Laboratorio di Basi di Dati per Bioinformatica

Laurea in Bioinformatica

Docente: Carlo Combi

Email: carlo.combi@univr.it

Lezione 10

Il linguaggio XML

eXtensible Markup Language

- XML è un linguaggio di marcatura proposto dal W3C
- XML definisce una sintassi generica per contrassegnare i dati di un documento elettronico con marcatori (tag) semplici e leggibili
- La sintassi XML viene utilizzata in contesti molto diversi:
  - ✓ pagine web
  - ✓ scambio di dati elettronici
  - ✓ grafica vettoriale
  - ✓ cataloghi di prodotti
  - ✓ sistemi di gestione di messaggi vocali
  - ✓ …
- http://www.w3c.org/XML

XML: evoluzione

- 1986: Standard Generalized Markup Language (SGML)
  - ✓ Linguaggio di marcatura strutturato, per la rappresentazione elettronica di documenti di tipo testuale
- 1995: HyperText Markup Language (HTML)
  - ✓ Applicazione di SGML che permette di descrivere come il contenuto di un documento verrà presentato
- 1998: eXtensible Markup Language (XML)
  - ✓ Versione “leggera” di SGML, che consente una formattazione semplice e molto flessibile.

HTML vs XML

- HTML
  - Insieme fisso di tag
  - Descrizione di come il documento verrà presentato
  - Usato solo per la costruzione di pagine web

- XML
  - Insieme non fisso di tag: i tag possono essere personalizzati
  - Descrizione del contenuto del documento
  - Usato in molti domini diversi

HTML vs XML: esempio

```
<h1>
    Essential XML: <br>
     Oltre il Markup
</h1>

<ul>
    <li>
     Don Box
    </li>
    <li>
     Aaron Skonnard
    </li>
    <li>
     John Lam
    </li>

</ul>

<i>
    Addison-Wesley
</i>
```

