XML - Extensible Markup Language

XML è **linguaggio estensibile e validante**

- permette definizione di
  - **nuovi tag XML**
  - **regole grammaticali** per verificare la correttezza sintattica dei documenti XML rispetto al linguaggio definito
- *Es: linguaggio usato per descrivere catalogo dei prezzi del caffè o tipi di piatti (pizza, pasta) offerti nel menu del ristorante*

Definizione di nuovi linguaggi markup

Partendo dalla sintassi di base di XML, è utile **definire nuovi linguaggi basati su XML** (dialetti di XML) per

- **descrivere tipi di dati** (es: linguaggio di descrizione prodotti in catalogo)
- **scambiare messaggi tra applicazioni** (es: WSDL, Web Services Description Language, descrive sintassi per invocare operazioni offerte da Web Service su internet)
- descrivere **presentazioni** di informazioni (es. SMIL per presentazioni multimediali)

Documenti XML validi

- Documenti XML **ben formati** che **rispettano una grammatica di linguaggio markup** specificata via
  - **Document Type Declaration (DTD),** oppure
  - **XML schema**
- Grammatica regola l'uso dei tag e la struttura documenti
  - non basta che i tag siano bilanciati, etc.
  - documento deve rispettare regole grammaticali che dicono
    - quali tag si possono usare (parole riservate del linguaggio)
    - quali tag possono essere inclusi in altri
    - quali tipi di dato possono essere racchiusi in un tag (numeri, caratteri, etc.)
    - ...
- ***Noi vedremo a fondo gli XML schema, mentre citeremo solo le DTD, che non sono piu' attualissime (ma ancora usate)***

XML e DTD: specifiche W3C

per la specifica della sintassi di XML e DTD vedere:

- **http://www.w3.org/TR/xml/**
- rappresenta la grammatica di XML e delle DTD in Backus Naur Format

DTD: Document Type Definition - CENNI

- Le **regole grammaticali** definiscono un (nuovo) ***linguaggio markup basato su XML***
- Esempio: **pizza.dtd** definisce la struttura di documenti XML che descrivono pizze (l'estensione *.dtd* non e' obbligatoria, ma e' consueta)

pizza.dtd

```
<?xml version='1.0' encoding='utf-8'?>

<!ELEMENT pizza (topping, price, size)>

<!ELEMENT topping (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT size (#PCDATA)>

<!ATTLIST topping extracheese (yes|no) 'no'>
```

Prologo

Definizione di elemento composto

Definizione di elemento semplice: prende valori di tipo Parsed Character Data

Definizione di attributo dell'elemento "topping"

PCDATA: Parsed Character Data

- **PCDATA:** l'elemento deve contenere una sequenza di caratteri riconosciuti dal parser XML (e.g., se contiene un testo di markup (<, >, etc.) viene trattato come tale

pizza.dtd

```
<?xml version='1.0' encoding='utf-8'?>

<!ELEMENT pizza (topping, price, size)>

<!ELEMENT topping (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT size (#PCDATA)>

<!ATTLIST topping extracheese (yes|no) 'no'>
```

Definizione di elemento semplice: prende valori di tipo Parsed Character Data

DTD: dichiarazioni di elementi – CENNI - I

- **<!ELEMENT *tagname definition*>**
- ***tagname***: nome del tag, corrisponde al nome di una componente logica del documento
- ***definition***: specifica tipo e struttura di un elemento. Es:
  - **<!ELEMENT persona (#PCDATA)>**

  ⇒ "persona" (elemento semplice) prende valori di tipo carattere. Es:
    - <persona> Giorgio Bianchi</persona>
  - **<!ELEMENT tagVuoto EMPTY>**

  ⇒ elemento senza valore (tag che non racchiude nulla)
    - <tagVuoto> </tagVuoto>
    - </tagVuoto>

DTD: dichiarazioni di elementi – CENNI - II

Dato: <!ELEMENT canzone (#PCDATA)>

- **<!ELEMENT disco (canzone)+>**

⇒ "disco": elemento complesso; contiene almeno un'occorrenza dell'elemento "canzone"

