Progetto di sistemi informatici 2004 - 2005

Tecnologie per XML

Paolo Papotti
papotti@dia.uniroma3.it
Lab. basi di dati

Cosa vedremo oggi

- Richiami di XML
  - Rappresentazione dati
  - Sintassi
  - Schemi
- Tecnologie per XML
  - Navigazione
  - Interrogazione
  - Trasformazione

Cosa vedremo domani

- Model management
- Chameleon

Dati strutturati vs semi-strutturati

- Dati strutturati
  - database relazionali
- Semi-strutturati
  - Grafi etichettati per rappresentare informazioni
- **XML** è una convenzione per rappresentare come testo il sottoinsieme degli alberi

XML cenni

- Progetto del W3C dal 1998 per il web semantico (sostituto dell'HTML)
- Sintassi basata su marcatori (*tag*)
- Modello basato su alberi
  - Elementi sui nodi
  - Attributi e valori sulle foglie
  - Visitati in preordine (entro nel primo ramo, lo visito tutto, passo al successivo)

Sintassi per alberi

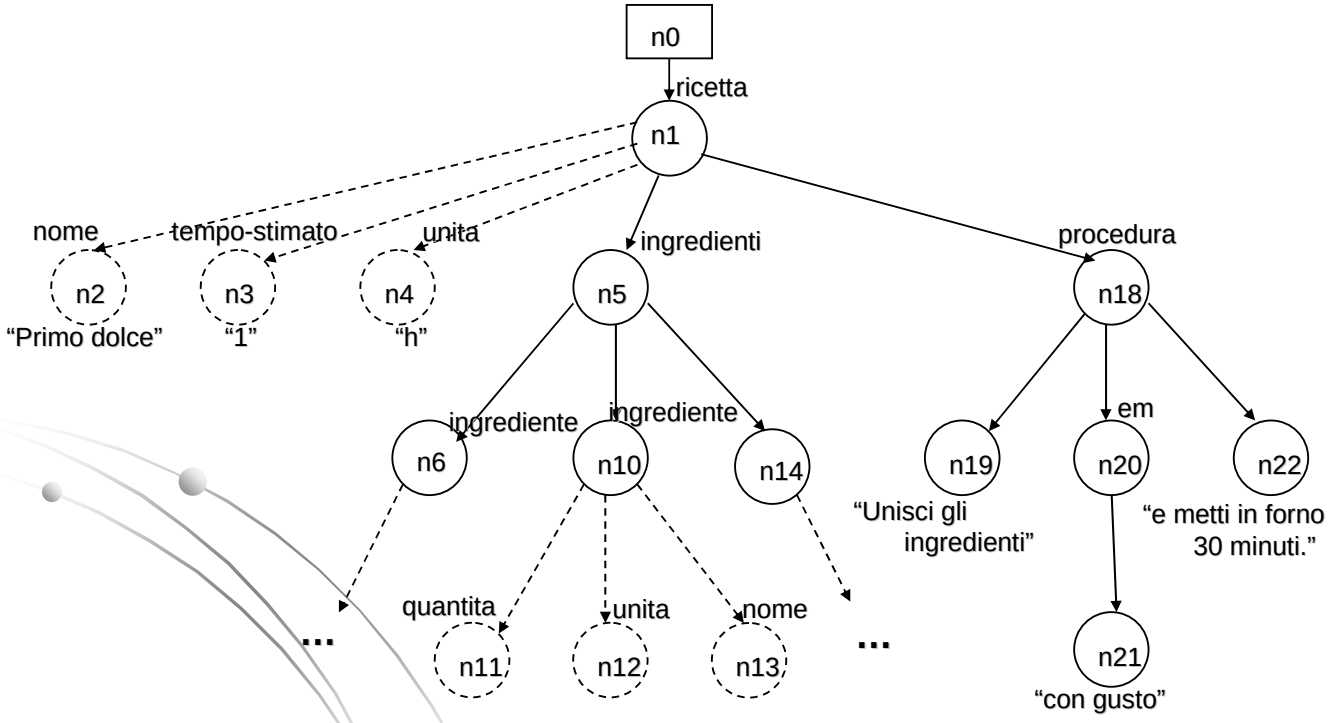
"It is intriguing that something as bland as a syntax for trees has become one of the leading buzzwords of the Internet era."

