


CORSO I.F.T.S.
"TECNICHE PER LA PROGETTAZIONE E LA GESTIONE DI DATABASE"

Matricola 2014LA0033

DISPENSE DIDATTICHE

MODULO di "DATABASE SEMANTICI"

Ing. Simone Menabeni

Lezione del 15/09/2014

Xml, Rdf

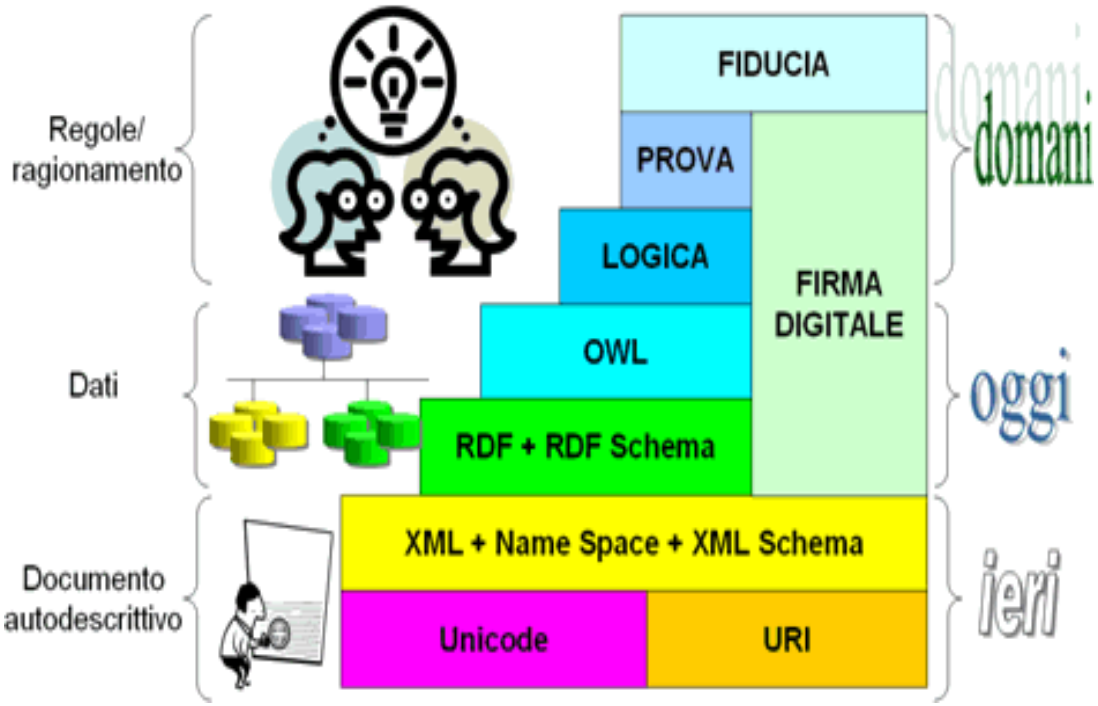
Ing. Simone Menabeni
Dipartimento di Ingegneria dell'informazione
Università di Firenze
simone.menabeni@unifi.it

DISIT Lab
http://www.disit.dinfo.unifi.it/

Extensible Markup Language (XML)

- Introduzione
- Classi e istanze
- Proprietà

Web Semantico: XML



- Massiccio utilizzo di XML e Xml Schema
- Fornisce al web semantico l'interoperabilità sintattica
- Nodi concettuali
  - RDF e OWL sono basati su XML ma hanno un DTD che ne limita l'espressività e aggiunge semantica ai singoli nodi

XML: cosa è

- XML: Extensible Markup Language:
  - è un meta-*linguaggio* che consente la creazione di linguaggi di mark-up
  - consente la rappresentazione di documenti e dati strutturati su supporto digitale
  - è uno dei più potenti e versatili sistemi per la creazione, archiviazione, preservazione e disseminazione di documenti digitali
  - …. ma è anche una famiglia di tecnologie complementari

XML: le origini

- XML è stato sviluppato dal World Wide Web Consortium (http://www.w3.org)
- Le specifiche sono state rilasciate come *W3C Recommendation* nel 1998 e aggiornate nel 2008
- 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 alla utilizzazione su World Wide Web…

  … ma ha assunto ormai un ruolo autonomo e una diffusione ben maggiore del suo progenitore
- XML è abbastanza generale per poter essere utilizzato nei più disparati contesti

XML: caratteristiche (1)

- XML è un metalinguaggio di mark-up, cioè un linguaggio che permette di definire sintatticamente altri linguaggi di mark-up.
- XML permette di esplicitare la struttura di un documento in modo formale mediante marcatori (mark-up) che vanno inclusi all'interno del testo
- A differenza di HTML, XML non ha tag predefiniti e non serve per definire pagine Web né per programmare
  - … serve esclusivamente per definire altri linguaggi
- In realtà, XML non è altro che un insieme standard di regole sintattiche per modellare la struttura di documenti e dati. Queste regole, dette ***specifiche***, definiscono le modalità secondo cui è possibile crearsi un proprio linguaggio di mark-up. Le specifiche ufficiali sono state definite dal W3C (World Wide Web Consortium, http://www.w3.org/XML)

Il concetto di metalinguaggio

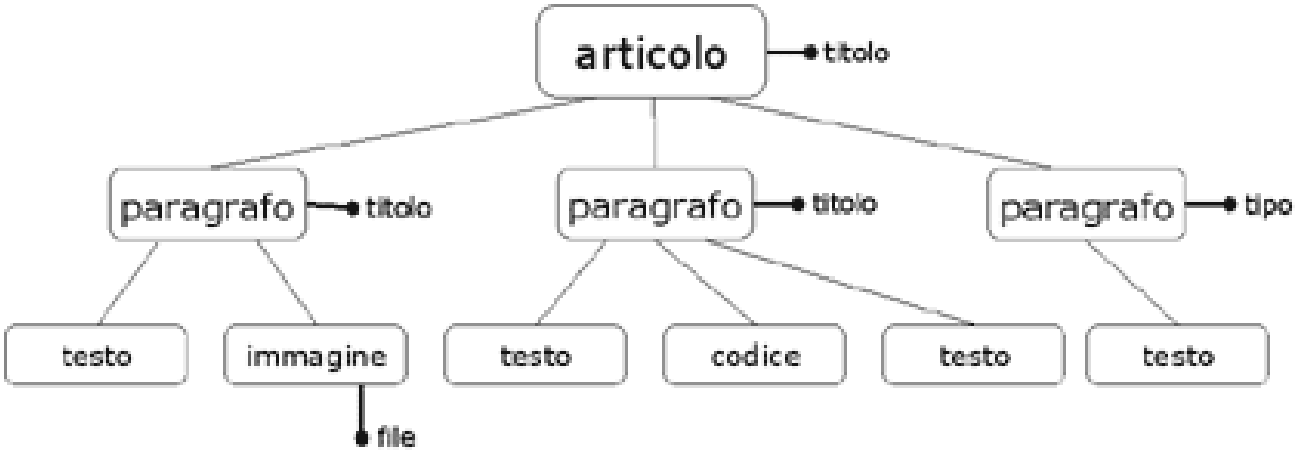
- XML è un **metalinguaggio**
  - XML definisce un insieme regole (meta)sintattiche, attraverso le quali è possibile descrivere formalmente un linguaggio di mark-up, detto “applicazione XML”
- ogni applicazione XML eredita un insieme di caratteristiche sintattiche comuni
- ogni applicazione XML a sua volta definisce una sintassi formale particolare
- ogni applicazione XML è dotata di una semantica specificata in modo non formale

XML: caratteristiche (2)

- XML è indipendente dal tipo di piattaforma hardware e software su cui viene utilizzato
- XML permette la rappresentazione di qualsiasi tipo di documento (e di struttura testuale) indipendentemente dalle finalità applicative
- XML è indipendente dai dispositivi di archiviazione e visualizzazione
  - un documento XML può essere archiviato su qualsiasi tipo di supporto digitale (attuale e… futuro!)
  - un documento XML può essere visualizzato su qualsiasi dispositivo di output

XML: caratteristiche (3)

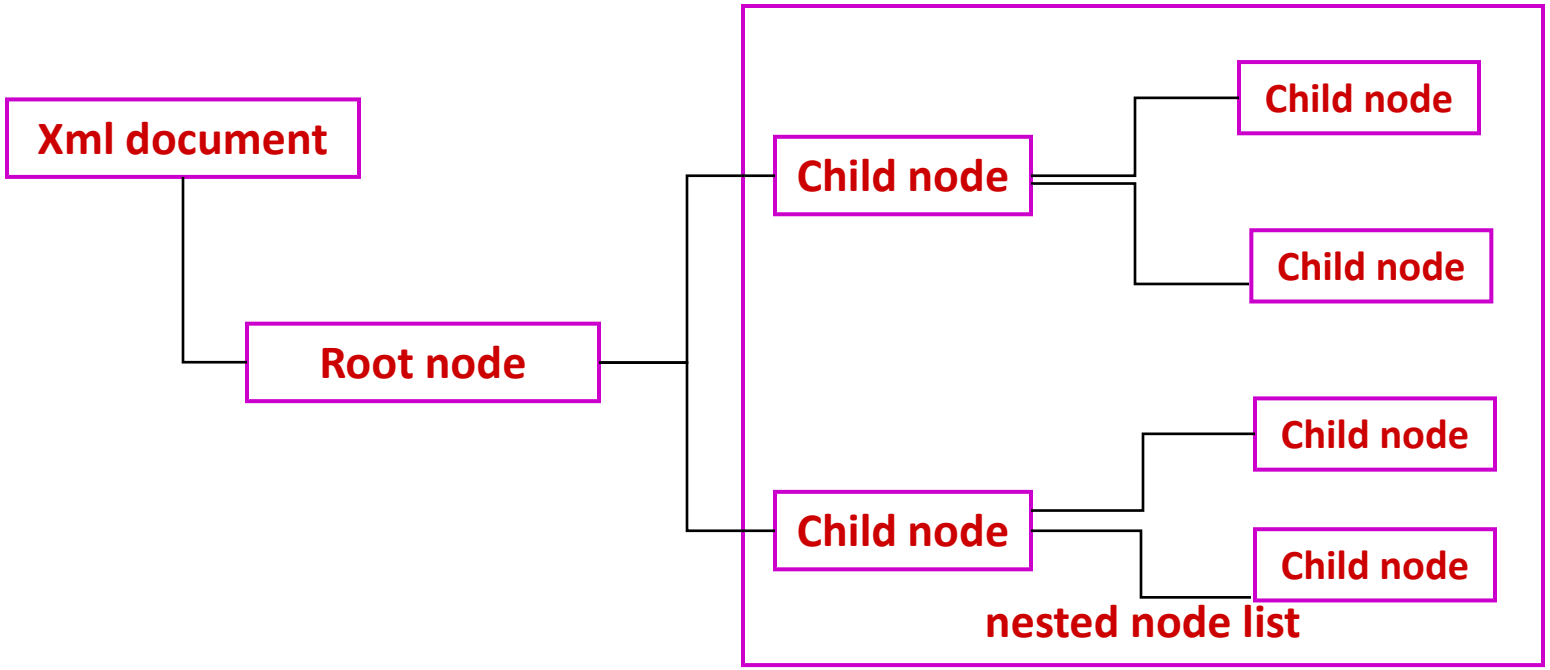
- XML adotta un formato di file di tipo testuale: sia il mark-up sia il testo, sono stringhe di caratteri
- XML si basa sul sistema di codifica dei caratteri ISO 10646/UNICODE
- Un documento XML è "leggibile" da un utente umano senza la mediazione di software specifico
- Concretamente, un documento XML è un file di testo che contiene una serie di tag, attributi e testo, secondo regole sintattiche ben definite.



