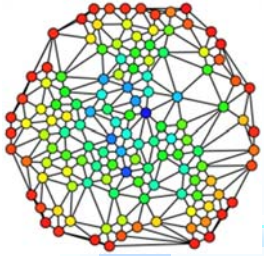




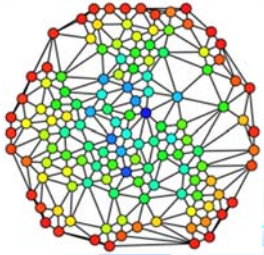
# *Knowledge Management and Protection Systems (KMaPS)*

# Corso di Laurea in Ingegneria

## *Xml, Rdf and Ontology Basic aspects (parte 3, 2015)*

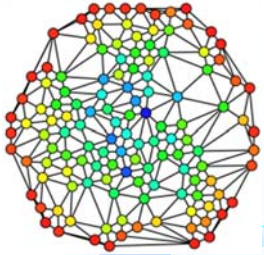
*Eng. Ph.D. Michela Paolucci*
*Eng. Ph.D. Gianni Pantaleo*

**DISIT Lab** http://www.disit.dinfo.unifi.it/
Department of Information Engineering, DINFO
University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570
michela.paolucci@unifi.it gianni.pantaleo@unifi.it



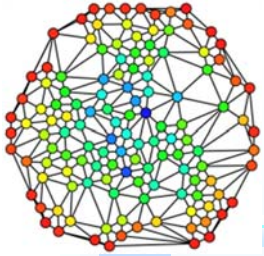
# Outline

- XML: *Extensible Markup Language*
  - Introduzione
  - Classi e Istanze
  - Proprietà
  - Applicazioni XML
- RDF: *Resource Description Language*
  - Introduzione
  - Classi e Istanze
  - Proprietà



# Outline

- XML: *Extensible Markup Language*
  - Introduzione
  - Classi e Istanze
  - Proprietà
  - Applicazioni XML
- RDF: *Resource Description Language*
  - Introduzione
  - Classi e Istanze
  - Proprietà

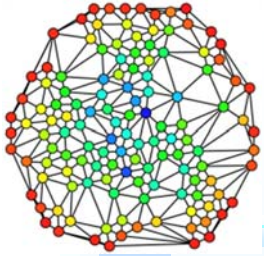


# XML: Extensible Markup Language

Introduzione

Classi e istanze

Proprietà

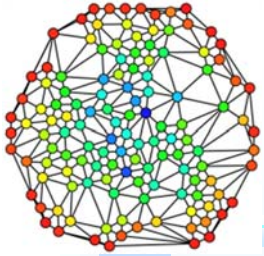


# XML: cosa è

XML: Extensible Markup Language:

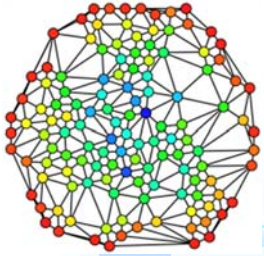
- è un meta-*linguaggio* che consente la creazione di linguaggi di mark-up
- consente la rappresentazione di documenti e dati strutturati su supporto digitale
- è uno dei più potenti e versatili sistemi per la creazione, archiviazione, preservazione e disseminazione di documenti digitali

- …. ma è anche una famiglia di tecnologie complementari



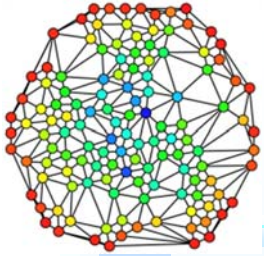
# XML: le origini

- 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



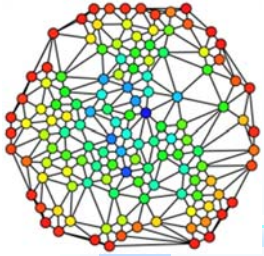
# XML: caratteristiche (1)

- 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)



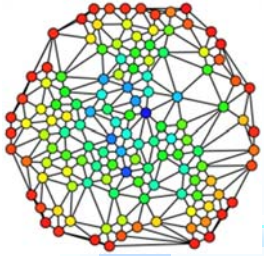
# Il concetto di metalinguaggio

- 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



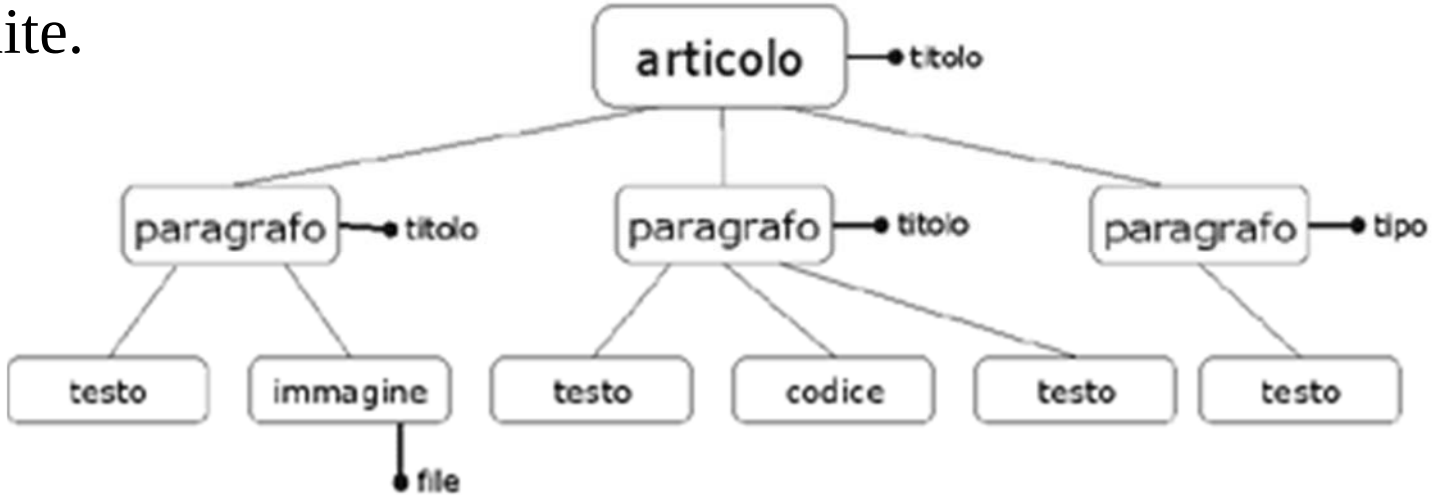
# XML: caratteristiche (2)

- 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)

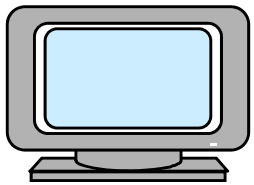
- XML adotta un formato di file di tipo testuale: sia il mark-up sia il testo, sono stringhe di caratteri
- XML si basa sul sistema di codifica dei caratteri ISO 10646/UNICODE
- Un documento XML è “leggibile” da un utente umano senza la mediazione di software specifico
- Concretamente, un documento XML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



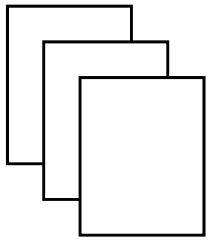
# XML: caratteristiche (4)

File XML

```
<Title> Titolo </title>

<p>Paragarafo ..
<p>Paragrafo
```



On-line WWW

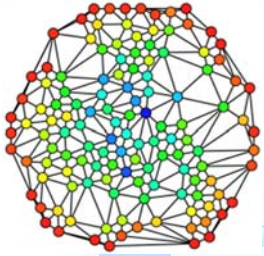


Carta

CD-ROM

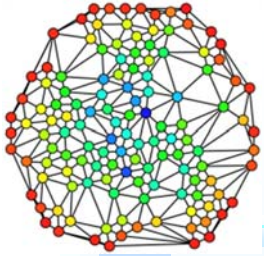
DBMS NATIVI O RELAZIONALI

DOCUMENT MANAGEMENT SYSTEMS

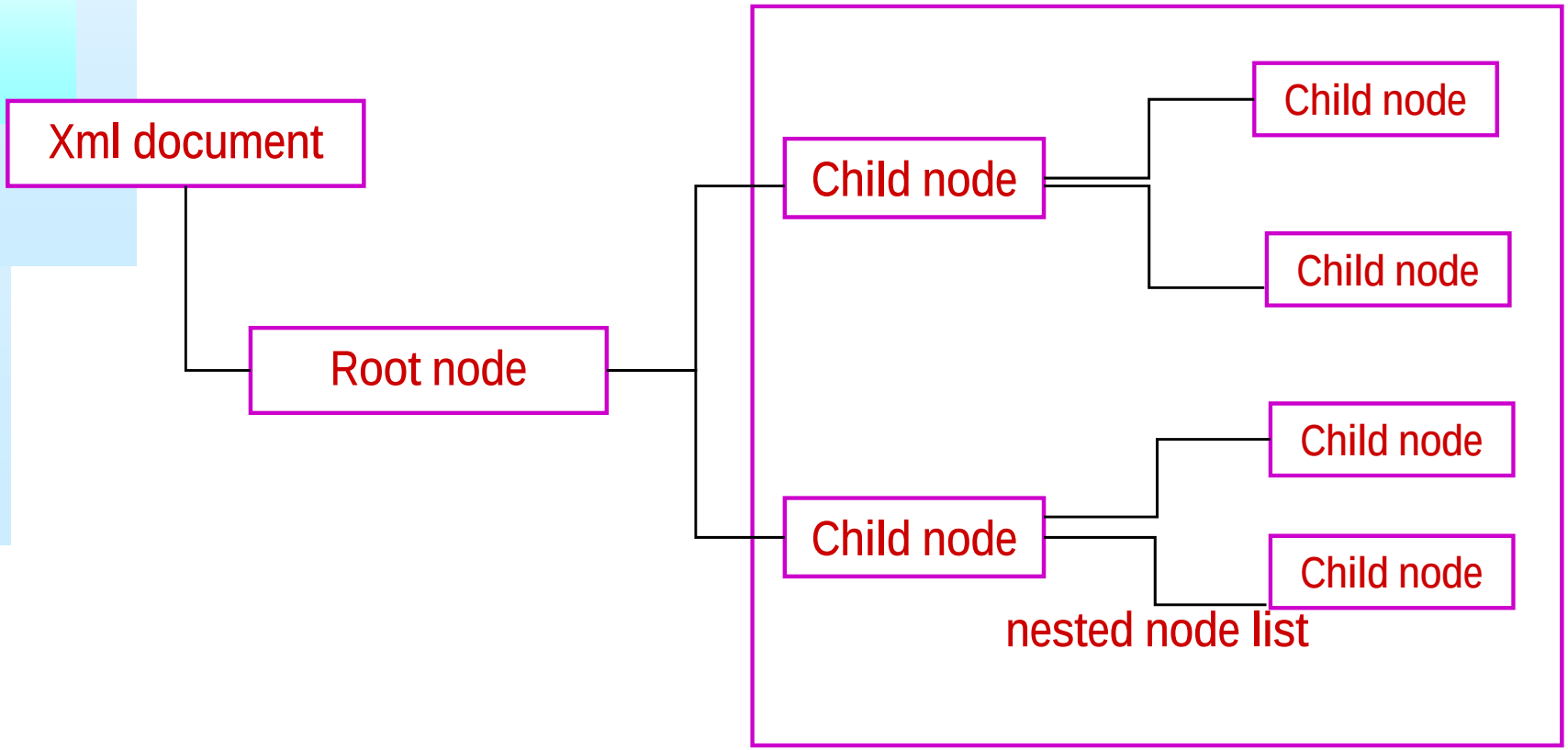


# XML secondo il W3C

- 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

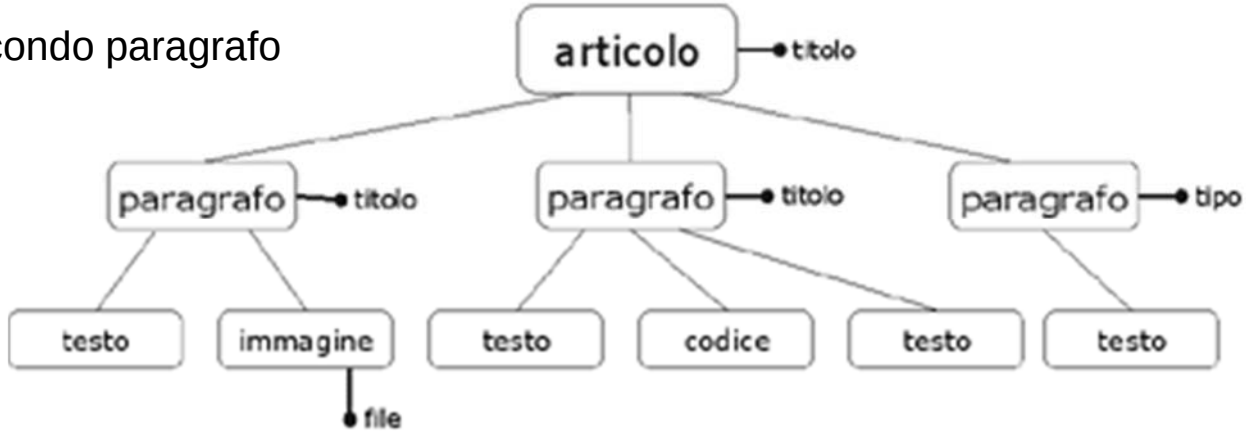


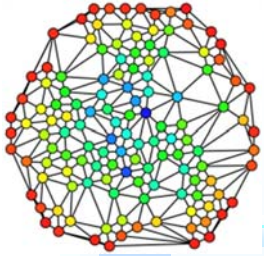
# La struttura gerarchica ordinata



# Esempio: articolo

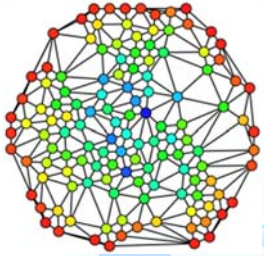
```
<?xml version = "1.0" ?>
  <articolo titolo = "Titolo dell'articolo">

    <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>
```





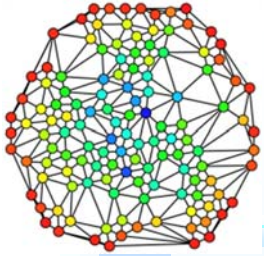
# Strutture XML: gli elementi (1)

- 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)

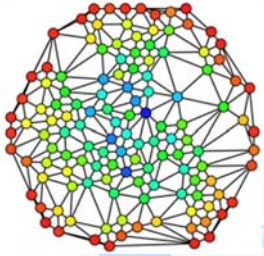
- Esiste uno e uno solo elemento radice (corrispondente al nodo radice dell'albero), che non è contenuto da nessun altro e che contiene direttamente o indirettamente tutti gli altri
- Ogni elemento, escluso l'elemento radice, deve essere contenuto da un solo elemento (elemento padre) e può contenere altri sotto-elementi (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



# Strutture XML: gli attributi

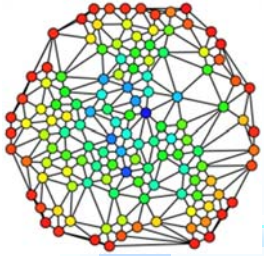
- Ad ogni elemento possono essere associati uno o più attributi che ne specificano ulteriori caratteristiche o proprietà non strutturali:
  - la lingua del suo contenuto testuale
  - un identificatore univoco
  - un numero di ordine
  - etc.

