XSLT

Fabio Vitali

Introduzione

Oggi esaminiamo in breve:

- XSLT, ovvero il linguaggio di stile di XML.

XSL: un linguaggio di stylesheet

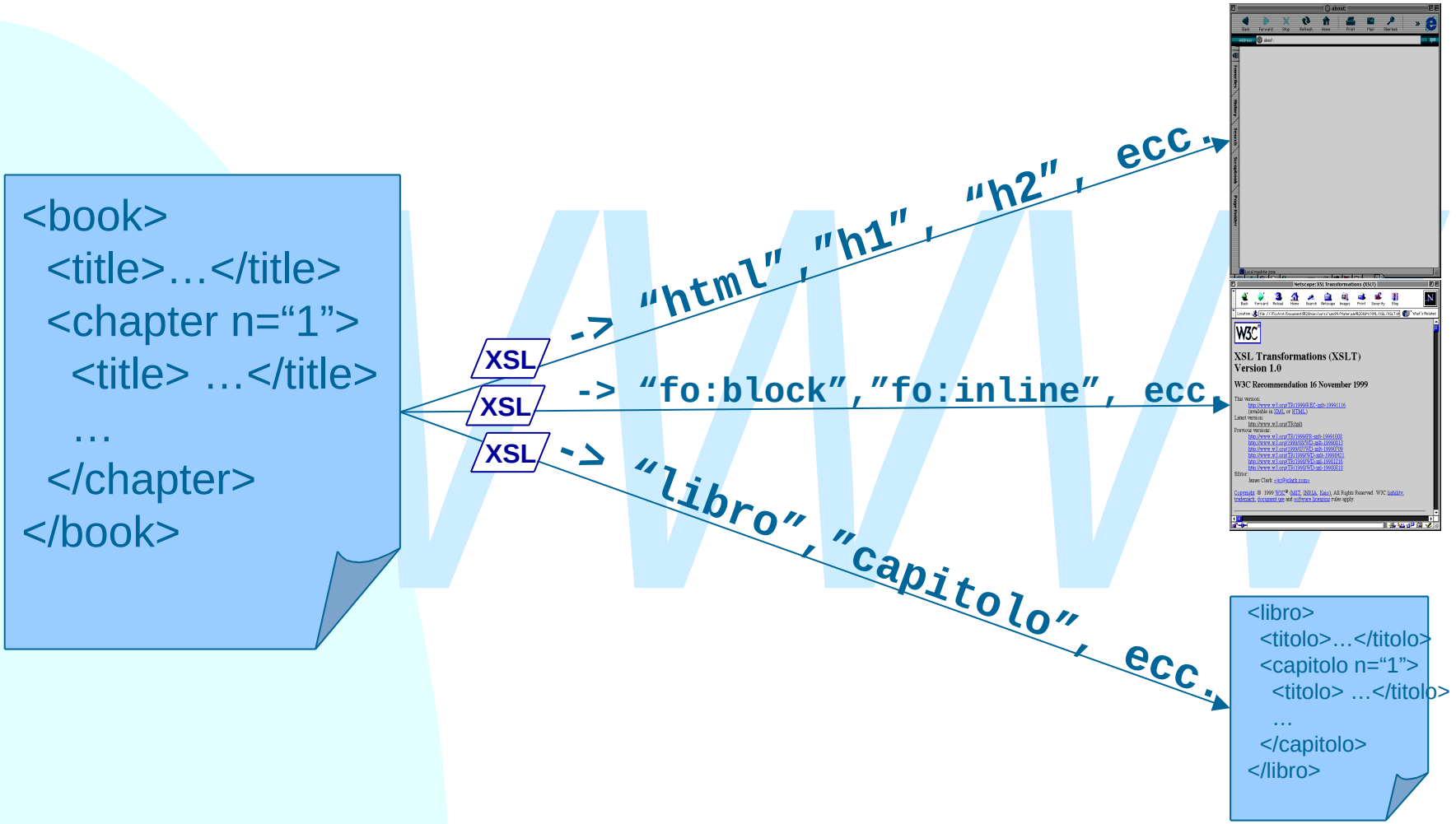
Poiché nessun elemento di XML possiede un significato predefinito, il linguaggio di stylesheet si occupa di dare un ***significato*** agli elementi di un documento XML.

