

XML

Uno standard per lo scambio di dati

Standard per lo scambio di dati in ambito distribuito

- Tipicamente definiscono:
 - Modalità per la definizione *astratta* di tipi di dati
 - Rappresentazioni dei dati “neutrali” (per mascherare le differenze tra i sistemi di rappresentazione): system independence
 - Meccanismi con i quali chi riceve i dati sia in grado di decodificarli/separarli correttamente
 - **Tipicamente il ricevitore deve conoscere il tipo dei dati, che viene espresso utilizzando un opportuno formalismo**
- Esempi: ASN.1, XDR, CORBA CDR, XML
- XML sta diventando lo standard più diffuso, non solo per l’ambito distribuito

Principali Caratteristiche di XML

- I dati assumono la forma di “documenti”, simili a quelli del web (documenti HTML: caso particolare)
- I dati hanno rappresentazioni a carattere
 - Human readable & machine readable
 - Occupazione di memoria/banda non ottimizzata
- I dati incorporano anche le informazioni sul loro tipo
 - La struttura dei dati non deve essere nota a priori al ricevitore

XML (eXtensible Markup Language)

- Linguaggio basato su SGML per la descrizione formale di **linguaggi di markup** (meta-linguaggio)
- Sviluppato dal w3c a partire dal 1996
- Applicazioni:
 - Pensato inizialmente per aumentare le potenzialità delle pagine web
 - Tende di fatto a diventare lo strumento standard per lo scambio di dati tra applicazioni eterogenee
- Versioni attuali:
 - XML 1.0 (third edition) Febbraio 2004
 - XML 1.1 Febbraio 2004

SGML

- SGML (Standard Generalized Markup Language) è un sistema per la descrizione di **linguaggi di markup**:
 - linguaggio di markup: linguaggio per la descrizione di documenti, basato su annotazioni (markup) che servono per rappresentare elementi *strutturali*, di *presentazione*, *semantici*
- I linguaggi descritti da SGML sono device/system independent
- SGML permette di definire gli aspetti sintattici/strutturali dei linguaggi
 - » **Come scrivere i markup e distinguerli dal testo e tra loro**
 - » **Quali markup sono ammessi, quali necessari, ...**
- Non permette di definirne la semantica

SGML

- SGML definisce
 - Una serie di caratteristiche generali dei linguaggi di markup che è in grado di descrivere:
 - » **I markup hanno stile dichiarativo e non procedurale**
 - » **I linguaggi descritti hanno una base sintattica comune**
 - Un formalismo (DTD - Document Type Definition) per descrivere le caratteristiche specifiche di un linguaggio
- In pratica SGML definisce un super-linguaggio generale (comune) di cui i linguaggi descrivibili sono sottoinsiemi

Terminologia SGML

- Documento SGML
 - E' un elemento del linguaggio generale (un qualunque testo conforme ai requisiti generali SGML)
 - Contiene testo (dati), markup e un riferimento al DTD
- Applicazione SGML
 - E' un linguaggio di markup descritto usando SGML. La descrizione comprende:
 - » **SGML Declaration**, che specifica caratteristiche di "basso livello" del linguaggio (character set, delimitatori, ecc.)
 - » **Document Type Definition (DTD)**, che definisce la sintassi dei costrutti di markup
 - » **La specifica della semantica dei costrutti di markup** (non descritta formalmente in SGML)

XML

- XML nasce con l'obiettivo di essere un SGML ...
 - ...utilizzabile in modo diretto su internet (tramite HTTP)
 - ...largamente aperto e compatibile
 - ...utilizzabile anche dalla maggior parte delle applicazioni in modo semplice
- Quindi XML incorpora solo gli aspetti più semplici di SGML
- Formalmente XML è un sottoinsieme di SGML (un profilo di applicazione)