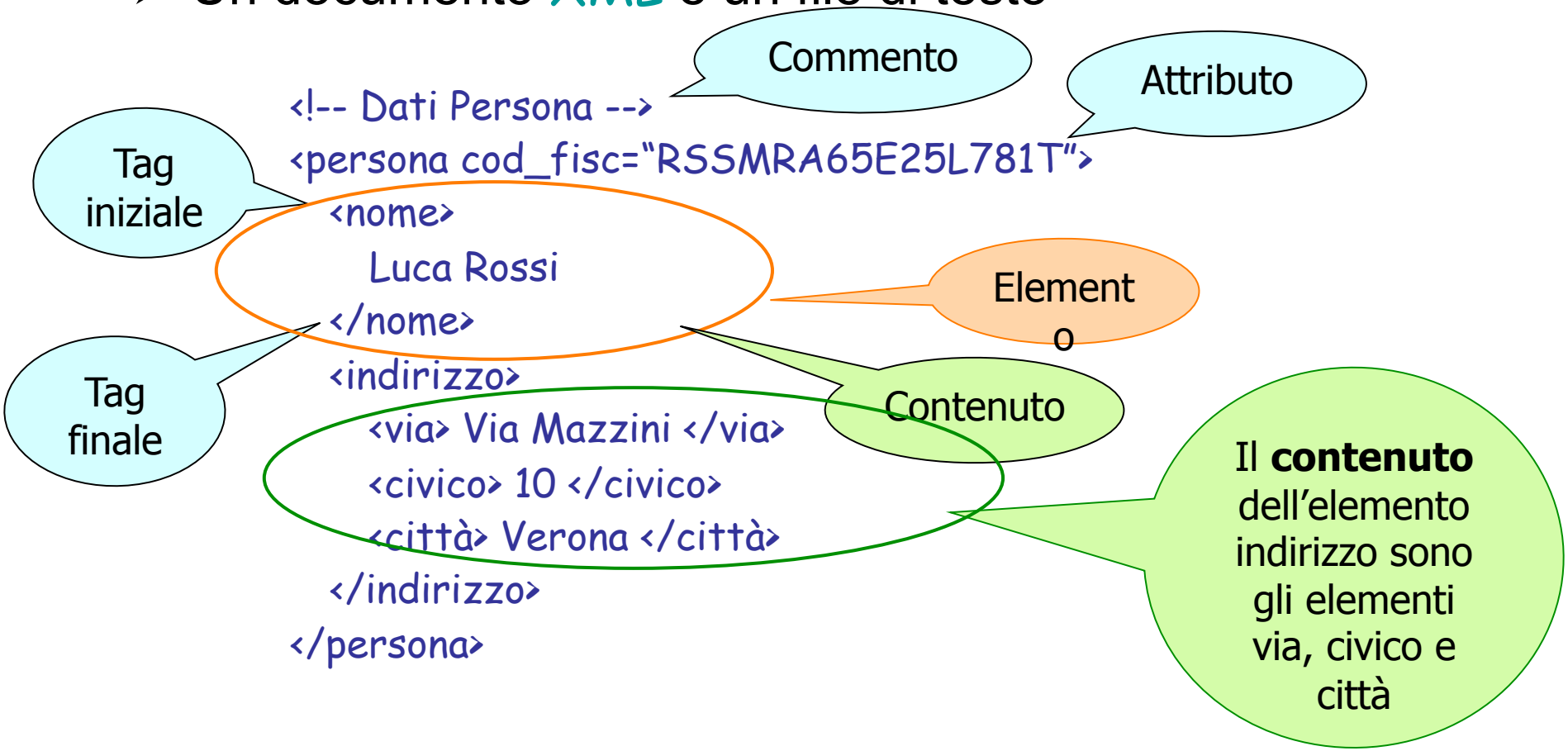
```
<titolo>
    Essential XML:
          Oltre il Markup
</titolo>

<autore>
    Don Box
</autore>
<autore>
    Aaron Skonnard
</autore>
<autore>
    John Lam
</autore>
<casa editrice>
    Addison-Wesley
</casa editrice>
```

Documenti *ben formati*

- XML è più restrittivo di HTML per quanto riguarda il posizionamento dei tag e il modo in cui vengono scritti
- Ogni documento XML deve essere *ben formato*
  - ✓ Ha una sola radice
  - ✓ L'annidamento dei tag deve essere corretto
  - ✓ Tutti i tag aperti devono essere chiusi
  - ✓ I valori degli attributi devono essere specificati tra virgolette

Esempio di documento XML: persona.xml

- Un documento XML è un file di testo



Sintassi dei tag

- Tag iniziale
  <nome>
- Tag finale
  </nome>
- Elementi vuoti: sono elementi privi di contenuto
  <maschio></maschio> oppure <maschio/>

- XML è sensibile alla differenza tra maiuscole e minuscole (è *case-sensitive*)

Contenuto di un elemento

- Contenuto di tipo carattere

```
<nome> Luca Rossi </nome>
```

- Contenuto costituito da altri elementi (figli)

```
<indirizzo>
    <via> Via Mazzini </via>
    <civico> 10 </civico>
    <città> Verona </città>
</indirizzo>
```

Contenuto di un elemento (2)

- Contenuto misto

```
<dati_anagrafici> Il signor
  <persona>
    <nome> Mario Rossi </nome> vive in
    <indirizzo>
        <via> Via Mazzini </via>
        <civico> 10 </civico>
        <città> Verona </città>
    </indirizzo>
  </persona>
</dati_anagrafici>
```

Attributi

- Un attributo consiste in una coppia nome-valore associata al tag iniziale di un elemento

```
<persona cod_fisc="RSSMRA65E25L781T">
```

- Si possono usare anche gli apici singoli

```
<persona cod_fisc='RSSMRA65E25L781T'>
```