- Gli attributi XML sono caratterizzati da:
  - un nome che li identifica
  - un valore



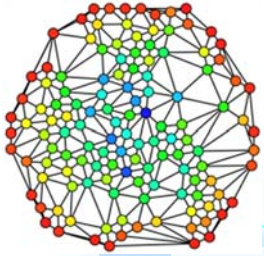
# Strutture XML: le entità (1)

- Un documento XML (in quanto oggetto digitale) ha una struttura fisica
- Dal punto di vista fisico un documento è composto da unità di archiviazione che sono denominate entità (entity)
- Esiste almeno una entità in ogni documento XML: la document entity, che contiene il documento stesso
- In generale una entità è 'una qualsiasi sequenza di byte considerata indipendentemente dalla sua funzione strutturale':
  - un singolo carattere UNICODE
  - una stringa di testo XML (caratteri e mark-up)
  - un intero file XML esterno
  - un intero file non XML (es. immagini digitali, etc.)
- È possibile ad esempio rappresentare nel contenuto di un documento caratteri non presenti sulla tastiera mediante entità



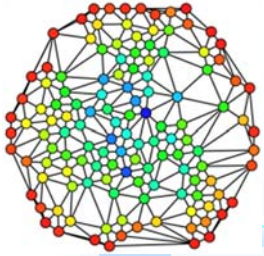
# Strutture XML: le entità (2)

- Le entità vanno definite con apposite dichiarazioni nel DTD (o nel XML-schema)
- Una entità ha un nome e un contenuto
- In un documento l'inserimento di una entità avviene mediante un riferimento a entità che ne specifica il nome
- Un processore XML sostituirà automaticamente il contenuto dell'entità al posto del riferimento



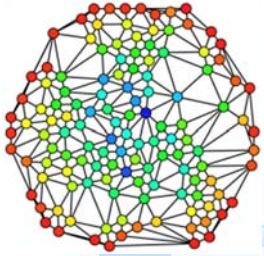
# XML: documenti ben formati e validi (1)

- 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



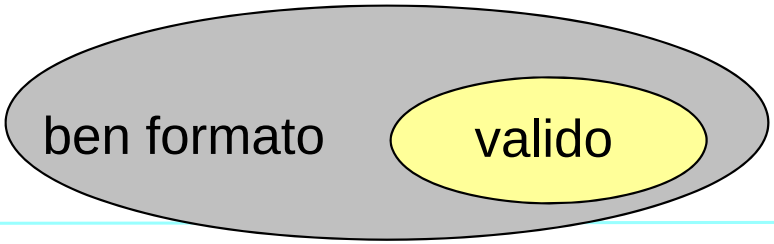
# XML: documenti ben formati e validi (2)

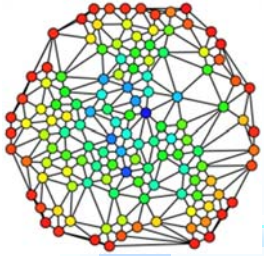
- Oltre ad essere ben formato un documento XML può anche essere **valido**
- Per stabilire la validità di un documento xml è necessario definire una **grammatica (tipo di documento)** per il linguaggio di mark-up che abbiamo ideato, a cui poi si farà riferimento per la validazione del documento xml
- 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 ben formati e validi (3)

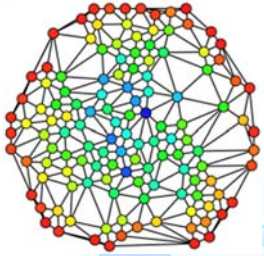
- Una **grammatica** definisce uno specifico linguaggio di mark-up => Se un documento XML rispetta le regole definite da una grammatica è detto **valido** per un particolare linguaggio
- Un documento ben formato può non essere valido rispetto ad una grammatica, mentre un documento valido è necessariamente 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**





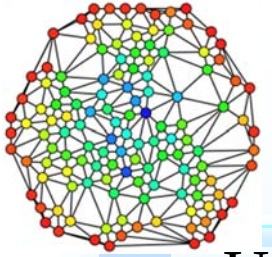
# XML: documenti ben formati e validi (4)

- 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 xml-schema)
- La maggior parte degli editor XML più recenti ha un parser integrato o si appoggia su parser esterni per effettuare la convalida dei documenti



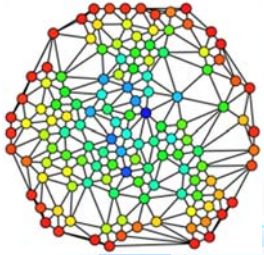
# Come si crea un documento XML

- Un documento XML contiene il mark-up (sotto forma di coppie di tag) che rappresenta linearmente la struttura gerarchica degli elementi, i loro eventuali attributi, e i caratteri del testo
- L'inserimento di mark-up e caratteri deve rispettare i vincoli di buona formazione indicati nelle specifiche XML
- Un documento valido deve rispettare anche i vincoli sintattici definiti nel DTD o xml-schema



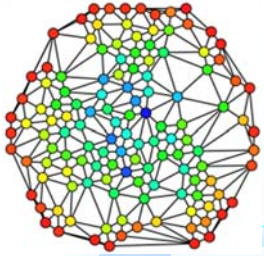
# Aspetti di sintassi generale e vincoli

- 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



# XML: documenti ben formati (1)

- Anche 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 (2)

- 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>**

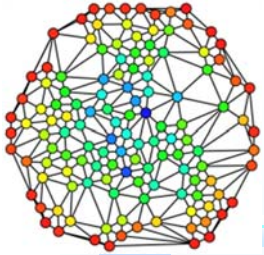
In questo caso, XML prevede l'uso delle entità che consentono di sostituire altri caratteri. Cinque entità sono predefinite e consentono l'uso di altrettanti caratteri riservati all'interno di un documento:

**&** definisce il carattere & | **<** definisce il carattere <

**>** definisce il carattere > **"** definisce il carattere " **'** definisce il carattere '

Sfruttando le entità, l'Esempio precedente diventa:

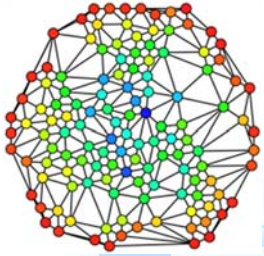
**<testo> il simbolo < indica minore di </testo>**



# XML: documenti ben formati (3)

- 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>



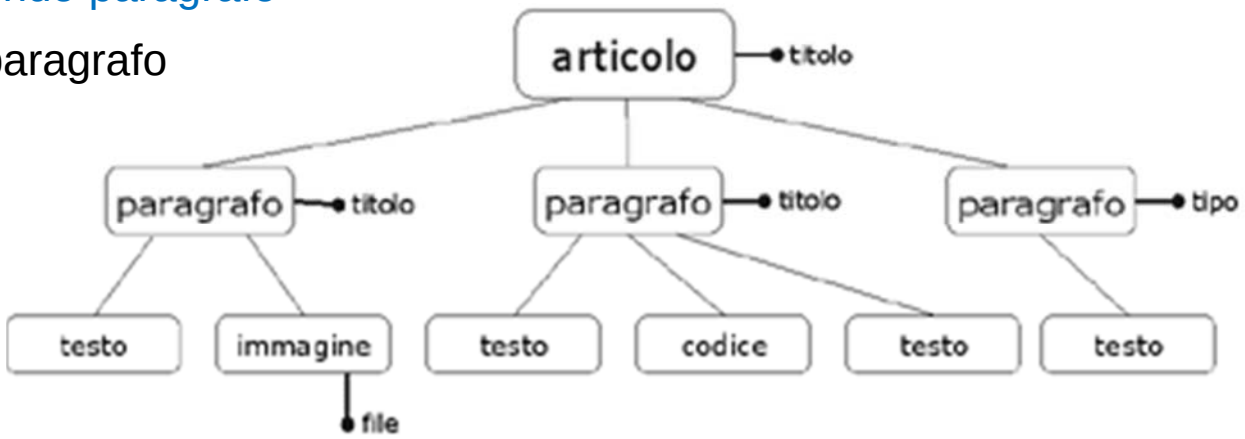
# La forma di un documento XML

- 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”?>

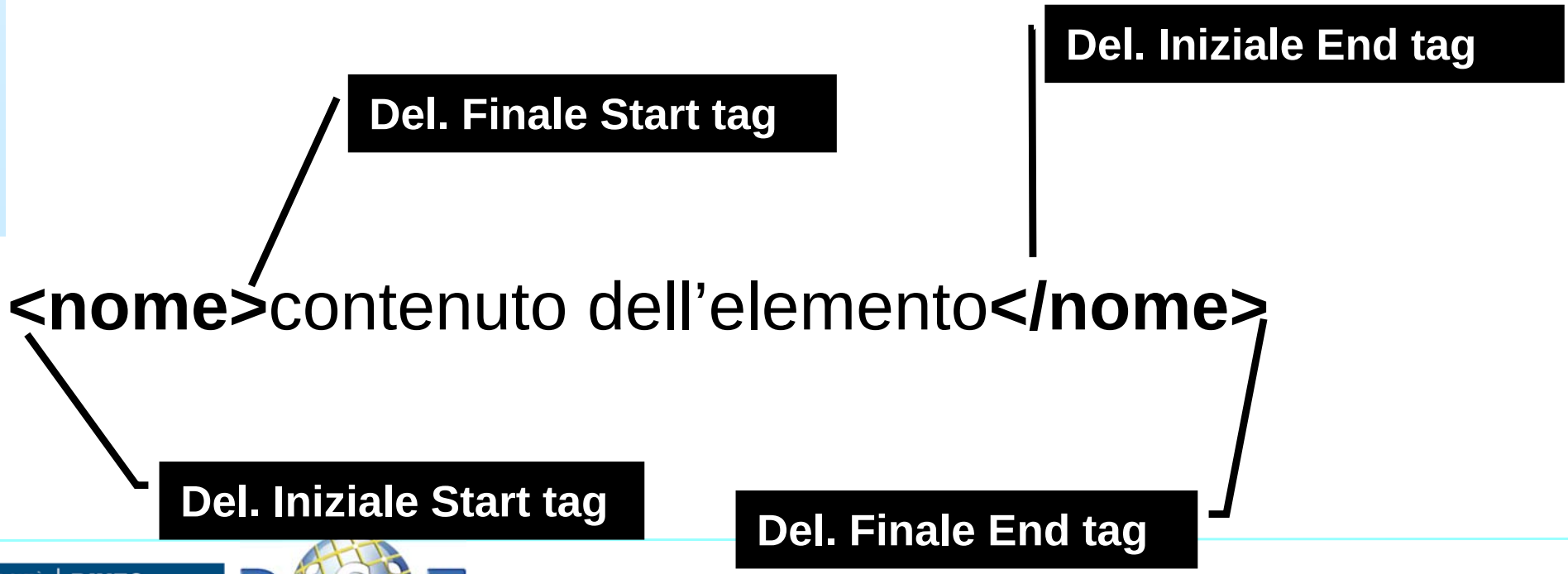
# Esempio: articolo

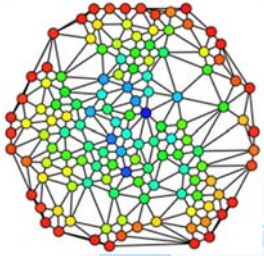
```
<?xml version="1.0" ?>
  <articolo titolo="Titolo dell'articolo">

    <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>
```



# La codifica degli elementi (1)

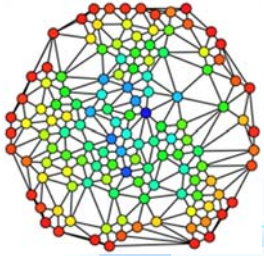
- Nel documento ogni elemento non vuoto (contenente cioè altri elementi o caratteri) deve essere marcato da un **tag iniziale** e da un **tag finale**
- Ogni tag è costituito da caratteri delimitatori e dal nome dell'elemento
- Sintassi di un elemento:





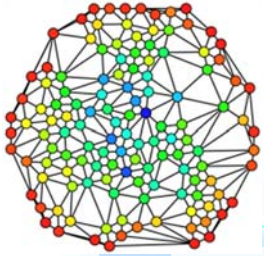
# La codifica degli elementi (2)

<text>
<div1>
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…
…
</div1>
</text>



# La codifica degli elementi (3)

- La relazione lineare tra i tag rappresenta la relazione gerarchica tra gli elementi
- Per ogni elemento, se il suo tag iniziale è nel contenuto di un elemento P allora il suo tag finale deve essere nel contenuto del medesimo elemento P
- Detto altrimenti: le coppie di tag devono annidarsi correttamene e mai sovrapporsi



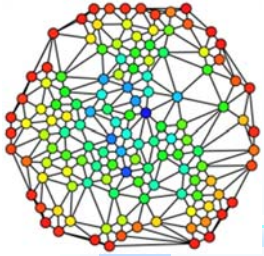
# La codifica degli elementi (4)

**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… 

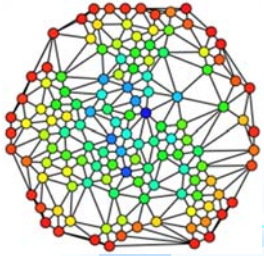
**CORRETTO!!!**

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… 



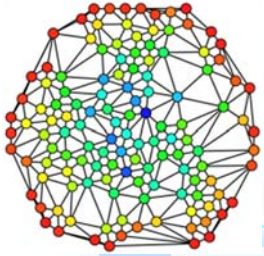
# La codifica degli elementi (5)

- Gli elementi vuoti
  - o sono rappresentati da entrambi i tag
    - …<nome_elemento> </nome_elemento>…
  - o assumono la seguente forma
    - <nome_elemento/>
  - Esempio:
    - **<img src='foo.gif'/>**



# 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">

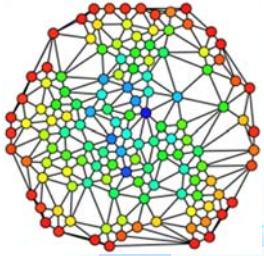
Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in una relazione troppo seria…

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"/>

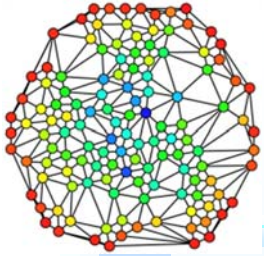
…

</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



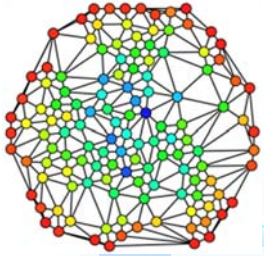
# Il riferimento alle entità

- 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

**&**nome_entità**;**

**Delimitatore di apertura**

**Delimitatore di chiusura**

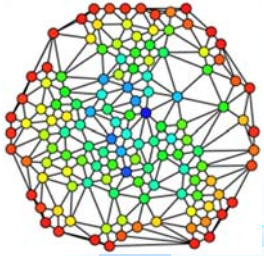


# Il riferimento alle entità

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

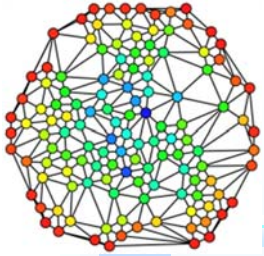
```
<p>La sua famiglia? Una sola sorella non ingombrante
n&eacute; fisicamente n&eacute; moralmente, piccola
e pallida, di qualche anno pi&ugrave; giovane di
lui…</p>
```

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



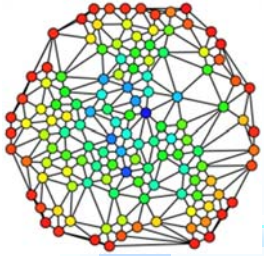
# Standard correlati a XML (1)

- XML adotta due linguaggi appositamente sviluppati per la specificazione di strutture ipertestuali complesse:
  - XML Linking Specification (XLink) specifica come costruire elementi e attributi di collegamento
  - XML Extended Pointer Specification (XPointer) specifica i sistemi di indirizzamento delle destinazioni di un link



# Standard correlati a XML (2)

- 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

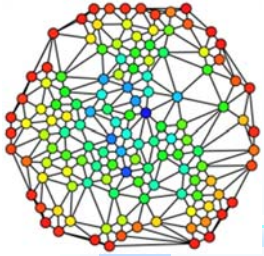


# Standard correlati a XML

- Famiglia di raccomandazioni che premette 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