Semantica Struttura e Stile

- Il **concetto** di documento incorpora due aspetti
 - Contenuto
 - » **Struttura** (organizzazione del contenuto)
 - » **Semantica** (significato del contenuto)
 - Stile (di presentazione)
- Per semplificare/razionalizzare sviluppo ed elaborazione di documenti è utile separare le 3 componenti
 - In HTML originariamente non erano separate
 - Con i CSS (Cascading Style Sheet) è diventato possibile separare Stile da Contenuto
 - Con XML si può descrivere separatamente anche la Struttura

Esempio di documento XML

```
<?xml version="1.0"?>
<bibliografia>
  <articolo>
    <autore> J. W. Cooley </autore>
    <autore> J. A. Tukey </autore>
    <titolo> An Algorithm for Machine Computation of Complex FFT
    </titolo>
    <rivista volume="19" numero="Aprile 1965"> Math. Computation
    </rivista>
  </articolo>
  <articolo>
    <autore> T. G. Stockham </autore>
    <titolo> High speed convolution and correlation </titolo>
    <atti anno="1966"> Spring Joint Computer Conference </atti>
  </articolo>
  <libro>
    <autore> D. A. Chappel </autore> <autore> T. Jewell </autore>
    <titolo> Java Web Services </titolo>
    <editore> Hops Libri </editore>
  </libro>
</bibliografia>
```

Organizzazione logica di un documento XML

- Descrive un contenuto strutturato ad albero
 - Ogni sotto-albero (o nodo) è un **elemento**
 - Gli elementi possono
 - » **avere attributi**
 - » **contenere dati**
- Può includere
 - dichiarazioni (p. es. specifica della DTD di riferimento)
 - istruzioni di elaborazione
 - commenti

Organizzazione fisica di un documento XML

- Un documento XML nel suo complesso può essere distribuito su più unità di memorizzazione chiamate **entità**
- Un'entità può contenere dati di varia natura (parsed, non parsed) e può far riferimento ad altre entità
- Vi è in ogni caso un'entità radice (**document entity**) cui nessun'altra fa riferimento
- Per semplicità, consideriamo inizialmente solo documenti mono-entità

Sintassi generale dei documenti XML

- I documenti XML seguono la sintassi generale SGML
- Un file di testo è un documento XML **well formed** (ben formato) se oltre a seguire la sintassi SGML rispetta alcuni vincoli aggiuntivi. I principali sono:
 - Ogni elemento non vuoto è delimitato da un tag iniziale e un tag finale
 - Esiste uno ed un solo elemento “radice” (elemento che contiene tutti gli altri elementi)
 - I valori degli attributi sono sempre racchiusi tra virgolette
 - I nomi degli attributi sono univoci nell’elemento (i linguaggi XML sono case-sensitive)

Strutture sintattiche

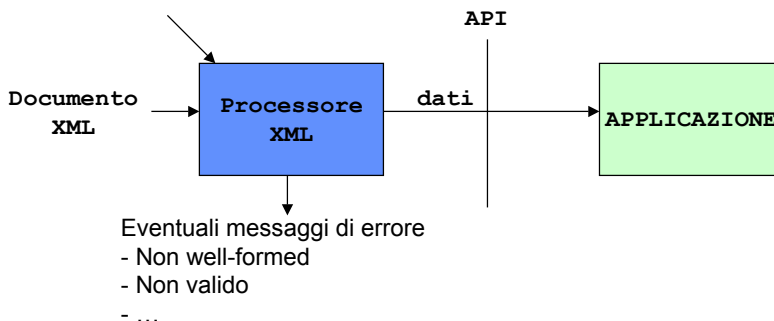
- Il testo di un documento è costituito da **dati** (sequenze di caratteri) e **annotazioni** (markup)
- I markup possono essere:
 - Tag di inizio/fine elemento (o di elemento vuoto) e delimitatori di sezione
 - Riferimenti ad entità e a caratteri speciali
 - Commenti
 - Dichiarazioni DTD
 - Dichiarazioni XML e Istruzioni di elaborazione