Klarlund, Schwentick, Suciu

Successo XML

- La struttura ad albero è sufficientemente generale da adattarsi a tutte le applicazioni, ma sufficientemente ben caratterizzata per la realizzazione di parser generici
- Es applicazione: informazioni sul web (SOAP), descrizione di web service (WSDL), …

Esempio1



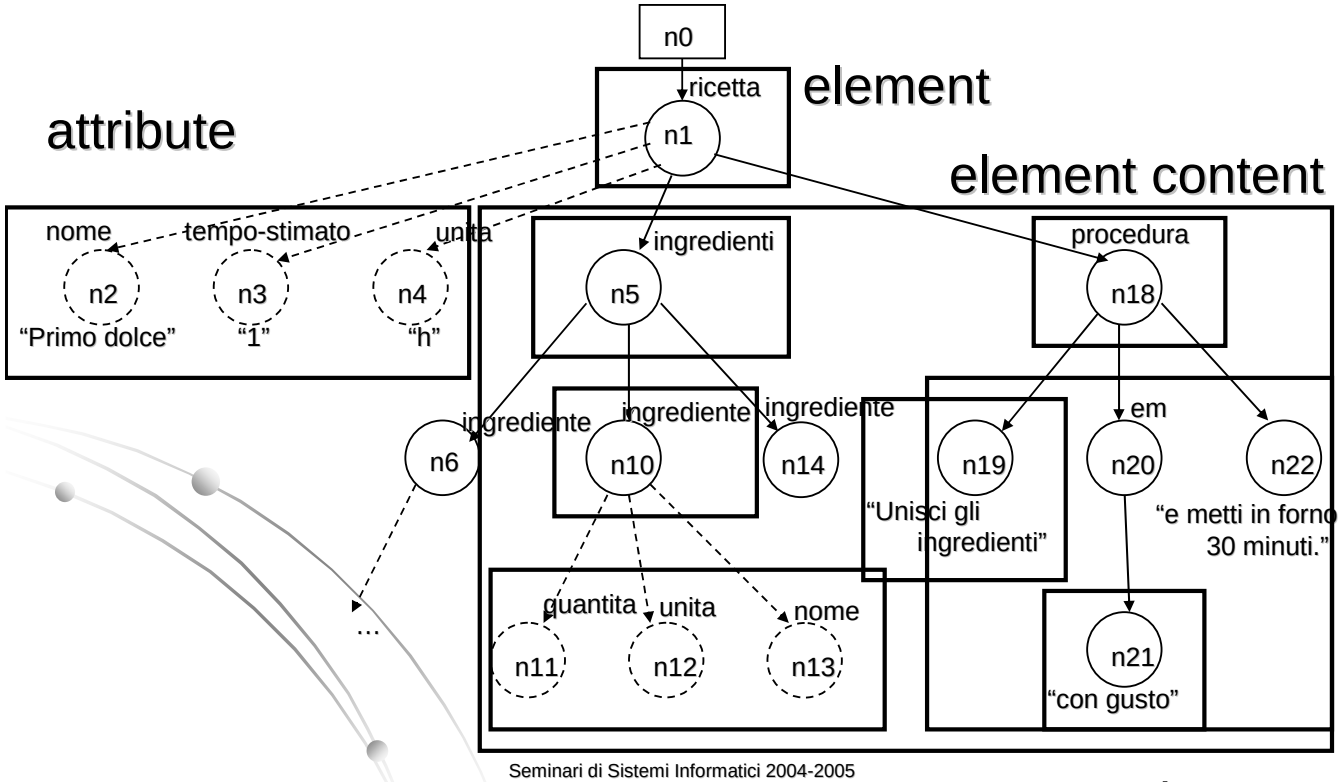
Esempio ricetta

```
<ricetta nome="Primo dolce" tempo-
  stimato="1" unita="h">
  <ingredienti>
     <ingrediente quantita="200" unita="g"
     nome="farina">
     <ingrediente quantita="200" unita="g"
     nome="zucchero">
  <\ingredienti>
  <procedura>Unisci gli ingredienti <em>con
  gusto<\em> e metti in forno 30 min.
  <\procedura>
<\ricetta>
```