Nomi XML

- Possono essere costituiti da qualsiasi carattere alfanumerico
- Possono includere:
  - ✓ Underscore _
  - ✓ Trattino -
  - ✓ Punto .
- Possono iniziare solo con lettere, ideogrammi o con il carattere underscore
- Non possono includere:
  - ✓ Altri caratteri di punteggiatura
  - ✓ Virgolette
  - ✓ Apostrofi
  - ✓ $ e %
  - ✓ < e >
  - ✓ Spazi

Esempi di nomi XML

- Nomi ben formati:
  - ✓ <Nome_persona> Maria </Nome_persona>
  - ✓ <Giorno-Mese-Anno> 10/06/2004 </Giorno-Mese-Anno>
  - ✓ <_indirizzo> Via Stella 10 </_indirizzo>

- Nomi NON ben formati:
  - ✓ <Nome persona> Maria </Nome persona>
  - ✓ <Giorno/Mese/Anno> 10/06/2004 </Giorno/Mese/Anno>
  - ✓ <citta'> Verona </citta'>
  - ✓ <1_telefono> 045 1234567 </1_telefono>
  - ✓ <%vendita> 20 </%vendita>

La dichiarazione XML

- I documenti XML dovrebbero iniziare con una dichiarazione XML

<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>

Document Type Definition (DTD)

- Un DTD descrive la struttura di un documento XML:
  - ✓i tag ammessi
  - ✓le regole di annidamento dei tag

- I DTD vengono utilizzati per la validazione di un documento XML
  - ✓Un documento XML è valido quando è conforme ad un dato DTD

Esempio di DTD: elenco.dtd

```
<!ELEMENT elenco (libro+)>
<!ELEMENT libro (titolo,prezzo?)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT prezzo (#PCDATA)>
```

- ✓ Ogni riga rappresenta la dichiarazione di elemento
- ✓ L'elemento elenco contiene uno o più elementi libro
- ✓ L'elemento libro contiene un titolo e zero o un prezzo
- ✓ titolo e prezzo possono contenere solo testo

Dichiarazione di elementi

<!ELEMENT nome_elemento (modello_di_contenuto)>

- ✓ Il modello di contenuto può essere:
  - #PCDATA
  - Elementi figli
  - Sequenze
  - Misto
  - EMPTY
  - ANY

Modello di contenuto: #PCDATA

- Specifica che l'elemento deve contenere solamente dati di tipo carattere

- L'elemento non può contenere elementi figli di alcun tipo

Esempio: L'elemento titolo deve contenere solo testo

<!ELEMENT titolo (#PCDATA)>

Modello di contenuto: Elementi Figli

- Specifica che un elemento deve contenere esattamente un elemento figlio di un determinato tipo

Esempio: L'elemento libro deve contenere esattamente un elemento titolo (né più né meno di uno)

<!ELEMENT libro (titolo)>

Modello di contenuto: Sequenze

- Specifica che un elemento deve contenere più elementi figli
- I figli vengono elencati in una sequenza separati da virgole
- Gli elementi figli devono apparire all'interno dell'elemento padre nell'ordine specificato

Esempio: L'elemento libro deve contenere un elemento titolo e un elemento prezzo

<!ELEMENT libro (titolo,prezzo)>

Il numero di figli

- Per indicare quante istanze di un elemento possono apparire si usa:
  - ✓ ? zero o una istanza
  - ✓ * zero o più istanze
  - ✓ + una o più istanze

Esempio: L'elemento libro deve contenere un elemento titolo, uno o più elementi autore e zero o un elemento prezzo

<!ELEMENT libro (titolo,autore+,prezzo?)>

Scelte

- Una scelta è un elenco di nomi di elementi (due o più) che possono apparire nell'elemento padre
- Gli elementi della scelta vengono separati da barre verticali
- L'elemento padre non può contenere entrambi gli elementi elencati nella scelta

Esempio: L'elemento contatto può contenere o un elemento telefono_casa o un elemento telefono_ufficio

```
<!ELEMENT contatto (telefono_casa |
telefono_ufficio)>
```

