.newSAXParser(); /* SAX */
```

```
DocumentBuilder builder = dbf.newDocumentBuilder(); /* DOM */
```

Interfacce Parser-to-Application (callback)

- Vengono implementate dall'applicazione per imporre un preciso comportamento a seguito del verificarsi di un evento
 - **ContentHandler**: metodi per elaborare gli eventi generati dal parser
 - **DTDHandler**: metodi per ricevere notifiche su entità esterne al documento e loro notazione dichiarata in DTD
 - **ErrorHandler**: metodi per gestire gli errori ed i warning nell'elaborazione di un documento
 - **EntityResolver**: metodi per personalizzare l'elaborazione di riferimenti ad entità esterne
- Se un'interfaccia non viene implementata il comportamento di default è *ignorare l'evento*

Interfacce Application-to-Parser

- Sono implementate dal parser
 - **XMLReader**: interfaccia che consente all'applicazione di invocare il parser e di registrare gli oggetti che implementano le interfacce di callback
 - **XMLFilter**: interfaccia che consente di porre in sequenza vari XMLReader come una serie di filtri

Interfacce Ausiliarie

- **Attributes:** Metodi per accedere ad una lista di attributi
- **Locator:** metodi per individuare l'origine degli eventi nel parsing dei documenti (es. systemID, numeri di linea e di colonna, ecc.)

Attivazione di un parser SAX

- Si crea prima una **factory**, poi il **parser** e infine il **reader**

```
SAXParserFactory spf =
    SAXParserFactory.newInstance();
try
{
    SAXParser saxParser =
        spf.newSAXParser();
    XMLReader xmlReader =
        saxParser.getXMLReader();
}
catch (Exception e)
{
    System.err.println(e.getMessage());
    System.exit(1);
};
```

Esempio SAX

- Consideriamo il seguente file XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<db>
    <person idnum="1234">
        <last>Rossi</last><first>Mario</first>
    </person>
    <person idnum="5678">
        <last>Bianchi</last><first>Elena</first>
    </person>
    <person idnum="9012">
        <last>Verdi</last><first>Giuseppe</first>
    </person>
    <person idnum="3456">
        <last>Rossi</last><first>Anna</first>
    </person>
</db>
```

Esempio SAX (2)

- Stampare a video nome, cognome e identificativo di ogni persona:

Mario Rossi (1234)

Elena Bianchi (5678)

Giuseppe Verdi (9012)

Anna Rossi (3456)

- Strategia di soluzione con un approccio event-based:

- All'inizio di **person**, si registra **idnum** (e.g., 1234)
- Si tiene traccia dell'inizio degli elementi **last** e **first**, per capire quando registrare nome e cognome (e.g., “Rossi” and “Mario”)
- Alla fine di **person**, si stampano i dati memorizzati

Esempio SAX (3)

- Incominciamo importando le classi che ci servono

```
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;

// Implementazione di default di ContentHandler:
import org.xml.sax.helpers.DefaultHandler;

// Classi JAXP usate per accedere al parser SAX:
import javax.xml.parsers.*;
```

Esempio SAX (4)

- Definiamo una classe che discende da **DefaultHandler** e ridefinisce solo i metodi di callback che sono rilevanti nella nostra applicazione

```
public class SAXDBApp extends DefaultHandler  
{  
    // Flag per ricordare dove siamo:  
    private boolean InFirst = false;  
    private boolean InLast = false;  
    // Attributi per i valori da visualizzare  
    private String FirstName, LastName, IdNum;  
  
    ...
```

Esempio SAX (5)

- Implementiamo i metodi di callback che ci servono
- **startElement** registra l'inizio degli elementi **first** e **last**, ed il valore dell'attributo **idnum** dell'elemento **person**

```
public void startElement (String namespaceURI,  
String localName, String rawName, Attributes atts)  
{  
    if (localName.equals("first"))  
        InFirst = true;  
    if (localName.equals("last"))  
        InLast = true;  
    if (localName.equals("person"))  
        IdNum = atts.getValue("idnum");  
}
```

Esempio SAX (6)