- <disco> <canzone>uno</canzone> </disco>
- <disco> <canzone>uno</canzone> <canzone>due</canzone> </disco>
- MA NON <disco> </disco>

- **<!ELEMENT disco (canzone)*>**

⇒ Elemento complesso contiene 0 o + occorrenze di "canzone"

- **<!ELEMENT disco (canzone)?>**

⇒ Elemento complesso contiene al + 1 occorrenza di "canzone"

- <disco> </disco>
- <disco> <canzone>uno</canzone> </disco>
- MA NON <disco> <canzone>uno</canzone> <canzone>due</canzone></disco>

DTD: dichiarazioni di elementi – CENNI - III

– **<!ELEMENT pizza (topping, price, size)>**

⇒ elemento complesso formato (esattamente) da elementi specificati nella sequenza (“,” indica una sequenza)

```
<pizza>
        <topping extracheese=“yes”> peperoni </topping>
        <price> 8.00 </price>
        <size> large </size>
</pizza>
```

MA NON

```
<pizza>
        <price> 8.00 </price>
        <size> large </size>
</pizza>
```

Esistono altri tipi di dichiarazione – vedere http://www.w3.org/TR/xml/

DTD: dichiarazioni di attributi

- Posso associare attributi ad un elemento, specificando quali valori possono assumere, se e’necessario specificare valore, ...
- **<!ATTLIST *tagName attrName values [defValue] [restrictions]*>**
  - ***tagName***: nome dell’elemento a cui associo l’attributo
  - ***attrName***: nome dell’attributo
  - ***values***: tipo dell’attributo (enumerazione di valori)
  - ***defValue***: valore di default
  - Es:
    - <!ATTLIST topping extracheese (yes|no) 'no'>
    - <!ATTLIST topping extracheese (yes|no) #REQUIRED>
    - <!ATTLIST topping extracheese CDATA>
    - <!ATTLIST pizza vendor (CDATA) #FIXED ‘Pizza-Hut’ >

Attributo prende valori di tipo Character Data

Tipi di attributi

- **Stringa: CDATA:** attributo di tipo stringa; può contenere una sequenza di caratteri qualunque (eccetto <, >, &, ‘, “)
- **Enumerati**: attributi di tipo enumerato; dichiarano la lista di valori che possono assumere
  - <!ATTLIST utente sesso (M | F) “F”>
    - Attributo può solo assumere valore M o F, non altri valori
- ***Altri tipi che non vediamo****. Gli interessati facciano riferimento alle specifiche delle DTD*

Documento valido vs. ben formato -I

pizza.dtd

```
<?xml version='1.0' encoding='utf-8'?>
<!ELEMENT pizza (topping, price, size)>
<!ELEMENT topping (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT size (#PCDATA)>
<!ATTLIST topping extracheese (yes|no) 'no'>
```

Grammatica

Documento valido rispetto a DTD

```
<?xml version='1.0' encoding='utf-8'?>
<pizza>
  <topping extracheese=“yes”> peperoni </topping>
  <price> 8.00 </price>
  <size> large </size>
</pizza>
```

Documento valido vs. ben formato - II

pizza.dtd

```
<?xml version='1.0' encoding='utf-8'?>
<!ELEMENT pizza (topping, price, size)>
<!ELEMENT topping (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT size (#PCDATA)>
<!ATTLIST topping extracheese (yes|no) 'no'>
```

Grammatica (di solito specificata in un file .dtd)

Documento ben formato, ma non valido rispetto alla DTD

```
<?xml version='1.0' encoding='utf-8'?>
<mypizza>
  <topping extracheese="boh"> peperoni </topping>
  <price> 8.00 </price>
  <size> large </size>
</mypizza>
```


Associazione di documenti XML a DTD

Serve per dichiarare a quale grammatica fa riferimento un documento XML (altrimenti non sarebbe possibile verificare se il documento e' valido o meno)

pizza.dtd

```
<?xml version='1.0' encoding='utf-8'?>
<!ELEMENT pizza (topping, price, size)>
……..
<!ATTLIST topping extracheese (yes|no) 'no'>
```