Documenti XML Validi

- Un documento XML well-formed si dice **valido** se
 - Contiene una dichiarazione DTD
 - E' conforme ai vincoli espressi nella DTD
 - Usando una DTD definiamo un linguaggio (costituito da tutti i documenti validi che vi si riferiscono)
- ⇒ Un documento XML è autocontenuto:
- contiene sia i dati che le regole strutturali e sintattiche con cui i dati sono organizzati

Elaborazione di documenti XML

Eventuali entità esterne



- **Non Validating Processor** (o Parser): controlla solo che il documento sia well-formed (Esempio: msxml)
- **Validating Processor** : controlla anche la validità (Esempio: JAXP)

I Dati di un documento XML

- Possono essere di diversi tipi:
 - PCDATA (Parsed Character Data)
 - » **Testi sottoposti all'analisi del Processore (devono rispettare le regole XML)**
 - CDATA (Character Data)
 - » **Testi non sottoposti all'analisi del Processore (passati direttamente all'applicazione)**

DTD

- E' una sequenza di regole (dichiarazioni di elementi ed attributi)
- Segue la sintassi prevista da SGML, ma con alcune restrizioni e differenze

Esempio di DTD

```
<!-- DTD per elenco utenti ed utenze telefoniche -->
<!ELEMENT elenco ((utente|utenza)*)>
<!ELEMENT utente (#PCDATA)>
<!ATTLIST utente
      cf                ID                #REQUIRED
      tipo              (privato|azienda)  #IMPLIED
      pubblicita        (ammessa|bloccata)  "bloccata"
>
<!ELEMENT utenza (indirizzo?, abilitazioni?)>
<!ATTLIST utenza
      numeroTel        NMTOKEN           #REQUIRED
      intestatario     IDREF             #REQUIRED
      mobile           NMTOKEN           #FIXED "si"
>
<!ELEMENT indirizzo EMPTY>
<!ATTLIST indirizzo
      via              CDATA             #REQUIRED
      numero          NMTOKEN           #IMPLIED
      frazione        CDATA             #IMPLIED
      comune          CDATA             #REQUIRED
      cap             NMTOKEN           #REQUIRED
>
```

Dichiarazione di Elemento

- Specifica
 - nome dell'elemento
 - modello del suo contenuto
- Sintassi:
`<!ELEMENT nome modello >`
- Tipi di modello:
 - EMPTY l'elemento è vuoto
 - ANY contenuto qualsiasi (nessun controllo)
 - Element l'elemento contiene solo elementi
 - Mixed Content l'elemento contiene elementi e dati

Modelli Element

Consentono di descrivere **nome**, **ordine**, **opzionalità** e **frequenza** degli elementi annidati con una semplice grammatica:

- Un modello può essere semplicemente un elemento oppure *una sequenza* o *un'alternativa* di modelli
 - La virgola indica sequenza
 - La barra verticale indica alternativa

La frequenza è specificata dagli operatori postfissi:

- + il modello deve comparire 1 o più volte
- * il modello deve comparire 0 o più volte
- ? il modello deve comparire 0 o 1 volta (opzionale)

Esempi di Modelli Element

```
<!ELEMENT pastoAllaCarta (portata*)>
<!ELEMENT portata (primo|secondo|dolce)>
<!ELEMENT pastoAPrezzoFisso (primo,secondo,dolce)>
<!ELEMENT primo EMPTY>
<!ELEMENT secondo EMPTY>
<!ELEMENT dolce EMPTY>
```

```
<!ELEMENT laboratorio ( nome, responsabile,
viceResponsabile?, (tecnico|operaio)+)>
```

Modelli Mixed

- In questo caso si possono specificare solo i nomi degli elementi annidati, non ordine e frequenza
- La sintassi è conforme a quella dei modelli Element, ma l'unica forma possibile è la seguente:

```
(#PCDATA | nome1 | nome2 | ... )*
```

Esempi:

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

```
<!ELEMENT DatiFont (#PCDATA | font)*>
```

```
<!ELEMENT DatiFontEColori (#PCDATA | font | colore)*>
```