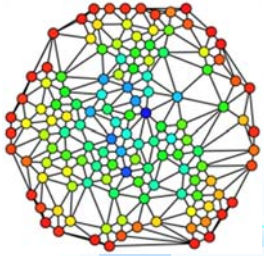
# XSLT

- 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



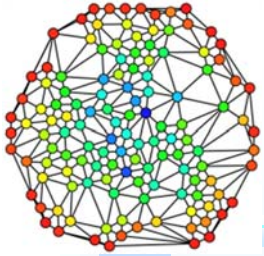
# XSLT

- 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()"/>



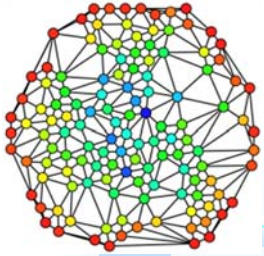
# XSLT - esempio

```
<?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>
```



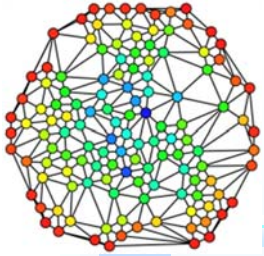
# Link utili

- 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



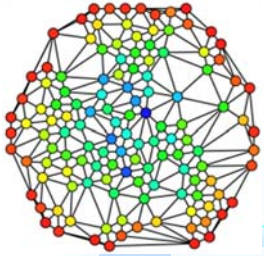
# Extensible Markup Language (XML)

## Parser, dtd e xml-schema



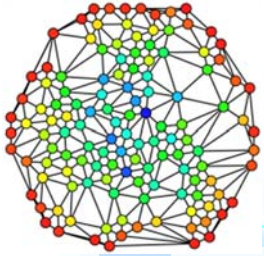
# Parser XML

- Permettono l'analisi sintattica di un documento XML
- **Parser validanti**
  - Permettono di verificare il contenuto di un documento attraverso un file esterno
    - XML Schema
    - DTD
- **Parser non validanti**
  - Delegano il controllo della struttura del documento all'applicazione
- Un parser può esporre i propri servizi principalmente attraverso due tipi di interfacce
  - **SAX (Simple Api for XML)**
  - **DOM (Document Object Model)**



# Parser SAX

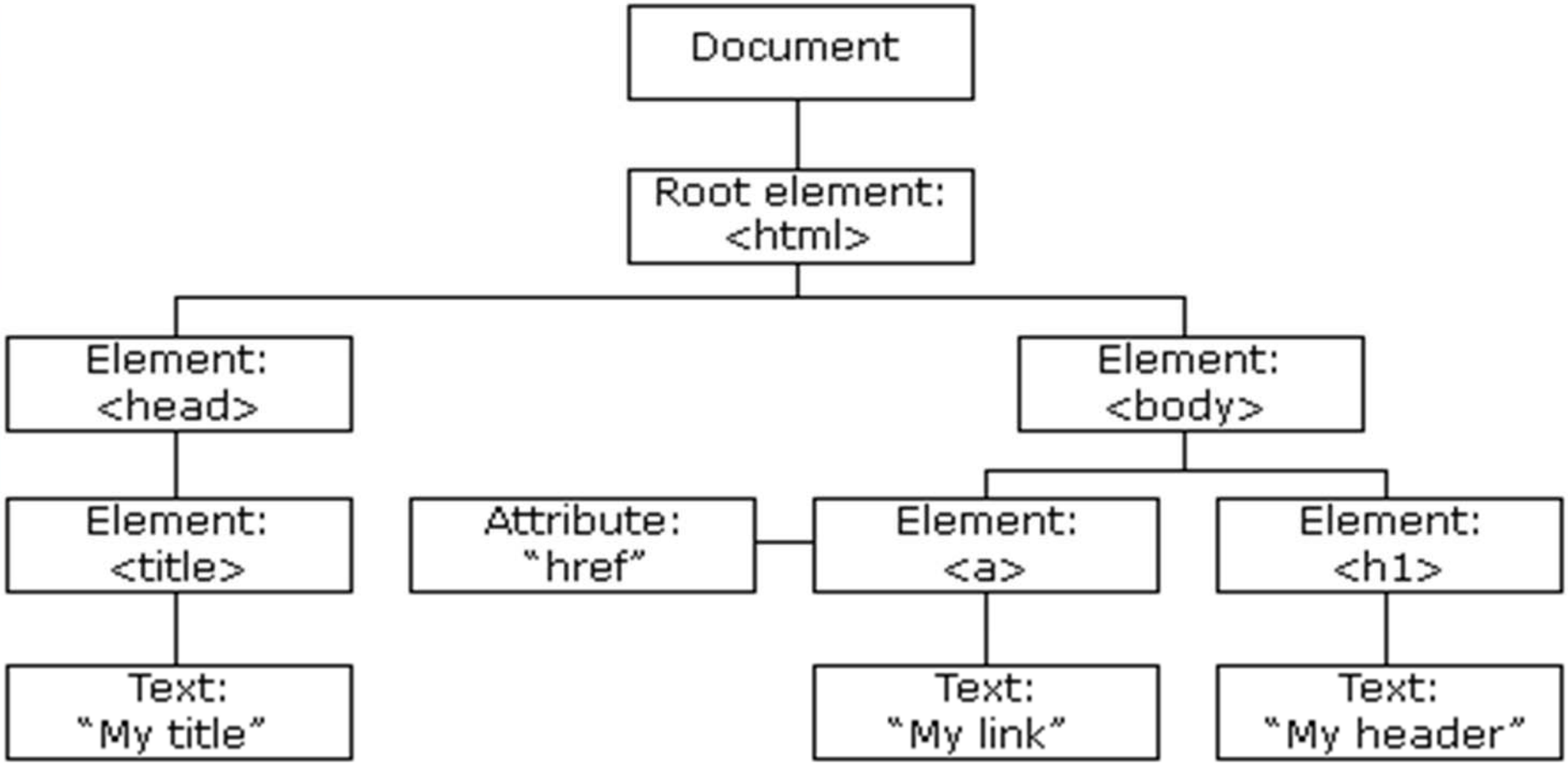
- Proposto dalla mailing list XML-Dev
  - .. recepita ed implementata da diversi produttori (Microsoft, IBM, Sun, …)
- Basato su un modello di programmazione orientato agli eventi
- Un documento XML viene visto come un flusso di dati
  - Gli eventi vengono segnalati, analizzati e processati man mano che si presentano
  - Inizio e fine del documento
  - Inizio e fine di un elemento
  - Errori
  - …
- Bisogna indicare le azioni da compiere al verificarsi di un determinato evento

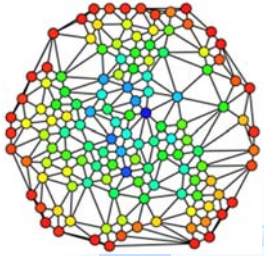


# Parser DOM

- Standard ufficiale del W3C per la rappresentazione di documenti strutturati
- Basato su un modello di programmazione orientato agli oggetti
  - Tree-based approach per la navigazione dei documenti XML
  - Costruzione esplicita dell'albero sintattico
  - Accesso casuale per la ricerca e la modifica degli elementi
- La struttura costruita dal parser viene completamente contenuta in memoria
  - …. il rilascio della memoria è a carico del programmatore!
- Specifiche suddivise in vari livelli
  - Ogni livello può  contenere moduli opzionali o obbligatori

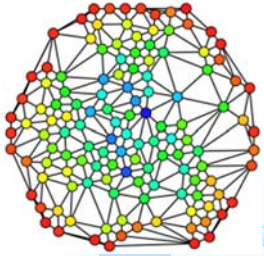
# Albero DOM


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
"href"
Element:
<a>
Element:
<h1>
Text:
"My title"
Text:
"My link"
Text:
"My header"



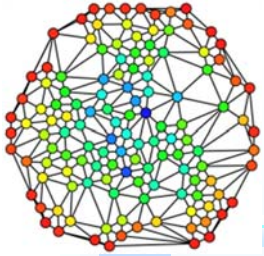
# DOM vs SAX

- Applicazioni SAX
  - Richiedono poche risorse di sistema
  - Molto efficienti
  - Adatte per documenti XML di dimensioni consistenti  o per dispositivi con memoria limitata
  - Non adatte se si ha la necessità di modificare il documento XML o ricordare eventi precedenti
- Applicazioni DOM
  - API semplici da utilizzare
  - Accesso casuale agli elementi del documento XML
  - Adatte per applicazioni che necessitano di navigare e/o modificare il documento
  - Richiedono un alto costo computazionale
    - ➔ …. la struttura rimane in memoria fino al rilascio….



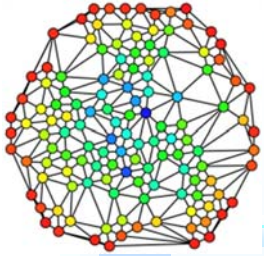
# Esempio DOM - validazione

```
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
public class DOMParser01 {
    public static void main(String args[]){
            if (args.length != 1) {
                System.err.println("Usage: java DOMParser01 filename");
                System.exit(1);
            }
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            factory.setValidating(true);
            try {
                DocumentBuilder builder = factory.newDocumentBuilder();
                builder.parse( new File(args[0]) );
                System.out.println();
                System.out.println(args[0] + " is a valid and well-formed XML file");
            } catch (Exception ex) {
                System.err.println(ex.getMessage());
                    System.err.println();
                    System.err.println(args[0] + " is NOT a valid and well-formed XML file");
            }
    }
}
```



# Esempio DOM - visualizzazione

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
public class DOMParser02{
    public static void main(String args[]){
            try {
                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                  factory.setValidating(true);
                 DocumentBuilder builder = factory.newDocumentBuilder();
                 Document doc = builder.parse( new File(args[0]) );
                 DOMSource source = new DOMSource(doc);
                 TransformerFactory tFactory = TransformerFactory.newInstance();
                 Transformer transformer = tFactory.newTransformer();
                  transformer.transform(source, new StreamResult(System.out));
            } catch (Exception ex) {
                  System.err.println(ex.getMessage());
            }
    }
}
```

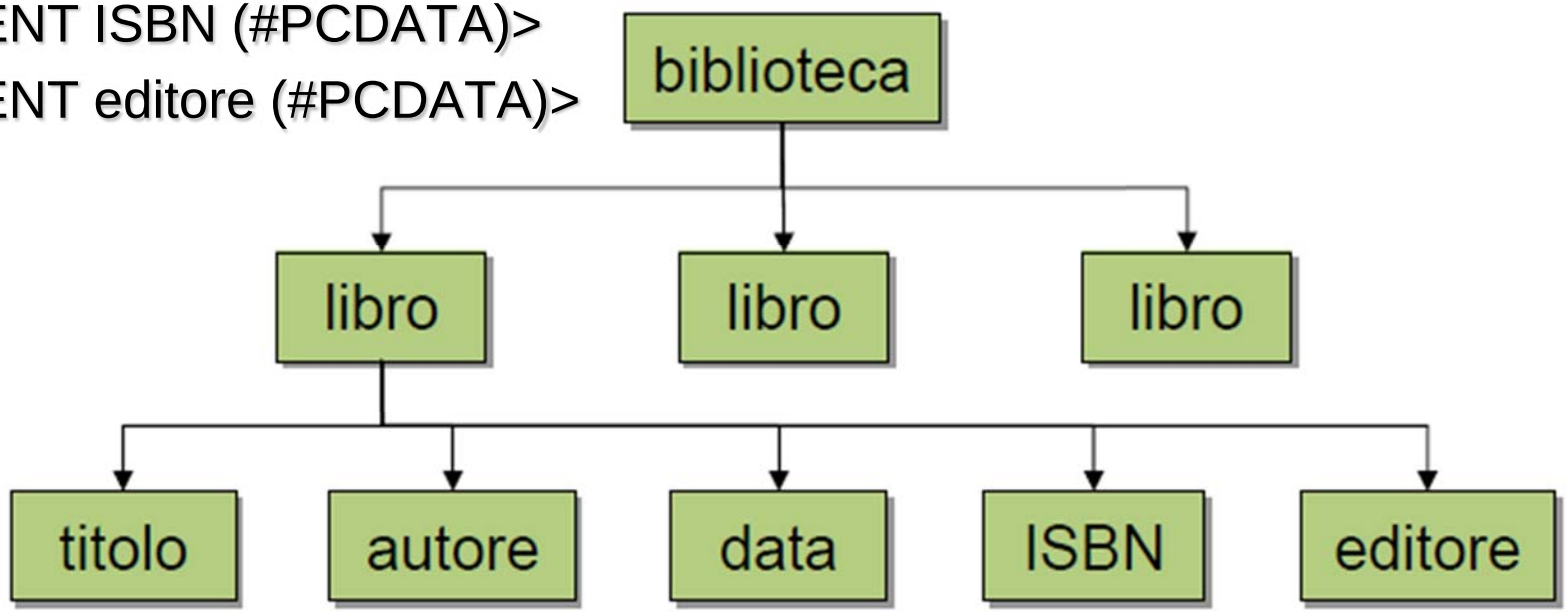


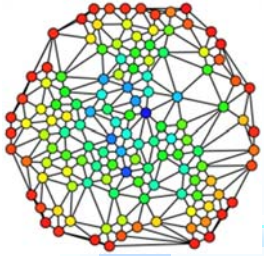
# DTD

- 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)>
]>
```





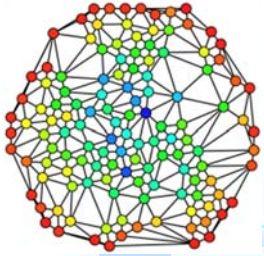
# Collegare un dtd ad un file xml

- 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>
```

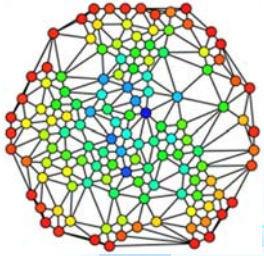
  - esternamente al documento XML:

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



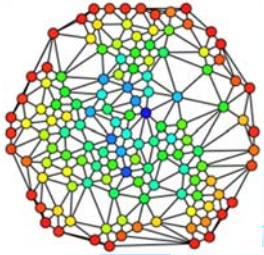
# Dai DTD agli xml-schema

- 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

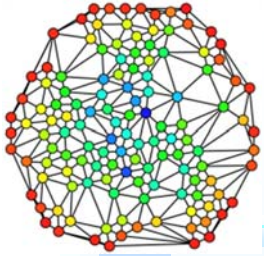


# Come definire un xml-schema (2)

- 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:

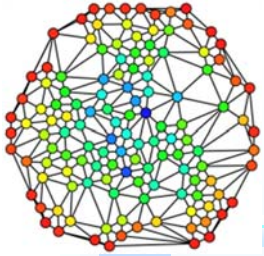
```
<?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    ... 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

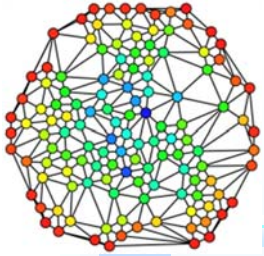
- XML Schema introduce il concetto di tipo di dato semplice per definire gli elementi che non possono contenere altri elementi e non prevedono attributi.
- Si possono usare tipi di dato semplici predefiniti oppure è possibile personalizzarli.
- Alcuni tipi di dato predefiniti sono riportati nella tabella

| | |
|---|---|
| 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 |

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

<quantita>123</quantita> …OK

<quantita>uno</quantita> ….NO

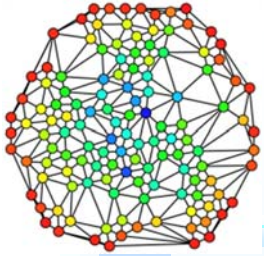


# XML-schema: tipo di dato semplice personalizzato

- 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>
```

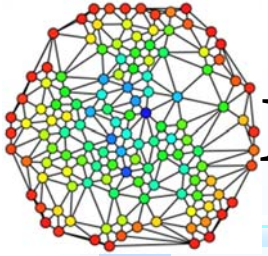
- In questo caso, la dichiarazione indica che l'elemento <quantita>:
  - è di tipo semplice
  - prevede una restrizione sul tipo di dato intero predefinito accettando valori compresi tra 1 e 100



# 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)

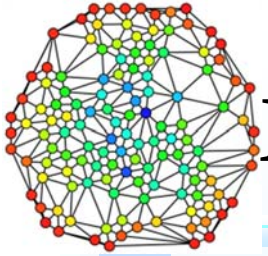
- I tipi di dato complesso sono elementi che ne possono contenere altri.
- E` possibile definire la sequenza di elementi che possono stare al suo interno utilizzando uno dei seguenti **costruttori di tipi complessi**:
  - **<xs:sequence>** Consente di definire una sequenza ordinata di sottoelementi
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

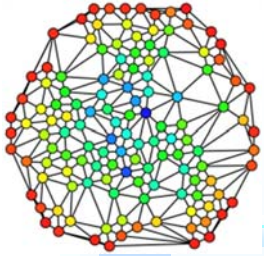
```
[…]
<articolo>
  <paragrafo>
     Questo è il paragrafo 1.. Introduzione…
  </paragrafo>
  <testo>
    Test effettivo contenuto nel paragrafo
  </testo>
</articolo>