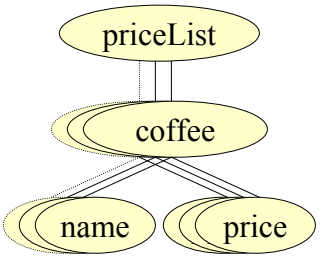
pizza.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE pizza SYSTEM "pizza.dtd">
<pizza>
  <topping extracheese="yes"> peperoni </topping>
  <price> 8.00 </price>
  <size> large </size>
</pizza>
```

Documenti XML diversi possono far riferimento ad una stessa grammatica

Embedded DTDs

- Se la grammatica serve in un solo documento la si può definire nel documento stesso




```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE priceList [
        <!ELEMENT priceList (coffee)+>
        <!ELEMENT coffee (name, price)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT price (#PCDATA)>
        ]>
 <priceList>
  <coffee>
    <name> Mocha Java </name>
    <price> 11.95 </price>
  </coffee>
  <coffee>
    <name> Sumatra </name>
    <price> 12.50 </price>
  </coffee>
 </priceList>
```

coffee.xml

Gestione documenti XML

XML richiede il rispetto di regole sintattiche ma permette di usare parser XML che verificano

- se documento è ben formato (sintassi di base di XML)
- se documento valido rispetto ad una DTD *(o XML schema)* data in input al parser

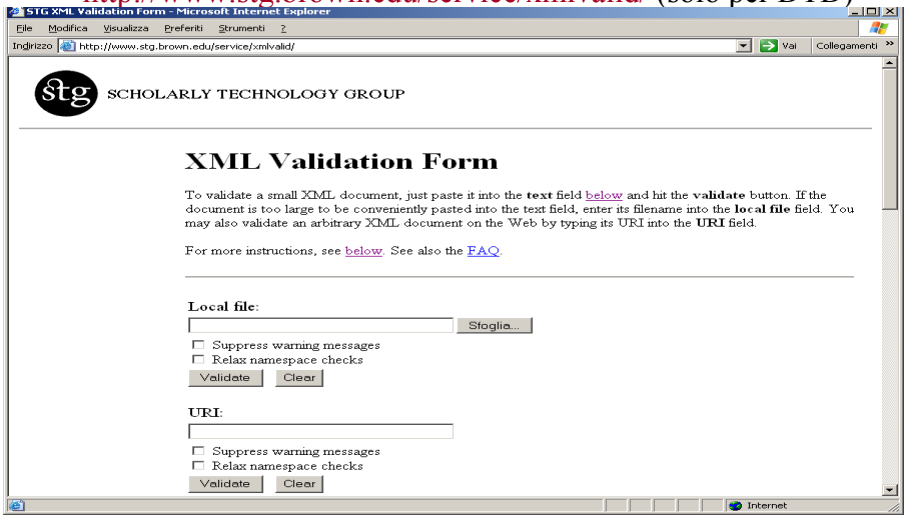
⇒***non dobbiamo sviluppare i parser*** *perchè, essendo XML uno standard, esistono analizzatori sintattici (anche open source) che operano su documenti XML, DTD (e XML Schema)*

Parsing di documenti XML

- Data 1 DTD *(o XML Schema)* e 1 documento XML
  - **Analisi sintattica isolata**: uso un parser per verificare la ben formatezza e validita' rispetto alla grammatica (mi interessa che il documento sia sintatticamente corretto)
  - **Analisi all'interno di un'applicazione** (e.g., Java):
    - l'analisi sintattica puo' servire per verificare che il documento non contenga errori che farebbero fallire l'applicazione, o per estrarre informazioni dal documento e utilizzarle nell'applicazione (e.g., per leggere il prezzo di un certo tipo di caffe')
    - in questo caso devo usare un parser invocabile all'interno dell'applicazione e catturare i risultati dell'analisi sintattica nell'applicazione stessa

Parser XML - I

- Ce ne sono molti (IBM, Sun Microsystems, ...); possono essere validanti o meno