Dichiarazioni di Attributo

- Ogni dichiarazione specifica le caratteristiche di uno o più attributi riferiti ad un tipo di elemento
- Sintassi:

```
<!ATTLIST nomeElemento listaAttributi >
```
- Per ogni attributo si specifica **nome**, **tipo di valore** e **dichiarazione di default**

Esempio:

```
<!ATTLIST soloDati
      id      ID          #REQUIRED
      type    (vettore|matrice) "vettore">
```

Specifica del tipo di valore

Tipo	Valore	Sintassi	Esempio
Stringa	una stringa non contenente i caratteri < > & ' "	CDATA	CDATA
Token	un token o una serie di token	ID IDREF ENTITY NMTOKEN IDREFS ENTITIES NMTOKENS	ID
Enumerativo	una delle stringhe specificate	serie di stringhe separate da	(siglsig.ra)

Significato dei Diversi Tipi Token

Tipo	Valore
ID	un nome che identifica <i>univocamente</i> l'elemento in tutto il documento XML
IDREF	l'ID di un elemento del documento XML (un riferimento ad un elemento)
ENTITY	il nome di un'entità dichiarata nel DTD
NMTOKEN	un generico nome
IDREFS	una serie di IDREF
ENTITIES	una serie di ENTITY
NMTOKENS	una serie di NMTOKEN

Dichiarazione di Default

- Indica se l'attributo è obbligatorio o qual è il suo valore di default. Può assumere 4 forme:

#REQUIRED	Attributo obbligatorio (no default)
default	Attributo opzionale: in assenza, viene usato il valore di default indicato
#IMPLIED	Attributo opzionale, default indefinito: in assenza può essere usato qualunque valore
#FIXED default	Attributo opzionale, ma fisso: se presente deve necessariamente avere il valore di default indicato

Esempio

```
<!ATTLIST      portata
                codice      ID          #REQUIRED
                nome        CDATA      #IMPLIED
                abbondante   (si|no)  "no"
>
```

Esempi di tag di inizio elemento validi:

```
<portata codice="A10" nome="Pasta al ragu">
```

```
<portata codice="A10" abbondante="si">
```

Esempi di tag di inizio elemento non validi:

```
<portata nome="Pasta alla carbonara">
```

```
<portata codice="A10" abbondante="ni">
```

```
<portata>
```

Uso delle Entità

- Le Entità sono meccanismi per
 - assegnare nomi simbolici a parti arbitrarie di un documento
 - memorizzare le parti su file separati riutilizzabili in documenti diversi
- I nomi simbolici sono rimpiazzati automaticamente dalle parti corrispondenti a cura del processore XML

Classificazione delle Entità

- In base al tipo di contenuto:
 - Parsed: rappresenta un testo che viene inserito nel documento e viene interpretato dal parser
 - Unparsed: rappresenta dati che possono anche non essere testo, non interpretati dal parser
- In base al tipo di contesto:
 - General: utilizzate per l'intero documento
 - Parameter: utilizzate solo in una DTD esterna
- In base al luogo dove fisicamente si trova il contenuto:
 - Internal il contenuto è nel file stesso
 - External il contenuto è in un file esterno

Dichiarazione delle Entità

- Sintassi:

- Generali: `<!ENTITY nome contenuto >`
- Parameter: `<!ENTITY % nome contenuto >`

- Dichiarazioni interne:

```
<!ENTITY pub-status "This is a pre-release">
<!ENTITY % item "(food|drink)">
```

- Dichiarazioni esterne:

```
<!ENTITY def1 SYSTEM "http://www.x.y/pub/def1.xml">
<!ENTITY im1 SYSTEM "/images/im1.gif" NDATA gif>
<!ENTITY % item PUBLIC "-//PUB//item//IT"
"http://www.x.y/pub/item.xml">
```

Riferimenti alle Entità

- Sintassi:

- Generali: `& nome ;`
- Parameter: `% nome ;`

- Esempi:

```
<nota> &pub-status; </nota>
<nota> si consideri la seguente definizione :
&def1; </nota>

<!ELEMENT omaggio %item; >
```