- Il metodo di callback **characters** intercetta il contenuto testuale
- Registriamo in modo opportuno il valore del testo a seconda che siamo dentro **first** o **last**

```
public void characters (char ch[], int start,  
    int length)  
{  
    if (InFirst) FirstName =  
        new String(ch, start, length);  
    if (InLast) LastName =  
        new String(ch, start, length);  
}
```

Esempio SAX (7)

- Quanto abbiamo trovato la fine dell'elemento person, scriviamo i dati;
 - quando troviamo la fine dell'elemento first o dell'elemento last aggiorniamo i flag in modo opportuno

```
public void endElement(String namespaceURI, String
localName, String qName)
{
    if (localName.equals("person"))
        System.out.println(FirstName + " " +
                           LastName + " (" + IdNum + ")" );
    //Aggiorna i flag di contesto
    if (localName.equals("first"))
        InFirst = false;
    if (localName.equals("last"))
        InLast = false;
}
```

Esempio SAX (8)

```
public static void main (String args[]) throws Exception
{
    // Usa SAXParserFactory per istanziare un XMLReader
    SAXParserFactory spf = SAXParserFactory.newInstance();
    try
    {
        spf.setNamespaceAware(true);
        SAXParser saxParser = spf.newSAXParser();
        XMLReader xmlReader = saxParser.getXMLReader();
        ContentHandler handler = new SAXDBApp();
        xmlReader.setContentHandler(handler);
        for (int i=0; i<args.length; i++)
            xmlReader.parse(args[i]);
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
        System.exit(1);
    };
}
```

Main per un singolo file

```
public static void main(String args[]) throws Exception {  
  
    SAXParserFactory spf = SAXParserFactory.newInstance();  
    spf.setNamespaceAware(true);  
  
    try {  
        SAXParser saxParser = spf.newSAXParser();  
        XMLReader xmlReader = saxParser.getXMLReader();  
        ContentHandler handler = new TestSAX();  
        xmlReader.setContentHandler(handler);  
        xmlReader.parse("/Users/macrita/Documents/workspace/web/xmlr/provasax.xml");  
    } catch (Exception e) {  
        System.err.println(e.getMessage());  
        System.exit(1);  
    }  
}
```

Altri metodi

- Inizio e fine documento:

- `startDocument();`
- `endDocument();`

```
public void startDocument() {  
  
    System.out.println("Start parsing.");  
  
}
```

```
public void endDocument() {  
  
    System.out.println("End parsing.");  
  
}
```

- `spf.setNamespaceAware(true)` specifica che il parser prodotto da questo codice fornirà il supporto per i namespace XML

Validating

- Only well-formed XML:

```
spf.setValidating(false);
```

- Parse the input document using only the **DTD**, as defined by the DOCTYPE in the input document, for validation:

```
spf.setValidating(true);
```

- Parse the input document using only the **XML Schema** as defined by the noNamespaceSchemaLocation attribute in the input document, for validation:

```
spf.setValidating(true);
```

```
spf.setNamespaceAware(true);
```