XML secondo il W3C

- Deve permettere l'interscambio di dati attraverso la rete internet
- Deve supportare per una grande varietà di applicazioni
- Deve essere compatibile con SGML
- Deve essere di estrema semplicità di elaborazione per le macchine
- Deve avere caratteristiche opzionali prossime allo 0
- Deve essere leggibile dagli umani
- La progettazione deve essere semplice ed intuitiva
- La progettazione deve essere formale e concisa
- Deve essere facile da creare
- Deve esser indipendente dalla piattaforma

La struttura gerarchica ordinata



Esempio: articolo

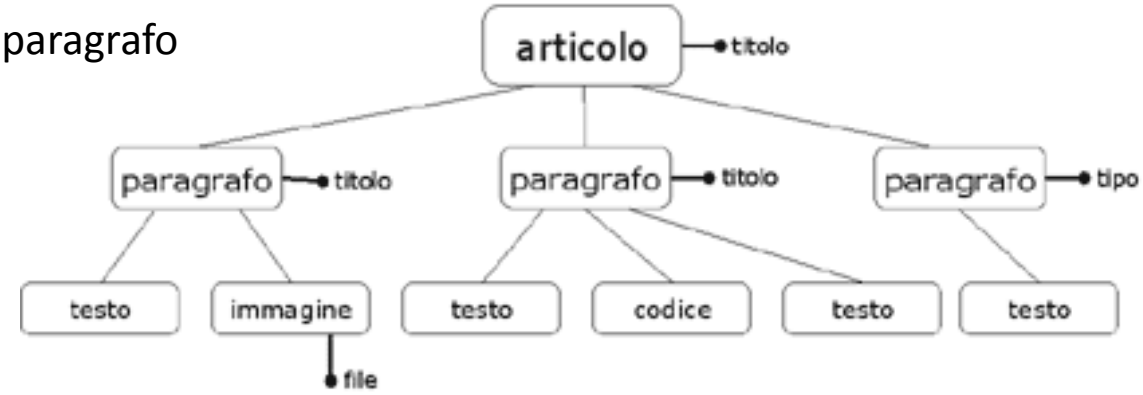
```
<?xml version="1.0" ?>
  <articolo titolo="Titolo dell'articolo">

    <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>
```



Aspetti di sintassi generale e vincoli

- Un documento XML è una stringa di caratteri UNICODE in codifica UTF-8 o UTF-16 (http://www.unicode.org/)
- I nomi di elementi, attributi e entità sono sensibili alla differenza tra maiuscolo e minuscolo
- Il mark-up è separato dal contenuto testuale mediante caratteri speciali: **< > &**
- Vincoli di buona formazione:
  - I caratteri speciali non possono comparire come contenuto testuale e devono essere eventualmente sostituiti mediante i riferimenti a entità: **< > & ' "**
  - Esiste un solo elemento radice
  - Tutti gli elementi non vuoti devono presentare sia il tag iniziale sia il tag finale
  - Tutti gli elementi devono essere correttamente annidati
  - Tutti i valori di attributo devono essere racchiusi tra apici doppi o singoli

La forma di un documento XML

- Ogni documento XML inizia con un prologo che contiene:
  - una XML declaration
  - eventualmente una Doctype declaration (la dichiarazione della DTD o del xml-schema a cui il documento si riferisce)
  - eventualmente una serie di processing instruction
- Forme di XML declaration:
  - <?xml version="1.0"?>
  - <?xml version="1.0“ encoding=“UTF-8”?>

Strutture XML: gli elementi (1)

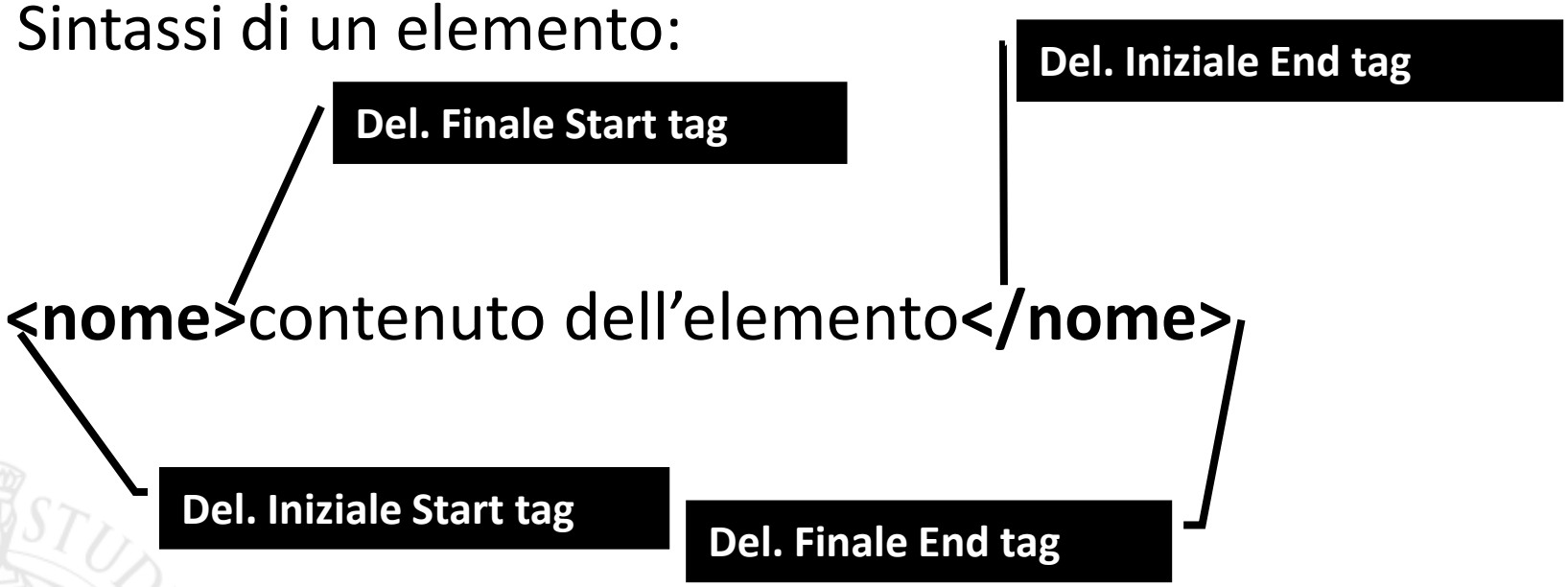
- I componenti strutturali di un documento sono denominati elementi (element)
- Ogni nodo dell'albero del documento è un (tipo di) elemento
- Ogni (tipo di) elemento è dotato di un nome (detto identificatore generico) che lo identifica
- Ciascun (tipo di) elemento rappresenta un componente logico del documento e può contenere altri (tipi di) elementi (sotto-elementi) o del testo
- L'organizzazione degli elementi segue un ordine gerarchico (ad albero) che prevede un elemento principale, chiamato root element o radice
- La radice contiene l'insieme degli altri elementi del documento
- E' possibile rappresentare graficamente la struttura di un documento XML tramite un albero, generalmente noto come ***document tree***

Strutture XML: gli elementi (2)

- Esiste uno e uno solo elemento radice (corrispondente al nodo radice dell'albero), che non è contenuto da nessun altro e che contiene direttamente o indirettamente tutti gli altri
- Ogni elemento, escluso l'elemento radice, deve essere contenuto da un solo elemento (elemento padre) e può contenere altri sottoelementi (elementi figli) e/o stringhe di caratteri
- Esiste un sottoinsieme di elementi che non contengono altri elementi e che possono:
  - essere vuoti
  - contenere esclusivamente stringhe di caratteri
- I (tipi di) elementi possono avere associate altre informazioni che ne descrivono le proprietà. Queste informazioni sono chiamate attributi

La codifica degli elementi (1)

- Nel documento ogni elemento non vuoto (contenente cioè altri elementi o caratteri) deve essere marcato da un **tag iniziale** e da un **tag finale**
- Ogni tag è costituito da caratteri delimitatori e dal nome dell'elemento
- Sintassi di un elemento:



La codifica degli elementi (2)

```
<text>
<div1>
<p>Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in una relazione troppo seria…</p>
<p>La sua famiglia? Una sola sorella non ingombrante né fisicamente né moralmente, piccola e pallida, di qualche anno più giovane di lui…</p>
…
</div1>
</text>
```

La codifica degli elementi (3)

SBAGLIATO!!!

Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in <emph>una relazione troppo seria…

La sua famiglia?</emph>Una sola sorella non ingombrante né fisicamente né moralmente, piccola e pallida, di qualche anno più giovane di lui…

CORRETTO!!!

Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in <emph>una relazione troppo seria… </emph>

 <emph>La sua famiglia?</emph> Una sola sorella non ingombrante né fisicamente né moralmente, piccola e pallida, di qualche anno più giovane di lui…

La codifica degli elementi (4)

- Gli elementi vuoti
  - o sono rappresentati da entrambi i tag
    - …<nome_elemento> </nome_elemento>…
  - o assumono la seguente forma
    - <nome_elemento/>
  - Esempio:
    - ****

La codifica degli attributi (1)

- Ogni elemento XML può avere uno o più attributi
- Un attributo ha un nome e un valore, che può assumere diverse tipologie
- Gli attributi devono essere associati agli elementi all'interno del tag iniziale dopo l'identificatore
  - <elemento attributo = "valore"> contenuto… </elemento>
- Altri eventuali attributi vanno collocati dopo il primo separati da uno o più spazi
- Non possono esservi più istanze dello stesso attributo per un elemento

La codifica degli attributi (2)

<text resp="Italo Svevo" n="Senilità">

Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in una relazione troppo seria…

La sua famiglia? Una sola sorella non ingombrante né fisicamente né moralmente, piccola e pallida, di qualche anno più giovane di lui…

<pb n="5"/>

…

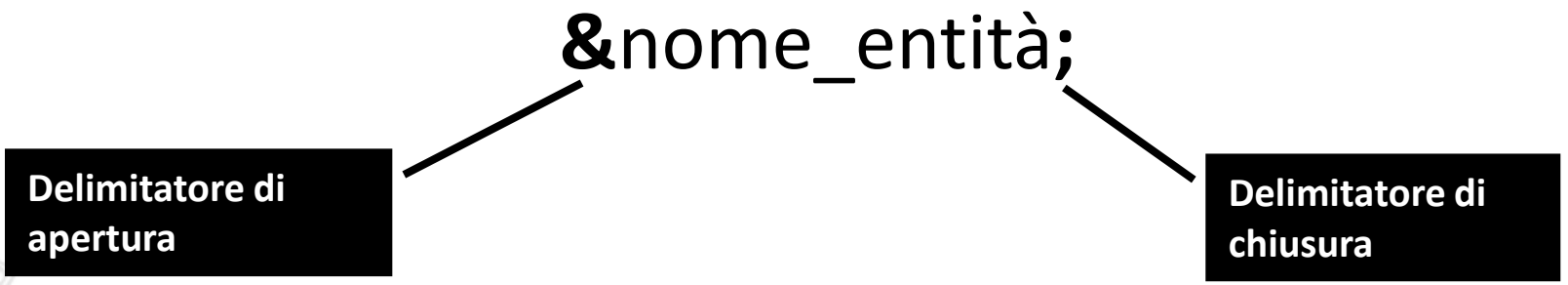
</text>

La codifica delle entità

- Per definire un'entità personalizzata si utilizza la dichiarazione: **<!ENTITY>**
- Il seguente esempio mostra la definizione di un'entità **&html;** che rappresenta un'abbreviazione per la stringa HyperText Markup Language:
  - <**!ENTITY** html "HyperText Markup Language">
- Grazie a questa dichiarazione possiamo utilizzare l'entità **&html;** al posto dell'intera stringa all'interno del documento XML che fa riferimento a questa grammatica

Il riferimento alle entità

- L'inclusione di una entità all'interno di un documento SGML si effettua mediante un **riferimento a entità** (entity reference)
- La sintassi di un riferimento, valida sia per entità esterne sia interne, è la seguente



Il riferimento alle entità

- In questi esempi i caratteri accentati sono stati sostituiti da riferimenti a entità carattere:

```
<p>La sua famiglia? Una sola sorella non ingombrante n&eacute; fisicamente n&eacute; moralmente, piccola e pallida, di qualche anno pi&ugrave; giovane di lui…</p>
```

```
<testo>
  il simbolo &lt; indica minore di
</testo>
```

XML: documenti ben formati (1)

- XML richiede un certo rigore sugli aspetti sintattici
- Ogni documento XML deve essere ben formato (well formed)
- Un documento, in generale, è ben formato se:
  - la sua struttura è implicita nel markup
  - rispetta i vincoli di buona formazione indicati nelle specifiche
- Più in dettaglio:
  - Ogni documento XML deve contenere un unico elemento di massimo livello (root) che contenga tutti gli altri elementi del documento. Le sole parti di XML che possono stare all'esterno di questo elemento sono i commenti e le direttive di elaborazione (per esempio, la dichiarazione della versione di XML)
  - Ogni elemento deve avere un tag di chiusura o, se vuoti, possono prevedere la forma abbreviata (/>)
  - Gli elementi devono essere opportunamente nidificati, cioè i tag di chiusura devono seguire l'ordine inverso dei rispettivi tag di apertura
  - XML fa distinzione tra maiuscole e minuscole, per cui i nomi dei tag e degli attributi devono coincidere nei tag di apertura e chiusura anche in relazione a questo aspetto
  - I valori degli attributi devono sempre essere racchiusi tra singoli o doppi apici
- Un documento XML ben formato non richiede la presenza di un DTD o xml-schema

XML: documenti ben formati (2)

- La scelta dei nomi dei tag deve seguire alcune regole:
  - Un tag può iniziare con un lettera o un underscore (_) e può contenere lettere, numeri, il punto, l'underscore (_) o il trattino (-). Non sono ammessi spazi o altri caratteri.
  - XML è sensibile all'uso di maiuscolo e minuscolo, quindi i tag <prova> e <Prova> sono considerati diversi.
- Per quanto riguarda il contenuto:
  - un documento XML può contenere potenzialmente qualsiasi carattere dell'alfabeto latino, cifre e punteggiatura. Normalmente vengono accettati come caratteri validi in un documento XML i primi 128 caratteri della codifica ASCII (lettere dell'alfabeto latino minuscole e maiuscole, cifre, segni di punteggiatura, ecc.)
  - Se un documento contiene caratteri che non rientrano tra questi (es.: lettere accentate, simboli di valuta, ecc.) è necessario specificare lo schema di codifica utilizzato tramite elementi speciali detti direttive di elaborazione o processing instruction
  - es: <?xml version="1.0" encoding="iso-8859-1"?>
- Le specifiche di XML prevedono esplicitamente la possibilità di utilizzare la codifica Unicode per rappresentare anche caratteri non latini, come ad esempio i caratteri greci, cirillici, gli ideogrammi cinesi e giapponesi

XML: documenti ben formati (3)

- Oltre alle direttive di elaborazione, in un documento XML possiamo trovare i commenti che seguono la stessa sintassi dell'HTML, sono cioè racchiusi tra le sequenze di caratteri <!-- e --> e possono trovarsi in qualsiasi punto del documento.
- Potrebbe essere necessario inserire in un documento XML dei caratteri particolari che potrebbero renderlo non ben formato
  - Ad esempio, se dobbiamo inserire del testo che contiene il simbolo < corriamo il rischio che possa venire interpretato come l'inizio di un nuovo tag. Esempio:

  <testo> il simbolo < indica minore di </testo>
- In questo caso, XML prevede l'uso delle entità che consentono di sostituire altri caratteri. Cinque entità sono predefinite e consentono l'uso di altrettanti caratteri riservati all'interno di un documento:
  - **&**; definisce il carattere &
  - **<** definisce il carattere <
  - **&gt**; definisce il carattere >
  - **"** definisce il carattere "
  - **'** definisce il carattere '
- Sfruttando le entità, l'Esempio precedente diventa:

  <testo> il simbolo < indica minore di </testo>

XML: documenti ben formati (4)

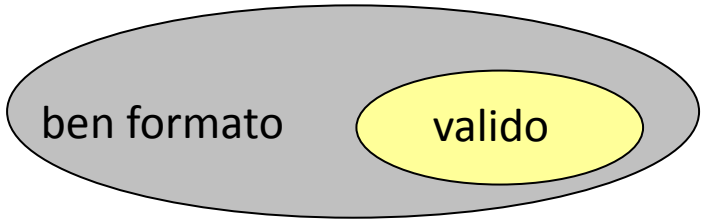
- In alcune situazioni gli elementi da sostituire con le entità possono essere molti, il che rischia di rendere illeggibile il testo (not human readable). Si consideri il caso in cui un blocco di testo illustri proprio del codice XML:
  - <codice>
    <libro>
    <capitolo>
    </capitolo>
    </libro>
    </codice>
- In questo caso, al posto di sostituire tutte le occorrenze dei simboli speciali con le corrispondenti entità è possibile utilizzare una **sezione CDATA** (un blocco di info che viene considerato sempre come testo)
- Per indicare una sezione CDATA è sufficiente racchiuderla tra le sequenze di caratteri **<![CDATA[e]]>:**
- <codice>
  <![CDATA[
  <libro>
  <capitolo>
  </capitolo>
  </libro>
  ]]>
  </codice>

XML: documenti validi (1)

- Oltre ad essere ben formato un documento XML può anche essere **valido**
- Per stabilire la validità di un documento xml è necessario definire una **grammatica (tipo di documento)** per il linguaggio di mark-up che abbiamo ideato, a cui poi si farà riferimento per la validazione del documento xml
- Una **grammatica** è un insieme di regole che indica quali vocaboli (elementi) possono essere utilizzati e con che struttura è possibile comporre frasi (documenti)

XML: documenti validi (2)

- Una **grammatica** definisce uno specifico linguaggio di mark-up => Se un documento XML rispetta le regole definite da una grammatica è detto **valido per un particolare** linguaggio
- Un documento ben formato può non essere valido rispetto ad una grammatica, mentre un documento valido è necessariamente ben formato
- Un documento valido per una grammatica può non essere valido per un'altra grammatica
- Una grammatica si definisce attraverso i due principali approcci:
  - **Dtd** - Document Type Definition
  - **XML Schema**



