

# XSLT



Moreno Marzolla  
INFN Sezione di Padova  
moreno.marzolla@pd.infn.it  
<http://www.dsi.unive.it/~marzolla>

## Testo di riferimento

- Erik T. Ray, *Learning XML*, O'Reilly, First Edition, January 2001 ISBN: 0-59600-046-4, 368 pages
- Parte del materiale presentato è tratto dal corso di Tecnologie Web, prof. Fabio Vitali, Università di Bologna

## Cos'è XSLT

- XSLT è un linguaggio di trasformazione
  - Dato un documento XML, è possibile generare un altro documento (non necessariamente) XML derivato applicando delle regole di trasformazione specificate nello stylesheet
  - Se nel documento XML di destinazione abbiamo scritto elementi i cui nomi ed attributi sono noti ad un browser (ad esempio, HTML), il documento può essere visualizzato da un browser.
- XSLT è un linguaggio per trasformare un documento XML in un altro documento secondo regole predefinite.

## Come funziona XSLT

- Un documento XSL è composto di *template di costruzione*, che permettono di riscrivere una selezione elementi del documento XML d'origine in altri elementi del documento destinazione
  - Ogni template individua un *pattern* da ricercare nel documento di partenza
  - Ad ogni pattern individuato, associa un blocco di elementi e testo da inserire nel documento di destinazione
- XSLT si basa su XPath per descrivere questi pattern

## I fogli di stile XSLT

- Esistono due filosofie di riscrittura disponibili in XSLT, *iterativo* e *ricorsivo*
- Iterativo, o Pull
  - Basata su *template*, viene usata tipicamente per trasformare dati. In un documento pre-formatto per l'output, si vanno ad inserire le parti di documento tratte dal file XML d'origine.
- Ricorsivo, o Push
  - Basata su *regole*, usata tipicamente per trasformare documenti. Per ogni elemento del documenti di input, si cerca la regola più appropriata e la si usa per scrivere il risultato.

## Stile iterativo / 1

```
<?xml version="1.0"?>
<portfolio>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <sym>ZCXM</sym>
    <pr>28.875</pr>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <sym>ZFFX</sym>
    <pr>92.250</pr>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <sym>ZYSZ</sym>
    <pr>20.313</pr>
  </stock>
</portfolio>
```

## Stile iterativo / 2

```
<HTML xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns="http://www.w3.org/HTML4/">
  <BODY>
    <TABLE BORDER="2">
      <TR><TD>Simbolo</TD><TD>Nome</TD><TD>Prezzo</TD></TR>
      <xsl:for-each select="portfolio/stock">
        <TR>
          <TD><xsl:value-of select="sym"/></TD>
          <TD><xsl:value-of select="name"/></TD>
          <TD><xsl:value-of select="pr"/></TD>
        </TR>
      </xsl:for-each>
    </TABLE>
  </BODY>
</HTML>
```

## Il foglio di stile iterativo

- Questo è sostanzialmente un documento HTML con qualche tag strano:
  - **xsl:for-each** va a cercare uno ad uno una sequenza di elementi ed applica le istruzioni al suo interno per ciascun elemento
  - **xsl:value-of** va a cercare il valore (cioè il contenuto) di ogni elemento all'interno di xsl:foreach, e lo inserisce al suo posto.

## Stile ricorsivo / 1

```
<?xml version="1.0"?>
<document>
  <title>To the Pole and Back</title>
  <section>
    <title>The First Day</title>
    <para>It was the <emph>best</emph>
      of days, it was the <emph>worst
        </emph> of days.</para>
    <para><emph>Best</emph> in that the
      sun was out, but <emph>worst</emph>
      in that it was 39 degrees below
      zero.</para>
  </section>
  <!-- ... -->
</document>
```