- Alcuni utilizzabili via web. Es: http://www.stg.brown.edu/service/xmlvalid/ (solo per DTD)

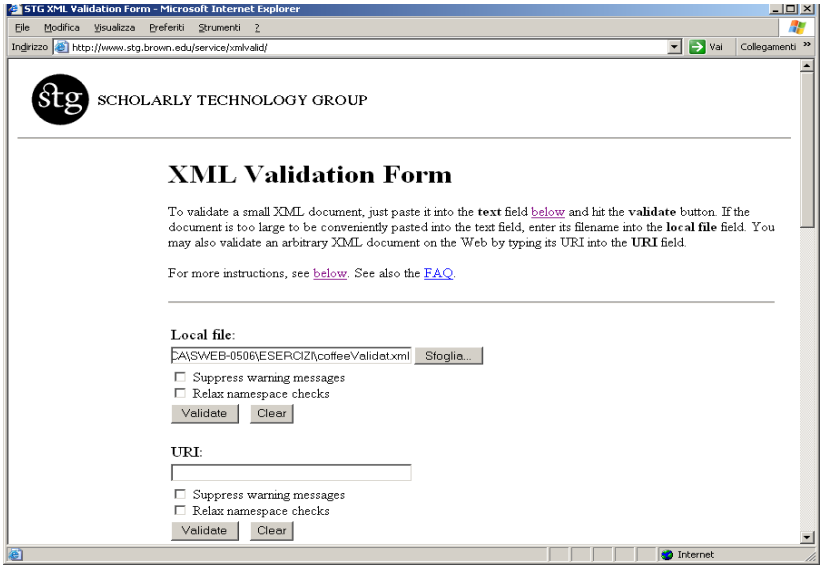


Validatore (DTD) – esempio di uso - I

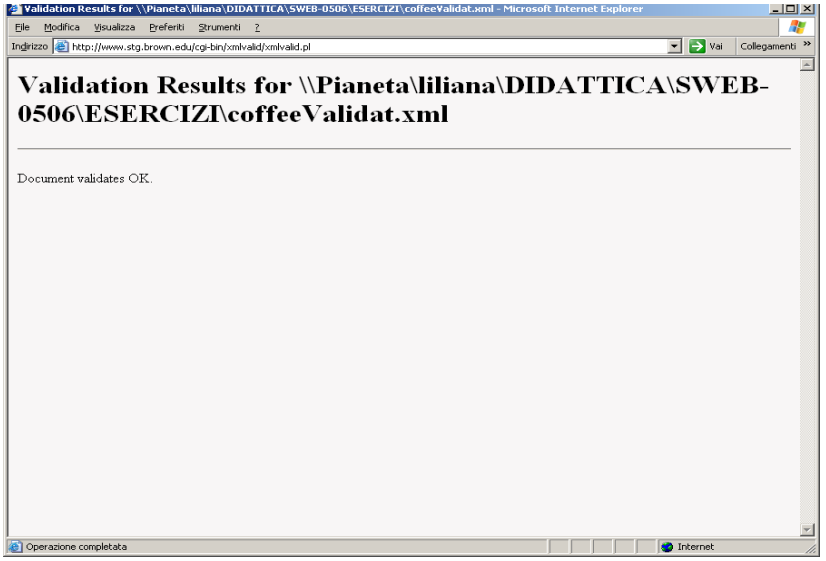
```
<?xml version='1.0' encoding='utf-8'?>
<!-- comment... -->
<!DOCTYPE priceList [
        <!ELEMENT priceList (coffee)+>
        <!ELEMENT coffee (name, price)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT price (#PCDATA)>
        ]>
  <priceList>
    <coffee>
       <name> Mocha Java </name>
       <price> 11.95 </price>
    </coffee>
    <coffee>
       <name> Sumatra </name>
       <price> 12.50 </price>
    </coffee>
  </priceList>
```

coffee.xml
ben formato
e valido…
*file **coffeeValidat.xml***

Validatore (DTD) – esempio di uso - II



Validatore (DTD) – esempio di uso - III

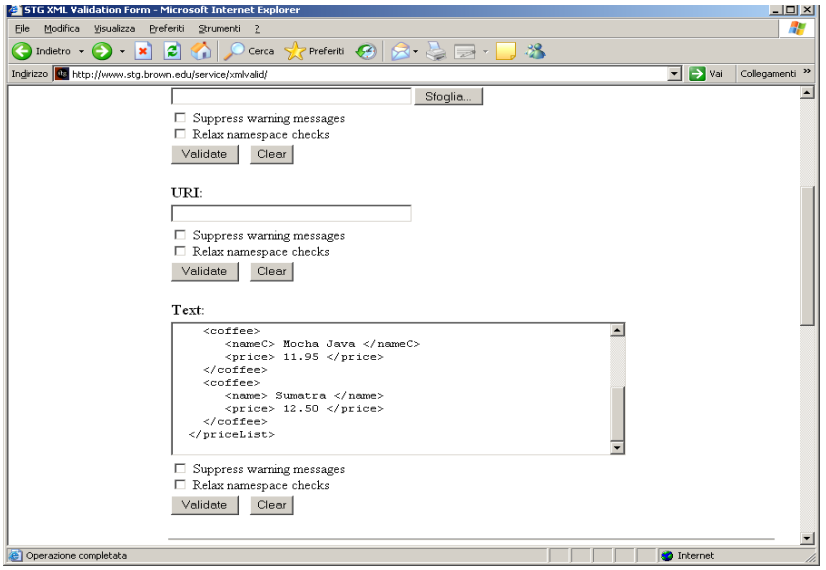


Validatore (DTD) – esempio di uso - IV

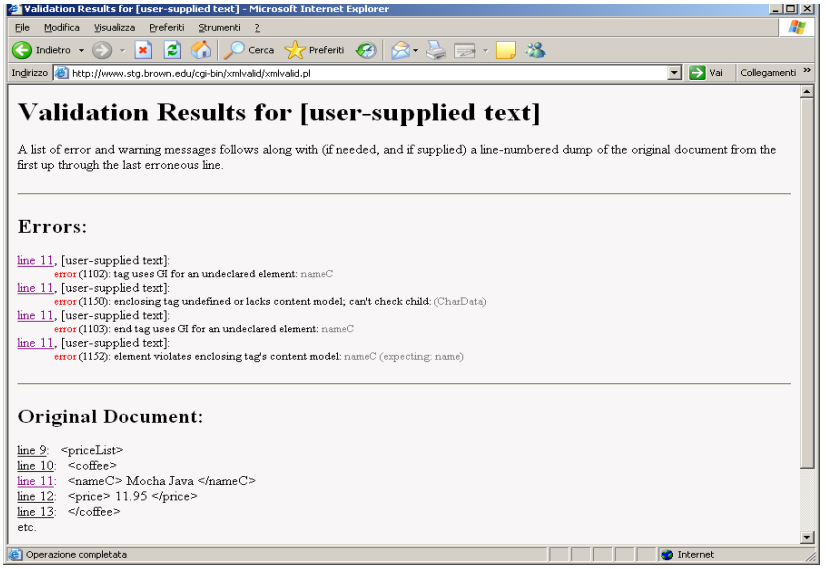
```
<?xml version='1.0' encoding='utf-8'?>
<!-- comment... -->
<!DOCTYPE priceList [
        <!ELEMENT priceList (coffee)+>
        <!ELEMENT coffee (name, price)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT price (#PCDATA)>
        ]>
  <priceList>
    <coffee>
       <nameC> Mocha Java </nameC>
       <price> 11.95 </price>
    </coffee>
    <coffee>
       <name> Sumatra </name>
       <price> 12.50 </price>
    </coffee>
  </priceList>
```

Ben formato ma non valido…

Validatore (DTD) – esempio di uso - V



Validatore (DTD) – esempio di uso-VI



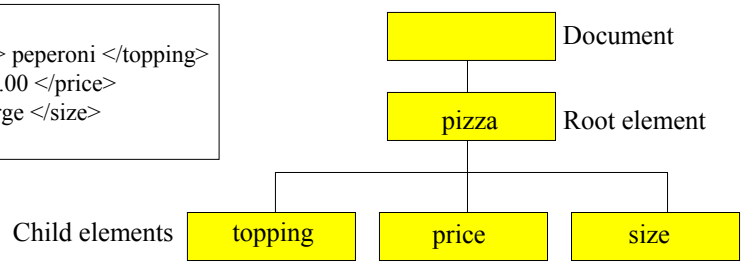
Rappresentazione ad oggetti di documenti XML

- W3C ha definito le specifiche per la rappresentazione strutturata di documenti XML: modello DOM
- Sono disponibili parser open source che analizzano documenti XML e ne costruiscono la rappresentazione DOM
- NB: modello DOM e' indipendente dal linguaggio di programmazione che si usa per gestire i documenti XML

DOM - Document Object Model - I

- **standard W3C per la gestione di documenti XML**
  - rappresenta i documenti con un modello ad oggetti
  - specifica l'interfaccia per accedere a parti dei documenti e "navigarli"
  - la rappresentazione DOM si basa su **struttura ad albero: elementi, attributi, etc**. rappresentati come **nodi di albero**

