Il Linguaggio XSLT
Corso ingegneria
Dicembre 2005

Andrea Marchetti
CNR/IIT - Pisa
Andrea.Marchetti@iit.cnr.it

Indice

Interazione Codice / Dati su web (1/42)

- **Come si elabora un documento?**

Opportunità (2/42)

- **Esperienza di LiveWire - LiveConnect**
- **DHTML: combinazione HTML, Stylesheet e Script**
- **"*XML da a Java qualcosa da fare*" - Jon Bosak**
- **DOM, SAX**
- **XSLT**

XSLT (3/42)

- **DOM e SAX sono le interfacce di programmazione per elaborare XML**
- **Sono a basso livello**
- **Richiedono l'uso di un linguaggio di programmazione**
  - Java
  - VisualBasic
  - Python
  - Perl
  - C#
  - C++
- **XSLT è un linguaggio XML semplice, potente e flessibile dedicato esclusivamente all'elaborazione/trasformazione di documenti XML**

XSLT (4/42)

- **XSLT è uno strumento non una religione**
- **If the only tool you have is a hammer, you tend to see every problem as a nail.*Abraham H. Maslow***

XSLT (5/42)

- **eXtensible Stylesheet Language for Transformations**
- **Official recommendation of W3C - novembre 1999**
- **Linguaggio per trasformare un documento XML in:**
  - nuovo documento XML - ex XHTML, SVG, XSL-FO...
  - documento HTML
  - testo ... ex VRML, Java code, ...

Stylesheets (6/42)

- **Uno stylesheet definisce le proprieta di rendering di un documento XML**
- **W3C definisce due linguaggi per scrivere stylesheet per XML**
  - CSS - 1996 per HTML e XML
  - XSL: XSLT (1999) + XSL-FO (2001)
    - XSLT: linguaggio di trasformazione
    - XSL-FO: linguaggio di rendering

CSS (7/42)

- **Consente di associare informazioni di rendering a documenti HTML e XML**
- **Più usato per HTML**
- **CSS ha alcuni limiti**
  - Non può modificare l'ordine con cui gli elementi appaiono in un documento (ex sort, filter)
  - Non può fare computazioni (ex la somma di alcuni campi)
  - Non può cominare più documenti
- **CSS comunque torna comodo quando trasfomo un documento xml in html + css**
- **CSS va visto come alternativa a XSL-FO**

XSLT - Modello (8/42)

- **Strumento potente e flessibile per trasformare documenti XML**
- **Linguaggio basato su regole di pattern matching**
  - ciascuna regola dice "se trovi una parte del documento simile a questo allora sostituiscilo con quest'altro
- **Nessun effetto collaterale ovvero l'esecuzione di una regola non deve interferire con l'esecuzione di una successiva**
  - come conseguenza avremo che le variabili non possono essere modificate dalle regole

XSLT - campi di applicazione (9/42)

- **Un sito web deve rendere i suoi documenti su dispositivi differenti (PC, PDA, Mobile Phone, Printers, qualsiasi dispositivo con differenti risoluzioni e differenti funzioni)**
- **Scambio dati tra differenti formati di partenza**
- **Continui cambi nello stile di un web**

Obiettivi (10/42)

- **Capire come un processor XSLT usa uno stylesheet per trasformare un documento XML**
- **La struttura di uno stylesheet XSLT**
- **Esercizio**

XSLT Processor (11/42)

Il documento XML da trasformare (12/42)

- ```
  <?xml version="1.0" ?>
  <greeting>Hello, World!</greeting>
  ```
- **greeting.xml**
- **quello che vogliamo fare è trasformarlo in un documento html**
- ```
  <html>
     <body>
        <h1>Hello World!</h1>
     </body>
  </html>
  ```
- **per visualizzarlo su qualsiasi browser**

Lo stylesheet XSLT (13/42)

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

  <xsl:output method="html"/>

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

  <xsl:template match="greeting">
    <html>
      <body>
        <h1>
          <xsl:value-of select="."/>
        </h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

## Trasformazione del documento XML (14/42)

- **Si usa Internet Explorer**
- ```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="greeting.xsl"?>
<greeting>Hello, World!</greeting>
```
- **greetingPlusStylesheet.xml**
- **Il processo si sviluppa in 3 fasi**