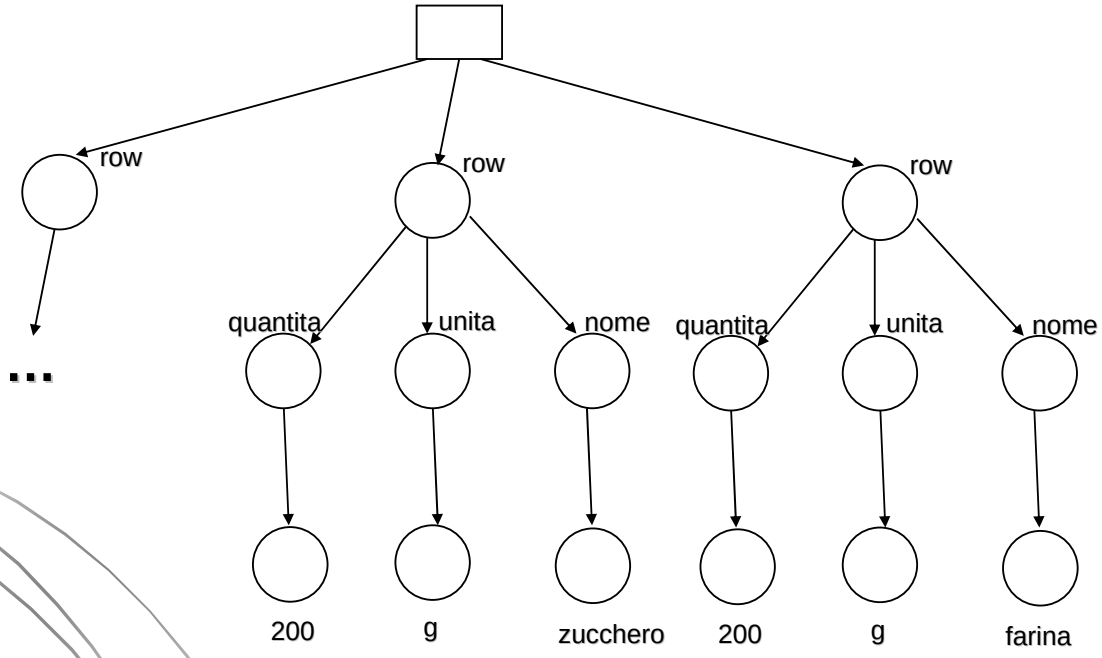
elemento

contenuto del nodo elemento

Esempio 2



e se li volessi strutturati?



Verso il relazionale

Spostati i dati da **attributi** a **sottoelementi**:

```
<row>
   <quantita>200<\quantita>
   <unita>g<\unita>
   <nome>farina<\nome>
<\row>
<row>
   <quantita>200<\quantita>
   <unita>g<\unita>
   <nome>zucchero<\nome>
<\row>
...
```

Attributi o sottoelementi?

Meglio usare un attributo o un elemento nidificato con un valore?

(una) filosofia: usare gli attributi per identificatori, riferimenti, metadati: “dati sui dati” (es. unità di misura usata)

Namespace

“Prezzo” con o senza IVA?

Che formato per la data (gg-mm-aaaa)?

