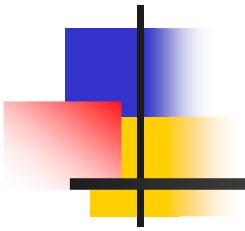
Linguaggi

Corso M-Z - Laurea in Ingegneria Informatica
A.A. 2009-2010

Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it



XML

Schema Lezione

- Generalità XML
- Struttura di un documento XML
- Document Type Definition (DTD)
- XML Schema
- Utilizzo di XML
- XLL, XSL
- Limiti di XML, RDF

XML - Generalità

- Lo sviluppo tecnologico relativo alle reti e il crescente sviluppo della distribuzione di hardware per la comunicazione sempre più sofisticato (personal computer ma anche palmari e telefoni dalle funzioni via via più complesse) ha reso necessaria la definizione di standard per lo scambio di documenti e informazioni.
- Da molti anni le organizzazioni per la standardizzazione lavorano alla definizione di linguaggi di scambio. Già nel 1986 l'International Organization for Standardization (ISO) produsse **SGML** (Standard Generalized Markup Language), un linguaggio finalizzato a consentire l'identificazione del ruolo che i vari brani di informazione hanno nei documenti scambiati, delle informazioni che debbono essere inserite nei documenti scambiati e di quali programmi vanno utilizzati per interpretare correttamente le informazioni scambiate. Tutto ciò prima che nascesse il world wide web e si moltiplicassero siti e portali.

XML - Generalità

- La **complessità dell'SGML** ne ha limitato l'utilizzo per diversi anni, fino alla definizione di una versione semplificata, **XML (eXtensible Markup Language)**, avvenuta nel 1996 ad opera del W3C (WWW Consortium) a seguito di un'iniziativa di Jon Bosak (Sun Microsystems).
- XML può essere visto sia come un linguaggio di interscambio di dati sia come un linguaggio per la definizione di linguaggi (meta-linguaggio). Probabilmente il successo che ha avuto come linguaggio di scambio dipende dal suo essere essenzialmente un meta-linguaggio. Per fare un esempio, WML (Wireless Markup Language, utilizzato per la comunicazione via WAP) è scritto in XML. Anche RDF e, di conseguenza, DAM+OIL (i linguaggi per il Semantic Web) sono realizzati in XML.

XML - Generalità

Caratteristiche di XML:

- Linguaggio di markup
- Standard del W3C
- Semplicità di HTML
- Flessibilità di SGML
- Permette una strutturazione dei documenti
- Permette di separare la struttura dalla rappresentazione (al contrario di HTML)

XML - Generalità

- XML consente di definire delle strutture dati indipendenti dalla piattaforma, che possono essere elaborate in modo automatico.
- XML consente di definire propri tag. I tag XML dovrebbero consentire di descrivere la struttura dei dati, e non la loro rappresentazione su browser (come avviene attualmente con l'HTML).
- XML non consente di specificare come le informazioni vanno presentate (visualizzate). A questo scopo è necessario definire opportuni fogli di stile. E' possibile definire fogli di stile o utilizzando il linguaggio XSL (eXtensible Stylesheet Language) oppure i CSS (Cascading Style Sheets).

XML - Generalità

Confronto XML – HTML:

- Dal punto di vista dell'organizzazione dei dati XML somiglia ad HTML. Infatti anche in questo caso si fa uso di tag, con la notazione <TAG> testo </TAG>. La differenza principale è che mentre in HTML i tag sono predefiniti, in XML ogni utente può definire i propri tag.
- Un'altra differenza (più subdola) è che mentre HTML è case-insensitive, XML è case-sensitive.
- In secondo luogo l'XML è molto più preciso dell'HTML riguardo la chiusura dei tag: ad ogni <TAG> corrisponde sempre una chiusura </TAG>.
- Possono inoltre essere definiti tag vuoti ma la loro sintassi è <TAG/>.

XML - Generalità

- **Esempio di file XML:**

Rex Stout è l'autore di molti romanzi gialli
aventi come protagonisti il famoso Nero Wolfe
e il suo aiutante e narratore Archie Goodwin

<?xml version="1.0"?>
<autore>Rex Stout**</autore>** è l'autore di molti
<genere>romanzi gialli**</genere>** aventi come protagonisti
il famoso **<protagonista>**Nero Wolfe**</protagonista>**
e il suo aiutante e narratore
<narratore><protagonista>
Archie Goodwin
</protagonista></narratore>