I Fase: Lettura dello sylesheet (15/42)

- **Contiene tre istruzioni**
-

```
<xsl:output method="html"/>
<xsl:template match="/">
  ...
</xsl:template>
<xsl:template match="greeting">
  ...
</xsl:template>
```

- ***<xsl:output>*** **specifica che il formato di output sarà** ***HTML***
- **due regole di trasformazione** ***<xsl:template>*** **specificano come le parti del documento XML devono essere trasformate**

II Fase: Lettura del documento XML di input (16/42)

- **Il processore XSLT costruisce un albero del documento sorgente**
- **Source Tree**

III Fase: Determinazione del contesto, il nodo corrente (17/42)

- **Hai qualche nodo del documento XML da elaborare?**
  - L'insieme dei nodi che devono essere elaborati ad ogni passo è detto *contesto*
  - Inizialmente il *contesto* è caricato dal parser con la radice del documento XML: *Contesto = { / }*
  - Si può modificare il contesto tramite l'istruzione *<xsl:apply-templates select=".."/>*
- **Per ogni nodo presente nel contesto seleziona un nodo che diventa il *nodo corrente***
  - Inizialmente quindi il nodo corrente è /

III Fase: Determinazione regola (template) (18/42)

- **Cerca una regola (template) che soddisfi (match) il nodo corrente**
  - Se esistono più regole che soddisfano prendi la più specifica (vedi dopo)
  - Se non esistono regole selezionala una tra le regole predefinite (vedi dopo)
  - Esistono template di default per gestire qualsiasi tipo di nodo
- **Un processor XSLT inizia sempre la sua elaborazione applicando la regola *<xsl:template match="/">***

III Fase: Applicazione regola (template) (19/42)

- **Determinata la regola applica la trasformazione associata**
  - Tutti gli elementi che non appartengono allo spazio *<xsl:... >*, compreso il testo, vengono copiati inalterati in output
  - Se all'interno del template il parser trova l'istruzione *<xsl:apply-templates select=".."/>* congela il contesto attuale e apre un nuovo contesto (ricorsione)

```
Old context {/}
New context {node list}
```

  - Il nuovo contesto viene riempito con i nodi selezionati tramite l'attributo *select*
  - Tramite l'attributo *select* si può selezionare quali nodi dell'albero sorgente elaborare
  - Una volta elaborati tutti i nodi del nuovo contesto si torna al vecchio

III Fase: Determinazione del nuovo nodo corrente (20/42)

- **Applicata la regola**
  - *Se* nel contesto corrente ci sono ancora nodi
    - prendi il nodo successivo che diventa il nuovo nodo corrente
  - *Altrimenti* torna al contesto precedente
  - *Se* non ci sono altri contesti hai finito

Rule/Template predefiniti o di default (built-in) (21/42)

- **Regola di default per il nodo radice e i nodi elementi**
- ```
  <xsl:template match="*|/">
     <xsl:apply-templates />
  </xsl:template>
  ```
  - questa regola assicura che il processo ricorsivo continui anche se non sono stati inseriti tutti i template necessari
- **Regola di default per i nodi testo e i nodi attributi**
- ```
  <xsl:template match="text()|@*">
     <xsl:value-of select="." />
  </xsl:template>
  ```
  - questa regola assicura che tutti i nodi testo e i nodi attributi vengano inseriti nel documento di output
- **Bisogna sempre tenerle presenti altrimenti si rischia di non capire il funzionamento del parser**
- **A volte conviene riscriverle per modificare il cambiamento di default**

Rule/Template predefiniti o di default (built-in) (22/42)

- **Regola di default per i nodi commenti e i nodi processing-instruction**
- ```
  <xsl:template match="comment()|processing-instruction()">
     <!-- non fare nulla -->
  </xsl:template>
  ```
  - questa regola assicura che i nodi commento e i nodi processing-instruction vengano esclusi dal documento di output
- **Regola di default per i nodi namespace**
- ```
  <xsl:template match="namespace::node()" />
     <!-- non fare nulla -->
  </xsl:template>
  ```
  - questa regola assicura che tutti i nodi namespace vengano esclusi dal documento di output
- **Regola di default per il nodo radice e i nodi elementi con modo X**
- ```
  <xsl:template match="*|/" mode="x">
     <xsl:apply-templates mode="x"/>
  </xsl:template>
  ```
  - questa regola assicura che il processo ricorsivo continui considerando anche i template con modo