```
<pizza>
  <topping> peperoni </topping>
  <price> 8.00 </price>
  <size> large </size>
</pizza>
```



DOM - Document Object Model - II

- Molti parser sono conformi a DOM ⇒ usano rappresentazione DOM per gestire documenti
- **DOM e' una interface** ⇒ *i produttori di parser sviluppano le implementazioni*
- **org.w3c.dom**
- DOM usa **interface Node** (e + specifici, come **Element**), che rappresenta i nodi dell'albero di un documento e offre i metodi per
  - trovare nodi i figli di un nodo
  - aggiungere/rimuovere figli
  - leggere/modificare il valore contenuto in un nodo, ...
- il documento si ispeziona (e modifica) navigando l'albero a partire dalla radice

Vedere API java (JAXP) per maggiori info su DOM

Creazione di oggetto DOM che rappresenta un documento XML

- dal nulla, **creando i nodi e inserendoli nell'albero**
  - si parte da un albero che rappresenta un documento XML vuoto e si utilizzano i metodi delle classi java di libreria per inserire i nodi
- **analizzando sintatticamente un documento XML**
  - si utilizza un DOM parser (e.g., JAXP parser) per analizzare sintatticamente il documento XML e produrre l'albero corrispondente

Vediamo un esempio in cui utilizziamo le librerie java per gestire un documento XML. E.g., stiamo sviluppando applicazione java che deve manipolare documenti XML ⇒ nell'applicazione si devono usare gli API di gestione di XML

Applicazione java che verifica se documento XML e' valido rispetto a DTD

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException; import org.xml.sax.SAXParseException;
import java.io.File; import java.io.IOException;
import org.w3c.dom.Document; import org.w3c.dom.DOMException;

public class DomEcho01{
  static Document document;
 public static void main(String argv[])  {
   DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
   factory.setValidating(true);   // per effettuare validazioni di documenti XML
   try {DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.parse( "coffee.xml" );
     } catch (SAXParseException spe) {System.out.println("\n** Parsing error");
       ….. Gestione di altre eccezioni... }
   }// end class DomEcho01
```