Esempio di DTD con entità: La Table HTML

```
<!ELEMENT table      (caption?, (col*|colgroup*), thead?,
                        tfoot?, (tbody+|tr+) )>

<!ATTLIST table
  %attrs;
  summary      %Text;          #IMPLIED
  width        %Length;        #IMPLIED
  border       %Pixels;        #IMPLIED
  frame        %TFrame;        #IMPLIED
  rules        %TRules;        #IMPLIED
  cellspacing %Length;        #IMPLIED
  cellpadding %Length;        #IMPLIED
>
```

Namespace

- Tecnica utilizzabile per evitare i conflitti sui nomi di elementi ed attributi
- Utile soprattutto quando un documento
 - deve poter essere elaborato da una molteplicità di software diversi
 - deve far riferimento a DTD esterne diverse (che quindi possono usare gli stessi nomi con significati diversi)
- Il principio è la creazione di spazi di nomi separati
 - Uno stesso nome può essere ri-usato in spazi separati con significati diversi

Caratteristiche dei Namespace

- Ogni namespace
 - è identificato univocamente da un *IRI* (forma estesa di URL)
 - può essere rappresentato localmente da un *nome simbolico*
- I nomi di elementi ed attributi possono appartenere ad un namespace
 - In questo caso, il namespace deve essere stato dichiarato

Dichiarazione di Namespace

- Un namespace si dichiara con un attributo
 - il cui **nome** è
 - » `xmlns` (dichiarazione del namespace di default)
 - oppure
 - » `xmlns:` seguito dal nome simbolico locale del namespace
 - il cui **valore** è la IRI che identifica il namespace
- Lo scope di una dichiarazione comprende l'intero elemento in cui si trova
- Esempi:

```
<sezione xmlns="http://www.w3c.org"> ... </sezione>  
<sezione xmlns:local="http://www.x.y">  
    xmlns:global="http://www.x.z"> ... </sezione>
```

Nomi Qualificati

- Con questo termine si intendono i nomi soggetti ad interpretazione secondo i namespace
- Un nome qualificato può assumere due forme:
 - Un semplice nome: il nome appartiene al namespace di default (se è stato dichiarato)
 - Un nome con prefisso: il nome appartiene al namespace avente come nome simbolico il prefisso
- Esempio di nomi qualificati con prefisso:

local:letter global:letter

Esempio di uso dei Namespace

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

<directory xmlns = "http://www.polito.it/xml/nsdef"
           xmlns:image = "http://www.polito.it/xml/nsImage">

  <file filename = "book.xml">
    <description>A book list</description>
  </file>

  <image:file filename = "funny.jpg">
    <image:description>A funny picture
    </image:description>
    <image:size width = "200" height = "100"/>
  </image:file>