Parentesi

- Per combinare scelte e sequenze si possono usare le parentesi

Esempio: L'elemento indirizzo deve contenere un elemento tra via e piazza e un elemento civico

```
<!ELEMENT indirizzo ((via | piazza), civico)>
```

L'elemento persona può contenere un elemento nome e un elemento cognome o un elemento cognome e un elemento nome

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

Modello di contenuto: Misto

- Specifica che un elemento deve contenere sia dati di tipo carattere che elementi figli
- Non è possibile specificare:
  - ✓l'ordine in cui appariranno
  - ✓quante istanze di essi appariranno
- L'elemento #PCDATA deve essere il primo della lista

Esempio: L'elemento libro può contenere dati di tipo carattere e elementi figli titolo e prezzo

<!ELEMENT libro (#PCDATA|titolo|prezzo)*>

Modello di contenuto: EMPTY

- Specifica che un elemento deve essere vuoto e quindi senza nessun tipo di contenuto

Esempio: L'elemento immagine deve essere un elemento vuoto

<!ELEMENT immagine EMPTY>

Modello di contenuto: ANY

- Specifica che un elemento può contenere qualsiasi cosa
  - ✓ Testo
  - ✓ Elementi figli
  - ✓ Contenuto misto
- Gli elementi che appaiono come figli devono comunque esser stati dichiarati

Esempio: L'elemento pagina può contenere qualsiasi cosa

<!ELEMENT pagina ANY>

Dichiarazione di attributi

```
<!ATTLIST nome_elemento
        nome_attributo1 CDATA #REQUIRED
        nome_attributo2 CDATA #IMPLIED
        nome_attributo3 CDATA #FIXED valore>
```

Esempio: L'elemento immagine ha un attributo codice obbligatorio ed un attributo titolo opzionale

```
<!ATTLIST immagine codice CDATA #REQUIRED
                   titolo CDATA #IMPLIED>
```

Valori di default per gli attributi

- #IMPLIED: il valore dell'attributo è opzionale
- #REQUIRED: il valore dell'attributo è obbligatorio
- #FIXED: il valore dell'attributo è costante e immutabile
- Literal: indica il valore di default sotto forma di stringa tra apici

Tipi di Attributi

- ✓ CDATA
- ✓ NMTOKEN
- ✓ NMTOKENS
- ✓ Enumerazione
- ✓ ID
- ✓ IDREF
- ✓ IDREFS
- ✓ ENTITY
- ✓ ENTITIES
- ✓ NOTATION

Tipi di Attributi (1)

- CDATA: può contenere qualsiasi tipo di stringa accettabile in un documento XML ben formato

```
<!ATTLIST immagine titolo CDATA #IMPLIED>

<immagine titolo="tramonto"/>
```

- NMTOKEN: può iniziare con qualsiasi carattere

```
<!ATTLIST libro anno_pubblicazione NMTOKEN #REQUIRED>

<libro anno_pubblicazione="1950d.c."/>
```

- NMTOKENS: può contenere uno o più token

```
<!ATTLIST esibizione date NMTOKENS #IMPLIED>

<esibizione date="10-07-2004 17-07-2004 24-07-2004"/>
```

Tipi di Attributi (2)

- Enumerazione: lista di tutti i possibili valori assegnabili all'attributo

```
<!ATTLIST data giorno (1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|
                       17|18|19|20|21|22|23|24|25|26|27|28|29|
                       30|31) #REQUIRED
           mese (gennaio|febbraio|marzo|aprile|maggio|
                 giugno|luglio|agosto|settembre|ottobre|
                 novembre|dicembre) #REQUIRED
           anno (2003|2004|2005|2006|2007) #REQUIRED>
```

```
<data giorno="15" mese="agosto" anno="2007"/>
```

Tipi di Attributi (3)