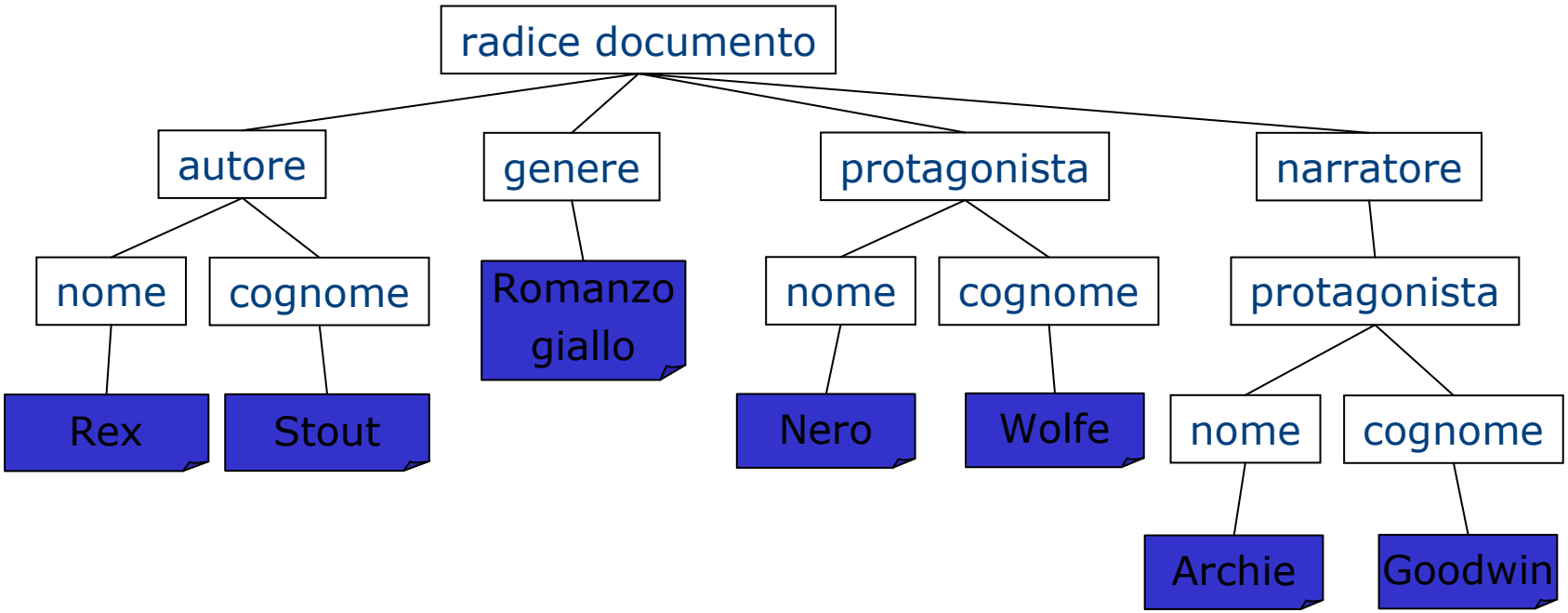
Struttura di un documento XML

- Permettendo l'annidamento, un documento XML può essere rappresentato mediante una struttura ad albero. Ad esempio:

```
<?xml version="1.0"?>
<autore><nome>Rex</nome><cognome>Stout</cognome></autore
    >
è l'autore di molti <genere>romanzi gialli</genere>
aventi come protagonisti il famoso
<protagonista attributo="famoso"><nome>Nero</nome>
    <cognome>Wolfe</cognome></protagonista>
e il suo aiutante e narratore
<narratore>
    <protagonista mansione="aiutante">
        <nome>Archie</nome>
        <cognome>Goodwin</cognome>
    </protagonista>
</narratore>
```

Struttura di un documento XML

- Graficamente



Struttura di un documento XML

- Un **documento XML è costituito da tre parti**, che possono essere contenute in tre file diversi:
  - **una Document Type Definition (DTD)**: è l'insieme delle regole che definisce la struttura dei dati, ovvero per la definizione degli elementi, degli attributi e delle loro correlazioni (la DTD definisce un vero linguaggio con un proprio vocabolario e regole sintattiche). Le DTD sono contenute alternativamente nel documento XML vero e proprio oppure in un file di testo con estensione dtd;
  - **un foglio di stile**: specifica le regole con cui un documento deve essere visualizzato da un applicativo (browser, word processor…);
  - **il documento XML vero e proprio: i dati**, organizzati secondo la struttura definita dalla DTD. I veri e propri documenti XML sono contenuti in file di solo testo con estensione xml.
- Come nel caso dell'HTML i dati possono alternativamente essere scritti in file oppure essere generati dinamicamente da applicativi quali i sistemi per la gestione di banche dati.

Struttura di un documento XML

- **Esempio di un documento dati XML**:

```
<persona>
    <nome>Giorgio</nome>
    <cognome>Rossi</cognome>
    <codice>ID2121</codice>
</persona>
<persona>
    <nome>Anna</nome>
    <cognome>Bianchi</cognome>
    <codice>ID3131</codice>
</persona>
```

- Per consentire a un applicativo di utilizzare i dati in questione, inviare i soli dati etichettati non è sufficiente: non è possibile che l'applicativo preveda tutti i possibili tag, strutture.
- D'altro canto i tag personalizzati in qualche modo specificano il ruolo che un brano di informazione ha all'interno del documento.
- I vari tag (<persona>, <nome>) sono definiti nella DTD del documento, che può essere memorizzata su un file a parte oppure inclusa nel documento XML.

Struttura di un documento XML

- In generale un **documento dati XML è diviso in tre parti**: un prologo, un corpo e un epilogo.
- Il **corpo** del documento XML contiene dei dati strutturati in un albero. L'albero è espresso da un elemento radice che contiene tutti gli altri elementi e i dati del documento. Tutto ciò che precede l'elemento radice costituisce il prologo del documento, ciò che lo segue l'epilogo.
- Il **prologo** di un documento è molto importante: in esso sono contenute direttive utili ai programmi che utilizzeranno il documento. In particolare un documento XML comincia con <?xml version="1.0"?>
  - Qualora il documento utilizzi tag definiti in una DTD esterna occorre indicarlo ai programmi In questo caso si ha
    <?xml standalone="no"?>
  - Occorrerà poi aggiungere al prologo l'informazione relativa a dove è memorizzata la DTD. A tal fine si utilizza <!DOCTYPE root SYSTEM "file.dtd" >, dove root è l'elemento che deve essere utilizzato come radice dell'albero in cui i dati sono strutturati, SYSTEM è una parola riservata del linguaggio, e file.dtd indica la posizione occupata dal nome del file in cui la DTD è memorizzata (ad esempio un'URL).

Struttura di un documento XML

- Esempio di documento dati XML con DTD esterno

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE personale SYSTEM "people.dtd">

<personale>
     <persona>
          <nome>Annalisa</nome>
          <cognome>Anselma</cognome>
          <codice>ID3121</codice>
     </persona>
     <persona>
          <nome>Ugo</nome>
          <cognome>Angeli</cognome>
          <codice>ID3122</codice>
     </persona>
     <persona>
          <nome>Bianca</nome>
          <cognome>De Giorgi</cognome>
          <codice>ID3123</codice>
     </persona>
     <persona>
          <nome>Oreste</nome>
          <cognome>Silvestri</cognome>
          <codice>ID3124</codice>
     </persona>
</personale>
```

Document Type Definition

- La DTD contiene la definizione della struttura dei dati di uno o più documenti. Quando si parla di struttura dei dati di un documento si assume che il documento possa essere ricorsivamente scomposto in parti (per esempio un libro può essere scomposto in capitoli, che possono essere suddivisi in sezioni e così via)
- in certi casi i vari elementi devono rispettare un particolare ordine di precedenza (per esempio le voci di una rubrica telefonica potrebbero prevedere il nome di una persona come primo elemento, seguito dal numero di telefono e dall'indirizzo).
- La strutturazione dei documenti XML avviene attraverso la definizione di tutti gli elementi che possono denotare il documento e delle loro inter-relazioni
- in altri termini la DTD di un documento contiene le definizioni dei tag e della struttura attesa dei dati.

Document Type Definition

- Un elemento è definito utilizzando la sintassi:
- <!ELEMENT *nome_elemento regola*>
- *nome_elemento* identifica un nuovo tag ed è una sequenza di caratteri che inizia con una lettera o un underscore e può poi contenere lettere, cifre, trattini, punti. Nel caso in cui si usino i namespace è anche possibile utilizzare ":"
- Nel caso più semplice la *regola* può valere alternativamente: ANY oppure #PCDATA
  - ANY è una parola chiave che indica che fra <nome_elemento> e </nome_elemento> può essere contenuta qualsiasi cosa, ovvero possono essere contenuti sia dati (tipicamente del testo, "PCDATA" sta per parsed character data) sia altri elementi. Se anziché ANY si utilizza #PCDATA non potranno essere contenuti altri elementi.

Document Type Definition

- Esempio