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



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

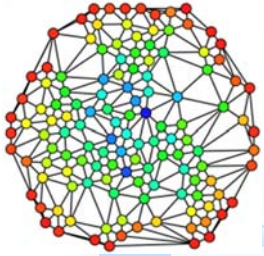
- Per ciascuno dei costruttori visti (**sequence, choice, all**) e per ciascun elemento è possibile definire il numero di occorrenze previste utilizzando gli attributi **minOccurs** e **maxOccurs**.
- Esempio: se l'elemento *testo* può essere presente una o infinite volte all'interno di un paragrafo 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. 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

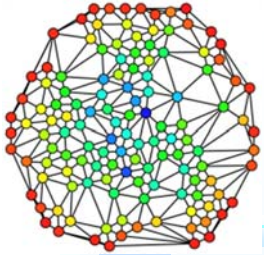


# XML-schema: def. modulare degli elementi (1)

- 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>
```

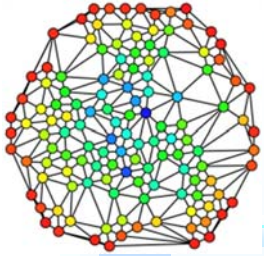
- Il riferimento ad una dichiarazione di tipo viene fatta come se fosse un tipo predefinito, come mostrato nel seguente esempio:
- <xs:element name="nome_elemento" type="nome_tipo" />



# 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>
```

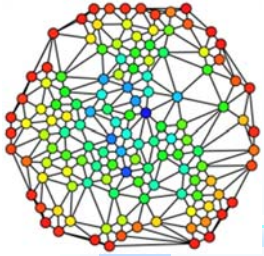


# XML-schema: namespace (1)

- 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

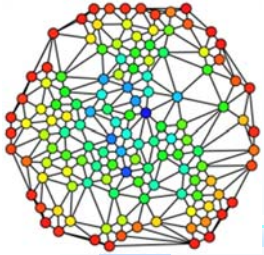


# XML-schema: namespace (2)

- **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

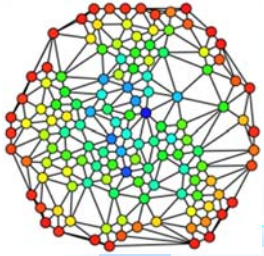


# Collegare un xml-schema ad un file xml

- A partire da una grammatica definita tramite un XML-Schema, è possibile sfruttare un parser XML validante per **verificare la validità** di un documento XML
- Il parser avrà bisogno:
  - del documento XML da validare
  - dello schema XML rispetto a cui effettuare la validazione
- Ci sono diversi modi per fornire al parser informazioni sullo schema da usare per la validazione. Uno di questi è il seguente:
  - inserire nel documento XML un riferimento allo schema da usare associato all'elemento root, come nel seguente esempio:

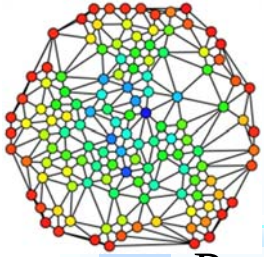
<articolo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="articolo.xsd" titolo="“documento XML" >, con :

    - **xmlns:xsi** indica un URL che specifica la modalità con cui si indicherà il riferimento allo schema XML
    - **xsi:noNamespaceSchemaLocation** indica il nome e l'eventuale percorso del file contenente lo schema XML di riferimento



# XML-schema: sintassi dei namespace (1)

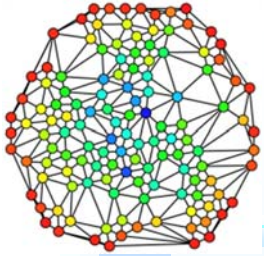
- In un documento XML si fa riferimento ad un namespace utilizzando un attributo speciale (**xmlns**) associato al root element:
  - <articolo **xmlns**="http://www.dominio.it/xml/articolo">
- Questo indica che l'elemento articolo ed i suoi sottoelementi usano i nomi definiti nel namespace identificato 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)

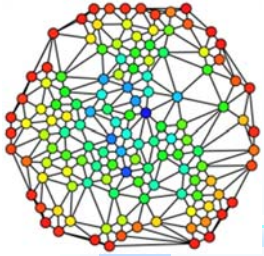
- Per mettere in relazione un namespace con il relativo XML Schema occorre dichiararlo nel root element:
  - <articolo
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:art="http://www.dominio.it/xml/articolo"
    xmlns:bibl="http://www.dominio.it/xml/bibliografia"
    xsi:schemaLocation="http://www.dominio.it/xml/articolo
    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



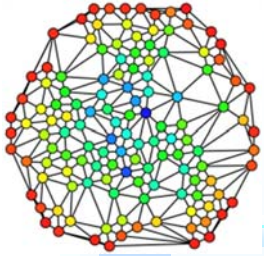
# Extensible Markup Language (XML)

## Applicazioni xml



# XML - Applicazioni

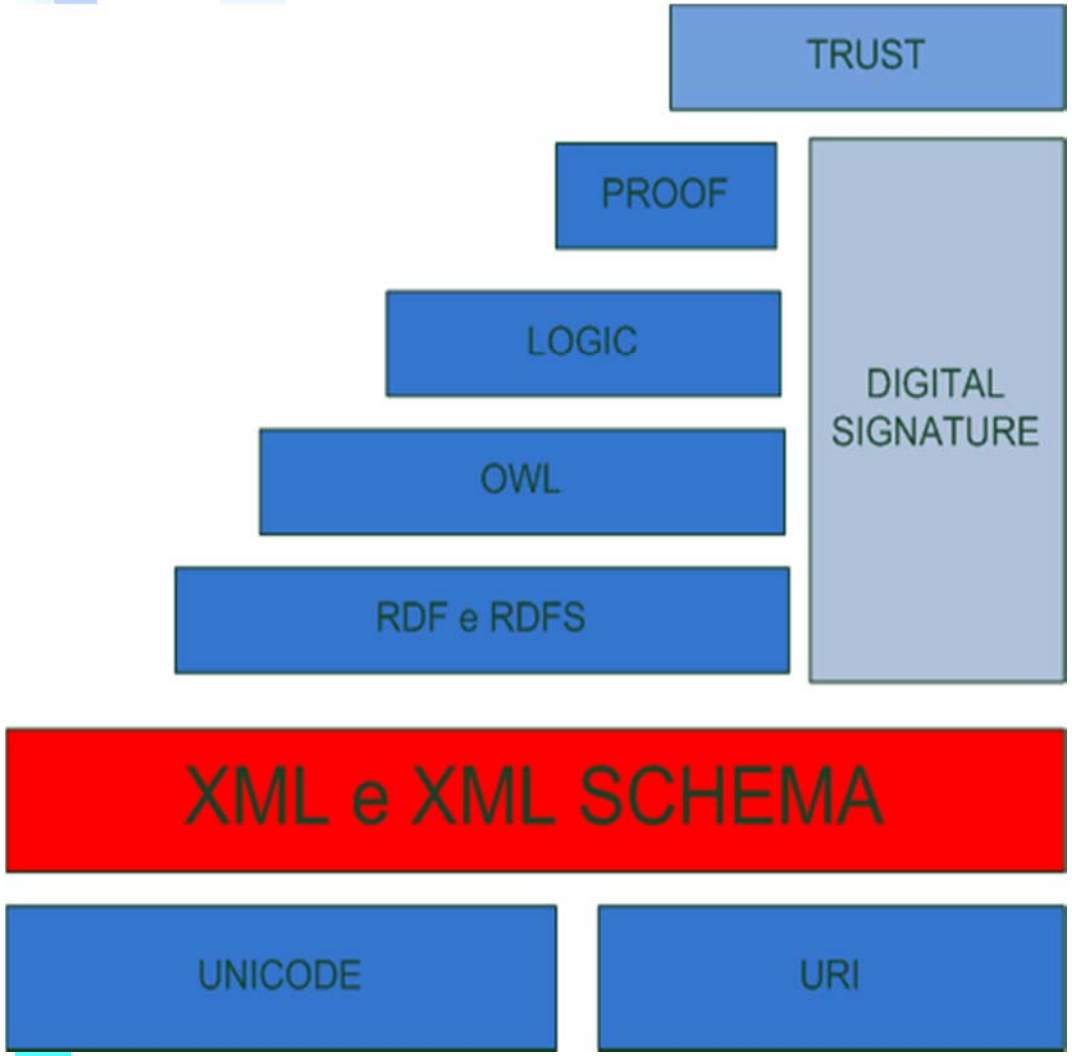
- Svariati contesti applicativi
  - Web Semantico
    - SKOS, RDF, OWL, OWL2
  - RSS
  - Scalable Vector Graphics (SVG)
  - Simple Object Access Protocol (SOAP)
  - Synchronized Multimedia Integration Language (SMIL)
  - WSDL
  - Documenti di definizione di metadati
  - UDDI
  - … e molti altri ancora



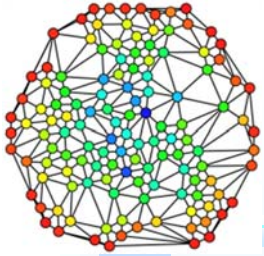
# XML – Applicazioni: web semantico

- Termine coniato da Tim Berners-Lee
- Estende l'attuale World Wide Web da machine-rapresentable a **machine-understandable**
  - Non lo sostituisce… ma lo migliora associando informazioni di carattere semantico alle risorse della rete
- Il Semantic Web è una ragnatela di informazioni connesse tramite relazioni **semantiche**
  - Logiche di connessione più elaborate rispetto ai collegamenti ipertestuali di HTML
  - L'idea è di generare dei documenti che possano essere letti al tempo stesso da esseri umani e da agenti automatici
- Massiccio utilizzo di linguaggi e grammatiche XML per rendere il formato machine-readable

# XML – Applicazioni: web semantico

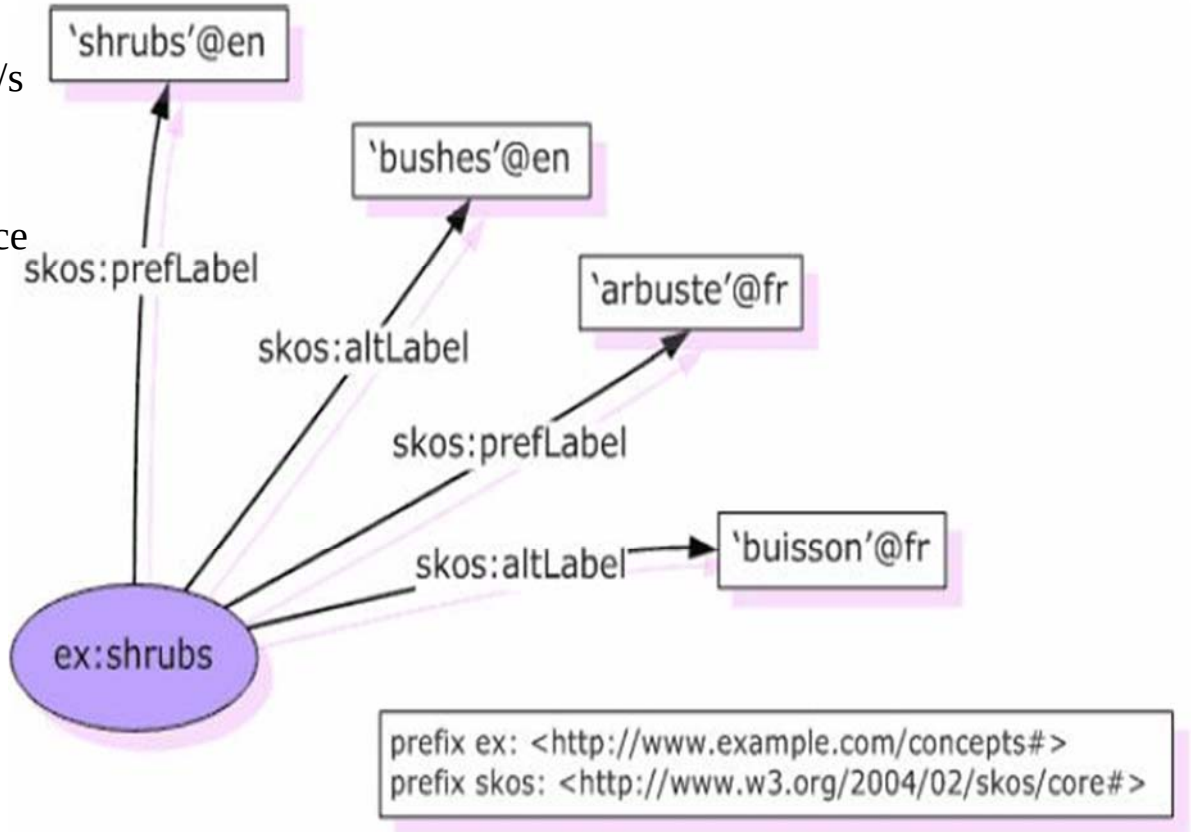


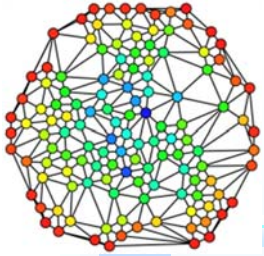

- Massiccio utilizzo di XML e Xml Schema
- Fornisce al web semantico l'interoperabilità sintattica
- Nodi concettuali
  - RDF e OWL sono basati su XML ma hanno un DTD che ne limita l'espressività e aggiunge semantica ai singoli nodi