Analizza sintatticamente il file coffee.xml e produce l'oggetto DOM

NB: l'esecuzione non da' output se documento e' ben formato e valido; altrimenti segnala errori

Visita di oggetto DOM - esempio

```
public static void visitDocument(Document document, String feature) {
   if (document!=null) {
     Element el = document.getDocumentElement();
     visit(el, feature);}}
public static void visit(Node el, String feature) {
    if (el!=null) {
     NodeList children = el.getChildNodes();
     for (int i=0; i<children.getLength(); i++) {
           Node n=children.item(i);
           getFeatureVal(n, feature);
           visit(children.item(i), feature);}}}
public static void getFeatureVal(Node domNode, String feature) {
    String out = "";
    if (domNode!=null) {
       String name = domNode.getNodeName();
       if (name.equals(feature)) {
          Element el = (Element)domNode;
          out = el.getFirstChild().getNodeValue();
          System.out.println("The value of feature "+feature+ " is: " +out);}
     }}
```

Feature: nome del tag (element) di cui si vuole conoscere il valore

DOM vs. JDOM e altri….

- DOM nato per trattare documenti
  - document-oriented, focalizzato sulle parti di cui documento si compone
  - gestione di strutture dati possibile, ma ostica
  - $\Rightarrow$ sviluppati altri modelli di rappresentazione che facilitano accesso e modifica. Es: **JDOM, DOM4J, …**