XML: documenti validi (3)

- Un documento è **valido se:**
  - si riferisce a una DTD esplicita mediante un Doctype declaration (o ad un xml-schema)
  - Soddisfa i vincoli sintattici del DTD o xml-schema (nome, sequenza, occorrenze ed attributi degli elementi)
- Il controllo di validità viene effettuato da un apposito programma detto parser
- I parser si possono dividere in due categorie:
  - parser **non validante**: verifica soltanto se un documento è ben formato
  - parser **validante**: oltre a verificare che un documento sia ben formato**,** verifica anche se è corretto rispetto ad una data grammatica (dtd o xmlschema)
- La maggior parte degli editor XML più recenti ha un parser integrato o si appoggia su parser esterni per effettuare la convalida dei documenti

Standard correlati a XML (1)

- La presentazione di un documento XML viene controllata da uno o più fogli di stile
- I linguaggi di stile utilizzabili con XML sono
  - **Extensible Stylesheet Language (XSL)**
  - **Cascading Style Sheet (CSS)**
- I CSS però hanno alcuni limiti:
  - Non possono cambiare l'ordine con il quale verranno visualizzati gli elementi di markup
  - Non posso effettuare operazioni logiche e computazionali
  - Non possono operare su più documenti contemporaneamente

```
<?xml-stylesheet type="text/css" href="stile.css" ?>
```

Standard correlati a XML

- Famiglia di raccomandazioni che permette di definire trasformazioni e presentazioni di documenti XML
- XSLT
  - Permette di definire delle regole per trasformare un documento XML
- XSL-FO
  - Permette di definire delle regole e delle specifiche di formattazione da applicare ad un documento XML
  - Formato adatto alla stampa, pdf,…
- XPATH
  - Permette di esprimere delle espressioni per indirizzare parti di un documento XML

XSLT

- Trasforma un documento XML in un nuovo documento
  - XML, HTML, PDF,….
  - Maggiore flessibilità rispetto ai CSS
- Si possono applicare diverse trasformazioni XSL allo stesso documento XML
  - Ogni trasformazione produce degli output differenti
- Netta separazione tra contenuto del documento e presentazione

XSLT

- Accesso al nodo radice
  - <xsl:template match="/">
- Accesso ad un nodo prefissato
  - <xsl:template match="nome_nodo">
- Accesso al contenuto di un elemento
  - <xsl:value-of select="nome_nodo">
- Accesso al testo contenuto in un nodo
  - <xsl:value-of select="text()"/>
- Accesso ad un determinato attributo di un nodo
  - <xsl:value-of select="@nome_attr"/>
- Accesso al commento di un nodo
  - <xsl:value-of select="comment()"/>

XSLT - esempio

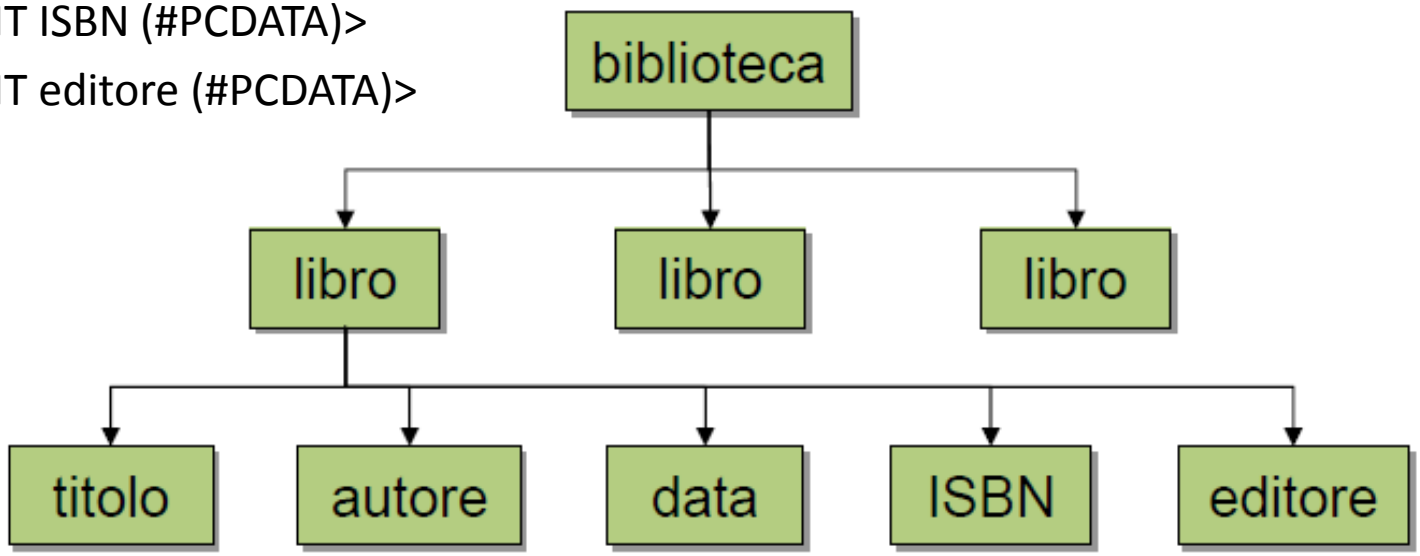
```
<?xml version="1.0" encoding="UTF-8"?><!-- Prologo XML -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/"> <html>
    <xsl:for-each select="//articolo">
      <b>Articolo : </b>
    <xsl:value-of select="@titolo"/>
      <br/>
    <xsl:for-each select="articolo/paragrafo">
      <b>- paragrafo: </b>
    <xsl:value-of select="titolo"/>
       -
    <xsl:value-of select="autore"/>
      <br/>
      </xsl:for-each>
      <br/>
      </xsl:for-each>
    </html></xsl:template></xsl:stylesheet>
```

DTD

- Grammatica che definisce quali tag sono validi in un documento XML
- Eredità di SGML
- Per descrivere un dtd è necessario utilizzare una sintassi particolare, diversa da quella descritta per i documenti xml
- Due modi possibili di specificare un DTD
  - All'interno del documento XML
  - All'esterno del documento XML
- Inconvenienti:
  - Non viene espresso mediante XML
  - Non permette di esprimere tutti i vincoli

DTD

```
<!DOCTYPE biblioteca [
    <!ELEMENT biblioteca (libro+)>
    <!ELEMENT libro (titolo, autore+, data , ISBN, editore)>
    <!ELEMENT titolo (#PCDATA)>
    <!ELEMENT autore (#PCDATA)>
    <!ELEMENT data (#PCDATA)>
    <!ELEMENT ISBN (#PCDATA)>
    <!ELEMENT editore (#PCDATA)>
]>
```



Collegare un dtd ad un file xml

- Tramite un Dtd è possibile definire la grammatica per un linguaggio di mark-up.
- Esistono **due modi** per indicare il Dtd cui un documento XML fa riferimento:
  - internamente al documento XML:

```
<?xml version="1.0">
 <!DOCTYPE articolo[
      ...Definizioni del Dtd...
 ]>
 <articolo>
 ...Contenuto del documento XML...
</articolo>
```

  - esternamente al documento XML:

```
<!DOCTYPE Report SYSTEM “Report.dtd”>

<!DOCTYPE Report PUBLIC “Report.dtd”>
```

Dai DTD agli xml-schema

- L'uso dei Dtd per definire la grammatica di un linguaggio di markup non sempre è del tutto soddisfacente
- A parte il fatto che la sintassi utilizzata per definire un Dtd non segue le regole stesse di XML, i Dtd non consentono di specificare:
  - un tipo di dato per il valore degli attributi
  - il numero minimo o massimo di occorrenze di un tag in un documento
  - altre caratteristiche che in determinati contesti consentirebbero di ottenere un controllo ancora più accurato sulla validità di un documento XML
- Queste limitazioni hanno spinto alla definizione di approcci alternativi per definire grammatiche per documenti XML. Tra questi approcci, il più noto è XML Schema

Come definire un xml-schema (1)

- **Analogamente ad un Dtd**, un XML Schema è una descrizione formale di una grammatica per un linguaggio di markup basato su XML
- Tuttavia, se abbiamo bisogno di un maggiore controllo sugli elementi che possono trovarsi all'interno di uno specifico tipo di documento XML, i Dtd non risultano più sufficienti
- **A differenza di un Dtd**, che utilizza una propria sintassi specifica, un XML Schema utilizza la stessa sintassi XML per definire la grammatica di un linguaggio di mark-up. Questo è invece indice dell'estrema flessibilità di XML.
- Un XML-Schema è un documento XML che descrive la grammatica di un linguaggio XML utilizzando un linguaggio di mark-up specifico

Come definire un xml-schema (2)

- In quanto documento XML, uno XML Schema ha un root element che contiene tutte le regole di definizione della grammatica
- La **struttura generale** di uno schema XML è la seguente:

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

    ... Definizione della grammatica ...