# XML – Applicazioni: web semantico

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22
    -rdf-syntax-ns#"
    xmlns:skos="http://www.w3.org/2004/02/s
    kos/core#">
    <skos:Concept
    rdf:about="http://www.example.com/conce
    pts#shrubs">
    <skos:prefLabel
    xml:lang="en">shrubs</skos:prefLabel>
    <skos:altLabel
    xml:lang="en">bushes</skos:altLabel>
    <skos:prefLabel
    xml:lang="fr">arbuste</skos:prefLabel>
    <skos:altLabel
    xml:lang="fr">buisson</skos:altLabel>
    </skos:Concept>
</rdf:RDF
```

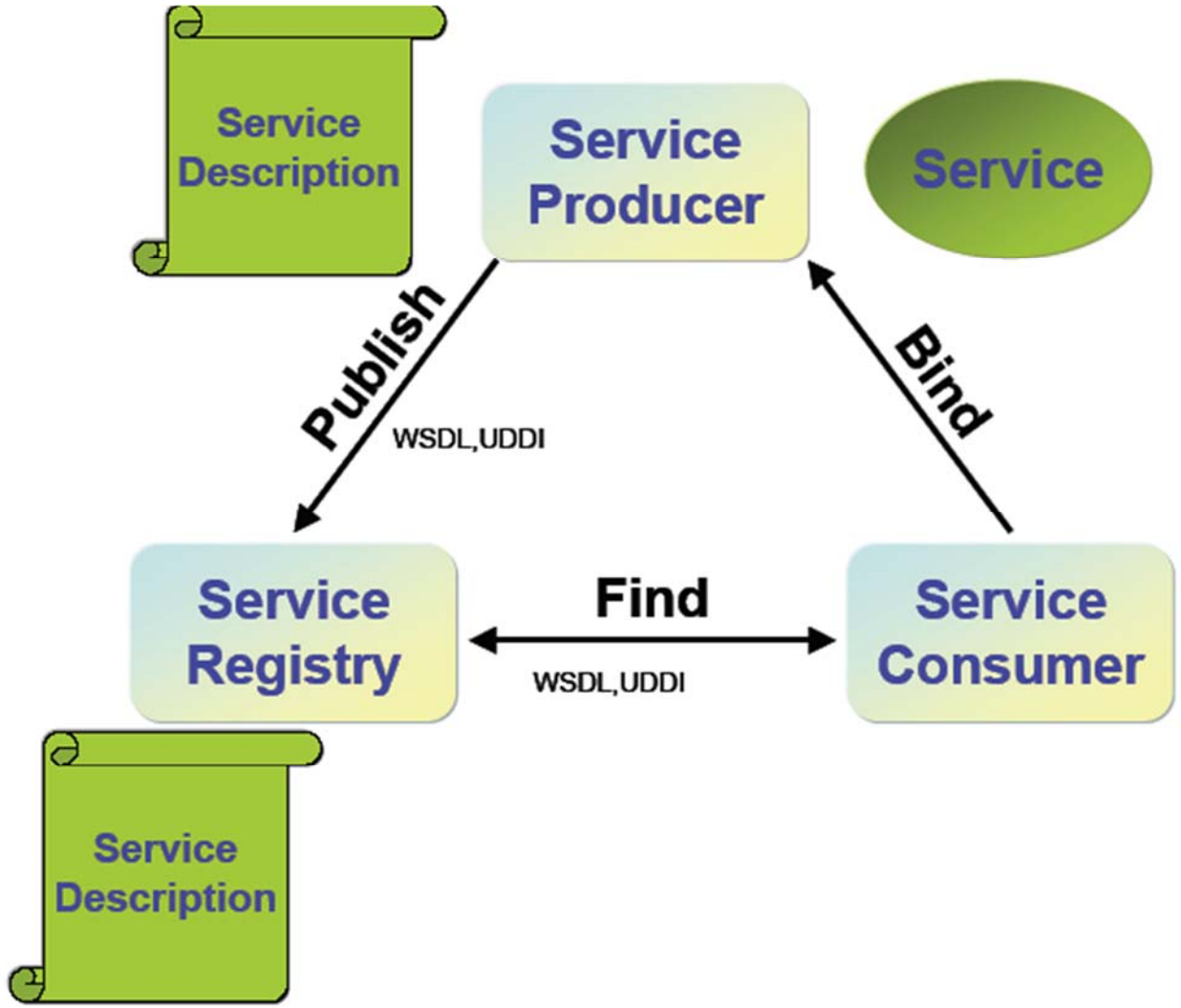


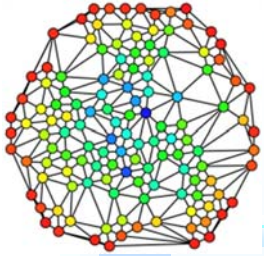


# Applicazioni XML: Web Service

- Un web service è un oggetto che offre un servizio (o alcuni servizi) ai client sulla rete attraverso una interfaccia ben definita
- Il servizio è posto fisicamente sul web
  - L'integrazione con il servizio avviene attraverso lo scambio di messaggi attraverso la rete
- Tre componenti principali:
  - Service producer
  - Service consumer
  - Service register

# Applicazioni XML: Web Service

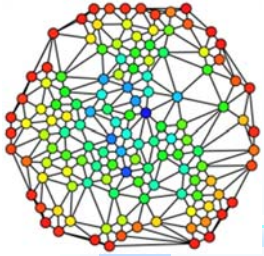




# Web Service: Tecnologie

- SOAP
  - Protocollo per lo scambio di informazioni in una architettura distribuita
- WSDL
  - Standard utilizzato per la descrizione di un servizio web
- UDDI
  - Standard utilizzato per la pubblicazione dei servizi in rete

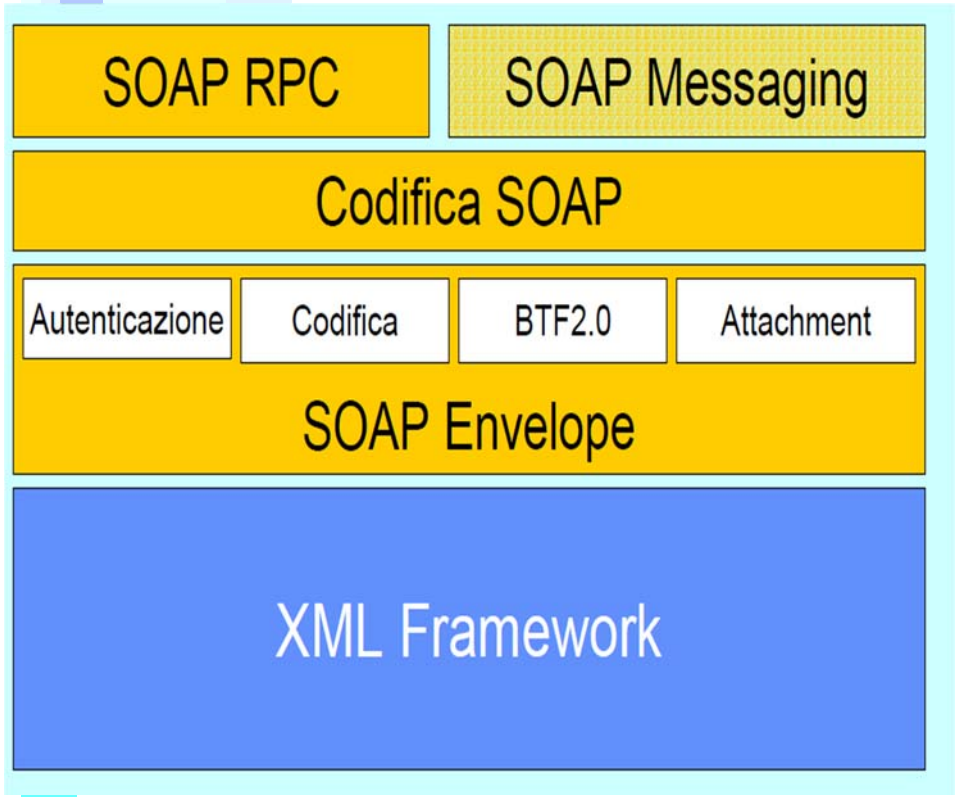
- Tutte e tre le tecnologie utilizzano lo standard XML



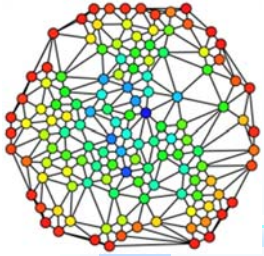
# SOAP

- protocollo SOAP (Simple Object Access Protocol)
- nasce nel dicembre 1999, da collaborazioni fra Microsoft e IBM
- interoperabilità di applicazioni su piattaforme diverse
- Sfrutta tecnologie già ampiamente consolidate e funzionanti:
  - HTTP
    - Per trasportare le informazioni
  - XML
    - Per esprimere le informazioni

# SOAP

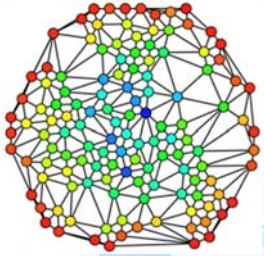



```
<soap:Envelope
    xmlns:soap="http://schemas.xmloap.or
    g/soap/envelope" soap:encodingStyle="
    http://schemas.xmlsoap.org/soap/enco
    ding">

<soap:Body
    xmlns:mybook="http://www.books.com/
    soapbook">
      <mybook:GetBookPrice>
        <mybook:ISBN>12-3456-789-
    </mybook:ISBN>
      </mybook:GetBookPrice>
  </soap:Body>
</soap:Envelope>
```



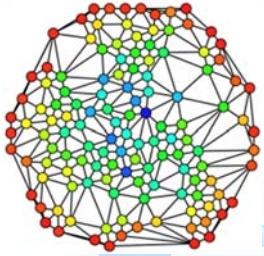
# WSDL

- Linguaggio basato su XML che serve a definire e descrivere web-service
- Descrive un servizio fornendo le informazioni necessarie per sviluppare client che intendono utilizzarlo seguendo la sintassi XML
- Specifica:
  - Locazione (URL del servizio)
  - Operazioni offerte
    - Descrizione astratta
    - Descrizione concreta



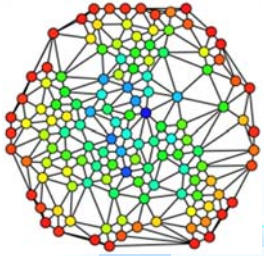
# WSDL – Descrizione astratta

- Types
  - Per definire i tipi di dati utilizzati all'interno del documento
- Message
  - Definizione astratta dei dati scambiati tra requestor e provider, contenente i parametri di richiesta e di risposta
- Port type
  - Di solito uno per WSDL
  - Corrisponde al servizio stesso
  - Composto da una serie di elementi operation
- Operation
  - Specifica i nomi delle operazioni, gli input e gli output
  - Vengono specificati i messaggi scambiati durante l'operazione



# WSDL – Descrizione concreta

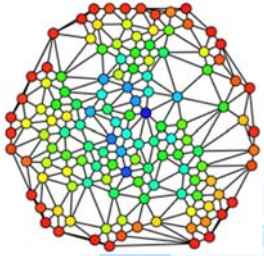
- Binding
  - Fornisce i dettagli per l'implementazione delle operazioni contenute in un portType, specificando il protocollo utilizzato ed il formato dei messaggi scambiati
- Port
  - Specifica l'indirizzo di rete del servizio con cui effettuare la connessione
- Service
  - Insieme di porte correlate che spesso offrono modalità differenti di accesso alla stessa porta



# WSDL – esempio

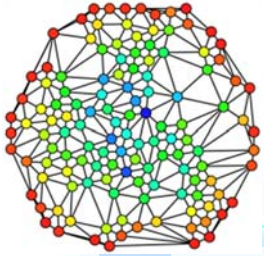
```
<message name="get_userRequest">
      <part name="id" type="xs:integer" />
</message>
<message name="get_userResponse">
      <part name="utente" type="tns:utente"
    />
</message>
<portType name="gestioneUtentiType">
  <operation name="getUserById">
    <input message="tns:get_userRequest"
     />
    <output
     message="tns:get_userResponse" />
  </operation>
</portType>
```