```
<!ELEMENT persona ANY>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT codice (#PCDATA)>
```

- Con riferimento ai tag utilizzati sopra, <persona> dovrà avere associata come regola ANY mentre <nome> potrebbe avere associato un semplice #PCDATA.
- Questi sono solo i casi di regola più semplici: regole che non ci permettono di strutturare i dati ma in generale le regole sono proprio utilizzate per strutturare l'informazione definendo un modo in cui gli elementi sono composti di altri elementi.

Document Type Definition

- La DTD prima definita non correla in alcun modo gli elementi definiti: per nessun motivo siamo tenuti a descrivere una persona in termini di nome, cognome e codice.
- Normalmente pero', nome, cognome e codice dovrebbero essere le componenti in cui è strutturata l'informazione relativa a persona.
- Se potessimo esprimere questa correlazione potremmo anche realizzare applicativi che controllano il modo in cui sono descritti i dati, verificando che la struttura intesa sia rispettata. In generale è possibile strutturare un elemento utilizzando sequenze di altri elementi, che possono ricorrere anche più volte.

Document Type Definition

- Nel caso di persona, per indicare che l'informazione relativa a una persona è strutturata nella sequenza di nome, cognome e codice si indica:

```
<!ELEMENT persona (nome, cognome, codice)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT codice (#PCDATA)>
```

- La strutturazione può avere più livelli ed essere di vario tipo. Consideriamo per esempio il documento che contiene informazioni relative a Giorgio Rossi, Anna Bianchi ed eventuali altre persone. E' possibile regolamentare la struttura dell'intero documento dichiarando che esso è costituito da una sequenza di persone.

Document Type Definition

```
<!ELEMENT elenco (persona*)>
<!ELEMENT persona (nome, cognome, codice)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT codice (#PCDATA)>
```

- il simbolo * (asterisco) indica una sequenza eventualmente vuota, costituita da un numero imprecisato di persone. Se si vuole indicare che la sequenza non può essere vuota (l'elenco deve contenere almeno una persona) si utilizza il simbolo +. Se invece l.elemento può occorrere al più una volta (quindi 0 oppure 1) si utilizza il simbolo ?
- L'elemento più astratto (quello definito in termini di altri elementi ma che non è utilizzato all'interno di nessuna regola; nell'esempio è proprio elenco) è detto radice. È necessario che in una DTD sia definito sempre un elemento radice.

Document Type Definition

- Oltre ai simboli visti le regole possono contenere anche l'operatore | (oppure) che indica un'alternativa. Supponiamo, per esempio, che il campo codice sia opzionale nella definizione di persona. Potremmo quindi scrivere:

```
<!ELEMENT persona ((nome, cognome, codice) | (nome, cognome))>
```

- Che si legge: un elemento persona è strutturato in un nome, seguito da un cognome, seguito da un codice oppure da un nome seguito da un cognome. Le parentesi tonde sono utilizzate per dare maggiore granularità alla sintassi definita.
- Si osservi che le regole per definire una certa struttura non sono necessariamente uniche, tornando al caso precedente è possibile definire anche la seguente regola, equivalente a ((nome, cognome, persona) | (nome, cognome))

```
<!ELEMENT persona (nome, cognome, codice?)>
```

- Le regole più compatte sono generalmente più efficienti, anche se non sempre sono altrettanto intuitive.

Document Type Definition

- In accordo al DTD ora definito, un esempio corretto ed uno scorretto di file XML:

```
<persona>
  <nome>Anna</nome>
  <cognome>Bianchi</cognome>
</persona>

<persona>
   <nome>Luca</nome>
   <cognome>Rossi</cognome>
   <codice>9876789</codice>
</persona>
```

```
<persona>
  <nome>Anna</nome>
  <nome>Maria</nome>
  <cognome>Bianchi</cognome>
</persona>
```

Document Type Definition

- Gli elementi vuoti sono definiti utilizzando la parola chiave EMPTY:

```
<!ELEMENT nome_elemento EMPTY>
```

  - gli elementi vuoti possono avere attributi ma non contengono nulla al loro interno; un esempio ripreso dall'HTML e' il tag IMG che descrive un'immagine:
  - <IMG src="./images/prova.jpg" border=1></IMG>
  - fra <IMG…> e </IMG> non si pone nulla; nel caso specifico, poiche' l'HTML lo permette, e' quasi sempre assente </IMG>
- nel caso di DTD che non si intendano utilizzare per descrivere il contenuto di più documenti XML è possibile inserire la DTD direttamente nel prologo del documento, all'interno della direttiva <!DOCTYPE ...>.
- In tal caso la specifica 'SYSTEM <URL_della_dtd> è sostituita dalla DTD medesima inserita fra parentesi quadre.

DTD

Esempio di DTD incorporato

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

<!DOCTYPE ricetta [
     <!ELEMENT ricetta (titolo,lista_ingredienti,preparazione)>
     <!ELEMENT titolo (#PCDATA)>
     <!ELEMENT lista_ingredienti (ingrediente+)>
     <!ELEMENT preparazione (passo+)>
     <!ELEMENT ingrediente (#PCDATA)>
     <!ELEMENT passo (#PCDATA)>
]>

<ricetta>
     <titolo> insalata </titolo>
     <lista_ingredienti>
          <ingrediente> indivia </ingrediente>
          <ingrediente> pomodoro </ingrediente>
          <ingrediente> cetriolo </ingrediente>
          <ingrediente> sedano </ingrediente>
          <ingrediente> olio </ingrediente>
          <ingrediente> aceto </ingrediente>
          <ingrediente> sale </ingrediente>
     </lista_ingredienti>
     <preparazione>
          <passo> lavare l'invidia, il sedano e il pomodoro</passo>
          <passo> pelare il cetriolo </passo>
          <passo> affettare cetriolo, pomodoro e sedano </passo>
          <passo> tagliare l'insalata</passo>
          <passo> mescolare le verdure in un'insalatiera </passo>
          <passo> versare olio e aceto, salare e mescolare</passo>
     </preparazione>
</ricetta>
```

Document Type Definition

- Gli elementi possono essere caratterizzati meglio tramite la specifica di una lista di attributi.
- Considerando il tag <persona> prima definito, consideriamo in particolare il campo codice. Codice è attualmente definito come un elemento della stessa DTD, potrebbe però tornare utile rappresentarlo in modo diverso, ad esempio: <persona codice="12343"> … </persona>. In questo caso codice non è più un elemento bensì è un attributo di <persona>.
- Dal punto di vista sintattico un attributo è così specificato:

```
<!ATTLIST nome_elem nome_attr tipo valore_default>
```

- *nome_elem* è un riferimento all'elemento caratterizzato dall'attributo (nell'esempio *persona*),
- *nome_attr* è il nome dell'attributo in questione (*codice*)
- *tipo* indica il tipo di valore che l'attributo può assumere
- *valore_default* è il valore che l'attributo assume quando non specificato diversamente nel documento.