XSL (Extended Stylesheet Language) è un working group di W3C che si occupa di attribuire significati “ben noti” (come caratteri, font, ecc.) agli elementi di un documento XML.

La proposta è divisa in due parti: un linguaggio di trasformazione da documenti XML a documenti XML (chiamato XSLT), ed un vocabolario di elementi XML con semantica di formattazione (chiamato XSL-FO, o anche solo XSL).

Il linguaggio XSL non ha ancora uno stato concluso. XSLT è una recommendation W3C del novembre 1999, mentre XSL-FO è ancora in fase di discussione.

Il modello di XSLT



Come funziona XSLT

XSLT è un linguaggio di trasformazione: dato un documento XML, è possibile generare un altro documento XML derivato applicando delle regole di trasformazione specificate nello stylesheet.

Se poi nel documento XML di destinazione abbiamo scritto elementi i cui nomi ed attributi sono noti ad un browser (ad esempio, HTML o XSL-FO), allora il documento può essere visualizzato da un browser.

XSLT, quindi, è molto più di un linguaggio di visualizzazione di documenti XML: è un linguaggio per trasformare un documento XML in un altro documento XML secondo regole predefinite.

I fogli di stile XSLT

Un foglio di stile XSL è un documento XML che utilizza un DTD i cui elementi hanno senso noto al motore XSLT.

Un foglio di stile XSL è composto sostanzialmente 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, e vi associa un blocco di elementi e testo da inserire nel documento di destinazione

XSLT si basa fondamentalmente su XPath per questi pattern. L'inclusione di XPath (del gruppo di lavoro XLink) ha costituito una notevole modifica rispetto alle prime proposte.

I fogli di stile XSLT

Nel documento XSLT si usano dunque elementi provenienti da almeno due ***namespace***: quello di XSLT e quello del modello di documenti di destinazione.

Esistono due filosofie di riscrittura disponibili in XSLT, che vengono dette ***pull*** e ***push***.

- **Pull**: basata su template, viene usata tipicamente per trasformare dati. In un documento pre-formattato per l'output, si vanno ad inserire le parti di documento tratte dal file XML d'origine. Si ispirano sostanzialmente ai linguaggi di inclusione (ad es. server-side include tipo PHP e simili).
- **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. Si ispirano ai linguaggi di riscrittura basati su regole (tipo DSSSL)

Un esempio di pull (1)

I fogli di stile tipo ***pull*** sono adatti per riscrivere documenti XML dalla struttura di database, molto ripetitivi:

```
<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>
```


Un esempio di pull (2)

Questo è un foglio di stile XSLT di tipo ***pull*** per generare un documento HTML con quei dati:

```
<xsl:stylesheet>
  <xsl:template match="/">
    <HTML><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="price"/></TD>
        </TR>
      </xsl:for-each>
    </TABLE> </BODY> </HTML>
  </xsl:template>
</xsl:stylesheet>
```


Il foglio di stile per il pull

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.
- **`xsl:template`** dice che questa regola è un template da applicare all'elemento radice del documento XML di partenza (**`match="/"`**). E' quindi possibile applicare la stessa regola anche a sottoparti di un documento XML.
- **`xsl:stylesheet`** è l'elemento radice del foglio di stile e specifica i namespace che vengono utilizzati nel foglio di stile stesso (nel nostro caso, uno per le istruzioni XSL ed uno per il set di tag di destinazione, cioè HTML.