Namespace: qualifichiamo i nomi (sintatticamente e semanticamente)

Attributo speciale xmlns seguito dai prefissi usati per comporre i nomi

Namespace

```
<?xml version="1.0" encoding="iso-8859-1" ?>
  <rss version="2.0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:sy="http://purl.org/rss/1.0/modules/syndicatio
  n/" xmlns:admin="http://webns.net/mvcb/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
  ns#">
  <channel>
      <title>I Miserabili</title>
      <link>http://www.miserabili.com/</link>
      <description />
      <dc:language>en-us</dc:language>
      <dc:creator>g*****@fastwebnet.it</dc:creator>
      <dc:date>2004-02-22T16:20:01+01:00</dc:date>
      <admin:generatorAgent
      rdf:resource="http://www.movabletype.org/?v=2.661
  " />
      <sy:updatePeriod>hourly</sy:updatePeriod>
      …
```


DOM (Document Object Model)

- API del W3C per programmare Infoset
- Binding per JAVA (metodi per costruire e navigare alberi)
- Rappresentato in memoria, ogni nodo è raggiunto in tempo unitario
- Alternativa: SAX (simple API for XML). Parser *event driven*

Schemi e Istanze

Finora abbiamo visto solo istanze: documenti XML che contengono dei dati e che devono essere ben formati (well formed) per essere utilizzabili. Definiti da:

- Prologo
- Corretta nidificazione dei tag
- Corretta codifica degli attributi
- Corretta codifica dei valori

Ma se vogliamo dei documenti facilmente interpretabili? Documenti che si rifacciano a una sorta di schema che conosciamo a priori e che siano validi rispetto a questo.

Schemi

Sono possibili diverse scelte per rappresentare dati in XML, con grandi differenze anche per le stesse informazioni.

Gli insiemi delle possibili strutture sono definiti con notazioni e regole nei **linguaggi per la definizione degli schemi**.

Schemi per XML

Il mondo (informatico) ha bisogno di un linguaggio per gli schemi comune. Ma ne abbiamo diversi:

- DTD: Document Type Description
- XML Schema
- Relax NG, XDuce, …

Per motivi storici e per scelte di design.

DTD

- (Ancora)Il più usato, il più semplice, il meno espressivo.
- Orientato ai documenti elettronici, parte dello standard XML 1.0
- Albero sintattico di grammatiche *context-free*, con espressioni regolari risultato delle produzioni
- Un solo tipo di valori: la stringa (#PCDATA)
- Simboli speciali: , + ? | *
- Vincoli sugli attributi

- Es. DTD per i documenti che descrivono ricette:

Es. DTD

```
<!ELEMENT ricetta (ingredienti, procedura)>
<!ELEMENT ingredienti (ingrediente*)>
<!ELEMENT ingrediente (#PCDATA)>
<!ELEMENT procedura ((#PCDATA|em)*)>
<!ELEMENT em (#PCDATA)|em>
<!ATTLIST ricetta nome CDATA #REQUIRED
                  tempo-stimato CDATA #REQUIRED
                   unita CDATA #REQUIRED>
<!ATTLIST ingrediente quantita CDATA #REQUIRED
                      unita CDATA #REQUIRED
                      nome CDATA #REQUIRED>
```


Es. grammatica DTD

Se ignoriamo gli attributi:

```
ricetta     ::= ingredienti, procedura
ingredienti ::= ingrediente*
ingrediente ::= #PCDATA
procedura   ::= (#PCDATA | em)*
em          ::= #PCDATA
```

Altro su DTD

- Case sensitive (minuscolo per convenzione)
- Entità: costanti (predefinite &agrave, …)
- DTD interni ed esterni al documento:
  - <!DOCTYPE ricetta [
    <!ELEMENT ricetta (ingredienti, procedura)>
    …
    ]>
  - <!DOCTYPE ricetta SYSTEM “ricetta.dtd”>
    …