Document Type Definition

- Il *tipo* di un attributo può essere alternativamente:
  - una lista di alternative: alt1 | alt2 | . | altN
  - CDATA: sequenza di caratteri (una stringa)
  - ENTITY: una delle entità dichiarate nella DTD (indicato nel seguito)
  - ENTITIES: più entità separate da spazi
  - ID: attributo che funge da identificatore di uno specifico elemento (es. il codice fiscale è un identificatore di una persona); utilizzando il valore degli attributi ID è possibile correlare dati presenti nel documento. Ogni entità può avere al più un identificatore.
  - IDREF: i valori ammessi sono identificatori di altri elementi (es. per identificare il padre di una persona posso utilizzarne il codice fiscale). La coppia di tipi IDREF/ID consente la correlazione dei dati.
  - IDREFS: sequenza di IDREF separati da spazi
  - NMTOKEN: contrassegno di un nome XML; è più restrittivo di CDATA in quanto indica che i valori che l'attributo può assumere devono essere costituiti esclusivamente da lettere, cifre, punti, caratteri di sottolineatura, trattini e due punti.
  - NMTOKENS: sequenza di NMTOKEN separati da spazi
  - NOTATION: notazione dichiarata nella DTD (illustrata in seguito)

Document Type Definition

- Il campo *valore_default* può assumere diversi valori:
  - una stringa di caratteri.
  - la parola chiave #REQUIRED. In questo caso, pur non esistendo un valore di default predefinito, è necessario che l'utente (o il sistema), che scrive il documento XML, specifichi sempre un valore per l'attributo in questione, per ogni occorrenza dell'elemento da esso caratterizzato.
  - parola chiave #IMPLIED (l'attributo è opzionale e il suo valore può non essere specificato)
  - coppia #FIXED valore, dove #FIXED è una parola chiave che indica che l'attributo ha il valore costante *valore*.

Document Type Definition

Alcuni esempi di definizione di attributi:

- <!ATTLIST persona cod_fis ID #REQUIRED>: codice fiscale è un identificatore di persona, è necessario specificarne il valore ma non esiste un valore di default
- <!ATTLIST persona azienda CDATA #FIXED .Pinco Palla spa.>: supponendo di scrivere una DTD per un file che conterrà le informazioni relative ai dipendenti della ditta "Pinco Palla spa.", l'attributo "azienda" è sempre uguale al valore "Pinco Palla spa".
- <!ATTLIST persona genere (uomo | donna) #REQUIRED>: attributo necessario, i valori ammissibili sono solo due: uomo oppure donna
- <!ATTLIST persona data_nascita CDATA #IMPLIED>: la data di nascita non è un attributo necessario, è di tipo CDATA, quindi sostanzialmente una qualsiasi sequenza di caratteri; non si è utilizzato NMTOKEN perché le date possono contenere il carattere slash (/)
- <!ATTLIST persona padre IDREF #IMPLIED>: l'attributo "padre" non è necessario ma qualora specificato deve essere l'identificatore di un altro elemento. Si noti che non si specifica di quale elemento si tratti (se una persona o altro)

Document Type Definition

- Un esempio di DTD completo:

```
<!ELEMENT organizer (rubrica, agenda)>

<!ELEMENT rubrica ((categoria, voce)*)>
<!ELEMENT categoria (#PCDATA)>
<!ELEMENT voce (((nome, cognome) | rag_sociale), tel*,
                indirizzo?)>
<!ATTLIST voce id_voce ID #REQUIRED>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT rag_sociale (#PCDATA)>
<!ELEMENT tel (#PCDATA)>
<!ELEMENT indirizzo (#PCDATA)>

<!ELEMENT agenda (appuntamento*)>
<!ELEMENT appuntamento (#PCDATA)>
<!ATTLIST appuntamento giorno_sett (lun | mar | mer | gio | ven |
sab | dom) #IMPLIED>
<!ATTLIST appuntamento luogo CDATA #REQUIRED>
<!ATTLIST appuntamento chi_incontro IDREF #REQUIRED>
<!ATTLIST appuntamento quando CDATA #REQUIRED>
```

DTD

Supponendo che la DTD precedente sia memorizzata nel file "organizer.dtd" possiamo scrivere un documento XML che la utilizza:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE organizer SYSTEM
"http://www.di.unito.it/~baroglio/proveXML/organizer.dtd">

<organizer>

<!-- parte di rubrica -->
<rubrica>
    <categoria>amici</categoria>
    <voce id_voce="voce_matt">
        <nome> Matteo </nome>
        <cognome> Baldoni </cognome>
        <tel> 12345 </tel>
        <tel> 67890 </tel>
        <indirizzo> via pinco palla 21 </indirizzo>
    </voce>
</rubrica>

<!-- parte di agenda -->
<agenda>

    <appuntamento luogo="ristorante X"
                  chi_incontro="voce_matt" quando="4/6">
        Compleanno Matteo!!!
    </appuntamento>

</agenda>

</organizer>
```

Document Type Definition

- come decidere se rappresentare una componente di un'entità come un attributo o come un'altra entità a parte? Nel caso degli identificatori occorre definirli come attributi in quanto solo in questo modo potranno essere utilizzati per fare riferimento da un elemento ad un altro. Anche nel caso in cui si conoscano a priori tutti i possibili valori che una componente dell'elemento può assumere è bene utilizzare un attributo, idem quando si specifica un valore di default.
- L'unico caso in cui potrebbe invece essere meglio utilizzare un altro elemento è il caso di CDATA #IMPLIED: una stringa di caratteri priva di valore di default, non costretta ad appartenere ad un insieme limitato di alternative.
- In generale, tuttavia, è difficile o addirittura impossibile effettuare una scelta oggettiva, in quanto non vi è un chiaro vantaggio nell'una oppure nell'altra soluzione. In questi casi il programmatore è libero di utilizzare la forma che preferisce.

Document Type Definition

- Oltre agli elementi e ai loro attributi, una DTD può contenere la definizione di un certo numero di **entità**. Le entità corrispondono sostanzialmente a delle **macro**: consentono di attribuire a una sequenza di caratteri un nome convenzionale, che può essere utilizzato (per comodità) al posto della sequenza stessa, e ve ne sono di **vari tipi**: generali, parametriche, esterne, non analizzate
- Il primo tipo è quello delle entità generali. **Un'entità generale** attribuisce un nome convenzionale a una stringa di caratteri utilizzata in uno o più documenti XML; viene definita nel seguente modo:

```
<!ENTITY nome "sequenza di caratteri">
```

Document Type Definition

- Per esempio supponiamo di dover gestire la modulistica di una ditta (tale Pinco Palla srl) e che il nome della ditta in questione compaia in tutti i moduli. Se i moduli in questione sono memorizzati come documenti XML, anziché inserire in ciascuno la stringa "Pinco Palla srl", potremmo definire un'entità id_ditta nel seguente modo:

```
<!ENTITY id_ditta "Pinco Palla srl">
```

- e poi utilizzare id_ditta usando la notazione &nome_entità; nei documenti XML veri e propri