Un esempio di push (1)

Supponiamo di avere questo documento XML dalla struttura di documento, molto diversificato:

```
<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>
```


Un esempio di push (2)

Questo è un foglio di stile XSLT di tipo ***push***:

```
<xsl:stylesheet>
    <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 />
    </xsl:template>
    <xsl:template match="para">
        <P><xsl:apply-templates /></P>
    </xsl:template>
    <xsl:template match="emph">
        <I><xsl:apply-templates /></I>
    </xsl:template>
</xsl:stylesheet>
```


Il foglio di stile per il push

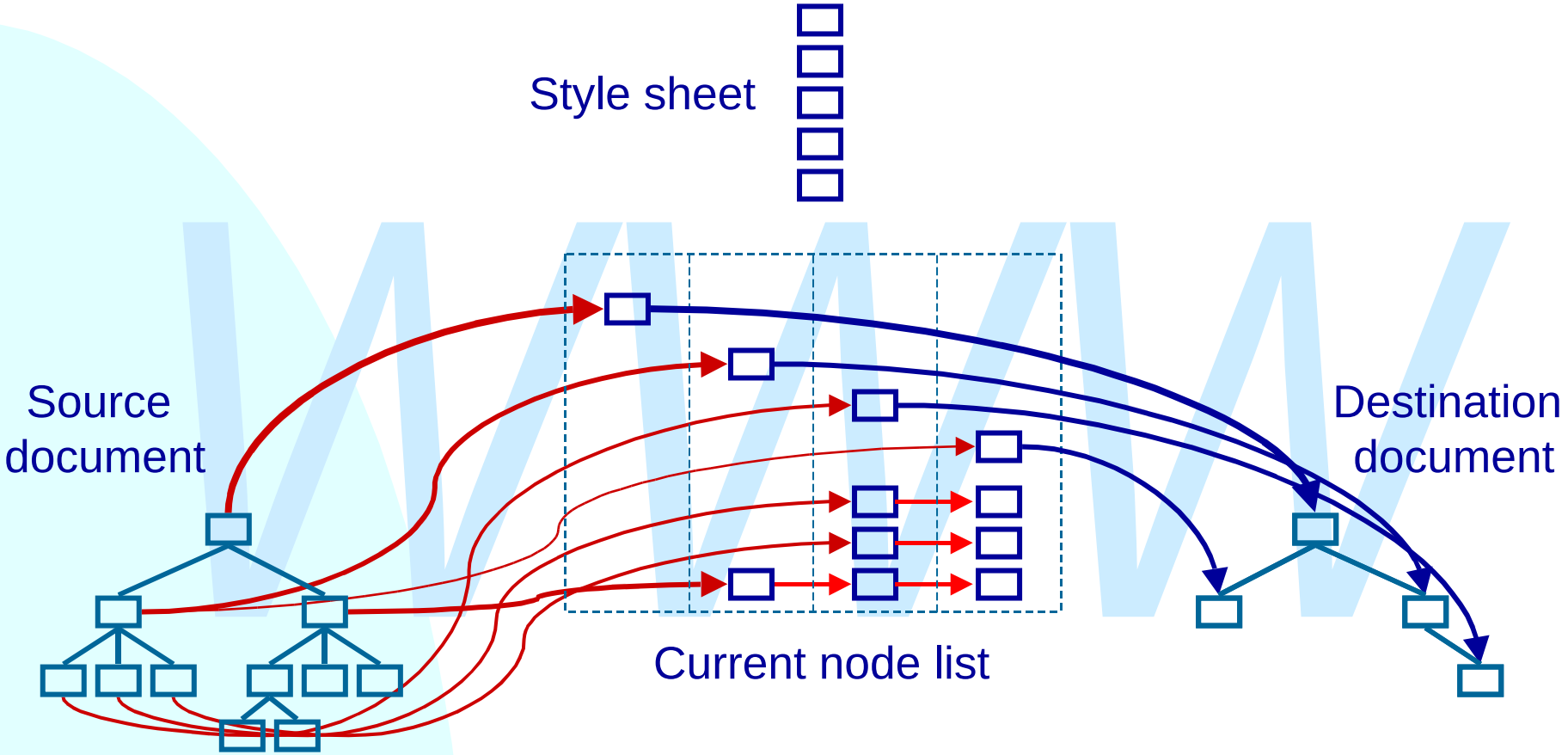
Questo non assomiglia ad un documento HTML, ma ad una serie di istruzioni di riscrittura separate:

- **`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 (1)

- Il parser costruisce una lista di nodi correnti e per default gli inserisce 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.

Modello di processing (2)



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, può essere attivato se il pattern fa match con il nodo corrente

Un template è indicato dall'elemento <template>:

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


I template (2)

Ad esempio, dato il frammento:

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

Il seguente template:

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

Fa match con l'elemento emph e scrive un elemento B di HTML ed inserisce tutti i nodi figlio del nodo di match nella lista dei nodi correnti.

Dentro ad un template

All'interno dell'elemento template vi sono una varietà di istruzioni che servono o per modificare la lista di nodi o per scrivere frammenti dell'albero di destinazione.

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

Scrivere l’albero di destinazione (1)

Poiché l’albero di destinazione è esso stesso un documento XML, debbo poter creare nodi elemento, attributi e testo in maniera sofisticata.

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. La stringa viene combinata con gli altri nodi di testo intorno.
- L'uso tipico è per convertire markup in testo (ad esempio il valore di attributi in contenuto).
- Dato il frammento

```
<persona nome="Fabio" cognome="Vitali"/>
```

e il template

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

ottengo il frammento

```
<p>Fabio Vitali</p>
```


Scrivere l’albero di destinazione (3)

Parentesi graffe {}

- Laddove non sia 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).
- Dato il frammento

```
<persona nome=”Fabio” cognome="Vitali”/>
```

e il template

```
<xsl:template match=“persona”>
  <mail to=“{@nome} {@cognome}”/>
</xsl:template>
```

ottengo il frammento

```
<mail to=“Fabio Vitali”/>
```


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 nel nome di un tag nel documento destinazione il valore di un attributo del dcumento di partenza.
- Dato il frammento

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

e il template

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

ottengo il frammento

```
<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.
- Dato il frammento

```
<persona id="1678.1245" nome="Mario Rossi"/>
```

e il template

```
<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>
```

ottengo il frammento

```
<A HREF="1678.1245.html">Mario Rossi</A>
```


Scrivere l'albero di destinazione (6)

<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.
- Dato il frammento

```
<persona id="1678.1245" nome="Mario Rossi"/>
```

e il template

```
<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>
```

ottengo il frammento

```
<A HREF="1678.1245.html">Mario Rossi</A>
```


Scrivere l'albero di destinazione (7)

<xsl:text>

- Inserisce esplicitamente il testo contenuto dentro al documento.
- E' vantaggioso rispetto a mettere il testo letterale perché rispetta il white space ed i caratteri speciali ("&" e "<", per esempio).

<xsl:processing-instruction>

- Le processing instruction dentro al foglio di stile vengono ignorate, non trasferite. Per scriverle esplicitamente nel documento di arrivo debbo usare <xsl:processing-instruction>
- Ad esempio,