Limiti DTD

- Namespace: è difficile far coesistere DTD e namespace:
  - i primi sono nati con XML, i namespace sono stati introdotti successivamente
- Elementi di testo: non è possibile imporre vincoli al contenuto testuale e, soprattutto, agli attributi. Non esiste il concetto di testo “tipizzato”
  es.: <corso codice=“Ing.Conoscenza”>
  <numeroIscritti>Marco</numeroIscritti>
  </corso>
- Content model misti: è possibile comporli solo come (#PCDATA|..|..)*
- Documentazione: con i DTD posso solo inserire commenti XML, che però possono essere ignorati dal parser
- NON sono scritti in XML

XML Schema

XSD (XML Schema Definition) è una particolare applicazione XML (linguaggio) che serve a descrivere le regole di validità di uno schema

Sviluppato dal W3C, recente, molto espressivo, verboso (rispetto al dtd, fattore 1:4).

Schemi espressi in XML, grande importanza ai tipi, largo uso dei namespace.

Vantaggi

- Supporto estensivo per la qualificazione tramite namespace
- Un sistema di tipi gerarchico
  - Tipizzazione del testo
  - Tipizzazione dei contenuti
- Definizione di frammenti di specifica riutilizzabili
- Permette di specificare vincoli per elementi strutturati ed offre grande flessibilità per Content Model misti
- Documentazione esplicita
- Scritto in XML

Es. XML Schema 1

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:r="http://www.lericettedelDIA/schema-ricette">

<simpleType name="unitType">
  <restriction base="string">
     <enumeration value="h"/>
     <enumeration value="m"/>
  </restriction>
</simpleType>
```


Es. XML Schema 2

```
<element name="ricetta">
  <complexType>
     <sequence>
          <element ref="r:ingredienti"/>
          <element ref="r:procedura"/>
     </sequence>
     <attribute name="nome" type="string"/>
     <attribute name="tempo-stimato"
                type="float"/>
     <attribute name="unita"
                type="r:unitType"/>
  </complexType>
</element>
```

Es. XML Schema 3

```
<element name="ingredienti">
  <complexType>
     <sequence>
          <element ref="r:ingrediente"
           maxOccurs="unbounded" minOccurs="0"/>
     </sequence>
  </complexType>
</element>