## Stile ricorsivo / 2

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/HTML4/">
  <xsl:template match="/">
    <HTML> <BODY>
      <H1><xsl:value-of select="document/title"/></H1>
      <xsl:apply-templates select="document/section"/>
    </BODY> </HTML>
  </xsl:template>
  <xsl:template match="section">
    <HR/> <H2><xsl:value-of select="title"/></H2>
    <xsl:apply-templates select="para"/>
  </xsl:template>
  <xsl:template match="para">
    <P><xsl:apply-templates /></P>
  </xsl:template>
  <xsl:template match="emph">
    <EM><xsl:apply-templates /></EM>
  </xsl:template>
</xsl:stylesheet>
```

## Il foglio di stile ricorsivo

- Questo non assomiglia ad un documento HTML, ma ad una serie di istruzioni di riscrittura separate:
  - **xsl:template** è la regola da applicare se l'elemento in esame corrisponde al valore dell'attributo match. Di volta in volta applicherò il template della radice ("/"), dell'elemento "section", dell'elemento "para", dell'elemento "emph", ecc.
  - **xsl:apply-templates** spinge a cercare, all'interno dell'elemento che stiamo considerando, se esistono altri template applicabili. E' il modo per far ripartire ricorsivamente la ricerca di altri template.

## Modello di processing

- Il parser costruisce una lista di nodi correnti che per default contiene il nodo radice
- Quindi cerca tutti i template che possono essere applicati alla testa della lista di nodi correnti e seleziona il più importante sulla base di criteri espliciti di preferenza
- L'applicazione del template può creare dei frammenti di albero di destinazione e può anche inserire altri nodi nella lista di nodi correnti.
- Poi il ciclo si ripete fino ad esaurimento della lista di nodi correnti.

## I template / 1

- Ogni foglio di stile contiene uno o più template. Un template ha o un nome o un pattern di attivazione
  - Se ha un nome, può essere esplicitamente attivato da un'altra azione
  - Se ha un pattern XPath, può essere attivato se il pattern fa match con il nodo corrente
- Un template è indicato dall'elemento `<template>`:

```
<xsl:template match=pattern name=gname priority=num mode=gname >
  <!-- azione -->
</xsl:template>
```

## I template / 2

- Attributo **name**
  - Può essere utilizzato per associare un nome univoco ai template, in modo da poterli richiamare in seguito
- Attributo **priority**
  - Utilizzato in caso di ambiguità per dare preferenza ad un template piuttosto che ad un altro
- Attributo **mode**
  - Sia `xsl:template` che `xsl:apply-templates` hanno questo attributo (opzionale)
  - Se `xsl:template` non ha un attributo `match`, deve avere un attributo `mode`
  - Se un elemento `xsl:apply-templates` ha un attributo `mode`, allora viene applicato solo alle regole di elementi `xsl:template` con lo stesso valore dell'attributo `mode`
  - Se un elemento `xsl:apply-templates` non ha attributo `mode`, allora si applica solo alle regole di elementi `xsl:template` che non hanno attributo `mode`

## I template / 3

- Ad esempio, dato il frammento:

```
Questo deve essere <emph>importante</emph>
```

- Il seguente template:

```
<xsl:template match="emph">
  <em>
    <xsl:apply-templates/>
  </em>
</xsl:template>
```

- Fa match con l'elemento `emph` e scrive un elemento `em` di HTML ed inserisce tutti i nodi figlio del nodo di match nella lista dei nodi correnti.
- Il pattern in questo caso è un XPath relativo (equivalente a `child::emph`). Qualunque XPath che ritorni un nodeset può essere inserito nell'attributo `match`.

## Contenuto di un template

- Modificare l'albero di destinazione
  - elementi letterali
  - `<xsl:value-of>`
  - `<xsl:element>`
  - `<xsl:attribute>`
  - `<xsl:text>`
  - `<xsl:comment>`
  - `<xsl:copy>`
  - `<xsl:number>`
- Modificare la lista dei nodi correnti
  - `<xsl:apply-templates>`
  - `<xsl:for-each>`
  - `<xsl:if>`
  - `<xsl:choose>`
  - `<xsl:sort>`

## Scrivere l'albero di destinazione / 1

- Poiché l'albero di destinazione è esso stesso un documento XML, devo poter creare nodi elemento, attributi e testo
- Nodi risultato letterali
  - Sono il modo più semplice: dentro al template scrivo direttamente il frammento XML richiesto.
  - Ogni elemento che non appartiene al namespace di xsl viene direttamente scritto nell'albero di destinazione così come appare nel template.
  - Analogamente viene fatto per ogni nodo di testo

```
<xsl:template match="pippo">
  <b>Viva Pippo</b>
</xsl:template>
```

## Scrivere l'albero di destinazione / 2

- **<xsl:value-of>**
  - **<xsl:value-of>** crea un nodo di testo nell'albero di destinazione. L'attributo **select** (obbligatorio) contiene un espressione XPath che viene valutata e convertita in stringa.
  - L'uso tipico è per convertire markup in testo (ad esempio il valore di attributi in contenuto).

XML 