```
<xsl:processing-instruction name="xml-stylesheet">
    href="book.css" type="text/css"
</xsl:processing-instruction>
```

genera il seguente output:

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

<xsl:comment>

- Inserisce esplicitamente del commento dentro al documento.

Scrivere l'albero di destinazione (8)

`<xsl:namespace-alias>`

- Permette di specificareesplicitamente il namespace di arrivo

`<xsl:copy>`

- Copia nell'output il nodo di riferimento, insieme al suo namespace, ma non al suo contenuto e ai 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.)

Priorità tra template

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


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


Cambiare la lista di nodi correnti (1)

Nel corso del processing di un nodo, debbo alimentare la lista dei nodi correnti con altri nodi, figli o altro. Qualunque nodo può essere inserito, anche più volte, nel documento, e in qualunque ordine (non è uno stack!). Alcuni tag XSL permettono di controllare la lista:

<xsl:apply-templates>

- <xsl:apply-templates inserisce nella lista dei nodi corrent i figli diretti dell'elemento considerato, nell'ordine in cui appaiono.
- Se usato con l'attributo select, inserisce solo i figli diretti 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> - segue

- 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>.
- Ad esempio il template:

```
<xsl:template match=”BODY">
    <xsl:apply-templates select=“H1”/>
</xsl:template>
<xsl:template match=”H1">
    <P><xsl:value-of select=“.”/></P>
</xsl:template>
```

viene più facilmente realizzato così:

```
<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 (4)

<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>.
- Gli elementi <xsl:sort> possono annidarsi per realizzare chiavi primarie, secondarie, ecc. per il sort.
- <xsl:sort> ha vari attributi:
  - Select: l'espressione in base alla quale fare il sort
  - Data-type: il tipo di dato da ordinare (numero o testo o altro)
  - Order: il tipo ascendete o discendente di ordine
  - Case-order: come trattare le maiuscole e le minuscole.

Cambiare la lista di nodi correnti (5)

<xsl:sort> - segue

```
<xsl:template match=”persona">
  <ul>
    <xsl:apply-templates select=“.”>
      <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)

E’ possibile raccogliere intere azioni o almeno bocchi di attributi riutilizzabili varie volte in maniera indiretta.

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 $

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


Indirezioni (2)

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 <xs: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>
```


Indirezioni (3)

<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>
```


Altri aspetti di XSLT (1)

Modi

- Permettono di avere template diversi se uso gli stessi nodi di input in più posti diversi. Per esempio, uno stile per le intestazioni negli indici e un altro nel corpo del testo. L'attributo "mode" di <xsl:apply-templates> e di <xsl:template> crea questo binding.

Merging

- E' possibile porre frammenti di folgi 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

Altri aspetti di XSLT (2)

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>.

Template di default

- Esistono delle regole di default che vengono applicate in mancanza di template più specifici. Esse ricopiano semplicemente l’input nell’output.

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="processing-instruction()”/>
<xsl:template match=" comment()"/>
```


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.xsl”?>
<doc> … </doc>
```


Internet Explorer 5.0 e XSLT

Ci sono varie differenze tra XSLT pubblicato nel novembre 1999 e il supporto di Internet Explorer 5.0 ad esso.

Namespace

- XSLT prevede la dicitura: “http://www.w3.org/1999/XSL/Tranform”, mentre Explorer richiede la precedente: “http://www.w3.org/TR/WD-xsl”

Tipo MIME

- Il documento dello stylesheet deve essere dichiarato nella PI <?xml-stylehseet?> di tipo “text/xsl”, invece che “text/xml”

Template di default

- Non esiste nessun template di default per Explorer. In particolare, è necessario fornire template sia per la radice che per l’elemento radice.

Conclusioni

Oggi abbiamo parlato di XSLT, concentrandoci su:

- I template
- Il meccanismo di processing
- Come modificare il documento di uscita
- Come modificare la lista di nodi correnti

Riferimenti

- James Clark, ***XSL Transformations (XSLT) Version 1.0***, W3C Recommendation 16 November 1999, **`http://www.w3.org/TR/xslt`**
- E.R. Harold, XSL Transformations (XSLT), capitolo 14 del libro ***XML Bible***, disponibile in rete: **`http://metalab.unc.edu/xml/books/bible/updates/14.html`**
- James Clark, ***XSLT in Perspective***, **`http://www.jclark.com/xml/xslt-talk.htm`**