Attivazione e uso di un parser DOM in JAXP

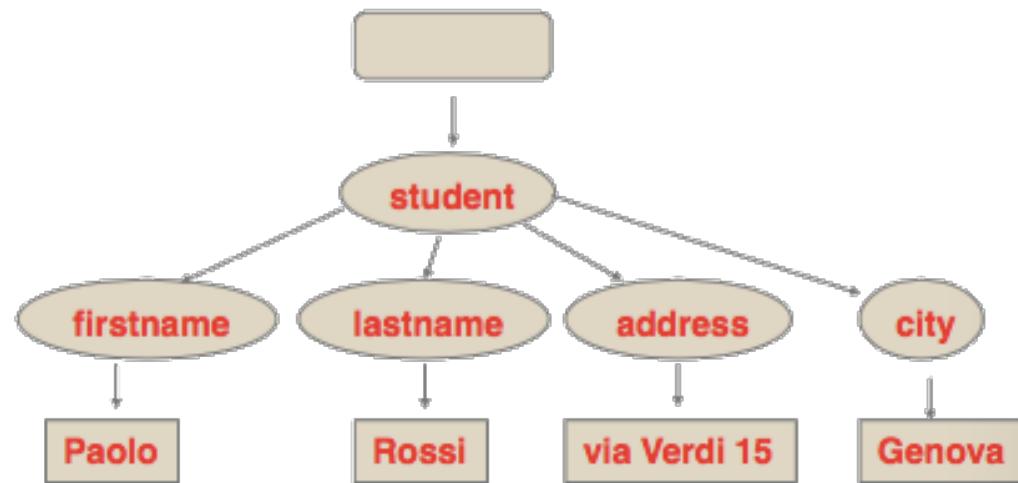
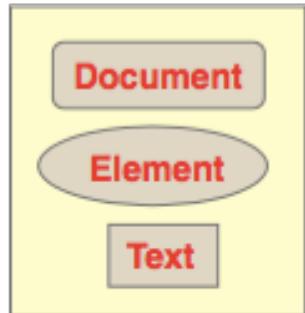
- Si crea prima una **factory**, poi il **document builder**
- Si crea un oggetto di tipo **File** passando il nome del file che contiene il documento
- Si invoca il metodo **parse** del parser per ottenere un'istanza di **Document** che può essere navigata

```
DocumentBuilderFactory dbf =  
    DocumentBuilderFactory.newInstance();  
  
DocumentBuilder builder =  
    dbf.newDocumentBuilder();  
  
File file = new File ("prova.xml");  
  
Document doc = builder.parse(file);
```

Esempio DOM – il documento XML

- Consideriamo un semplice documento XML

```
<?xml version="1.0"?>
<student>
  <firstname>Paolo</firstname>
  <lastname>Rossi</lastname>
  <address>via Verdi 15</address>
  <city>Genova</city>
</student>
```



Caricamento del documento

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class AccessingXmlFile
{
    public static Document loadDocument(String fileName)
    {
        try
        {
            File file = new File(fileName);
            DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            return db.parse(file);
        }
        catch (Exception e)
            e.printStackTrace();
    }
    ...
}
```

Navigazione nel documento

```
...
public static void main(String argv[])
{
    Document document = loadDocument(...);
    Element root = document.getDocumentElement();
    root.normalize();
    System.out.println("Root " + root.getNodeName());
    System.out.println("Informazioni sugli studenti");
    NodeList nodes = root.getChildNodes();
    for (int i = 0; i < nodes.getLength(); i++)
    {
        Node el = nodes.item(i);
        String elName= el.getNodeName();
        Node tx = el.getFirstChild();
        String elText=tx.getNodeValue();
        System.out.println(elName+"="+elText);
    }
}
```

Navigazione nel documento

```
...
public static void main(String[] args) {
    Document document =
    Element root = docu
    root.normalize();
    System.out.println('
    System.out.println('
    NodeList nodes = root.getElementsByTagName("p");
    for (int i = 0; i < nodes.getLength(); i++) {
        Node el = nodes.item(i);
        String elName= el.getNodeName();
        Node tx = el.getFirstChild();
        String elText=tx.getNodeValue();
        System.out.println(elName+"="+elText);
    }
}
```

Normalize: Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes.

if(tx != null) {...}

Metodi dell'interfaccia Node

- `getNodeType()`
- `getNodeValue()`
- `getOwnerDocument()`
- `getParentNode()`
- `hasChildNodes()`
- `getChildNodes()`
- `getFirstChild()`
- `getLastChild()`
- `getPreviousSibling()`
- `getNextSibling()`
- `hasAttributes()`
- `getAttributes()`
- `appendChild(newChild)`
- `insertBefore(newChild,refChild)`
- `replaceChild(newChild,oldChild)`
- `removeChild(oldChild)`

Altre API Java per il parsing di XML e riferimenti

- JDOM
 - variante di DOM; maggiormente allineato alla programmazione orientata agli oggetti

<http://www.jdom.org/>

- JAXP Documentation

<https://docs.oracle.com/javase/tutorial/jaxp/>