  </xs:schema>
```

- L'elemento root del documento è rappresentato dal tag **<xs:schema>**. Esso indica al parser che in questo documento saranno utilizzati dei tag definiti dal namespace standard del W3C
- I namespace rappresentano un meccanismo per identificare tag appartenenti ad una specifica grammatica. Nel nostro caso questi **tag speciali** sono caratterizzati dal prefisso **xs:**

Come definire un xml-schema (3)

- XML Schema prevede il tag **<xs:element>** per la definizione degli elementi utilizzabili in un documento XML, specificando nell'attributo name il nome del relativo tag.
- All'interno di ciascun tag **<xs:element>** si può indicare il tipo di dato dell'elemento e definire gli eventuali attributi
- Ad esempio, la seguente definizione specifica l'elemento testo che può contenere soltanto stringhe:
  - **<xs:element name="testo" type="xs:string" />**
- **NOTA:** Questa dichiarazione corrisponde alla seguente dichiarazione Dtd:
- <!ELEMENT testo (#PCDATA)>
- Per comprendere meglio ed apprezzare la potenza degli XML Schema, occorre analizzare nel dettaglio il concetto di tipo di dato. Esistono due categorie di tipi di dato: **semplici e complessi**

XML-schema: tipo di dato semplice

- XML Schema introduce il concetto di tipo di dato semplice per definire gli elementi che non possono contenere altri elementi e non prevedono attributi.
- Si possono usare tipi di dato semplici predefiniti oppure è possibile personalizzarli.
- Alcuni tipi di dato predefiniti sono riportati nella tabella

EX: <xs:element name="quantita" type="xs:integer" />

<quantita>123</quantita> …OK

<quantita>uno</quantita> ….NO

xs:string	Stringa di caratteri
xs:integer	Numero intero
xs:decimal	Numero decimale
xs:boolean	Valore booleano
xs:date	Data
xs:time	Ora
xs:uriReference	URL

XML-schema: tipo di dato semplice personalizzato

- Attraverso XML-Schema è possibile definire tipi di dato semplici personalizzati come derivazione da quelli predefiniti
- Se, ad esempio, si ha bisogno di limitare il valore che può essere assegnato all'elemento <quantita>, è possibile definirlo nel seguente modo:
  -
```
<xs:element name="quantita" >
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="100" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
- In questo caso, la dichiarazione indica che l'elemento <quantita>:
  - è di tipo semplice
  - prevede una restrizione sul tipo di dato intero predefinito accettando valori compresi tra 1 e 100

XML-schema: tipo di dato complesso

- I **tipi di dato complessi** si riferiscono ad elementi che possono contenere altri elementi e possono avere attributi
- Definire un elemento di tipo complesso corrisponde a definire la relativa struttura
- Lo schema generale per la definizione di un **elemento di tipo complesso** è il seguente:

```
<xs:element name="NOME_ELEMENTO">
  <xs:complexType>
    ... Definizione del tipo complesso ...
    ... Definizione degli attributi ...
  </xs:complexType>
</xs:element>
```

XML-schema: def. tipo di dato complesso (1)

- I tipi di dato complesso sono elementi che ne possono contenere altri.
- E` possibile definire la sequenza di elementi che possono stare al suo interno utilizzando uno dei seguenti **costruttori di tipi complessi**:
  - **<xs:sequence>** Consente di definire una sequenza ordinata di sottoelementi
  - **<xs:choice>** Consente di definire un elenco di sottoelementi alternativi
  - **<xs:all>** Consente di definire una sequenza non ordinata di sottoelementi

File.xsd

```
<xs:element name="articolo">
<xs:complexType>
 <xs:sequence>
       <xs:element name="paragrafo"/>
       <xs:element name="testo"/>
 </xs:sequence>
</xs:complexType>
</xs:element>


<xs:complexType>
<xs:choice>
       <xs:element name="paragrafo"/>
       <xs:element name="testo"/>
 </ xs:choice>
</xs:complexType>
```

File.xml

```
[…]
<articolo>
  <paragrafo>
      Questo è il paragrafo 1.. Introduzione…
  </paragrafo>
  <testo>
    Testo effettivo contenuto nel paragrafo
  </testo>
</articolo>

[…]
<articolo>
  <testo>
    Testo effettivo contenuto nell`articolo
  </testo>
</articolo>
[…]
```

XML-schema: def. tipo di dato complesso (2)

- Per ciascuno dei costruttori visti (**sequence, choice, all**) e per ciascun elemento è possibile definire il numero di occorrenze previste utilizzando gli attributi **minOccurs** e **maxOccurs**.
- Esempio: se l'elemento *testo* può essere presente una o infinite volte all'interno di un paragrafo possiamo esprimere questa condizione nel seguente modo:
  -
```
<xs:element name="paragrafo">
  <xs:complexType>
    <xs:element name="testo" minOccurs="1"
    maxOccurs="unbounded"/>
  </xs:complexType>
</xs:element>
```
- In questo caso il valore **unbounded** indica che non è stabilito un massimo numero di elementi testo che possono stare all'interno di un paragrafo

XML-schema: def. tipo di dato complesso (3)

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XML-schema: def. degli attributi del tipo di dato complesso

- La definizione degli attributi è basata sull'uso del tag **<xs:attribute>**, come nel seguente esempio:
  - <xs:attribute name="titolo" type="xs:string" use="required" />
- dove l'attributo use serve per specificare alcune caratteristiche come la presenza obbligatoria (**required**) o un valore predefinito (**default**) in combinazione con l'attributo value.
- Si noti che: se non si specifica esplicitamente l'obbligatorietà dell'attributo, esso è considerato opzionale

XML-schema: def. modulare degli elementi (1)

- XML Schema prevede di rendere modulare la definizione della struttura di un documento XML con la dichiarazione di tipi e elementi
- Questo contribuisce a fornire una **struttura modulare** allo schema, più ordinata, più comprensibile e semplice da modificare: un XML Schema diventa una sequenza di dichiarazioni di tipi ed elementi
- Esempio:

```
<xs:complexType name="nome_tipo">
...
</xs:complexType>
```

- Il riferimento ad una dichiarazione di tipo viene fatta come se fosse un tipo predefinito, come mostrato nel seguente esempio:
- <xs:element name="nome_elemento" type="nome_tipo" />

XML-schema: def. modulare degli elementi (2)

- La possibilità di dichiarare elementi e tipi di dato implica l'esistenza di un **ambito di visibilità**
- I componenti di uno schema dichiarati al livello massimo, cioè come sotto elementi di root, sono dichiarati a livello globale e possono essere utilizzati nel resto dello schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/... ">
    <xs:complexType name="paragrafoType">
            […]
    </xs:complexType>

    <xs:element name="articolo">
      <xs:complexType>
      <xs:sequence>
          <xs:element name="paragrafo"
                type="paragrafoType"
                maxOccurs="unbounded"/>
        </xs:sequence>
          <xs:attribute name="titolo" type="xs:string"
                use="required"/>
      </xs:complexType>
    </xs:element>
    </xs:schema>
```

Esempio: articolo

```
<?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>
       <![CDATA[
            <libro>
                  <capitolo>
                  </capitolo>
            </libro>
       ]]>
      </codice>
      <testo>
        Altro blocco di testo
      </testo>
    </paragrafo>

    <paragrafo tipo="bibliografia">
      <testo>
        Riferimento ad un articolo
      </testo>
    </paragrafo>