```
<binding name="gestioneUtentiBinding"
    type="tns:gestioneUtentiType">
<soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="getUserById">
  <soap:operation soapAction=”definizioneAction"
   style="rpc"/>
   <input>
    <soap:body use="encoded"
   namespace=“mynamespaceWS"
   encodingStyle="http://schemas.xmlsoap.org/soap/en
   coding/"/>
   </input>
   <output>
    <soap:body use="encoded" namespace="
   mynamespaceWS"
   encodingStyle="http://schemas.xmlsoap.org/soap/en
   coding/"/>
   </output>
 </operation>
</binding>
```



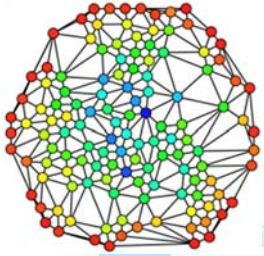
# Web Service: UDDI

- Universal Description Discovery and Integration
- Base di dati indicizzata, basta su XML
- Permette alle aziende di pubblicare i propri dati e servizi offerti sulla rete internet
- Registrazione
  - ♣ Pagine gialle
    - ➔ Categorizzazione dei servizi (tassonomie di servizi standardizzate)
  - ♣ Pagine bianche
    - ➔ Contatti e dettagli dell'azienda
  - ♣ Pagine verdi
    - ➔ Informazioni tecniche sui servizi



# Applicazioni XML: SMIL

- Le prime specifiche (SMIL 1.0) vengono pubblicate nel 1998 come W3C recommendation
- Il 15 giugno del 2001 viene pubblicato SMIL 2.0 come working draft W3C
- Cerca di superare alcuni limiti di HTML nella gestione dell'occupazione della pagina
  - ♣ Non esiste un sistema di coordinate temporali
    - ➔ Si può fare qualche tentativo con javascript
  - ♣ Non esiste un sistema di coordinate spaziali che consenta di collocare degli oggetti nella pagina



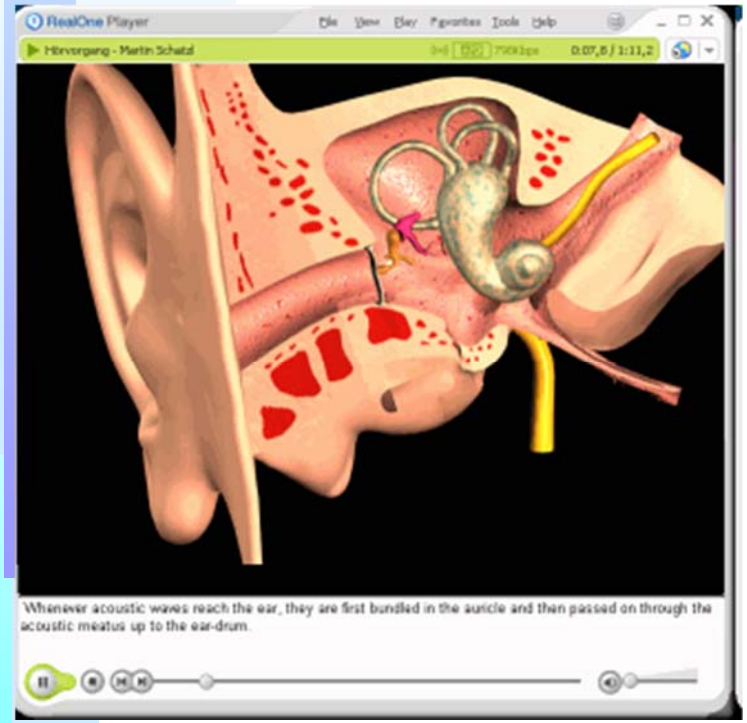
# Applicazioni XML: SMIL

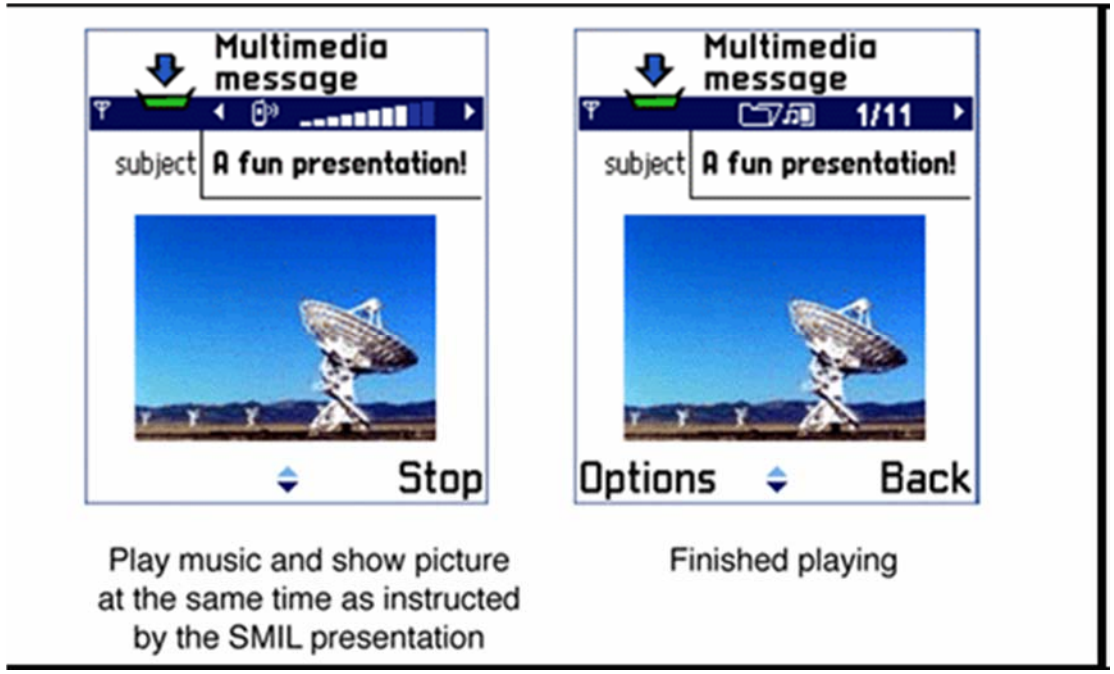
```
<smil>
<head>
<meta name= “Pippo" content= “Pluto" />
<layout>
<root-layout width=“300” height=“200”/>
</layout>
</head>
<body>
<img src=“images/image1.jpg” dur=“2s” />
<img src=“images/image2.jpg” dur=“1.5s”
    />
<img src=“images/image3.jpg” dur=“2.5s”
    />
<!-- ... -->
</body>
</smil>
```

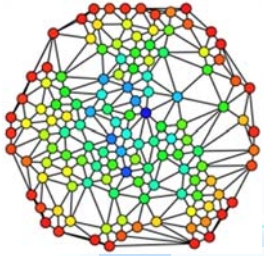
## SMIL permette:

- Disporre gli oggetti multimediali in punti precisi dello schermo
- Descrivere il comportamento temporale dei diversi elementi di una presentazione
- Modificare la riproduzione secondo alcuni parametri relativi alla stazione di lavoro dell’utente
- Inserire dei link ad altre presentazioni o parti di esse

# Applicazioni XML: SMIL



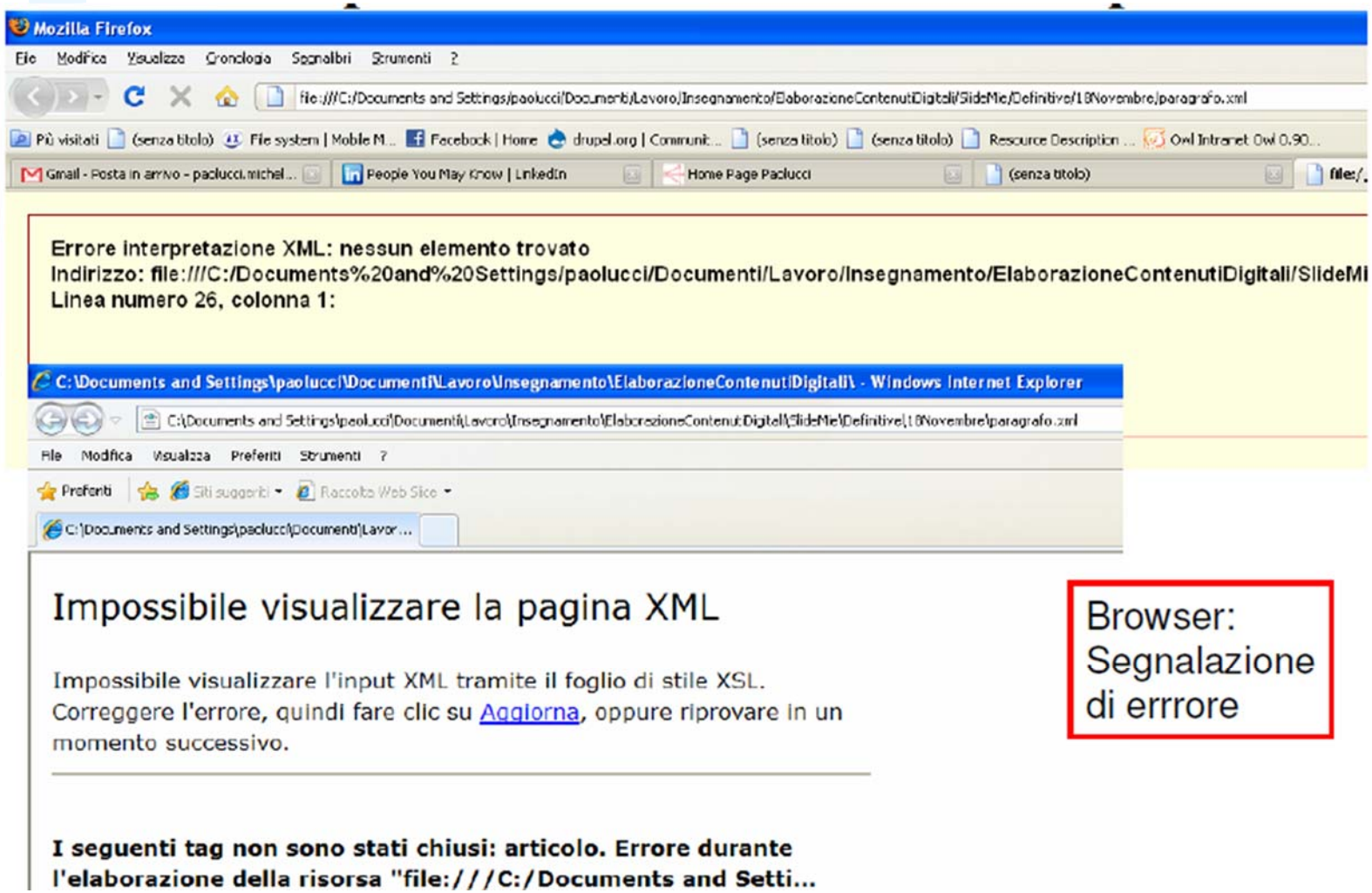



# Esempi XML

- 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/

# Firefox + Open XML Viewer



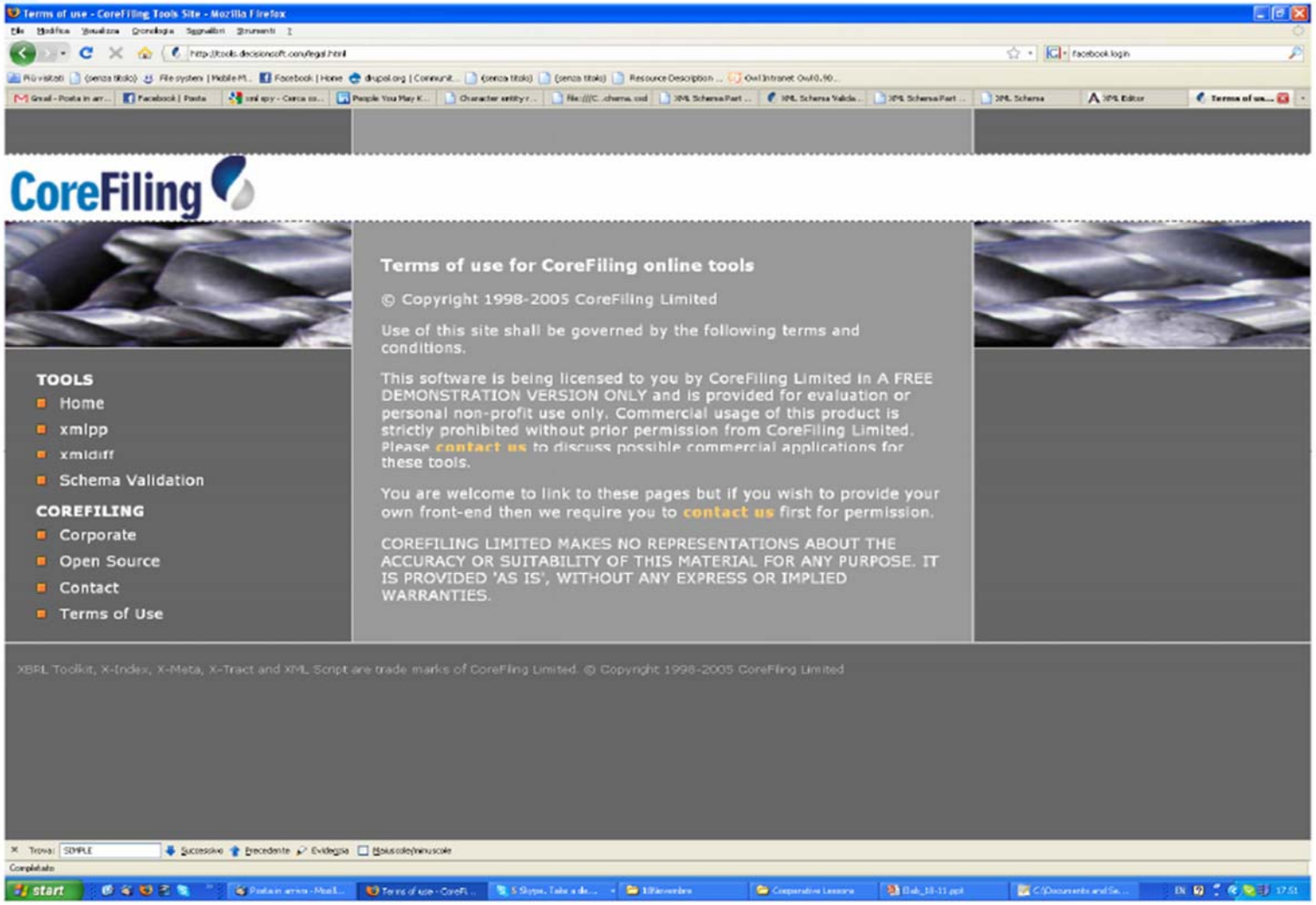

Browser: Segnalazione di errrore

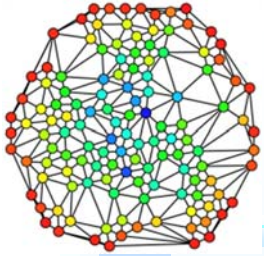
# Firefox + Open XML Viewer


```
− <articolo titolo="Come definire un documento xml">
  − <paragrafo titolo="Introduzione">
    − <testo>
        Questo documento è di prova per la definizione di file xml …
      </testo>
    </paragrafo>
  − <paragrafo titolo="Descrizione di un documento xml">
    − <testo>
        In questo paragrafo oltre al testo metto un'immagine:
      </testo>
      <immagine titolo="immagine" formato="jpg"/>
      <testo>Adesso commento l'immagine</testo>
    </paragrafo>
  − <paragrafo titolo="Ringraziamenti">
      <testo>Ringrazio …… </testo>
    </paragrafo>
</articolo>
```

Verificare che il documento XML sia ben formato aprendolo sul browser

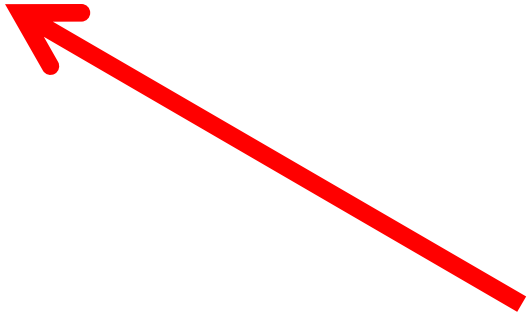
# Internet Explorer

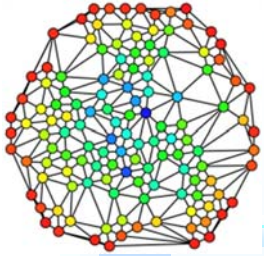




# Outline

- XML: *Extensible Markup Language*
  - Introduzione
  - Classi e Istanze
  - Proprietà
  - Applicazioni XML
- RDF: *Resource Description Language*
  - Introduzione
  - Classi e Istanze
  - Proprietà



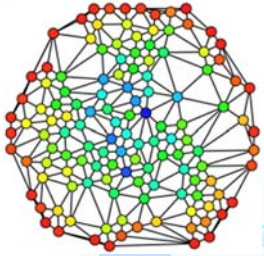


# RDF: Resource Description Language

Introduzione

Classi e istanze

Proprietà



# RDF

XML serve per:

strutturare l'informazione

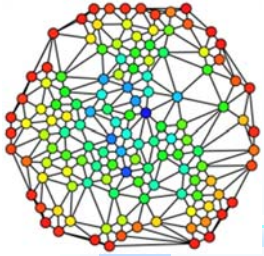
XMLS serve per:

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



# Risorse

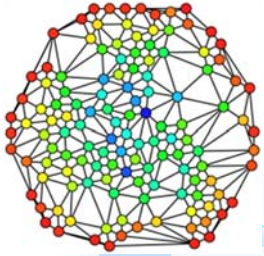
Tutte le cose descritte con espressioni RDF vengono dette **Risorse**.

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

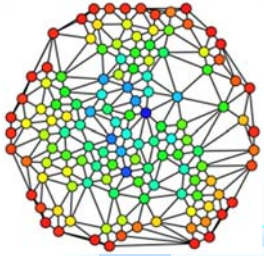
Qualsiasi cosa può avere associato un URI



# Proprietà

Una **Proprietà** viene usata per descrivere una **risorsa** e può consistere in:

- un aspetto specifico
- una caratteristica
- un attributo
- una relazione

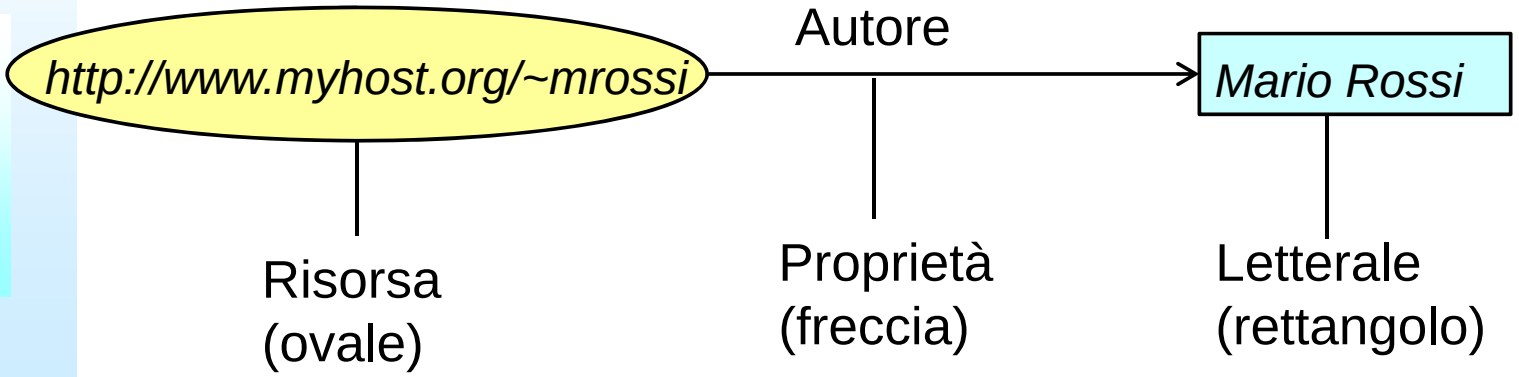


# Asserzioni

Una **Asserzione** RDF è composta da:

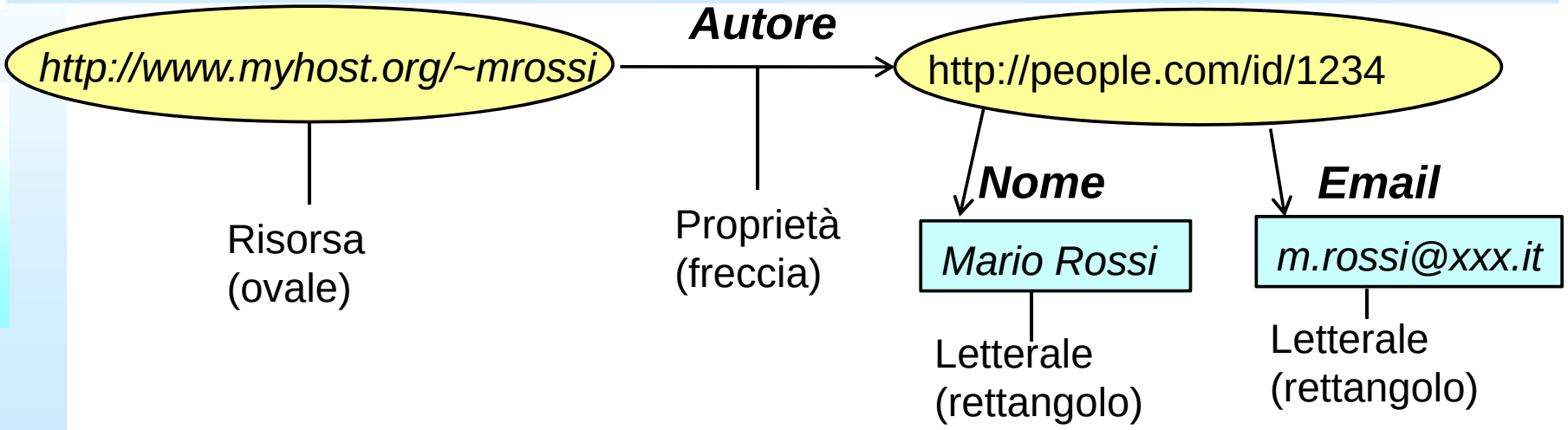
- Soggetto: una determinata risorsa
- Predicato: una proprietà della risorsa
- Oggetto: un valore relativo ad una proprietà
  - una risorsa definita da un URI
  - una stringa di caratteri o altro tipo di dato primitivo definito da XML

# Rappresentazione Grafica & sintassi (1)

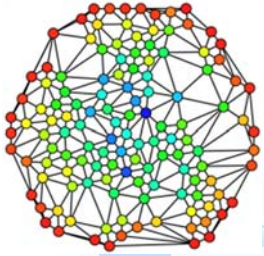