- ID: contiene un nome XML che ha valore univoco all'interno del documento

```
<!ATTLIST persona cod_fisc ID #REQUIRED>

<persona cod_fisc="RSSMRA65E25L781T"/>
```

- IDREF: riferimento all'attributo di tipo ID di un elemento del documento

```
<!ATTLIST persona cod_fisc ID #REQUIRED>
<!ATTLIST docente persona IDREF #REQUIRED>

<persona cod_fisc="RSSMRA65E25L781T"/>
<docente persona="RSSMRA65E25L781T"/>
```

Tipi di Attributi (4)

- IDREFS: contiene una lista di nomi XML ognuno dei quali deve essere un ID valido di un elemento del documento

```
<!ATTLIST persona cod_fisc ID #REQUIRED>
<!ATTLIST sposi persone IDREFS #REQUIRED>

<persona cod_fisc="RSSMRA65E25L781T"/>
<persona cod_fisc="BNCFRN63D45L781T"/>
<sposi persone="RSSMRA65E25L781T
   BNCFRN63D45L781T"/>
```

Validazione

- Un documento XML per il quale è richiesta la validazione deve includere un riferimento al DTD con cui deve essere messo a confronto
- Il riferimento deve essere fornito nella dichiarazione del tipo di documento

  <!DOCTYPE elenco SYSTEM "http://ibiblio.org/xml/dtds/elenco.dtd">
- Questa dichiarazione afferma che elenco è la radice del documento e il DTD si trova all'URL http://ibiblio.org/xml/dtds/elenco.dtd
- La dichiarazione del tipo di documento si trova dopo la dichiarazione XML

Dichiarazione del tipo di documento

- Se il DTD di riferimento si trova ad un certo URL:

  <!DOCTYPE elenco SYSTEM "http://ibiblio.org/xml/dtds/elenco.dtd">

- Se il DTD si trova allo stesso URL del documento:

  <!DOCTYPE elenco SYSTEM "/dtds/elenco.dtd">

- Se il DTD si trova nella stessa directory del documento

  <!DOCTYPE elenco SYSTEM "elenco.dtd">

Esempio di documento valido

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE elenco SYSTEM "elenco.dtd">
<elenco>
   <libro>
    <titolo> XML Guida di riferimento </titolo>
    <prezzo> 35 </prezzo>
   </libro>

   <libro>
    <titolo> Basi di dati </titolo>
   </libro>
</elenco>
```

Dichiarazione del tipo di documento (2)

- Il DTD può essere inserito direttamente nella dichiarazione del tipo di documento

```
<?xml version="1.0"?>
<!DOCTYPE elenco [
    <!ELEMENT elenco (libro+)>
    <!ELEMENT libro (titolo,prezzo?)>
    <!ELEMENT titolo (#PCDATA)>
    <!ELEMENT prezzo (#PCDATA)>
] >
<elenco>
    <libro><titolo> XML Guida di riferimento </titolo>
        <prezzo> 35 </prezzo>
    </libro>
    <libro><titolo> Basi di dati </titolo>
    </libro>
</elenco>
```

Validazione di un documento

- Validatori online:
  - Servizio di validazione del W3C

    http://validator.w3.org/
  - Verificatore di documenti XML/XHTML e altro rispetto a DTD o XMLSchema

    http://www.validome.org/xml
- Per utilizzare questi validatori i documenti e i DTD associati possono essere su un server Web accessibile al pubblico o inviati all'applicazione in vari modi.

Esempio: file prova.xml

http://validator.w3.org/

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<!DOCTYPE elenco
[<!ELEMENT elenco (libro+)>
<!ELEMENT libro (titolo,prezzo?)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT prezzo (#PCDATA)>
]>
<elenco>
    <libro>
        <titolo> XML Guida di riferimento </titolo>
        <prezzo> 35 </prezzo>
    </libro>
    <libro>
        <titolo> Basi di dati </titolo>
    </libro>
</elenco>
```