  </articolo>
```

XML-schema: namespace (1)

- Una delle caratteristiche auspicabili nella creazione di un nuovo linguaggio è la possibilità di integrare elementi derivanti da grammatiche diverse (definite in xml-schema differenti) in modo da **riutilizzare parti di grammatiche** già definite

- Tuttavia la **composizione di linguaggi** pone almeno due tipi di problemi:
  - un documento che usa due grammatiche presenta il **problema della validazione**: a quale schema si deve fare riferimento per validare un documento XML "ibrido"?
  - due linguaggi potrebbero avere tag ed attributi con lo stesso nome, anche se utilizzabili in contesti diversi: come fare a risolvere questo tipo di **ambiguità**?

- La soluzione a questi problemi deriva dai **namespace**. Un namespace è un insieme di nomi di elementi e nomi di attributi identificati univocamente da un identificatore

XML-schema: namespace (2)

- **L'identificatore univoco** individua l'insieme dei nomi distinguendoli da eventuali omonimie in altri namespace

- Il concetto non è nuovo nell'informatica. Esempio: **definizione** dei nomi dei campi in una tabella di un database. Non è possibile avere campi con lo stesso nome all'interno di una tabella, ma è possibile avere gli stessi nomi in tabelle diverse. In questo modo si risolve l'ambiguità tra due campi omonimi facendoli precedere dal nome della tabella (il namespace)

- Se in un documento XML si utilizzano **elementi definiti in schemi diversi** abbiamo bisogno di un meccanismo che permetta di identificare ciascun namespace e il relativo XML Schema che lo definisce

XML-schema: sintassi dei namespace (1)

- In un documento XML si fa riferimento ad un namespace utilizzando un attributo speciale (**xmlns**) associato al root element:
  - <articolo **xmlns**="http://www.dominio.it/xml/articolo">

- Questo indica che l'elemento articolo ed i suoi sottoelementi usano i nomi definiti nel namespace identificato dall'identificatore http://www.dominio.it/xml/articolo

- L'identificatore di un namespace può essere rappresentato da una qualsiasi stringa **univoca**. Solitamente si usa **un URI** (Uniform Resource Identifier)

XML-schema: sintassi dei namespace (2)

- Per mettere in relazione un namespace con il relativo XML Schema occorre dichiararlo nel root element:
  - <articolo
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:art="http://www.dominio.it/xml/articolo"
    xmlns:bibl="http://www.dominio.it/xml/bibliografia"
    xsi:schemaLocation="http://www.dominio.it/xml/articolo
    articolo.xsd"
    xsi:schemaLocation="http://www.dominio.it/xml/bibliografia
    bibliografia.xsd“>
  - dove:
    - **xmlns:xsi** specifica la modalità con cui viene indicato il riferimento allo schema
    - **xsi:schemaLocation** indica il namespace ed il file in cui è definito il relativo XML Schema separati da uno spazio
- E’ possibile **combinare più namespace** facendo in modo che ciascun elemento utilizzato faccia riferimento al proprio namespace
- Si noti che quando si fa **riferimento ad un namespace**, questo riferimento vale per l'elemento corrente e per tutti gli elementi contenuti, a meno che non venga specificato un diverso namespace

XML - Applicazioni

- L' XML è usato principalmente per l'interscambio di dati attraverso Internet.
- Uno degli obiettivi degli sviluppatori dell'XML è: "…dovrebbe essere facile scrivere programmi che elaborino i documenti scritti in XML".
- Nonostante questo, la specifica XML non contiene informazioni su come i programmatori dovrebbero fare tale procedimento. L'XML Infoset fornisce un vocabolario per riferirsi ai costrutti interni di un documento XML, ma non fornisce una guida su come accedere direttamente a queste informazioni.
- A tal fine sono state sviluppate, e sono quindi utilizzate, un certo numero di API per elaborare (leggere e modificare) i documenti scritti in XML. Talune di queste sono state anche standardizzate. Possono essere divise in 4 categorie:
  - Stream-oriented APIs: accessibili da un linguaggio di programmazione quale SAX;
  - Tree-transversal APIs: accessibili da un linguaggio di programmazione quale DOM;
  - XML data binding: che forniscono una "traduzione" automatica tra documento XML e linguaggio di programmazione ad oggetti;
  - Declarative transformation languages: quali, ad esempio, XSLT e Xquery.

XML - Applicazioni

- SAX (Simple API for XML): questa interfaccia permette di leggere in maniera “sequenziale” e di modificare i documenti XML. Attraverso di essa è possibile implementare degli XML parser specifici. E’, inoltre, “event-based” (basata sugli eventi) e reagisce quindi agli eventi di parsing facendo rapporto all’applicazione, ma lasciando al programmatore la gestione dell’evento di parsing.
- DOM (Document Object Model): è più semplice da utilizzare rispetto a SAX. Esso costruisce, partendo dal file XML, un albero dove ogni nodo dell’albero corrisponde ad un elemento del file: per questo motivo fa parte della categoria tree-transversal. Permette la lettura “trasversale” del contenuto del documento.
- Data Binding: qui i dati XML sono resi disponibili come una gerarchia di “custom”; è fortemente tipizzato in classi, al contrario degli oggetti generici creati dal DOM. Questo approccio semplifica lo sviluppo del codice e, talvolta, permette di risolvere problemi durante la compilazione e non durante l’esecuzione del codice.

Esempi XML

- Per verificare che un file XML sia ben formato
  - Scaricare il plugin per firefox: OpenXMLVievew
  - Usare direttamente Internet Explorer
- Per la validazione
  - Xml-spy (http://www.altova.com/xml-editor/) è proprietario ma esiste una versione free per 30 gg
  - Online schema validator
    - http://tools.decisionsoft.com/schemaValidate/

RDF: Resource Description Language

Introduzione

Classi e istanze

Proprietà

RDF

- XML serve per:
  - strutturare l'informazione
- XMLS serve per:
  - definire come formare "messaggi" corretti, ovvero per descrivere in modo formale una grammatica per un linguaggio di markup basato su XML
- RDF consente:
  - l'interoperabilità ***tra applicazioni*** permette di esprimere relazioni tra istanze

RDF

- Lo sviluppo di RDF nell'ambito del Web Semantico è stato condotto per consentire i seguenti utilizzi:
  - Web Metadata: fornire informazioni su risorse Web e i sistemi che utilizzano (restrizioni per la privacy, descrizioni di capacità, valutazione del contenuto);
  - Applicazioni che richiedono modelli dell'informazione aperti invece che chiusi (annotazione di risorse Web, descrizione di processi organizzativi, attività di avvicendamento);
  - Consentire che le applicazioni possano lavorare in interdipendenza, rendendo possibile la combinazione di nuovi dati per creare nuova informazione;
  - Elaborazione automatica delle informazioni per gli agenti software; il Web dovrebbe trasformarsi da un ambiente caratterizzato esclusivamente da informazione human-readable ad un network di processi di cooperazione attraverso il linguaggio RDF.

RDF

- La specifica principale dell'RDF è costituita da due componenti: l'RDF Model and Syntax che è una raccomandazione pubblicata nel 1999 e l'RDF Schema che è una raccomandazione candidata uscita nel 2000.
- **L'RDF Model and Syntax** espone la struttura del modello RDF e ne descrive una possibile sintassi. Si basa su tre principi chiave:
  1. Qualunque cosa può essere identificata da un URI (Universal Resource Identifier);
  2. *"The least power"*: utilizzare il linguaggio meno espressivo per definire qualunque cosa;
  3. Qualunque cosa può dire qualunque cosa su qualunque cosa.
- **L'RDF Schema** espone la sintassi per definire schemi e vocabolari per i metadati.

RDF

- Ogni documento RDF è anche un documento XML
- Qualunque cosa che è descritta dall'RDF è detta "**risorsa**". Principalmente una risorsa è reperibile sul Web, ma l'RDF può descrivere anche risorse che non si trovano direttamente sul Web. Ogni risorsa è identificata da un URI (Uniform Resource Identifier).
- Il modello dei dati dell'RDF è formato da risorse, proprietà e valori.

Risorse

- Tutte le cose descritte con espressioni RDF vengono dette **Risorse.**
- Una risorsa può essere:
  - un'intera pagina Web
  - una parte di una pagina Web
  - un'intera collezione di pagine (un sito Web)
  - un oggetto non direttamente accessibile via Web (un libro stampato)
- Le risorse sono sempre definite da URI
- Qualsiasi cosa può avere associato un URI

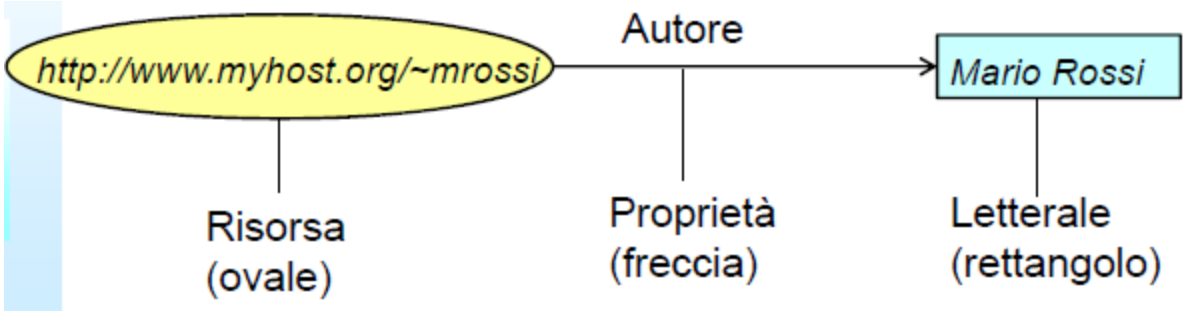
Proprietà

- Una **Proprietà** viene usata per descrivere una **risorsa** e può consistere in:
  - un aspetto specifico
  - una caratteristica
  - un attributo
  - una relazione

Asserzioni

- Una **Asserzione** RDF è composta da:
  - Soggetto: una determinata risorsa
  - Predicato: una proprietà della risorsa
  - Oggetto: un valore relativo ad una proprietà
    - una risorsa definita da un URI
    - una stringa di caratteri o altro tipo di dato primitivo definito da XML

Rappresentazione Grafica



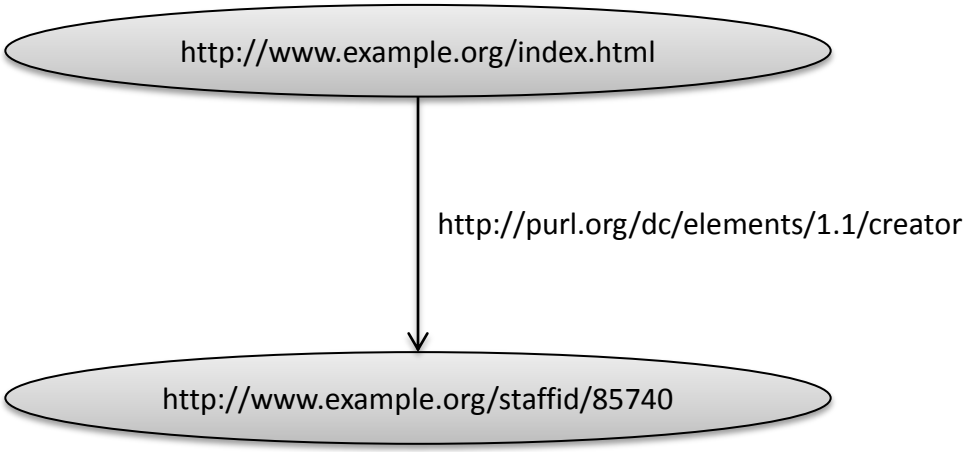
- i nodi che sono URI sono rappresentati da elissi, mentre i nodi che rappresentano literal sono disegnati con dei rettangoli. I predicati sono indicati con archi e frecce che ne indicano il verso.

Serializzazione

- Per rappresentare e memorizzare gli statement, l'RDF utilizza delle serializzazioni. Le principali sono:
  - **XML/RDF**: è un formato basato sull'XML, più spesso indicato con il solo termine RDF, visto che è stata introdotto assieme ad altre specifiche che definiscono l'RDF: si distingue, tuttavia, questo modello da quello astratto dell'RDF
  - **Notation 3**: abbreviata in N3, è una serializzazione di modelli RDF, non basata sull'XML. E' stata introdotta sempre dal W3C e risulta molto semplice da scrivere a mano. Si basa sull'uso di una notazione tabellare. In essa si serializza il grafo descrivendo una risorsa per volta con tutte le sue proprietà.
  - **Turtle** (Terse RDF Triple Language): è anch'essa una serializzazione di grafi RDF. E' un sottoinsieme di N3, proposta da Davie Beckett. Essa è limitata solo al modello dei grafi dell'RDF e non ha nessuno "stato ufficiale" con nessuna organizzazione di standard.
  - **N-Triples**: è un formato per memorizzare e trasmettere dati. In questa serializzazione il grafo è rappresentato come un insieme di triple nella forma soggetto - predicato - oggetto. Essa è un sottoinsieme di Turtle. E' stata progettata per essere un formato più semplice rispetto all'N3 e al Turtle stesso.

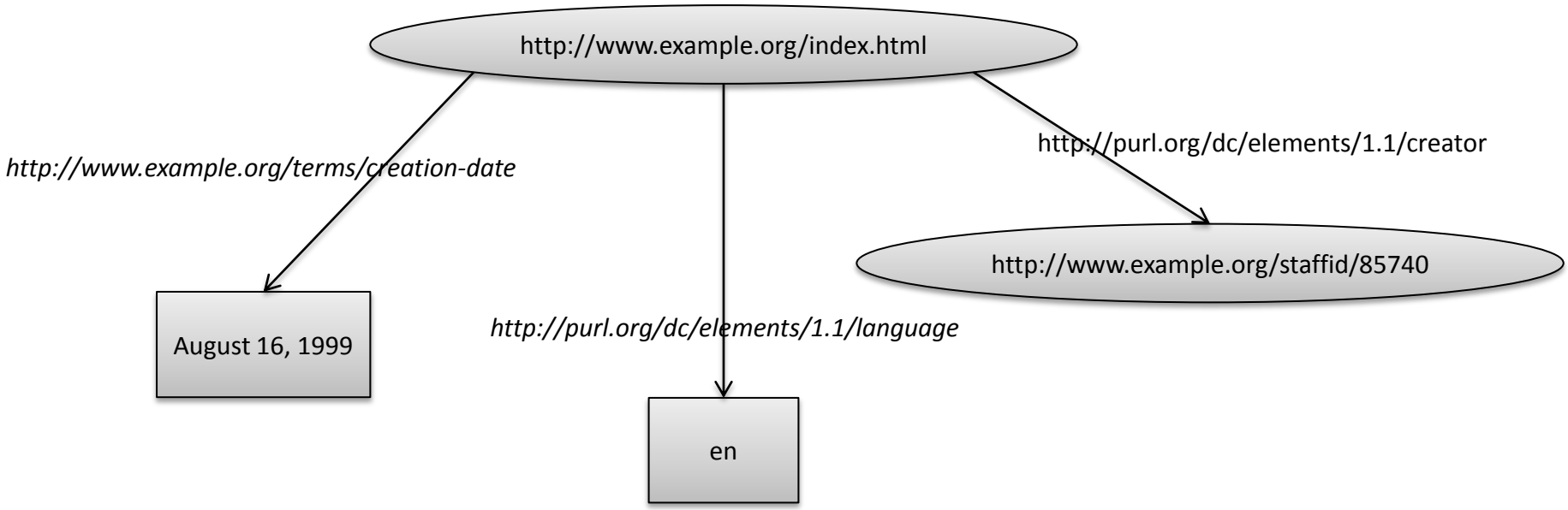
Esempio

- Si consideri l'affermazione nella quale si dice che una determinata pagina web, riferita dall'indirizzo web http://www.example.org/index.html , è stata creata da una persona di nome John Smith.
- Questa affermazione può essere rappresentata in uno statement RDF dove:
  - il soggetto è http://www.example.org/index.html
  - il predicato è http://purl.org/dc/elements/1.1/creator
  - l'oggetto è http://www.example.org/staffid/85740



Esempio

- Se l'affermazione è più complessa e comprende anche la data di creazione del documento (16 Agosto 1999) e la lingua (inglese = en) in cui il documento è stato scritto, gli statement diventano tre.



dove gli oggetti delle nuove affermazioni non sono URI, bensì valori costanti (detti "literal"), rappresentati da stringhe di caratteri che rappresentano certi tipi di proprietà dei valori.

N-Triples

- Talvolta, la notazione grafica non è la più felice per rappresentare le dichiarazioni RDF. Si può usare delle notazioni alternative per scrivere le triple. Una di queste è la N-Triple, nella quale ciascuna dichiarazione del grafo è scritta come una semplice tripla di soggetto - predicato - oggetto nell'ordine.

<http://www.example.org/index.html>
<http://purl.org/dc/elements/1.1/creator>
<http://www.example.org/staffid/85740> .
<http://www.example.org/index.html>
<http://www.example.org/terms/creation-date>
"August 16, 1999" .
<http://www.example.org/index.html>
<http://purl.org/dc/elements/1.1/language>
"en" .

N-Triples

- La notazione completa delle triple richiede che gli URI reference siano scritti per esteso e racchiusi tra parentesi angolari. Questo si trasforma spesso in una riga di caratteri troppo lunga che, talvolta, esce dalla visualizzazione della pagina.
- Esiste una scrittura abbreviata che è utilizzata in alcune specifiche RDF. Questa forma abbreviata costituisce un "XML qualified name" (o Qname) scritto senza parentesi angolari, come abbreviazione di un URI reference completo.
- Un Qname contiene un "prefix", che è stato assegnato ad un namespace URI, seguito dai due punti e poi da un "local name". Esistono diversi prefissi Qname che sono molto utilizzati; sono definiti come segue:
  - *prefix rdf: , namespace URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#*
  - *prefix rdfs: , namespace URI: http://www.w3.org/2000/01/rdf-schema#*
  - *prefix dc: , namespace URI: http://purl.org/dc/elements/1.1/*
  - *prefix owl: , namespace URI: http://www.w3.org/2002/07/owl#*
  - *prefix ex: , namespace URI: http://www.example.org/*
  - *prefix xsd: , namespace URI: http://www.w3.org/2001/XMLSchema#*

N-triples

- Con queste shorthand (abbreviazioni), l'esempio di prima può essere scritto come:

ex:index.html	*dc:creator*	*exstaff:85740*
ex:index.html	*exterms:creation-date*	*"August 16, 1999"*
ex:index.html	*dc:language*	*"en" .*

- dove sono stato utilizzati altri prefissi:
  - *prefix exterms: , namespace URI: http://www.example.org/terms/ (per i termini usati da un'organizzazione dell'esempio),*
  - *prefix exstaff: , namespace URI: http://www.example.org/staffid/ (per gli identificatori delle persone dello staff dell'organizzazione dell'esempio).*

RDF/XML

- L' "RDF/XML" è una sintassi, basata sull'XML. Diversamente dalle triple, che vengono intese come una notazione abbreviativa, l'RDF/XML è la normativa sintattica per scrivere le dichiarazioni RDF.
- Prendiamo per esempio la seguente tripla scritta in formato N-Triples:

  ex:index.html exterms:creation-date "August 16, 1999"

- La sintassi XML/RDF corrispondente è:

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdfsyntax-ns#"
3. xmlns:exterms="http://www.example.org/terms/">
4. <rdf:Description rdf:about="http://www.example.org/index.html">
5. <exterms:creation-date>August 16,1999</exterms:creation-date>
6. </rdf:Description>
7. </rdf:RDF>
```