<element name="ingrediente" type="string"/>
   … </schema>
```


Applicazioni XML

- **Documenti**. Prima e più semplice applicazione: markup dei documenti da mostrare attraverso browser. (vedi poi XSLT)
- **Scambio dati**. Linguaggio comune per far viaggiare informazioni. Necessario uno schema comune.
- **XML nativo**. Applicazioni e database che usano solo XML.
- **Altro**: XML content-based routing (grande scambio di informazioni su file XML: facciamo pacchetti XML e routing con XPath), SOAP, …

Oltre il DOM

- I dati XML sono spessi tradotti in DOM per essere manipolati con API standard in Java o C++
- Lavorare con la notazione del DOM non è molto comodo e a volte poco conveniente:
  - non si possono caricare in memoria centrale XML oltre una certa dimensione
  - possono esserci dati remoti
  - possono esserci dati aggiornati dinamicamente da altre applicazioni

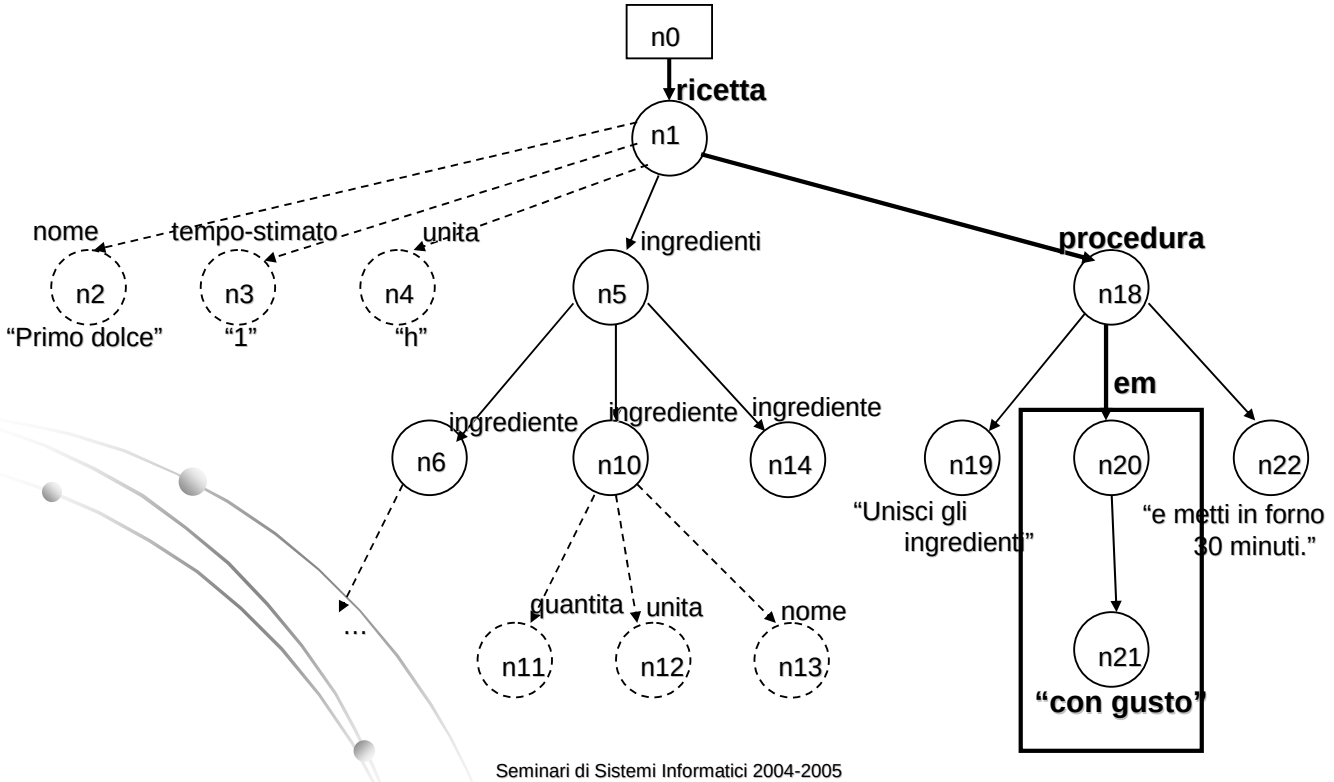
Oltre il DOM

- Sono quindi utili nuovi linguaggi specifici per programmare e interrogare l'XML
- Naturalmente è il W3C a definirli:
  - XPath
  - XQuery
  - XSLT

XPath

- Linguaggio per definire espressioni per navigare attraverso documenti XML (molto simile alle espressioni dei cammini per accedere alle directory sotto Linux)
- Pensato per essere semplice: modulo utile per linguaggi più complessi
- Es. espressione XPath:
  - `/ricetta/procedura/em`
    - risultato: `<em>con gusto</em>`

Primo esempio XPath



Es. XPath

Se vogliamo andare più in profondità possiamo selezionare direttamente il figlio di em che contiene il testo:

```
/ricetta/procedura/em/text()
```

che restituisce:

```
“con gusto”
```


Es. XPath

In generale, le espressioni restituiscono **sequenze di nodi**. L’espressione