Esempio: file libri1.xml ed elenco.dtd

http://validator.w3.org/

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<!DOCTYPE elenco SYSTEM 'elenco.dtd'>
<elenco>
      <libro>
            <titolo> XML Guida di riferimento </titolo>
            <prezzo> 35 </prezzo>
      </libro>
      <libro>
            <titolo> Basi di dati </titolo>
      </libro>
</elenco>
```


```
<!ELEMENT elenco (libro+)>
<!ELEMENT libro (titolo,prezzo?)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT prezzo (#PCDATA)>
```

Esempio: file libri2.xml http://validator.w3.org/

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<!DOCTYPE elenco
[<!ELEMENT elenco (libro+)>
<!ELEMENT libro (titolo,prezzo?)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT prezzo (#PCDATA)>
]>
<elenco>
    <libro>
        <titolo> XML Guida di riferimento </titolo>
        <prezzo> 35 </prezzo>
    </libro>
    <libro>
        <titolo> Basi di dati </titolo>
        <titolo> Basi di dati2 </titolo>
    </libro>
</elenco>
```

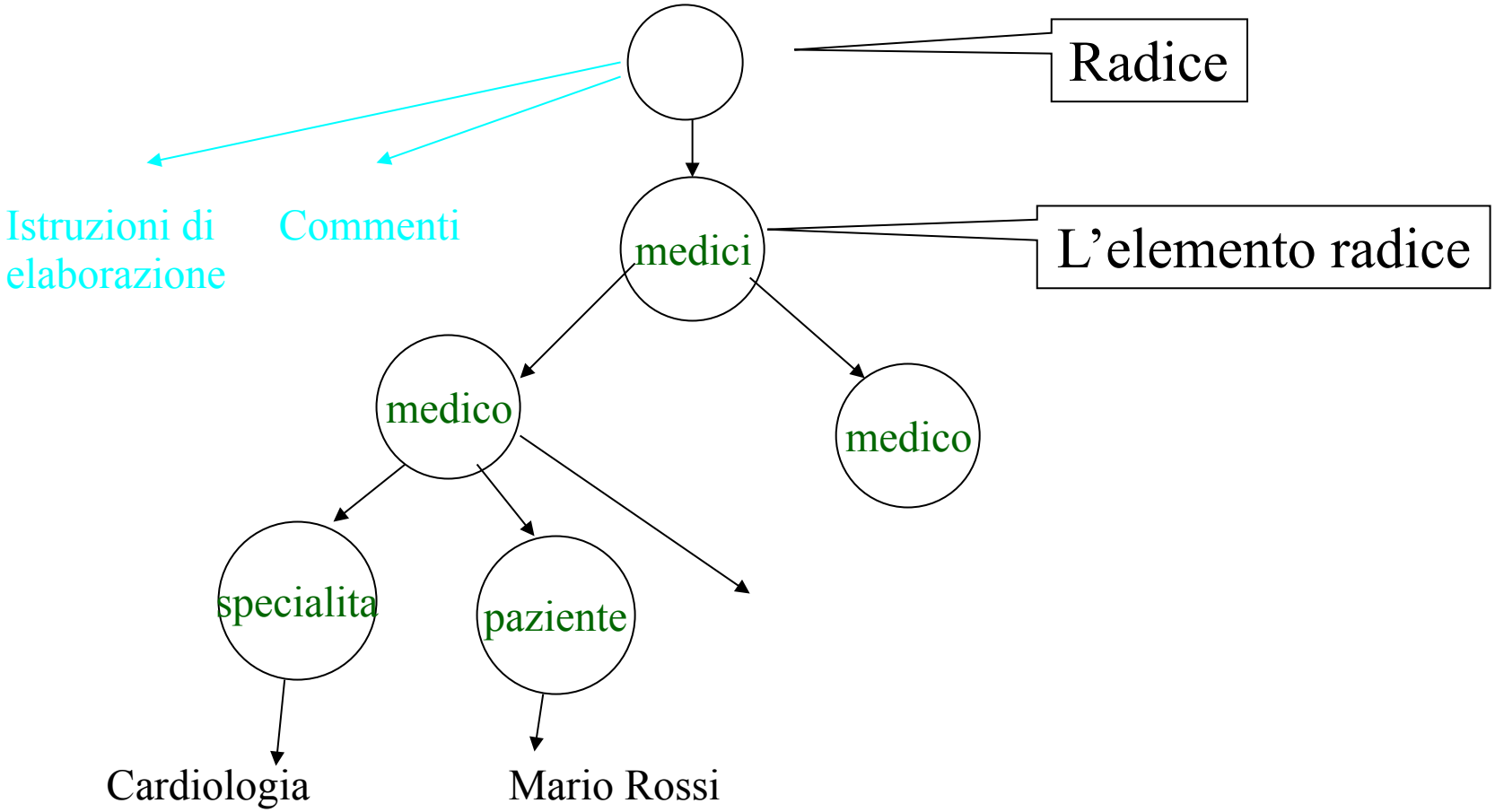
XPath

Un linguaggio per
l'interrogazione di documenti
XML

Un esempio per interrogazioni XPath

```
<medici>
    <medico>  <specialita> Cardiologia</specialita>
                <paziente> Mario Rossi</paziente>
                <paziente> <cognome> Rossi</cognome>
                            <nome> Maria</nome>
                </paziente>
                <paziente> Ada Verdi </paziente>
                <CognomeNome> Carlo Neri </CognomeNome>
                <inizioAtt> 1995 </inizioAtt>
    </medico>
    <medico eta="55">
                <specialita> Pediatria</specialita>