```
<persona nome="Moreno" cognome="Marzolla"/>
```

XSLT 

```
<xsl:template match="persona">
  <p><xsl:value-of select="@nome"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="@cognome"/></p>
</xsl:template>
```

Risultato 

```
<p>Moreno Marzolla</p>
```

## Scrivere l'albero di destinazione / 3

- Parentesi graffe {}
  - Dove non è possibile usare del markup (ad esempio come valore di un attributo), è possibile usare le parentesi graffe, che hanno in questo senso lo stesso significato di **<xsl:value-of>**
  - L'uso tipico è per convertire markup in altro markup (ad esempio il valore di un attributo nel nome di un tag).

XML 

```
<persona nome="Moreno" cognome="Marzolla"/>
```

XSLT 

```
<xsl:template match="persona">
  <a href="mailto:{@nome}.{@cognome}@pd.infn.it">
    <xsl:value-of select="@nome"/>
  </a>
</xsl:template>
```

Risultato 

```
<a href="mailto:Moreno.Marzolla@pd.infn.it">Moreno</a>
```

## Scrivere l'albero di destinazione / 4

- **<xsl:element>**
  - Se è necessario scrivere un elemento complesso o calcolato uso **<xsl:element>**
  - Ad esempio può servire per trasformare il valore di un attributo del documento di partenza nel nome di un tag nel documento destinazione

XML 

```
<persona tipo="studente" nome="Mario Rossi"/>
```

XSLT 

```
<xsl:template match="persona">
  <xsl:element name="{@tipo}">
    <xsl:value-of select="@nome"/>
  </xsl:element>
</xsl:template>
```

Risultato 

```
<studente>Mario Rossi</studente>
```

## Scrivere l'albero di destinazione / 5

- **<xsl:attribute>**

- All'interno di un elemento (sia letterale che **<xsl:element>**) è possibile specificare degli attributi in maniera esplicita con il tag **<xsl:attribute>**
- E' più chiaro e più potente delle parentesi graffe. Lo si può usare per stabilire con espressione anche il nome dell'attributo.

XML 

```
<persona id="12345" nome="Mario Rossi"/>
```

XSLT 

```
<xsl:template match="persona">
  <a><xsl:attribute name="href">
    <xsl:value-of select="@id"/>.html
  </xsl:attribute>
  <xsl:value-of select="@nome" /></a>
</xsl:template>
```

Risultato 

```
<a href="12345.html">Mario Rossi</a>
```

## Scrivere l'albero di destinazione / 6

- **<xsl:text>**

- Inserisce esplicitamente il testo contenuto dentro al documento.
- E' vantaggioso rispetto a mettere il testo letterale perché rispetta il white space. Inoltre con l'attributo `disable-output-escaping="yes"` rispetta anche i caratteri speciali ("&" e "<").

- **<xsl:comment>**

- Inserisce esplicitamente del commento dentro al documento.

## Scrivere l'albero di destinazione / 7

- **<xsl:copy>**

- Copia nell'output il nodo di riferimento, ma non il suo contenuto e i suoi attributi, che vanno copiati esplicitamente.

- **<xsl:number>**

- Viene usato per inserire esplicitamente numeri formattati dentro all'albero dei risultati. Ha vari attributi, tra cui:
  - `Level`: quanti livelli dell'albero sorgente vanno considerati
  - `Count`: quale pattern di nodi vanno contati per trovare il numero
  - `From`: da dove partire nel conto
  - `Value`: come identificare il numero (se diverso dalla posizione)
  - `Format`: il formato del numero (1, 2, 3 oppure A, B, C, ecc.)

## Cambiare la lista di nodi correnti / 1

- Nel corso del processing di un nodo, è possibile alimentare la lista dei nodi correnti con altri nodi. Qualunque nodo può essere inserito nel documento, anche più volte

- **<xsl:apply-templates/>**

- **<xsl:apply-templates>** inserisce nella lista dei nodi correnti i figli diretti dell'elemento considerato, nell'ordine in cui appaiono.
- Se usato con l'attributo `select` (con un XPath per selezionare i nodi richiesti), inserisce i nodi che fanno match con il pattern.
- Nel momento in cui incontra un elemento **<xsl:apply-templates>**, il parser sospende il processing del template in corso e procede ricorsivamente ad esaminare i figli.

## Cambiare la lista di nodi correnti / 2

### • `<xsl:apply-templates>`

- Questo template trasforma un "para" in un "p" di HTML:

```
<xsl:template match="para">
  <p><xsl:apply-templates/></p>
</xsl:template>
```

- Questo template crea un indice delle intestazioni di primo livello di un documento HTML e lo pone prima del testo:

```
<xsl:template match="body">
  <xsl:apply-templates select="h1"/>
  <hr/>
  <xsl:apply-templates />
</xsl:template>
```

## Cambiare la lista di nodi correnti / 3

### • `<xsl:for-each>`

- `<xsl:apply-templates>` mette i nodi figlio dentro alla lista dei nodi correnti e procede cercando template da applicare.
- Se invece voglio che un comportamento specifico venga applicato ad ognuno dei figli direttamente dentro al template, uso `<xsl:for-each>`

```
<xsl:template match="body">
  <xsl:apply-templates select="h1"/>
</xsl:template>
<xsl:template match="h1">
  <p><xsl:value-of select="."/></p>
</xsl:template>
```

Queste due trasformazioni sono equivalenti

```
<xsl:template match="body">
  <xsl:for-each select="h1"/>
    <p><xsl:value-of select="."/></p>
  </xsl:for-each>
</xsl:template>
```

## Cambiare la lista di nodi correnti / 4

### `<xsl:if>`

- `<xsl:if>` attiva condizionalmente dei comportamenti a seconda della verità di un XPath di test.
- Ad esempio il template seguente colora di giallo lo sfondo di una riga ogni due di una tabella HTML:

```
<xsl:template match="item">
  <tr>
    <xsl:if test="position() mod 2 = 0">
      <xsl:attribute name="bgcolor">
        yellow
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </tr>
</xsl:template>
```

## Cambiare la lista di nodi correnti / 5

### • `<xsl:choose>`, `<xsl:when>`, `<xsl:otherwise>`

- `<xsl:choose>` seleziona una tra molte alternative (la funzione di switch in C).

```
<xsl:template match="item"><tr>
  <xsl:choose>
    <xsl:when test="position() mod 3 = 0">
      <xsl:attribute name="bgcolor">blue</xsl:attribute>
    </xsl:when>
    <xsl:when test="position() mod 3 = 1">
      <xsl:attribute name="bgcolor">green</xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
      <xsl:attribute name="bgcolor">red</xsl:attribute>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:apply-templates/>
</tr></xsl:template>
```

## Cambiare la lista di nodi correnti / 6

- **<xsl:sort>**

- **<xsl:sort>** ordina i nodi nella lista dei nodi correnti. Esso può essere soltanto figlio di un **<xsl:apply-templates>** o di un **<xsl:for-each>**
- Il primo figlio **xsl:sort** specifica la chiave primaria di ordinamento, il secondo figlio **xsl:sort** specifica la chiave secondaria e così via
- **<xsl:sort>** ha vari attributi:
  - **Select**: l'espressione in base alla quale fare il sort
  - **Data-type**: il tipo di dato da ordinare ("text", "number")
  - **Order**: il tipo ascendente o discendente di ordine ("ascending", "descending")
  - **Case-order**: come trattare le maiuscole e le minuscole ("upper-first", "lower-first")

## Cambiare la lista di nodi correnti / 7

```
<xsl:template match="lista">
  <ul>
    <xsl:apply-templates select="persona">
      <xsl:sort select="cognome"/>
      <xsl:sort select="nome"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>

<xsl:template match="persona">
  <li>
    <xsl:value-of select="nome"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="cognome"/>
  </li>
</xsl:template>
```

## Indirezioni / 1

- **Variabili**

- Posso definire delle variabili. Il valore di una variabile è quello di qualunque espressione.
- La variabile può essere usata nel sottoalbero in cui è definita e deve essere richiamata con l'uso delle graffe e del segno \$
- Attenzione: la variabile XSLT non è *variabile*, ovvero il suo valore non può essere cambiato durante la computazione. E' una costante ottenuta attraverso la valutazione di un'espressione, non una variabile. .