```
… testo testo &id_ditta; testo testo …
```

- l'utilizzo è analogo ai caratteri speciali utilizzati in HTML, come ad esempio "à" che produce "à"

Document Type Definition

- Il vantaggio è duplice: da un lato avremmo uniformità di presentazione, dall'altro si puo' aggiornare tutta la modulistica in un colpo solo, semplicemente sostituendo la stringa associata ad id_ditta nell'unica definizione dell'entità contenuta nella DTD.
- L'operazione di sostituzione del nome di un'entità con la sequenza di caratteri corrispondente (sequenza di sostituzione) è detta risoluzione dell'entità ed è effettuata automaticamente dagli applicativi preposti ad elaborare documenti XML (browser, editor). All'atto della fruizione dei dati l'utente non leggerà il nome dell'entità ma la sequenza di caratteri che rappresenta.
- Le entità generali vengono risolte esclusivamente nei documenti XML, quindi è vietato utilizzare un'entità generale in qualche punto di una DTD. Per esempio, il codice seguente, nel quale si associa un nome a una regola utilizzando un'entità generale e poi si usa il nome al posto della regola nella definizione di un'entità è sbagliato:

```
<!ENTITY regola "(#PCDATA)">
<!ELEMENT personaggio &regola;>
```

Document Type Definition

- **Un'entità parametrica** può essere utilizzata solo all'interno delle DTD. La definizione di un'entità parametrica si distingue per la presenza di un carattere percentuale % posto dopo !ENTITY e da applicare anche nell'utilizzo con %nome;. Il funzionamento è analogo a quello delle entità generali, ma si applicano alle DTD. L'esempio precedente può essere riscritto utilizzando un'entità parametrica:

```
<!ENTITY % regola "(#PCDATA)">
<!ELEMENT personaggio %regola;>
```

- **Le entità esterne** consentono di includere in un documento il contenuto di un altro documento accessibile attraverso un URL. Per includere il documento esterno dovremo definire un'entità esterna corrispondente all'URL della pagina ed utilizzarlo all'interno della pagina da noi definita. Quello che otterremo non sarà un link, bensì una pagina parte del cui contenuto sarà stato importato da un'altra fonte. Le entità esterne sono definite nella DTD come segue:

```
<!ENTITY nome SYSTEM url>
```

- *nome* è il nome dell'entità definita, *url* il suo indirizzo. Per richiamare un'entità esterna si utilizza la notazione già vista &nome;

Document Type Definition

- L'ultimo tipo è **l'entità non analizzata**. Le entità non analizzate sono simili a quelle esterne, la differenza è che mentre le prime fanno riferimento ad altri documenti XML, queste ultime fanno riferimento a documenti non-XML. È opportuno che i due casi vengano distinti in quanto mentre i documenti XML devono essere analizzati dal parser, i documenti non-XML sono direttamente inviati all'applicativo finale (browser, data-base manager, editor, ...). La loro definizione segue lo schema:

```
<!ENTITY nome SYSTEM url NDATA tipo>
```

- La differenza rispetto alle entità esterne è la parola chiave NDATA seguita da un nome convenzionale che indica il tipo di informazione contenuta nel documento non analizzato. Nel caso di un'immagine in formato gif, per esempio, il tipo sarà GIF89a.

Document Type Definition

- Le **notazioni** sono una componente del linguaggio usato per scrivere DTD, e sono legati alle entità non analizzabili. Sono utilizzati per compiere delle associazioni fra un nome utilizzato come tipo di NDATA e una specifica più fine. La differenza rispetto alle entità è che tali associazioni non sono finalizzate alla sostituzione. Un uso tipico consiste nell'indicare quale applicativo consente di trattare un certo tipo di documenti. Per esempio:

```
<!NOTATION jpeg SYSTEM “netscape.exe”>
```

- Questa specifica associa l'applicativo “netscape.exe” al tipo di documento non analizzabile jpeg.

Document Type Definition

- Tornando agli attributi, un attributo ENTITY assume come valori entità definite all'interno della DTD. Per esempio:

```
<!ATTLIST ditta motto ENTITY #IMPLIED>
```

- Nell'esempio, *motto* è un attributo dell'elemento *ditta* ed assume come valori delle entità. Se in qualche punto della DTD è definita l'entità:

```
<!ENTITY timeo "timeo Danaos dona ferentes">
```

- Nel documento XML potremo scrivere:

```
<ditta motto="timeo"> Nome della Ditta </ditta>
```

- Si osservi che quando si usa un'entità come valore di un attributo non compaiono i delimitatori "&" e ";"

Document Type Definition

- Nell'esempio è stata utilizzata un'entità generale, tuttavia il caso più frequente di utilizzo degli attributi entità è finalizzato a creare dei collegamenti con dati esterni da non analizzare, come per esempio le immagini.
- Si pensi al tag <IMG> di HTML: questo tag ha un modificatore (src) il cui valore è l'identificatore di un file immagine, che verrà mostrato dal browser. L'immagine non è un documento HTML, quindi il browser deve trattarla in un modo particolare, adeguato al formato dei dati che contiene.
- Supponiamo di voler realizzare qualcosa di analogo in XML. In particolare, supponiamo di voler definire un elemento *mia_img*, che circonda del testo che costituirà la didascalia dell'immagine e che ha un attributo *sorgente* analogo a SRC di <IMG>: tale attributo assumerà come valore l'URL di un file immagine

Document Type Definition

- Nella DTD si avrà allora:

```
<!ELEMENT mia_img>
<!ATTLIST mia_img sorgente ENTITY #REQUIRED>
...
<!ENTITY im_logo SYSTEM “immagini/logo.gif” NDATA GIF89a>
<!ENTITY im_help SYSTEM “immagini/help.gif” NDATA GIF89a>
```

- L'elemento mia_img in un file XML verrà usato nel seguente modo:

```
<mia_img sorgente=”im_logo”>La nostra ditta</img>
<mia_img  sorgente=”im_help>se  hai  bisogno  di  aiuto
</mia_img>
```

Document Type Definition

- L'ultimo caso di attributo lasciato in sospeso è l'attributo di tipo NOTATION. Le notazoni sono usate per associare tipi di documenti non XML a strumenti necessari per gestire tali documenti. Consideriamo il caso di documenti multimediali. Un esempio potrebbe essere:

```
<!NOTATION avi SYSTEM "movieplayer">
<!NOTATION jpg SYSTEM "imgplayer">
<!NOTATION wav SYSTEM "musicplayer">
```

- Supponiamo di avere definito nella nostra DTD un elemento *file*, che identifica in modo generico un elemento multimediale compreso in un documento. Supponiamo inoltre di voler associare a tale elemento una proprietà che indica quale player va utilizzato. In questo caso possiamo utilizzare un attributo di tipo NOTATION, che assume valori nell'insieme {avi, jpg, wav}:

```
<!ATTLIST file player NOTATION (avi | jpg | wav) #REQUIRED>
```

DTD

Un esempio riassuntivo:

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

<!DOCTYPE ricettario [
    <!NOTATION video SYSTEM "myvideoplayer.exe">
    <!NOTATION img SYSTEM "netscape.exe">

    <!ENTITY pri "PrimaVera!!">
    <!ENTITY ric_pasta SYSTEM "http://www.mysite.it/pasta.mpg"
                NDATA video>

    <!ELEMENT ricettario (ricetta*)>
    <!ELEMENT ricetta (titolo,lista_ingredienti,preparazione)>
    <!ELEMENT titolo (#PCDATA)>
    <!ELEMENT lista_ingredienti (ingrediente+)>
    <!ELEMENT preparazione (passo+)>
    <!ELEMENT ingrediente (#PCDATA)>
    <!ATTLIST ingrediente
            numero CDATA #REQUIRED
            misura (g | hg | cucchiaio | cucchiai) #IMPLIED>
    <!ELEMENT passo (#PCDATA)>
    <!ATTLIST passo
            genere NOTATION (video | img) #IMPLIED
            sorgente ENTITY #IMPLIED>
]>
```