                <paziente>John Smith </paziente>
                <CognomeNome> Bianchi Giorgio</CognomeNome>
                <inizioAtt> 1998 </inizioAtt>
    </medico>
</medici>
```

Il modello dei dati di XPath



XPath: Espressioni semplici

/medici/medico/inizioAtt

Risultato: <inizioAtt> 1995 </inizioAtt>
<inizioAtt> 1998 </inizioAtt>

/medici/medicobase/inizioAtt

Risultato: vuoto (non ci sono medici di base)

XPath: Espressioni complesse

//paziente

Risultato:<paziente> Mario Rossi </paziente>
<paziente> <cognome> Rossi </cognome>
<nome> Maria</nome> </paziente>
<paziente> Ada Verdi</paziente>
<paziente> John Smith </paziente>

/medici//cognome

Risultato: <cognome> Rossi </cognome>

Xpath: Funzioni

/medici/medico/paziente/text()

Risultato: Mario Rossi
Ada Verdi
John Smith

Rossi Maria non è presente perchè ha cognome, nome

Funzioni in XPath:

- text() = ritorna contenuti testo
- node() = ritorna un qualsiasi nodo (= * or @* or text())
- name() = ritorna il nome del tag corrente

Xpath: Wildcard

//paziente/*

Risultato: <cognome> Rossi </cognome>
<nome> Maria</nome>

* Ritorna un qualsiasi elemento

Xpath: nodi attributo

/medici/medico/@eta

Risultato: “55”

@eta significa che eta deve essere un attributo

Xpath: qualificatori

/medici/medico/paziente[cognome]

Risultato: <paziente> <cognome> Rossi </cognome>
<nome> Maria</nome>
</paziente>

Xpath: più qualificatori

/medici/medico/paziente[cognome][indirizzo[//cap][citta]]/
cognome

Risultato: <cognome> … </cognome>
<cognome> … </cognome>

Xpath: più qualificatori

/medici/medico[@eta< “60”]

/medici/medico[paziente/@eta < “25”]

/medici/medico[paziente/text()]