```
<xsl:variable name="fs">12pt</xsl:variable>
<xsl:template match="para">
  <p style="font-size: {$fs}">
    <xsl:apply-templates/>
  </p>
</xsl:template>
```

## Indirezioni / 2

- L'esempio precedente si poteva anche scrivere come:

```
<xsl:variable name="fs">12pt</xsl:variable>
<xsl:template match="para">
  <p>
    <xsl:attribute name="style">
      <xsl:text>font-size: </xsl:text>
      <xsl:copy-of select="$fs"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </p>
</xsl:template>
```

## Indirezioni / 3

- Template nominali
  - Se in un elemento template uso l'attributo name, invece che match, ottengo un template nominale che viene esplicitamente attivato con il tag `<xsl:call-template>`
  - Posso specificare anche dei parametri per ottimizzare l'uso dei template nominali

```
<xsl:template name="numbered-block">
  <xsl:param name="format">1. </xsl:param>
  <p> <xsl:number format="{format}" />
  <xsl:apply-templates/></p>
</xsl:template>
<xsl:template match="ol//ol/li">
  <xsl:call-template name="numbered-block">
    <xsl:with-param name="format">a.
  </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Moreno M

33

## Indirezioni / 4

- `<xsl:attribute-set>`
  - Posso avere una lista nominale di attributi con l'elemento `<xsl:attribute-set>`. Con l'attributo `<xsl:use-attribute-sets>` di elementi testuali e nei tag `<xsl:element>` e `<xsl:copy>` uso la lista predefinita di attributi.

```
<xsl:attribute-set name="ts">
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="heading">
  <fo:block xsl:use-attribute-sets="ts">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Moreno Marzolla

Tecnologie Web

34

## Modi

- I modi permettono di avere template diversi se è necessario usare gli stessi nodi di input in più posti diversi.
- L'attributo "mode" di `<xsl:apply-templates>` e di `<xsl:template>` crea questo binding.

```
<xsl:template match="/"><html>
  <xsl:apply-templates select="//title" mode="toc"/>
  <xsl:apply-templates/></html>
</xsl:template>
<xsl:template match="title" mode="toc">
  <div><a href="#{@id}"><xsl:apply-templates/></a></div>
</xsl:template>
<xsl:template match="chapter/title">
  <h1><a name="{@id}"><xsl:apply-templates/></a></h1>
</xsl:template>
<xsl:template match="section/title">
  <h2><a name="{@id}"><xsl:apply-templates/></a></h2>
</xsl:template>
```

## Template di default

- Esistono delle regole di default che vengono applicate in mancanza di template più specifici. Esse ricopiano semplicemente l'input nell'output.
- Il loro uso è fondamentale per i fogli di stile ricorsivi.

```
<xsl:template match="*" />
  <xsl:apply-templates />
</xsl:template>
Applica ricorsivamente i template a tutti i nodi che non fanno match con alcun template esplicito

<xsl:template match="text()|@" />
  <xsl:value-of select="." />
</xsl:template>
Per ogni nodo testo e attributo che non matchano template espliciti, si copia il valore nel risultato

<xsl:template match="processing-instruction()" />
<xsl:template match="comment()" />
Ignora commenti e direttive di preprocessing
```

Moreno Marzolla

Tecnologie Web

36

## Attenzione

- Che cosa produce la trasformazione seguente su questo documento?

XML

```
<html>
<p style="font-weight: bold">Prova</p>
</html>
```

XSLT

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/xsl/transform">
</xsl:stylesheet>
```

Risultato

```
Prova
```

## Altri aspetti di XSLT

- Merging
  - E' possibile porre frammenti di fogli di stile in file esterni. Con gli elementi `<xsl:import>` e `<xsl:include>` è possibile inserire frammenti esterni con due significati leggermente diversi: `<xsl:import>` aumenta la priorità degli elementi inclusi, mentre `<xsl:include>` la mantiene.
- Metodi di output
  - E' possibile specificare che il documento risultante è XML, HTML o testo con l'elemento `<xsl:output>`. Se l'output è HTML o testo, il processore è meno rigoroso nel valutare la buona forma del documento risultante.
- White space
  - E' possibile specificare quali elementi debbano preservare e quali debbano collassare il white space con due appositi elementi, `<xsl:preserve-space>` e `<xsl:strip-space>`.