```
/ricetta/ingredienti/ingrediente
```

restituisce

```
<ingrediente quantita=“200” unita=“g”
  nome=“farina”>
<ingrediente quantita=“200” unita=“g”
  nome=“zucchero”>
<ingrediente quantita=“2” unita=“dl”
  nome=“latte”>
```

Es. XPath

Se vogliamo selezionare solo gli attributi utilizziamo il prefisso “@” davanti al nome:

`/ricetta/ingredienti/ingrediente/@`**`nome`**

che restituisce

`“farina”, “zucchero”, “latte”`

XPath

- Operatori per i confronti:
  - Con ogni nodo elemento: *
  - Con ogni nodo attributo: @*
  - Con ogni nodo testo: text()
  - Con ogni nodo: node()
- L’operatore ***/*** restituisce ogni figlio dell’operando sinistro ed è chiamato **navigation step**
- L’operatore ***//*** restituisce ogni discendente del nodo a sinistra, a prescindere dalla sua posizione e profondità

Ancora Es. XPath

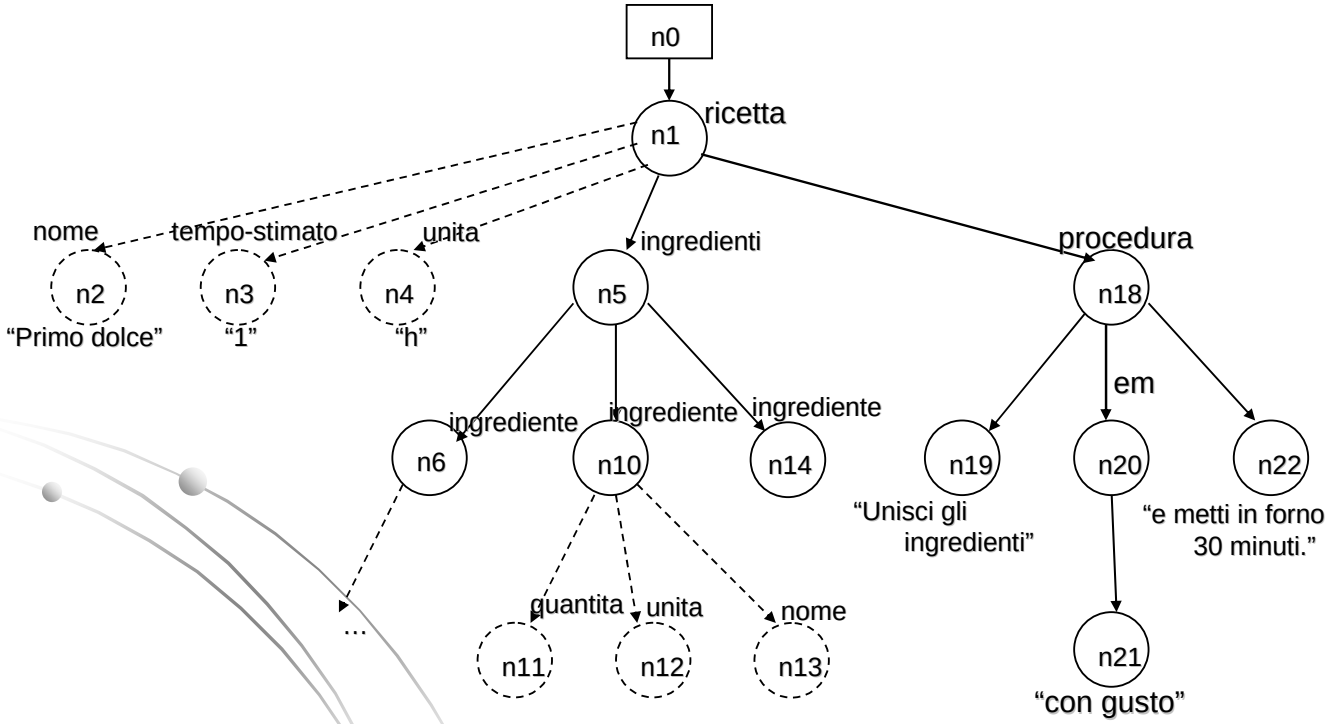
`//ingrediente/descrizione//tempo/*/text()`

Cerca un nodo chiamato `ingrediente` a partire dalla radice, entra nel figlio `descrizione` e cerca un discendente chiamato `tempo`. Recupera infine tutti gli elementi figli e restituisce i loro valori testuali.

Possono esserci anche filtri (predicati):

`//ingrediente[produttore/indirizzo]/descrizione/tempo[@timezone]/text()`

Esempio XPath



Es. filtri XPath