Vocabolari

- L'RDF usa l'approccio dei namespace per definire il proprio vocabolario dei termini che hanno un significato speciale nell'RDF stesso. Gli URIref in questo vocabolario RDF iniziano tutti con http://www.w3.org/1999/02/22-rdf-syntax-ns# , convenzionalmente associato con il prefisso Qname rdf: .
- L' "RDF Vocabulary Description Language" definisce un insieme aggiuntivo di termini che hanno URIref che iniziano con http://www.w3.org/2000/01/rdfschema#, ed è convenzionalmente associato con il prefisso Qname rdfs: .
- Gli URIref di diversi vocabolari possono essere liberamente mescolati in un grafo RDF.

RDF Schema

- L' "RDF Schema", talvolta abbreviato in RDFS o RDF-S, è un linguaggio estensibile per la rappresentazione della conoscenza, che fornisce gli elementi di base per la descrizione dei vocabolari RDF, che sono a loro volta progettati per strutturare le risorse RDF.
- L'RDF Schema non fornisce un vocabolario di classi specifiche per le applicazioni, quali ex:Person e di proprietà quali ex:author . Esso fornisce, invece, i mezzi necessari per descrivere tali classi e proprietà e per indicare quali classi e proprietà ci si aspetta che possano essere utilizzate assieme.
- L'RDF Schema permette alle risorse di esse definite come istanze di una o più classi. Inoltre, le classi possono essere organizzate in modo gerarchico.

Esempio

- Si supponga che un'organizzazione, chiamata "example.org" voglia utilizzare l'RDF per fornire informazioni sui diversi tipi di veicoli a motore (motor vehicle). Nell'RDF Schema, example.org necessiterà di una classe per rappresentare la categoria di quelle cose che sono veicoli a motore. Le risorse che appartengono a una classe sono chiamate istanze. L'organizzazione intende per istanze di questa classe essere quelle risorse che sono dei veicoli a motore.

Classi

- ***rdfs:Resource*** Tutto ciò che viene descritto in RDF è detto risorsa. Ogni risorsa è istanza della classe rdfs:Resource.
  - ***rdfs:Literal*** Sottoclasse di rdfs:Resource, rappresenta un letterale (stringa di testo)
  - ***rdf:Property*** Sottoclasse di rdfs:Resource. Rappresenta le proprietà
- ***rdfs:Class*** Quando viene definita una nuova classe, la risorsa che la rappresenta deve avere la proprietà rdf:type impostata a rdfs:Class
  - ***rdfs:subClassOf*** Specifica la relazione di ereditarietà fra classi. Questa proprietà può essere assegnata solo a istanze di rdfs:Class. Una classe può essere sottoclasse di una o più classi (concetto di ereditarietà multipla)

Esempio - Classi

- Nell'RDF Schema una classe è qualsiasi risorsa che ha una proprietà rdf:type il cui valore è la risorsa rdfs:Class.
- Così la classe dei motoveicoli sarà descritta assegnando la classe ad un URIref, chiamato ex:MotorVehicle e descrivendo quella risorsa con una proprietà rdf:type il cui valore è la risorsa rdfs:Class .

ex:MotorVehicle rdf:type rdfs:Class

- si è utilizzato ex: al posto dell'URIref http://www.example.org/schemas/vehicles che è usato come prefisso per gli URIref dal vocabolario example.org

Esempio - Classi

- La risorsa exthings:companyCar potrebbe essere descritta come un veicolo a motore tramite lo statement RDF:

  exthings:companyCar rdf:type ex:MotorVehicle

- È possibile introdurre classi aggiuntive che rappresentano vari tipi speciali di veicoli a motore.

  ex:Van rdf:type rdfs:Class .
  ex:Truck rdf:type rdfs:Class

Esempio - Classi

- Una proprietà importante predefinita nell'RDF Schema è quella di sottoclasse. In questo caso si può esplicitare che le due classi sono dei tipi speciali di veicoli a motore. Ciò si specifica con lo statement RDF:

  ex:Van rdfs:subClassOf ex:MotorVehicle
  ex:Truck rdfs:subClassOf ex:MotorVehicle

- Il significato della relazione rdfs:subClassOf è che ogni istanza della classe ex:Van è anche una istanza della classe ex:MotorVehicle.

rdfs:subClassOf

- La proprietà rdfs:subClassOf è transitiva. Nell'esempio dati gli statement:

ex:Van rdfs:subClassOf ex:MotorVehicle
ex:MiniVan rdfs:subClassOf ex:Van

- l'RDF Schema definisce ex:MiniVan essere anche una sottoclasse di ex:MotorVehicle .
- Una classe può essere una sottoclasse di più classi. L'RDF Schema definisce tutte le classi come sottoclasse della classe rdfs:Resource . Questo perché le istanze appartenenti a tutte le classi sono risorse.