## Specificare il foglio di stile XSLT

- E' necessario indicare al browser dove trovare lo stylesheet da usare. Questo può essere fatto in tre modi:
  - Specificando nell'intestazione MIME del collegamento HTTP la locazione del foglio di stile
  - Specificando un gruppo di documenti XLink, uno dei quali è il foglio di stile
  - Specificando con una Processing Instruction (PI) il collegamento:

```
<?xml-stylesheet type="text/xml" href="style.xml"?>
<doc> ... </doc>
```

## Alcuni suggerimenti

Eliminare un elemento e il suo contenuto

```
<xsl:template match="X"></xsl:template>
```

Eliminare un elemento mantenendo il suo contenuto

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

Trasformare l'elemento Q nell'elemento W

```
<xsl:template match="Q">
<W><xsl:apply-templates/></W>
</xsl:template>
```

Eliminare un attributo

```
<xsl:template match="@X"></xsl:template>
```

Cambiare il nome di un attributo mantenendone il valore

```
<xsl:template match="@Y">
<xsl:attribute name="Z">
<xsl:value-of select="."/ />
</xsl:attribute>
</xsl:template>
```

## Il foglio di stile identità

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/*|*|text()">
    <xsl:copy>
      <xsl:apply-templates select="@*" />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

## Il foglio di stile base / 1

- Dato un documento XML semplice da trasformare in un documento HTML graficamente complesso:

```
<xsl:template match="/">
  <html>
    <body>
      <!-- Parti fisse prima del contenuto -->
      <xsl:apply-templates />
      <!-- Parti fisse dopo il contenuto -->
    </body>
  </html>
</xsl:template>
<xsl:template match="para">
  <p><xsl:apply-templates/></p>
</xsl:template>
<xsl:template match="emph">
  <em><xsl:apply-templates/></em>
</xsl:template>
```

## Il foglio di stile base / 2

- Perché inventarsi i tag PARA e EMPH solo per non usare HTML? Se le esigenze sono semplici, è possibile generare un HTML minimale ed arricchirlo modificando il foglio di stile identità per ottenere un HTML complesso:

```
<xsl:template match="/">
  <html>
    <body>
      <!-- Parti fisse prima del contenuto -->
      <xsl:apply-templates />
      <!-- Parti fisse dopo il contenuto -->
    </body>
  </html>
</xsl:template>
<xsl:template match="*|*|text()">
  <xsl:copy>
    <xsl:apply-templates select="@*" />
  </xsl:copy>
</xsl:template>
```

## Trovare elementi non gestiti

```
<xsl:template match="*">
  <xsl:choose>
    <xsl:when test="count(./*) > 0">
      <span style="background:yellow;">
        &lt;&lt;xsl:value-of select="name()" />&gt;
      </span>
      <xsl:apply-templates />
      <span style="background:yellow;">
        &lt;&lt;xsl:value-of select="name()" />&gt;
      </span>
    </xsl:when>
    <xsl:when test="count(./*) = 0">
      <span style="background:yellow;">
        &lt;&lt;xsl:value-of select="name()" />&gt;
      </span>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

## Esempio

- Considerare la pagina XHTML del corso di Tecnologie Web.
- Voglio scrivere una trasformazione per estrarre tutti gli approfondimenti importanti, e metterli uno dietro l'altro in una lista non numerata

```
<div class="approf">
<ul>

<li><!-- blah --></li>
<li><!-- blah --></li>
<li class="important"><!-- blah --></li>
<li><!-- blah --></li>
<li class="important"><!-- blah --></li>

</ul>
</div>
```

Moreno Marzolla

45

## Soluzione

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<head><title>Approfondimenti importanti</title></head>
<body>
<ul>
<!-- applica i template solo a questi nodi -->
<xsl:apply-templates select="//dd/div[@class='approf']/ul/li[@class='important']"/>
</ul>
</body>
</html>
</xsl:template>

<xsl:template match="li">
<li><xsl:apply-templates/></li>
</xsl:template>

<!-- identita -->
<xsl:template match="**|@*">
<xsl:copy>
<xsl:apply-templates select="**"/>
<xsl:apply-templates/>
</xsl:copy>
</xsl:template>

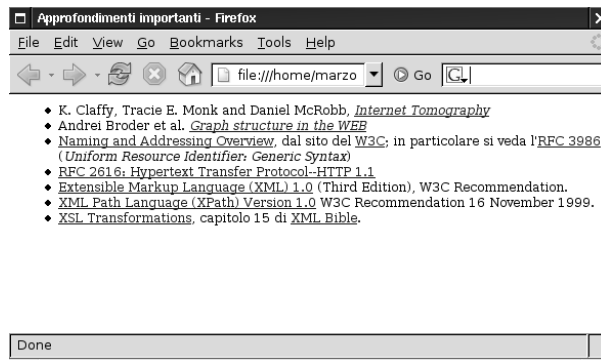
</xsl:stylesheet>
```

Moreno Marzolla

Tecnologie Web

46

## Risultato



Moreno Marzolla

Tecnologie Web

47