</directory>
```

Esercizio

- Scrivere una DTD che descriva un documento in grado di memorizzare i dati dei conti correnti di un'agenzia bancaria, con le seguenti regole:
 - Ogni conto corrente è caratterizzato da
 - » **uno e un solo numero di conto**
 - » **uno o più intestatari**
 - » **una serie di operazioni, raggruppate per anno**
 - Ogni operazione è caratterizzata da
 - » **data, importo, causale**
 - Ogni intestatario è caratterizzato da
 - » **nome, indirizzo**

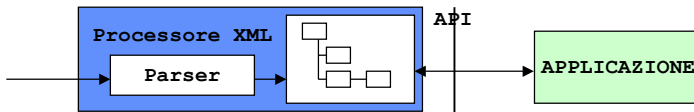
Gli Schema

- Sono descrizioni di tipi di documenti alternative alle DTD
- Migliorano le DTD in alcuni aspetti:
 - Gli Schema sono essi stessi documenti XML
 - ⇒ **Sono manipolabili/verificabili con le stesse tecniche**
 - Gli Schema consentono di esprimere in modo più preciso i vincoli sul documento
 - » **Per esempio, si possono specificare i tipi di dato del contenuto di un elemento**
 - E' possibile creare diverse classi di Schema, ciascuna con la sua sintassi (descritta da una DTD)

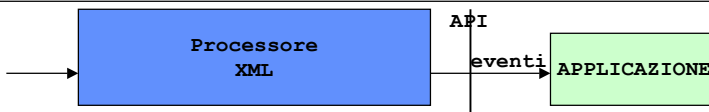
Strumenti per l'elaborazione di Documenti XML

- Le applicazioni XML possono essere realizzate con diversi tipi di API standard

DOM: Il parser costruisce in memoria l'albero dell'intero documento XML. L'applicazione accede all'albero tramite la API



SAX: Il parser passa direttamente i tag e i dati all'applicazione in sequenza man mano che li riconosce



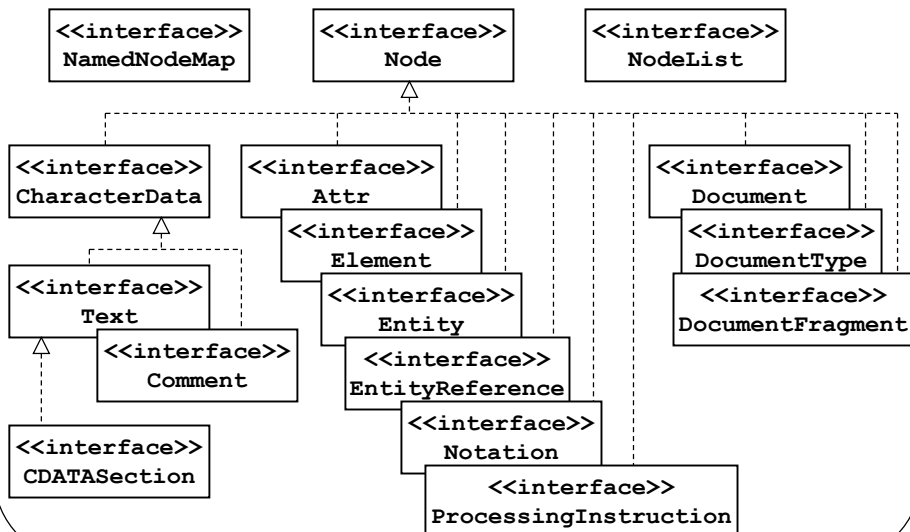
Document Object Model (DOM)

- Specifica creata da W3C per definire una modalità standard di memorizzazione della struttura ad albero di un documento XML o HTML
- La specifica è ad oggetti, ma sufficientemente astratta
 - Non fa riferimento a linguaggi di programmazione specifici
 - Definisce le interfacce tramite IDL CORBA
 - Esistono implementazioni in C, C++, Java, Perl, ...
- Documenti di riferimento:
 - Raccomandazioni w3c

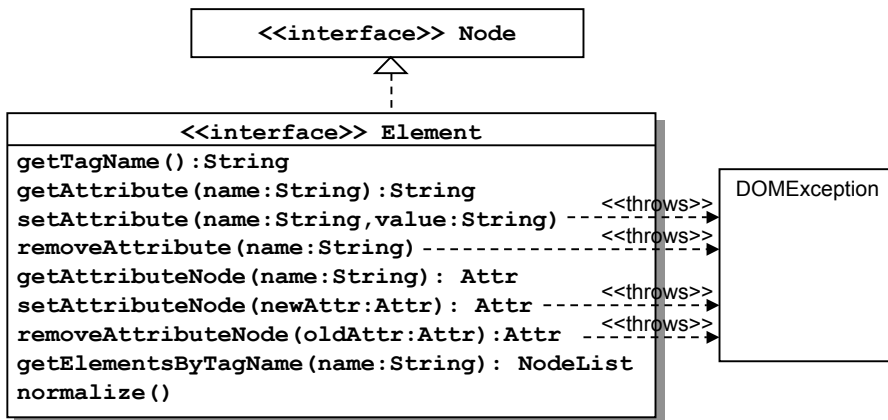
Organizzazione delle specifiche DOM

- DOM level 1
 - Funzionalità di base (semplice navigazione e manipolazione dinamica dei documenti)
- DOM level 2
 - Ulteriori strumenti di navigazione
 - Manipolazione di fogli di stile
 - Supporto per i namespace
 - Modello ad eventi
- DOM level 3
 - Supporto per convalida di documenti secondo DTD e schema