Xpath: Sommario

medici fa riferimento agli elementi medici

* fa riferimento a un qualunque elemento

/ fa riferimento all'elemento root

/medici fa riferimento a medici sotto root

medici/paper fa riferimento a paper in medici

medici//paper fa riferimento a paper in medici, a qualunque profondità

//paper fa riferimento a paper, a qualunque profondità

paper|medico fa riferimento a paper o a medico

@eta fa riferimento all'attributo eta

medici/medico/@eta fa riferimento all'attributo eta in medico, in medici

Xpath: Ulteriori dettagli

- Una espressione Xpath, p, stabilisce una relazione fra:
  - Un *context node*, e
  - Un nodo nell'*answer set*
- In altre parole, p denota una funzione:
  - S[p] : Nodes -> {Nodes}
- Examples:
  - paziente/nome
  - . = self
  - .. = parent
  - part/*/*/subpart/../name = part/*/*[subpart]/name

La radice e la radice

- <medici> <medicoBase> …. </medicoBase> <medicoBase> 2 <medicoBase> </medici>
- Medici è il "***document element***"
- La radice è sopra medici

- /medici = ritorna il "document element"
- / = ritorna la radice

- Perchè? ….

XML e PostgreSql

- Una tabella d'esempio:

```
CREATE TABLE table_exp(
id integer PRIMARY KEY,
created timestamp NOT NULL DEFAULT
   CURRENT_TIMESTAMP,
xperiment xml
);
```

XML e PostgreSql

```
INSERT INTO table_exp(id, xperiment) VALUES(2,
    '<experiment xmlns:xper="http://experiment.com"
    xper:xid="CCXX89 IT"><name>IT</name>
        <samples>
            <sample xper:id="111" xper:date="11-2-2010">
                <name>WBC</name><value>7000</value>
            </sample>
            <sample xper:id="115" xper:date="12-3-2010">
                <name>WBC</name><value>5000</value>
            </sample>
        </samples>
    </experiment>');
```

XML e PostgreSql

```
INSERT INTO table_exp(id, xperiment) VALUES(1,
    '<experiment xmlns:xper="http://experiment.com"
    xper:xid="CCXX88 IT"><name>IT</name>
        <samples>
            <sample xper:id="112" xper:date="12-2-2010">
                <name>SBP</name><value>160</value>
            </sample>
        </samples>
    </experiment>');
```

XML e PostgreSql

- Semplici query di esempio:

```
SELECT * FROM table_exp
WHERE (xpath('//sample/name/text()', xperiment))[1]::text = 'WBC';

SELECT id FROM table_exp
WHERE '160' = any((xpath('//sample/value/text()', xperiment))::text[]);
```

- Usando i namespace:

```
SELECT * FROM table_exp
WHERE (xpath('//sample/@xper:id', xperiment, ARRAY[ARRAY['xper',
    'http://experiment.com']]))::text like '%111%';
```

XML e PostgreSql

- per inserire in un attributo *xml* di postgresql un documento XML direttamente da file è necessario dichiarare una variabile che legga il file
  - \set *varname* `cat *filename*`
  - (attenzione agli apici "americani") in questo modo il contenuto del file viene copiato nella variabile.

XML e PostgreSql

- Poi si usa la variabile durante gli insert
  - INSERT INTO tablename (..., *xmldoc*, ...) VALUES (..., :*varname*, ...);
  - Attenzione, se si modifica il file è necessario ridefinire la variabile perché il file viene letto in fase di definizione della variabile. La variabile può essere usata anche per elaborare il doc XML usando gli altri comandi di postgresql per xml, (per esempio, xmlroot).

XML e PostgreSql

- Per i dettagli su PostgreSql e XML si veda:
  - http://www.postgresql.org/docs/8.4/static/functions-xml.html

Riferimenti

- Essential XML - Oltre il Markup

  Box Don, Skonnard Aaron, Lam John. Addison Wesley
- Data on the web: from relations to semistructured data and XML

  Serge Abiteboul,Peter Buneman,Dan Suciu. Morgan Kaufmann