```
<rdf:RDF  xmlns:s="http://www.example.com/myschema#"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Description rdf:about="http://www.myhost.org/~mrossi">
     <s:Autore>Mario Rossi</s:Autore>
</rdf:Description>
```

# Rappresentazione Grafica & sintassi (2)




```
<rdf:Description rdf: about="http://www.myhost.org/~mrossi">
      <s:Autore rdf:resource=" http://people.com/id/1234"/>
</rdf:Description>

<rdf:Description rdf:about="http://people.com/id/1234">
<s:Nome>Mario Rossi</s:Nome>
<s:Email>m.rossi@xxx.it </s:Email>
</rdf:Description>
```

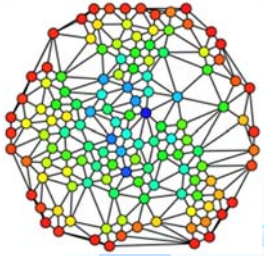


# Tipizzazione

- E’ possibile assegnare alle risorse un tipo (rdf:type) che appartiene ad uno schema di meta informazioni:

```
<rdf:Description rdf:about="http://www.myhost.org/~mrossi">
    <s:Autore>
        <rdf:Description rdf:about=”http://people.com/id/1234">
        <rdf:type rdf:resource="/myschema.rdf#Persona"/>
            <s:Nome>Mario Rossi</s:Nome>
            <s:Email>m.rossi@xxx.it </s:Email>
        </rdf:Description>
    </s:Autore>
</rdf:Description>
```

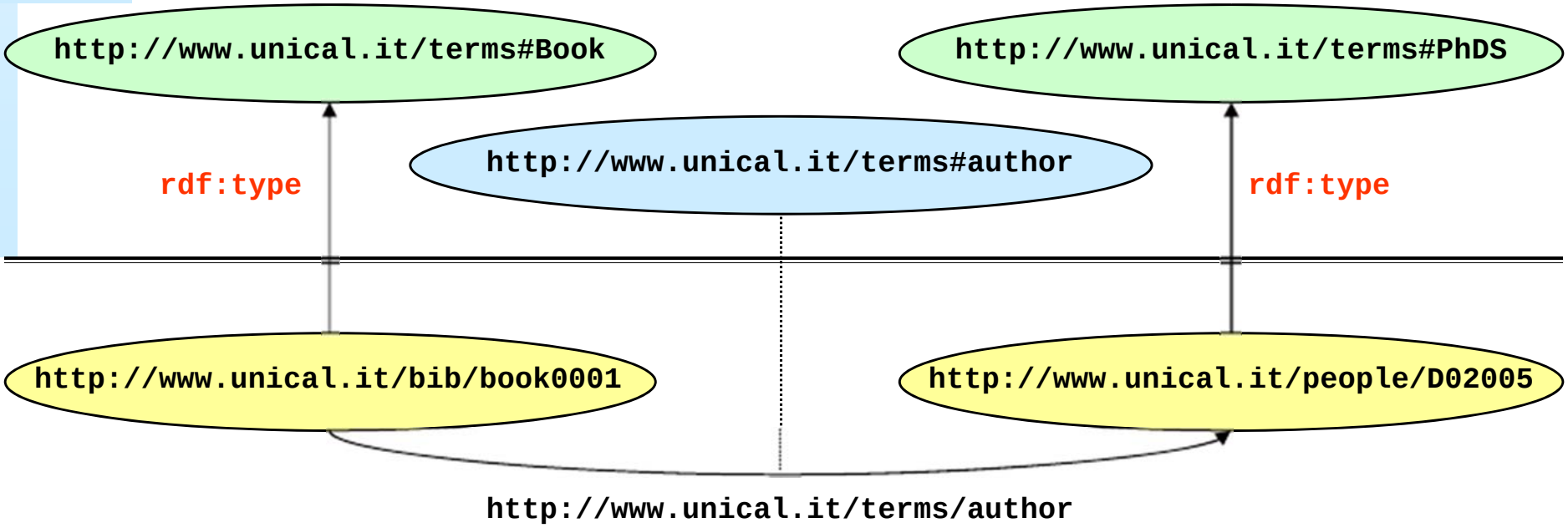
- L’attributo rdf:type specifica l’URI di definizione del tipo

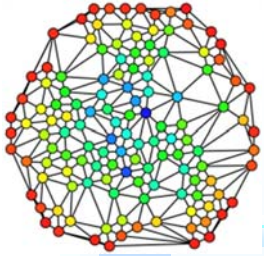


# Uno schema per RDF

RDF fornisce un modo per descrivere generiche asserzioni su risorse e proprietà

È spesso necessario indicare il fatto che **queste si riferiscono a particolari tipi** di risorse ed usano specifiche proprietà





# RDF Schema (RDFS)

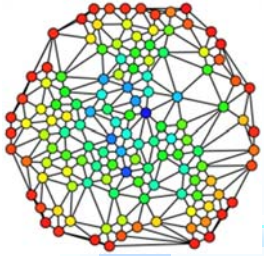
Per descrivere le **classi** e le **relazioni** *(utilizzate per costruire il particolare modello RDF)* si utilizza ancora RDF

Il particolare vocabolario è definito dall'*RDF Vocabulary Description Language*: RDF-Schema

Non un vocabolario specifico per l'applicazione (***`terms:Book`***, ***`terms:PhDS`***, ***`terms:author`***, *ecc.)*…

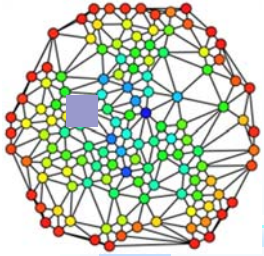
… ma un **meccanismo** per specificare **classi**, **proprietà** e costruire lo specifico **vocabolario**

RDF(S) definisce un sistema di tipi *(semantici)* per RDF



# Classi e Proprietà (1)

- ***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)



# Il meta-modello RDFS

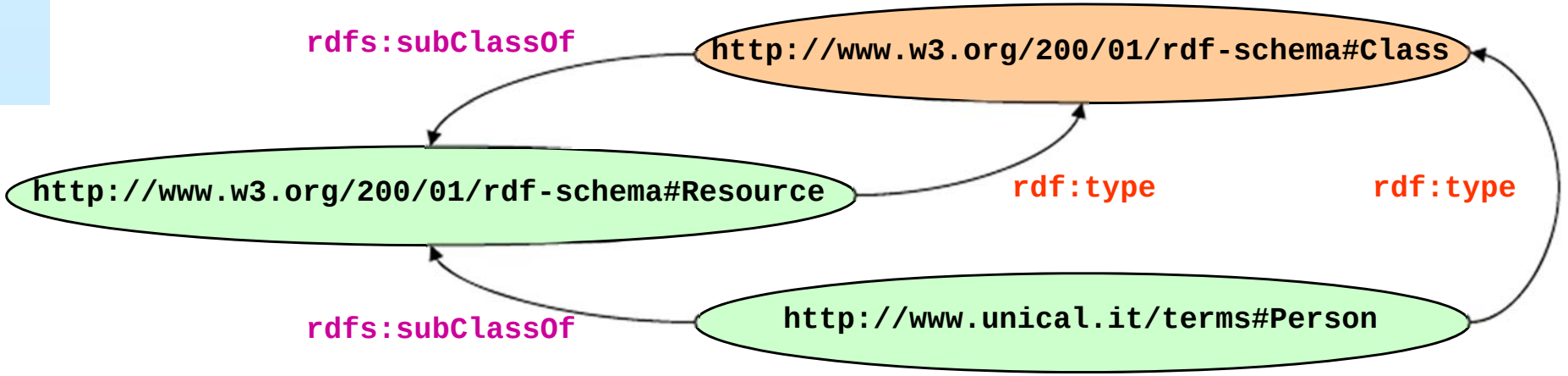
Le stesse risorse che usiamo per descrivere lo schema sono classi

**`rdfs:Class`** è la *(meta)* classe a cui appartengono tutte le classi

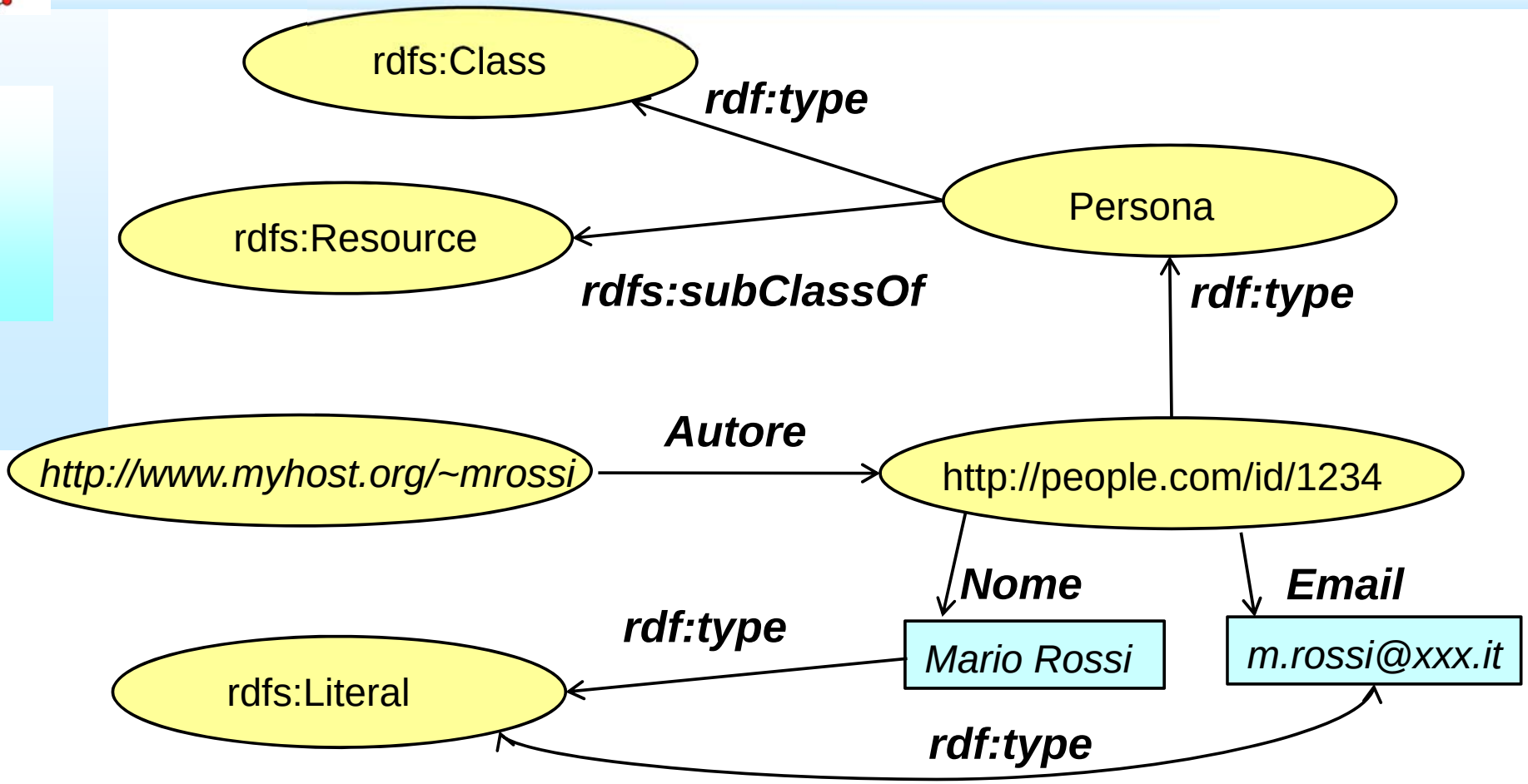
NON è l'analogo dell'*Object* di Java

**`rdfs:Resource`** è la classe "universo" da cui derivano tutte le classi di un modello. Se non indico alcun **`rdfs:subClassOf`** la classe deriva implicitamente da **`rdfs:Resource`**

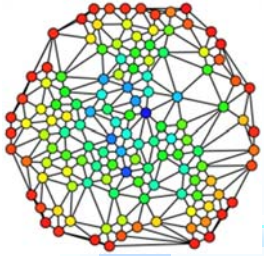
È l'analogo dell'*Object* di Java



# Classi e Proprietà (2)