Esempio

- L'esempio presentato in precedenza si può riassumere nelle triple:

ex:MotorVehicle rdf:type rdfs:Class .
ex:PassengerVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
ex:MiniVan rdf:type rdfs:Class .
ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle .
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .
ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle

Esempio

- L'esempio può essere anche scritto in RDF/XML:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
"http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://example.org/schemas/vehicles">
<rdf:Description rdf:ID="MotorVehicle">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdfschema#Class"/>
</rdf:Description>
<rdf:Description rdf:ID="PassengerVehicle">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdfschema#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
```

Esempio

```
<rdf:Description rdf:ID="Truck">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdfschema#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
<rdf:Description rdf:ID="Van">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdfschema#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
<rdf:Description rdf:ID="MiniVan">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdfschema#Class"/>
    <rdfs:subClassOf rdf:resource="#Van"/>
    <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>
</rdf:RDF>
```

Esempio - Istanze

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
"http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntaxns#"
xmlns:ex="http://example.org/schemas/vehicles#"
xml:base="http://example.org/things">
<ex:PassengerVehicle rdf:ID="johnSmithsCar">
    <ex:registeredTo rdf:resource="http://www.example.org/staffid/85740"/>
    <ex:rearSeatLegRoom rdf:datatype="&xsd;integer">127</ex:rearSeatLegRoom>
    <ex:primaryDriver rdf:resource="http://www.example.org/staffid/85740"/>
</ex:PassengerVehicle>
</rdf:RDF>
```

Proprietà

- **rdf:Property** Sottoclasse di rdfs:Resource. Rappresenta le proprietà
- ***rdfs:subPropertyOf*** Istanza di rdf:Property, usata per specificare che una proprietà è una specializzazione di un'altra. Ogni proprietà può essere la specializzazione di zero o più proprietà
- ***rdfs:seeAlso*** fa riferimento ad una una risorsa che fornisce ulteriori informazioni sul soggetto dell'asserzione
  - ***rdfs:isDefinedBy*** Sottoproprietà di rdfs:seeAlso, indica una risorsa che definisce il soggetto di un'asserzione

Esempio - Proprietà

- Ad esempio la proprietà exterm:weightInKg viene descritta assegnando alla proprietà un URIref e descrivendo quella risorsa con una proprietà rdf:type il cui valore è la risorsa rdf:Property .

  exterm:weightInKg rdf:type rdf:Property

rdfs:range

- L'RDFS fornisce anche dei vocabolari per descrivere come le proprietà e le classi devono essere intese per essere usate assieme nei dati RDF. L'informazione più importante di questo tipo è fornita attraverso l'uso delle proprietà rdfs:range e rdfs:domain , usate per descrivere ulteriori specifiche delle proprietà per le applicazioni. Si analizza ognuna di esse nei dettagli.
- La proprietà rdfs:range è utilizzata per indicare che i valori di una particolare proprietà sono istanze di una classe designata. Ad esempio se example.org vuole indicare che la proprietà ex:author ha valori che sono istanze della classe ex:Person , essa scriverà i tre statement:

ex:Person rdf:type rdfs:Class .
ex:author rdf:type rdf:Property .
ex:author rdfs:range ex:Person .

- Queste dichiarazioni indicano che ex:Person è una classe, ex:author è una proprietà e gli statement che utilizzano la proprietà ex:author hanno istanze di ex:Person come oggetti.

rdfs:range

- Una data proprietà può avere nessuna, una o più di una range property. Nell'ultimo caso, il valore delle proprietà sono risorse che sono istanze di tutte le classi che sono state specificate come il range. Una proprietà di tipo rdfs:range può anche essere utilizzata per indicare che il valore di una proprietà è dato da un tipo literal. Per esempio, se example.org vuole indicare che la proprietà ex:age ha valori dal tipo di dato dell'XML Schema xsd:integer , essa scriverà gli statement RDF:

ex:age rdf:type rdf:Property .
ex:age rdfs:range xsd:integer .

- Il datatype xsd:integer è identificato dal suo URIref (l'URIref completo è http://www.w3.org/2001/XMLSchema#integer). Questo URIref può essere usato senza dichiarare esplicitamente nello schema che esso identifica un datatype.

rdfs:Datatype

- Comunque, è buona norma dichiarare esplicitamente che un dato URIref identifica un datatype. Questo può essere fatto usando la classe dell'RDF Schema rdfs:Datatype. Per dichiarare che xsd:integer è un datatype, example.org scriverà lo statement RDF:

xsd:integer rdf:type rdfs:Datatype .

- Questo statement dice che xsd:integer è l'URIref di un datatype (il quale è assunto essere conforme ai requisiti per i datatype RDF descritti nella specifica RDFConcepts. E' da notare che una tale dichiarazione non costituisce una definizione di un tipo di dato. Non c'è possibilità di definire datatype nell'RDF Schema. Essi vengono definiti esternamente all'RDF e all'RDFS e sono attribuiti nelle dichiarazioni RDF tramite i loro URIref.

rdfs:domain

- La proprietà rdfs:domain è usata per indicare che una particolare proprietà si applica ad una classe designata. Se, nel solito esempio, example.org vuole indicare che la proprietà ex:author si applica ad istanze della classe ex:Book , essa dichiarerà:

ex:Book rdf:type rdfs:Class .
ex:author rdf:type rdf:Property .
ex:author rdfs:domain ex:Book .

- Questa dichiarazione indica che ex:Book è una classe, ex:author è una proprietà che la dichiarazione RDF che usa la proprietà ex:author ha istanze di ex:Book come soggetti.
- Una data proprietà può avere nessuna, una o più di una proprietà di dominio. Nell'ultimo caso si può dire che una risorsa che ha quella proprietà è un'istanza di tutte le classi specificate come dominio.

rdfs:subPropertyOf

- L'RDF Schema fornisce anche una maniera di specializzare le proprietà come fatto per le classi. Questa relazione di specializzazione tra due proprietà è descritta usando la proprietà, gia predefinita, rdfs:subPropertyOf .
- Ad esempio, se ex:primaryDriver e ex:driver sono entrambe proprietà, example.org potrebbe descrivere queste proprietà, ed il fatto che ex:primaryDriver è una specializzazione di ex:driver , scrivendo lo statement RDF:

ex:driver rdf:type rdf:Property .
ex:primaryDriver rdf:type rdf:Property .
ex:primaryDriver rdfs:subPropertyOf ex:driver .

- Il significato della relazione rdfs:subPropertyOf è che se un'istanza exstaff:fred è un ex:primaryDriver dell'istanza ex:companyVan , allora l'RDF Schema definisce exstaff:fred come essere anche un ex:driver di ex:companyVan .
- Una proprietà può essere una sottoproprietà di nessuna, una o di più proprietà. Tutte le proprietà dell'RDFS rdfs:range e rdfs:domain che si applicano ad una proprietà RDF si applicano anche a ciascuna delle sue sottoproprietà.

Altri costrutti

- Esistono anche altri costrutti dell'RDF Schema, che sono definiti nella specifica come "Utility Property", che sono utilizzati per fornire documentazione ed ulteriori informazioni su di uno schema RDF o sulle istanze. I principali sono:
- ***rdfs:comment***
  - rdfs:comment è un'istanza di rdf:Property che può essere utilizzata per fornire una descrizione di una risorsa in un modo che è leggibile dall'uomo. Questi commenti possono aiutare a capire il significato di classi e proprietà.
- ***rdfs:label***
  - rdfs:label è un'istanza di rdf:Property che può essere usata per fornire una versione leggibile dall'uomo del nome di una risorsa.
- ***rdf:value***
  - rdf:value è una istanza di rdf:Property che può essere usata per descrivere valori strutturati.

Proprietà non presenti in RDF

- Altre proprietà che arricchirebbero lo schema, che sono state identificate come utili, ma che non sono fornite dall'RDF Schema, includono:
- **"cardinality constraints"** (vincoli sulla cardinalità) sulle proprietà, quali ad esempio dire che una persona ha esattamente un padre;
- specificare che una data proprietà, quale ex:hasAncestor (ha antenato), è "**transitiva**", cioè che se A ex:hasAncestor B, e B ex:hasAncestor C, allora A ex:hasAncestor C;
- specificare che una data proprietà è **un identificatore univoco** (o una key) per le istanze di una classe particolare;
- specificare che due classi diverse (che hanno URIref diverso) in realtà rappresentano la stessa classe;
- specificare che due istanze diverse (che hanno URIrefs diverso) in realtà rappresentano lo stesso individuo;
- specificare vincoli sul range o sulla cardinalità di una proprietà che dipende dalla classe delle risorse a cui le proprietà è applicata, ad esempio essere in grado di dire che per una squadra di calcio la proprietà ex:hasPlayers ha 11 valori, mentre per una squadra di basket la stessa proprietà dovrebbe avere solo 5 valori;
- L'abilità di descrivere nuove classi in termini delle combinazioni (ad esempio unioni e intersezioni) di altre classi, o dire che due classi sono disgiunte (ad esempio che nessuna risorsa è un'istanza di entrambe le classi).

Link utili

- http://www.w3.org/XML
- http://www.w3.org/2004/11/uri-iri-pressrelease.html.en
- http://www.ietf.org/rfc/rfc3986.txt
- http://www.w3.org/TR/xpath
- http://www.w3.org/TR/xlink/
- http://www.w3.org/TR/xptr/
- http://www.w3.org/TR/xslt
- http://www.w3.org/TR/rdf-schema/
- http://www.w3.org/TR/rdf-syntax-grammar/
- http://www.w3.org/TR/n-triples/