DTD

Un esempio riassuntivo:

```
<ricettario>
 <ricetta>
   <titolo> insalata &pri; </titolo>
     <lista_ingredienti>
          <ingrediente numero="un cespo"> indivia </ingrediente>
          <ingrediente numero="1"> pomodoro </ingrediente>
          <ingrediente numero="1"> cetriolo </ingrediente>
          <ingrediente numero="1"> sedano </ingrediente>
          <ingrediente numero="2" misura="cucchiai">
                    olio </ingrediente>
          <ingrediente numero="1" misura="cucchiaio">
                    aceto </ingrediente>
          <ingrediente numero="q.b."> sale </ingrediente>
     </lista_ingredienti>
     <preparazione>
          <passo> lavare l'invidia, il sedano e il pomodoro</passo>
          <passo> pelare il cetriolo </passo>
          <passo> affettare cetriolo, pomodoro e sedano </passo>
          <passo> tagliare l'insalata</passo>
          <passo> mescolare le verdure in un'insalatiera </passo>
          <passo> versare olio e aceto, salare e mescolare</passo>
     </preparazione>
 </ricetta>
 <ricetta>
   <titolo> spaghetti &pri; </titolo>
     <lista_ingredienti>
          <ingrediente numero="q.b."> pasta </ingrediente>
          <ingrediente numero="q.b."> aglio </ingrediente>
          <ingrediente numero="q.b."> olio </ingrediente>
          <ingrediente numero="q.b."> peperoncino </ingrediente>
     </lista_ingredienti>
     <preparazione>
          <passo genere="video" sorgente="ric_pasta">Segui il filmato</passo>
     </preparazione>

 </ricetta>
</ricettario>
```

Validazione di documenti XML

- Un documento è **valido** quando rispetta le regole espresse dalla DTD.
- Un documento si dice invece **ben formato** quando rispetta le generiche regole di scrittura di un documento XML, che sono:
  - i valori degli attributi devono essere racchiusi fra doppi apici
  - i tag devono sempre essere chiusi
  - aperture e chiusure dei tag non devono intersecarsi (es. <nome> Anna <cognome></nome>Rossi </cognome> è sbagliato)
- I sistemi che verificano la correttezza (sintattica) dei documenti XML sono detti parser. A seconda che tali strumenti verifichino la correttezza del documento rispetto alla DTD indicata oppure rispetto alle generiche regole sintattiche di XML solamente, vengono detti parser validanti o parser non-validanti.

XML Schema

- le DTD non sono l'unico modo per definire la struttura di un documento XML. Un'alternativa è data dagli **XML Schema**. Le DTD derivano dal linguaggio SGML, di cui XML è una semplificazione, gli schemi XML sono nati successivamente.
- Le principali differenze rispetto alle DTD sono che da un lato è possibile esprimere vincoli di tipo, dall'altro è possibile esprimere vincoli di cardinalità. Riguardo il primo tipo di vincolo, consideriamo un elemento peso definito come <!ELEMENT peso (#PCDATA)>. In un documento, sono ammessi utilizzi come <peso>15</peso>, <peso>Anna</peso>, <peso>una giornata in barca</peso>. Intuitivamente potremmo pensare che si tratti di una misura, quindi che il primo uso sia l'unico corretto però ci è impossibile far sì che il parser controlli che peso racchiuda solo dei numeri, quindi tutti e tre gli usi sono formalmente corretti. La seconda differenza riguarda i vincoli di cardinalità: abbiamo visto che nelle regole che specificano la struttura degli elementi possono essere utilizzati gli operatori di ricorrenza * e +. Non possiamo però esprimere strutture del tipo "l'elemento A è costituito da una lista di almeno 2 ma non più di 5 elementi B".

XML Schema

- DTD (Document Type Definition)
  - Specifica annidamenti e combinazioni permesse di elementi, attributi ecc.
  - Un doc XML conforme a DTD = valido
- XML Schema
  - Definisce lo schema sintattico con la stessa sintassi XML
  - Fa uso di namespace e definizione di tipi
  - Un doc XML conforme a XML Schema = schema valido

XML Schema

- Definizioni di tipo
  - Semplici

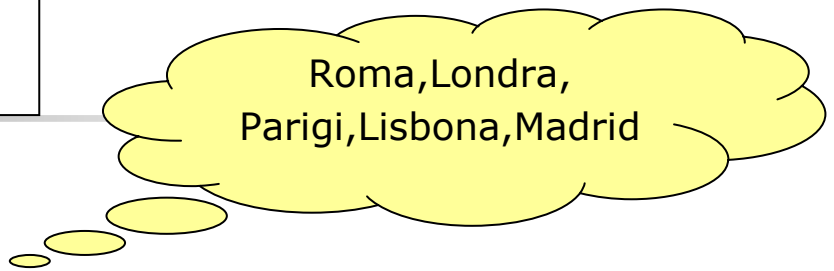
```
<simpleType name='stringlist' base='string' derivedBy='list'
/>
```

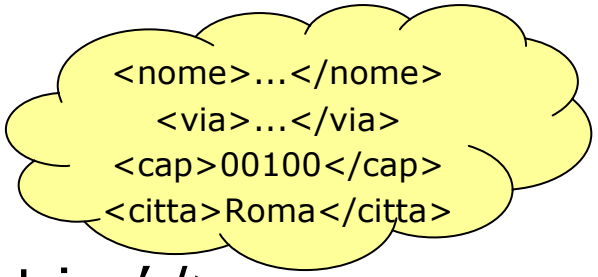
  - Complessi

```
<complexType name='indirizzo'>
  <sequence>
        <element name='nome' type='string' />
        <element name='via' type='string' />
        <element name='cap' type='decimal' />
        <element name='citta' type='string' />
  </sequence>
</complexType>
```

- Dichiarazioni di elementi e attributi

```
<element name='elenco' type='stringlist'>
```