```
<rdf:Description rdf:ID="Persona">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdfschema#Resource"/>
</rdf:Description>
```



# Classi (1)

Le classi sono aggregati di individui

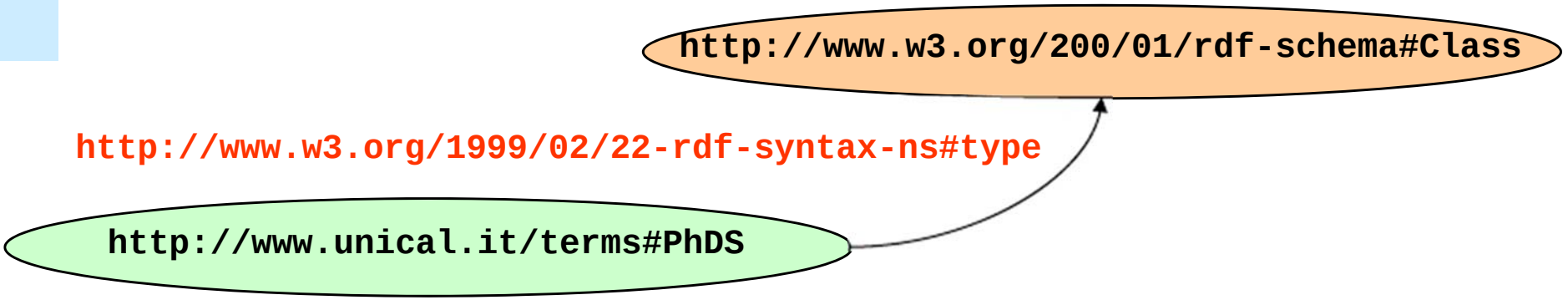
Ogni classe rappresenta un tipo di risorsa su cui si costruisce il modello RDF

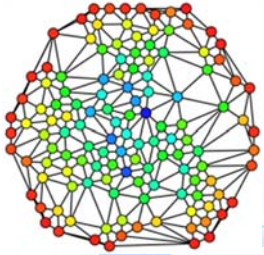
Il *namespace* di riferimento è

**`http://www.w3.org/2000/01/rdf-schema#`**

Ogni classe è una risorsa in relazione **`rdf:type`** con la risorsa

**`http://www.w3.org/2000/01/rdf-schema#Class`**





# Classi (2)

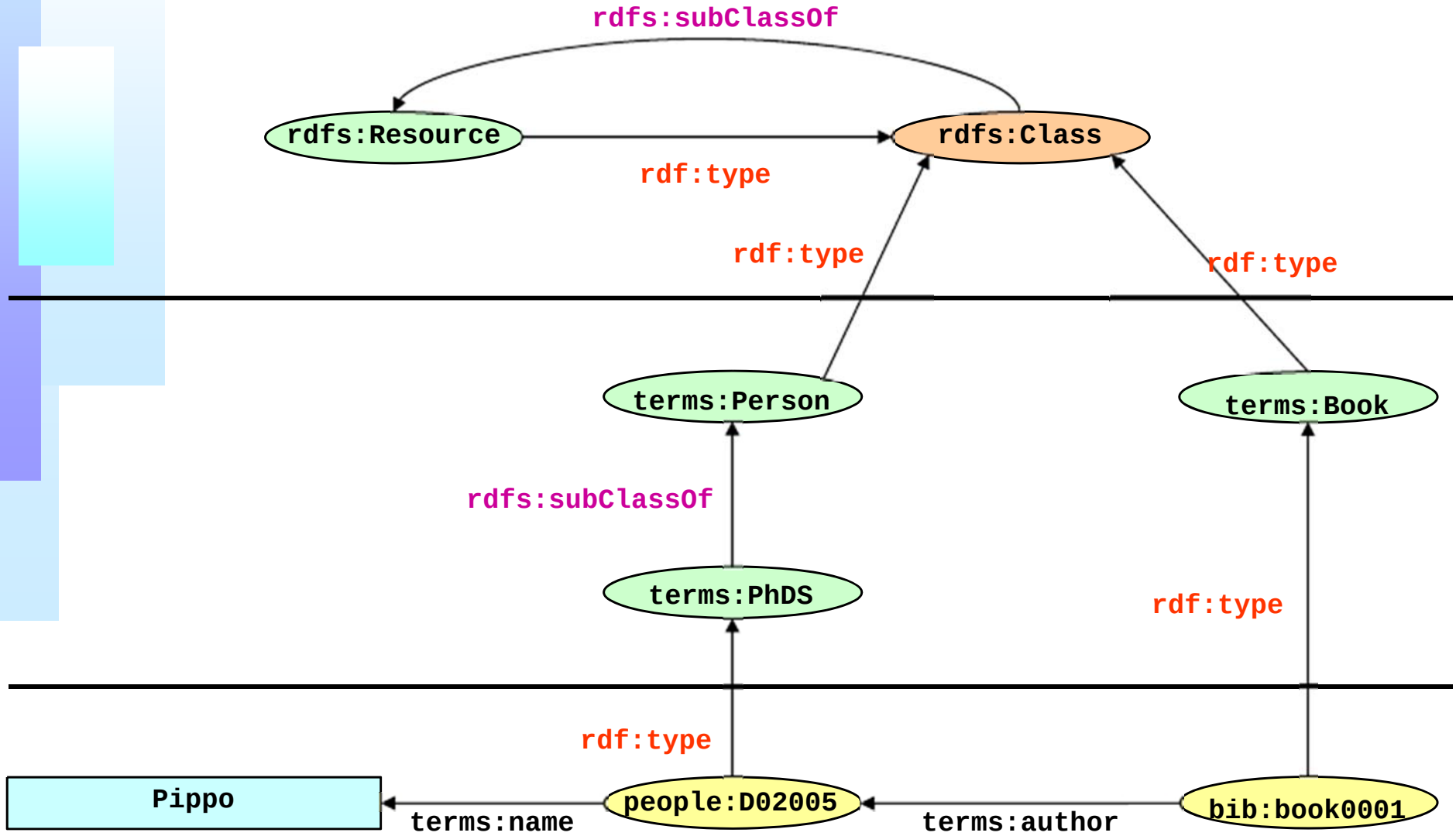
Esempio

```
<rdf:RDF xmlns:terms="http://www.unical.it/terms#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 <rdf:Description rdf:about="http://www.unical.it/terms#PhDS">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 </rdf:Description>
</rdf:RDF>
```

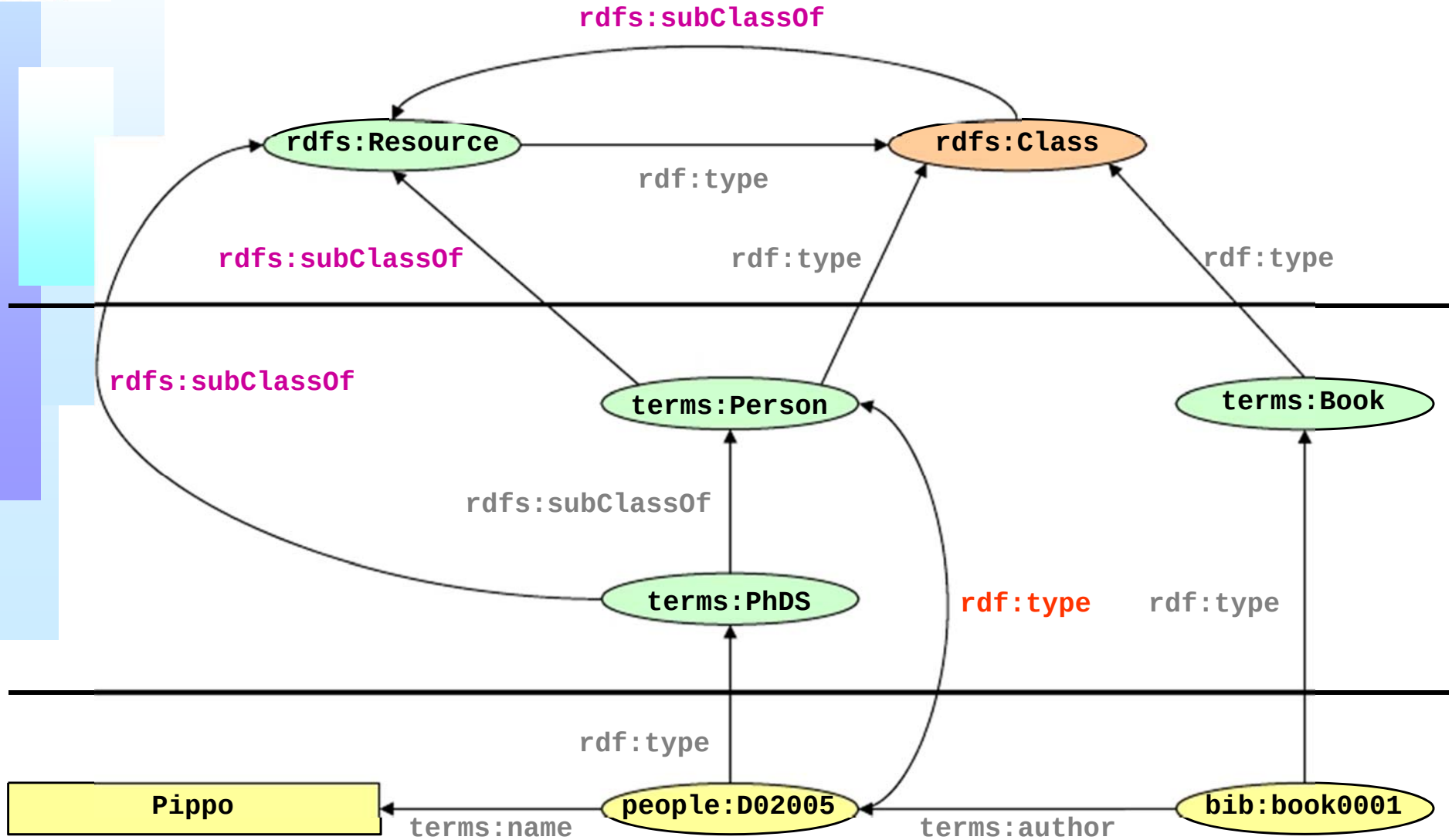
Forma abbreviata di RDF ed **`rdf:ID`** per riferirmi alla descrizione locale *(occorre strutturare correttamente il ns)*

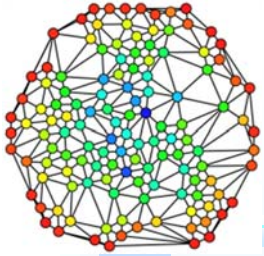
```
<rdf:RDF xml:base="http://www.unical.it/terms"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 <rdfs:Class rdf:ID="PhDS"/>
</rdf:RDF>
```

# Istanze – classi - metaclassi



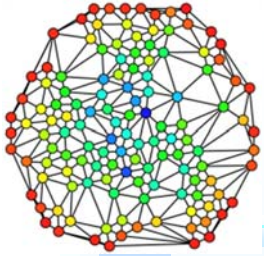
# Esempio di reasoning





# Proprietà (1)

- **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



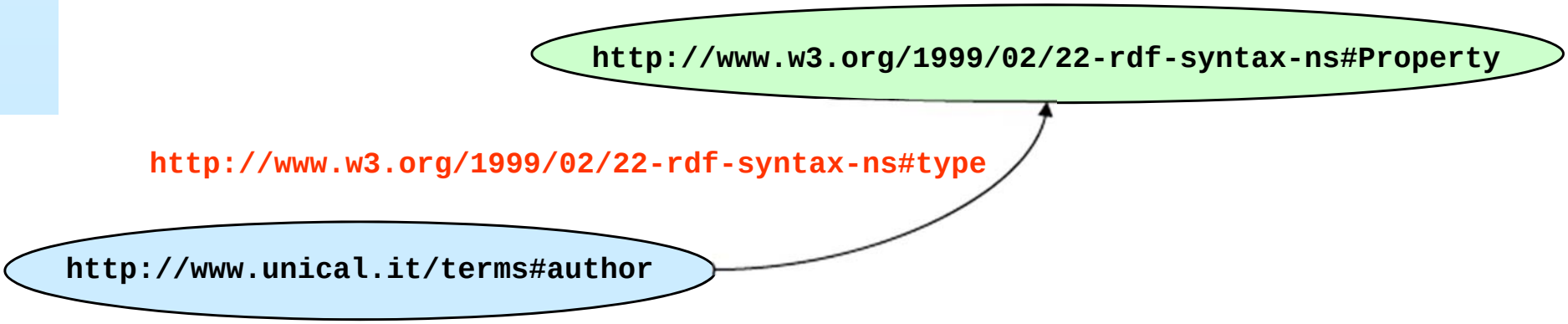
# Proprietà (2)

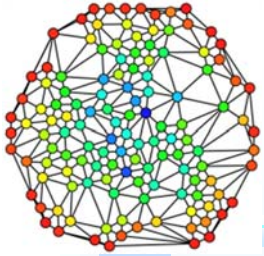
Oltre a descrivere le classi a cui appartengono gli oggetti del modello abbiamo bisogno di descrivere specifiche proprietà

Ogni proprietà RDF è istanza della classe predefinita **rdf:Property**

```
<rdf:Property rdf:ID="author" />
```

Es: la relazione (proprietà) **author** sussiste tra un **libro** ed una **persona**





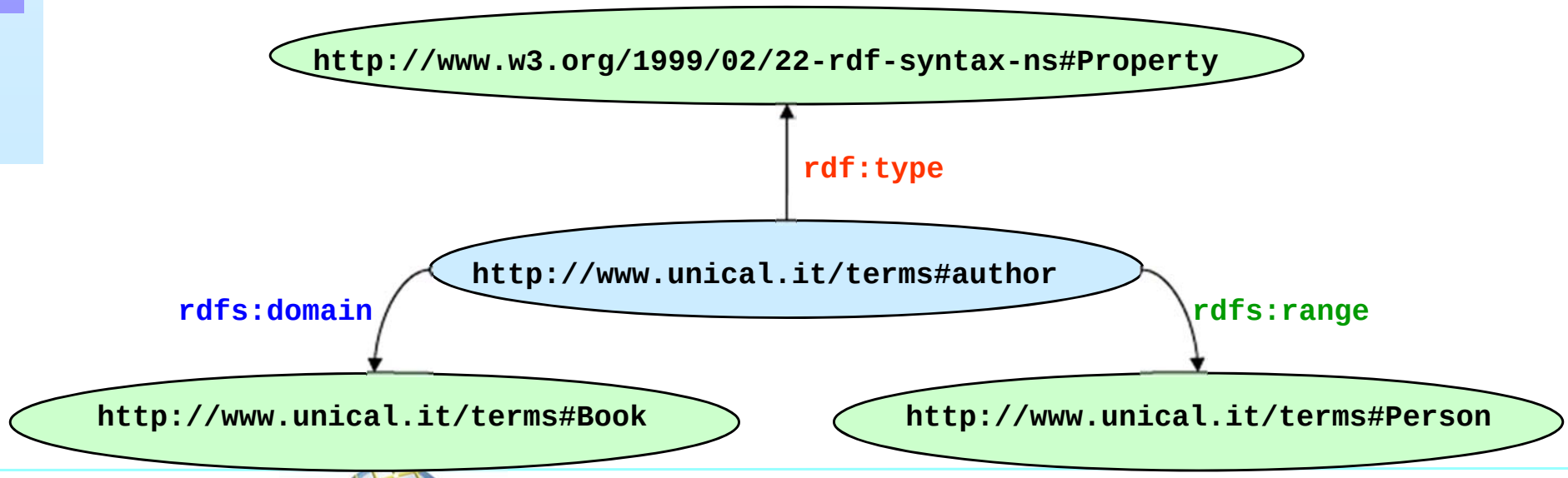
# Domain e Range (1)

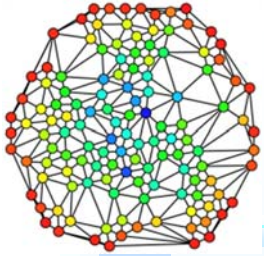
Legame (o vincoli) tra **classi** e **proprietà**

RDF(S) fornisce anche un vocabolario per descrivere come ci si aspetta che proprietà e classi si combinino tra di loro

Proprietà predefinite:

- rdfs:domain (dominio) Usato come predicato di una risorsa r, indica le classi (soggetto) a cui può essere applicata r
- rdfs:range (codominio) Usato come predicato di una risorsa r, indica le classi che saranno oggetto di un' asserzione che ha r come predicato

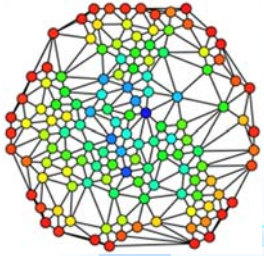




# Domain e Range (2)

```
<rdf:Description rdf:ID="Autore">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
    ns#Property"/>
    <rdfs:domain rdf:resource="#Book"/>
    <rdfs:range rdf:resource="#Person"/>
</rdf:Description>
```

```
<rdf:Description rdf:ID="Book">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-
    schema#Resource"/>
</rdf:Description>
<rdf:Description rdf:ID="Persona">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf
    rdf:resource="http://www.w3.org/2000/01/rdfschema#Resource"/>
</rdf:Description>
```



# Link utili

- http://www.w3.org/XML
- http://www.w3.org/2004/11/uri-iripressrelease.html.en
- http://www.w3.org/TR/wsdl#_document-s
- http://www.w3.org/2004/02/skos/
- http://www.w3.org/TR/wsdl
- https://www.mat.unical.it/informatica/Gestione%20della%20Conoscenza?action=AttachFile&do=view&target=GC060704101.pdf
- http://www.cs.unibo.it/~fabio/corsi/ltw01/slides/19-RDF/19-RDF.pdf