```
//ingrediente[produttore/indirizzo]/descr
   izione/tempo[@timezone]/text()
```

Si risolve in due passi:

1) Si rimuovono i filtri per avere un'espressione lineare
   - `//ingrediente/descrizione/tempo/text()`
2) Si considerano solo i path che soddisfano la condizione `[produttore/indirizzo]` (hanno quindi almeno un figlio produttore con un figlio address) e la condizione `[@timezone]`, cioè l'attributo con questo nome.

Es. filtri XPath

In pratica i filtri si usano per confrontare valori:

```
//ingrediente[produttore/indirizzo/zi
  p="00142"]/descrizione/tempo[@timezo
  ne="EST"]/text()
```

XQuery

Estende XPath con caratteristiche più avanzate: join, unione e interrogazioni annidate.

Il costrutto base è ***for-let-where-return*** (analogia con *select-from-where*): espressioni FLWR

```
for $x in /ricetta/ingredienti/ingrediente
where $x/@quantita > 150
return
<ingredienteSpeciale>
 <dose> { $x/@quantita } </dose>
 <nome> { $x/@nome } </nome>
</ingredienteSpeciale>
```


Es. XQuery

Risultato:

```
<ingredienteSpeciale>
    <dose>200</dose>
    <nome>farina</nome>
</ingredienteSpeciale>
<ingredienteSpeciale>
    <dose>200</dose>
    <nome>zucchero</nome>
</ingredienteSpeciale>
```

Altro es. in XQuery

```
FOR $e IN document("libri.xml")//editore
LET $l :=
document("libri.xml")//libro[editore=$e]
WHERE count($l) > 5
RETURN
  <risultato>
    {$e}
  </risultato>
```

- Crea lista (associata a '$e') contenente gli editori; associa a ciascun editore la lista dei libri da lui editi ('$l'), creando lista ordinata di tuple formate da ($e,$l).

- **count()** funzione già definita, come **avg()**, if-then-else- o sugli insiemi: **union()**, **intersection()**, **difference(); …**

Es. Join in XQuery

```
for $x in /ricetta/ingredienti/ingrediente
    $y in /ricetta/prodotti/prodotto
where $x/@nome = $y/ingredienti/nome/text()
return <costo>
       <nome>   { $x/@nome }          </nome>
       <dose>   { $x/@quantita }      </dose>
       <prezzo> { $y/@prezzo/text() } </prezzo>
       </costo>
```

XQuery e SQL

In SQL:

```
select x.nome, x.quantita, y.prezzo
from ingrediente x, prodotto t
where x.nome = y.nome
```

Fortissimo legame: traduzioni automatiche, tecniche miste XQuery – Relazionale, …

XSLT

eXtensible Style Language Trasformation

- Inizialmente nato come foglio di stile per trasformare documenti XML in un linguaggio visualizzabile con browser (HTML)
- Evoluto in un linguaggio per trasformare dati XML in dati XML attraverso funzioni ricorsive
- Idea: attraversare l'albero ricorsivamente applicando una trasformazione in ogni nodo

XSLT

N.B. anche con XPath possiamo navigare attraverso l'albero con un singolo comando (*//*) ma non c'è nessuna trasformazione fino a quando non incontro il nodo destinazione

Es. XSLT: sostituisci ad ogni tag il tag <bf>

XSLT

Due regole template

Match pattern

Template

Exp XPath

XSLT function

1. return node text value
2. invoca programma ricorsivamente su ogni figlio (copia tutti **elementi** dell'albero)
3. copia elemento

Output

```
<xsl:template match="em">
  <bf> <xsl:value-of/> </bf>
</xsl:template>
```

```
<xsl:template match="*">
  <xsl:element>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

XSLT

Nell' es. ci può essere un conflitto fra le due regole:

A un elemento chiamato `em` quale template applico?

XSLT sceglie la **regola più specifica** (nel nostro caso la prima)

A domani

Come usare tutto questo nel model management?