XML Schema

- Schema

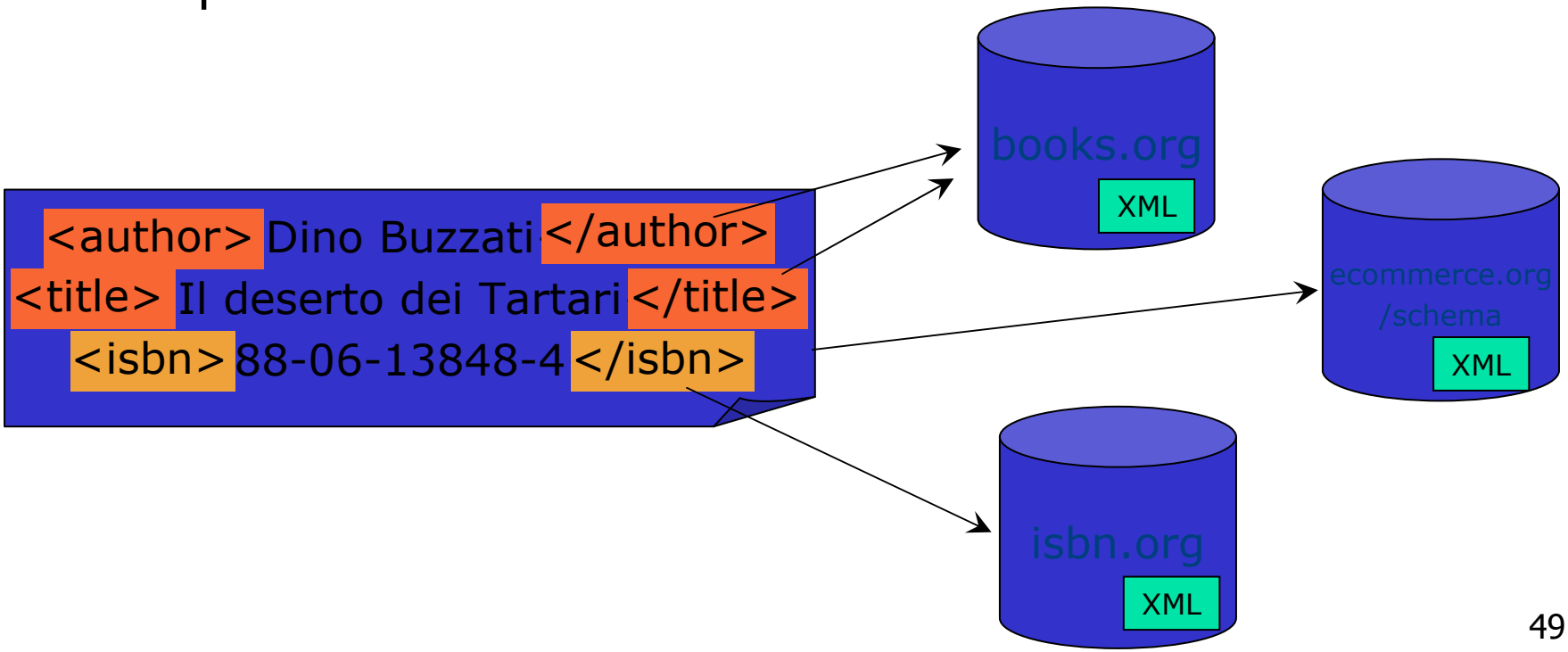
```
<schema xmlns='http://www.w3.org/1999/XMLSchema'
      targetNamespace='http://lazoo.org'>
   <simpleType name='comment' base='string'>
     <maxLength value='1024'>
   </simpleType>
   <element name='about' type='comment'>
   <element name='autore' type='string'>
</schema>
```

- Istanza schema-valida

```
<?xml version='1.0' xmlns='http://lazoo.org' ?>
<autore>Luigi Pirandello</autore>
<about>romanziere e commediografo</about>
```

XML Namespaces

- Namespace = collezione di nomi identificati da URI (Uniform Resource Identifier)
- Associati a elementi e attributi di documenti XML
- Permettono la visibilità al di fuori del documento
- Esempio di libreria online



XML Namespaces

```
<?xml version='1.0'
    xmlns:books='http://www.books.org/'
    xmlns:store='http://ecommerce.org/schema'
    xmlns:isbn='http://www.isbn.org/' ?>
<store:bookstore>
    <books:book>
        <books:title>Il deserto dei Tartari</books:title>
        <books:author>Dino Buzzati</books:author>
        <isbn:number>88-06-13848-4</isbn:number>
    </books:book>
    <books:book>
        <books:title>L'isola di Arturo</books:title>
        <books:author>Elsa Morante</books:author>
        <isbn:number>88-06-13838-3</isbn:number>
    </books:book>
</store:bookstore>
```

Utilizzo di XML

- E' stato introdotto il linguaggio XML, ponendo particolare attenzione alla costruzione di DTD. Una domanda che potrebbe sorgere spontanea è: ora che abbiamo strutturato l'informazione relativa ad un dominio, che cosa ce ne facciamo?
- Alcuni utilizzi di XML:
  - Come sintassi di serializzazione per altri linguaggi
  - Come markup per risorse (documenti, pagine web, configurazioni ecc.)
  - Come formato uniforme per lo scambio dei dati, anche al fine di accedere alle informazioni in maniera strutturata (stile database)

Utilizzo di XML

- Come sintassi di serializzazione per altri linguaggi
  - Esempio: Java → XML

```
class Hello
{
    static public void main(String[] args)
    {
      System.out.println("Hello world!");
    }
}
```

Utilizzo di XML

- Come sintassi di serializzazione per altri linguaggi
  - Esempio: Java → XML

```
<?xml version="1.0" ?>
<class name="Hello">
    <method name="main" static="true" scope="public"
      returnType="void" input="String[]">
        System.out.println("Hello world!");
    </method>
</class>
```

Utilizzo di XML

- Come markup per risorse (documenti, pagine web, configurazioni ecc.)
  - Es. classificazione delle istanze di un documento:

    Tra i precursori del genere giallo troviamo

    <autore>Edgar Allan Poe**</autore>**, con **<romanzo genere="giallo">**I delitti della Rue Morgue**</romanzo>**, e **<autore>**Arthur Conan Doyle**</autore>** con il suo **<personaggio>**Sherlock Holmes**</personaggio>**.
  - Altro esempio quello degli editor di testo: diversi editor utilizzano (in modo trasparente all'utente) il linguaggio XML per mantenere le informazioni relative ai documenti costruiti. Questo significa che il testo scritto viene man mano opportunamente "taggato" in XML dall'editor.
  - Ai vari tag sono inoltre associate regole di visualizzazione (o presentazione) e l'editor le usa per far sì che ad esmpio un testo dichiarato in grassetto, oltre ad essere descritto in XML, appaia agli occhi dell'utente in grassetto.

Utilizzo di XML

- Come sintassi di serializzazione per altri linguaggi
- Come markup per risorse (documenti, pagine web, configurazioni ecc.)
- Come formato uniforme per lo scambio dei dati
  - Es. SOAP (Simple Object Access Protocol)

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.stock.org/stock">
  <m:GetStockPrice>
        <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

XML - XLL

- L'XML fornisce un complesso meccanismo per la gestione dei link, definiti dallo standard XML Linking Language (XLL), tramite cui:
- specificare quando un link deve attivarsi (manualmente o meno);
  - specificare cosa effettuare a seguito dell'attivazione (aprire una nuova finestra, sostituire la precedente, o sostituire l'elemento iniziale;
  - collegare l'elemento iniziale a più risorse (link semplici o estesi), specificando la regola per ogni singolo locator;
  - definire una gerarchia di documenti tramite l'X-pointer, così da poter supportare link assoluti (es puntare alla radice, puntare al terzo figlio), o relativi alla gerarchia (es puntare al precedente documento), eventualmente completandoli con string matching, ad esempio:
  - ROOT.CHILD(3,CHAP).STRING(7,"Napoleone")
    trova la settima occorrenza della stringa "Napoleone" all'interno del terzo capitolo del documento puntato.