 - Perfezionamento dei meccanismi di caricamento e salvataggio

Le Interfacce DOM di Livello 1



Esempio: L'interfaccia Element



Simple API for XML (SAX)

- Metodo sviluppato nell'ambito della XML-DEV mailing list per un accesso più veloce ai documenti XML
- L'applicazione espone dei metodi che vengono chiamati dal Parser al verificarsi di determinati eventi
- Esempi di eventi:
 - `setDocumentLocator` avvio del parsing
 - `startDocument` inizio di documento XML
 - `endDocument`
 - `startElement` inizio di elemento XML
 - `endElement`
 - `characters` testo

Confronto DOM/SAX

	DOM	SAX
Velocità di Elaborazione	bassa	alta
Occupazione di memoria	alta	bassa
Facilità di accesso ai dati	alta	bassa
	Più adatto a modificare i dati	Più adatto a leggere i dati

Presentazione di documenti XML

- Lo stile di presentazione di un documento XML viene specificato a parte
- E' possibile usare
 - Cascading Style Sheet (CSS)
 - EXtensible Stylesheet Language (XSL)
- Lo stile viene associato al documento con un'apposita processing instruction (posizionata prima della radice)

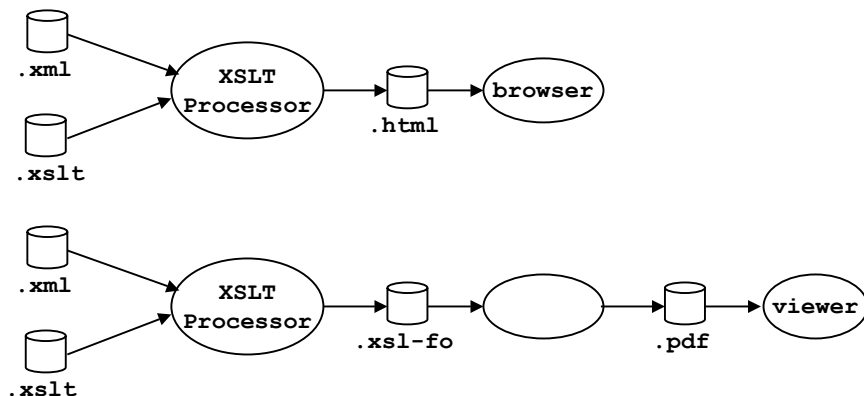
```
<?xml:stylesheet type="text/css" href="mystyle.css"?>
```

```
<?xml:stylesheet type="text/xsl" href="mystyle.xsl"?>
```

XSL (eXtensible Stylesheet Language)

- E' un linguaggio ideato per descrivere la **formattazione** dei documenti XML (presentazione)
 - più avanzato rispetto a CSS
- E' formato da due parti:
 - **XSLT** (XSL Transformations)
Applicazione XML per specificare le regole di trasformazione per convertire un documento XML in un altro tipo di documento
 - **XSL-FO** (XSL Formatting Objects)
Applicazione XML per descrivere il layout di un documento (su pagine, blocchi di testo, figure, ecc.)

Esempi di uso di XSL



Altri Possibili Usi

- XSLT è di fatto un vero e proprio metodo alternativo per l'elaborazione di documenti XML
 - Trasformazioni XML-XML
 - Estrazione di report a partire da grossi documenti
 - ...

XSLT

- La trasformazione viene descritta con stile *dichiarativo*:
 - Si dichiarano le **regole** della trasformazione tramite i “template”, che sono modelli di parti del documento cui è associato un testo trasformato
 - Ogni volta che viene riconosciuto un template nel documento di input, viene scritto il testo trasformato nell'output