XML - XSL

- L'XML fornisce un complesso meccanismo per la rappresentazione dei documenti, l'eXtensible Stylesheet Language (XSL).
- l'XSL si basa sulle template rules, regole che consentono di intercettare una sequenza richiesta di tag all'interno del documento XML da formattare e di associarvi un opportuno oggetto di formattazione.
- L'XSL processor prende in ingresso il documento XML da rappresentare (source tree), vi applica le template rules e genera il documento con i soli oggetti di formattazione (output tree). Esso viene dato ad un formatter in cui viene dettagliatamente descritto come rappresentare ogni oggetto di formattazione. Il formatter deve essere incluso nei browser.

Semantica Interdocumentale

- Sebbene un documento sia un buon modo per specificare informazioni, un documento, ancorché espresso in formato XML, è poco adatto al Web che per sua natura è distribuito e decentralizzato e, quindi, informazioni su una particolare entità possono essere localizzate ovunque. Infatti, con XML è possibile descrivere adeguatamente i contenuti di un documento ma la sintassi XML non definisce alcun meccanismo esplicito per qualificare le relazioni tra documenti.
- In questo non è di aiuto neppure il meccanismo dei collegamenti ipertestuali reso popolare dall'HTML perché amorfo, cioè non prevede la possibilità di descrivere il legame definito.

Semantica Interdocumentale

- Ad esempio, sebbene in un documento (ad es. una pagina HTML) sia possibile parlare di un Signor Napolitano ed esprimere semanticamente questo con opportuni tag, è poi difficile capire se due documenti che parlano di un Signor Napolitano si riferiscono alla stessa persona con conseguente scarsa qualità dei risultati restituiti dai motore di ricerca.
- Nella migliore delle ipotesi sarebbe possibile dedurlo se, tra gli altri, vi fossero dati anagrafici semanticamente definiti e sufficientemente precisi (ad es. il Codice Fiscale) o collegamenti ipertestuali debitamente descritti che li collegano.
- Poiché, però, i diversi documenti sono redatti per scopi differenti, indipendentemente gli uni dagli altri e normalmente senza condividere un comune formato XML, informazioni utili quali l'indirizzo postale o la data di nascita finiscono per essere espresse in modo non uniforme, rendendo ardua ogni deduzione automatica.

Semantica Interdocumentale

- Per comprendere il problema della rappresentazione dell'informazione semantica dei documenti online, consideriamo un motore di ricerca: i motori di ricerca consentono di reperire pagine presenti in rete ed inerenti un certo argomento (per esempio "il linguaggio XML"), in quale modo può un sistema di questo tipo valutare che una certa pagina sarà di interesse per l'utente?
- Tale valutazione riguarda il contenuto della pagina, non il nome dell'autore o l'URL alla quale è disponibile.
- Una soluzione grezza consiste nello scorrere il testo contenuto in tutte le pagine di tutti i siti web del mondo e restituire l'elenco di quelle che contengono la parola di interesse.

Semantica Interdocumentale

- Questa soluzione non è molto raffinata, per una discreta varietà di motivi:
  - Il testo potrebbe contenere la scritta "questa pagina non riguarda il linguaggio XML"
  - Il testo potrebbe riguardare tutt'altro argomento e poi contenere una frase del tipo "Suggeriamo al lettore interessato di studiare anche il linguaggio XML"
  - Il testo potrebbe riguardare il linguaggio XML ma essere scritto in inglese, per cui si parlerebbe di "XML language"
  - Il testo potrebbe riguardare l'argomento di interesse ma approfondimento di aspetti diversi (XML per principianti, XML per la realizzazione di banche dati, XML e creazione di nuovi linguaggi)

Semantica Interdocumentale

- Il linguaggio HTML fornisce una soluzione primitiva al problema consentendo di esplicitare informazioni inerenti il documento intero nella sezione HEAD, utilizzando il tag META (che sta per meta-dato). I tag meta non vengono visualizzati a chi fruisce della pagina HTML come lettore bensì da taluni motori di ricerca per valutare l'appropriatezza di una pagina come risposta ad una certa domanda.
- In generale un tag meta ha la seguente forma:
- <meta NAME="nome" CONTENT="valore">
- Con questa indicazione l'autore di una pagina associa la stringa "valore" alla stringa "nome", dove "nome" indica una proprietà del documento e "valore" il valore di tale proprietà.

Semantica Interdocumentale

- Esempi di utilizzo di metatag sono:
- <head>
- <meta NAME="AUTHOR" CONTENT="C. Bianchi">
- <meta NAME="KEYWORDS" CONTENT="XML, programmazione web, query">
- <meta NAME="DESCRIPTION" CONTENT="introduzione all'XML">
- </head>
- L'idea è quindi trasferire nella sezione HEAD le meta-informazioni relative ad un documento, privilegiando il contenuto di tale sezione come informazione utile durante una ricerca.

Semantica Interdocumentale

- L'utilizzo del tag HTML <meta> presenta un primo limite, ossia non esistono convenzioni né sui nomi né sulla struttura del contenuto delle proprietà da definire, limitando fortemente l'utilizzo di sistemi automatici.
- Se un sistema non è in grado di identificare o distinguere nomi di proprietà non potrà, per esempio, distinguere il caso in cui le meta-informazioni specificano che il sito è stato scritto da "C. Bianchi" da quello in cui si dichiara che il sito riguarda la vita e le opere di "C. Bianchi"
- Riassumendo, una specifica semantica deve basarsi su di un linguaggio (o vocabolario) comune, condiviso.

Semantica Interdocumentale

- Il secondo limite dell'uso del tag <meta> è che non permette di esprimere legami semantici:
- supponiamo di essere in cerca di materiale riguardante XML. Al di là delle specifiche sintattiche che ci spiegano come deve essere scritta, per esempio, una DTD, per comprendere bene la portata di XML potrebbe essere utile accedere ad altri argomenti ad esso correlati: per esempio a una descrizione di SGML oppure a qualche panoramica riguardante la programmazione dei siti web (con relativo uso di linguaggi quali javascript o perl). È assai inverosimile che nella descrizione del contenuto semantico dei siti inerenti questi argomenti sia specificato anche XML, tuttavia esiste una correlazione con XML: se il meccanismo che utilizziamo ci consentisse di mantenere non solo informazioni del tipo proprietà-valore ma strutture più complesse (in altri termini non solo concetti ma anche relazioni fra concetti) il nostro meccanismo di ricerca potrebbe svolgere ricerche più intelligenti. Da questo problema nasce RDF