Priorità tra template (23/42)

- **Quando più template soddisfano tramite l'attributo match il nodo corrente il processore XSLT segue alcune regole di priorità per selezionare un solo template**
  ```

Caricamento stylesheet (24/42)

- **Ci sono due regole**
-

```
<!-- Prima regola -->
<xsl:template match="/">
  <xsl:apply-templates select="greeting"/>
</xsl:template>
```

-

```
<!--  Seconda regola -->
<xsl:template match="greeting">
  <html>
    <body>
      <h1>
        <xsl:value-of select="."/>
      </h1>
    </body>
  </html>
</xsl:template>
```

Caricamento XML input (25/42)

-

```
<?xml version="1.0" ?>
<greeting>Hello, World!</greeting>
```

- **Si costituisce il contesto iniziale formato dal solo nodo radice *{ / }***
  - In xslt il *nodo radice* non coincide con il *document element* il quale è un figlio del nodo radice.
  - Altri figli possono essere le processing instruction come *<?xml-stylesheet ... ?>*

Elaborazione (26/42)

- **3) Prendi il primo ed unico nodo dal contesto, che diventa il *nodo corrente* e verifica se esistono delle regole che lo soddisfano**
  - La prima regola soddisfa il nodo corrente
  -
```
<xsl:template match="/">
  <xsl:apply-templates select="greeting"/>
</xsl:template>
```
- **4) Si esegue la prima regola (template)**
- **5) Contiene una sola istruzione *<xsl:apply-templates select="greeting"/>***
  - A partire dal nodo corrente controlla se esistono elementi di tipo *greeting* che costituiranno il nuovo contesto su cui applicare le regole
- **6) Poichè esiste un solo elemento greeting figlio del nodo radice il nuovo contesto sarà *{ greeting }***

Elaborazione (27/42)

- **7) greeting diventa il nuovo nodo corrente**
- **8) Si cerca di nuovo una regola che soddisfi questo nodo corrente**
  - La seconda regola soddisfa il nodo corrente
  -
```
<xsl:template match="greeting">
  <html>
    <body>
      <h1>
        <xsl:value-of select="."/>
      </h1>
    </body>
  </html>
</xsl:template>
```

Elaborazione (28/42)

- **9) Si esegue la seconda regola (template)**
  - I primi tre oggetti (<html>, <body> e <h1>) sono i tag di apertura di elementi HTML
  - Poichè non appartengono al namespace di xsl allora il processor XSLT li passa in uscita inalterati
  - Nel mezzo troviamo l'istruzione *<xsl:value-of select="."/>*.
  - Questa istruzione scrive il contenuto di qualche nodo in uscita
  - Tale nodo è indicato con l'attributo *select*
  - Il nodo . indica il nodo corrente, che nel nostro caso è *greeting*
  - Per ultimo abbiamo i tag di chiusura, anch'essi passati inalterati in uscita

Elaborazione (29/42)

- **10) Abbiamo terminato il template, abbiamo anche esaurito i nodi del contesto *{greeting}*, si torna al template chiamante**
- **11) Abbiamo terminato anche il template chiamante anche il contesto è esaurito *{ / }***
- **12) The End**

Altri approcci (30/42)

- **Naturalmente si potevano seguire altre strade nel disegnare lo stylesheet per ottenere gli stessi risultati**
- **Un esempio di stylesheet equivalente**
-

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

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <body>
          <xsl:apply-templates select="greeting"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="greeting">
     <h1><xsl:value-of select="."/></h1>
  </xsl:template>

</xsl:stylesheet>
```

- **greetingPlusStylesheet1.xml**

Altri stylesheet per lo stesso documento di input (31/42)

- **Uno stylesheet XSLT è composto**
  - da istruzioni appartenenti al namespace di XSLT di solito indicato con il prefisso *xsl:*
  - da qualsiasi altra cosa basta che rimanga un documento XML valido
-

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

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <body>
           <p>Finalmente qui posso sbizzarrirmi a fare quello che voglio ...</p>
           <xsl:apply-templates select="greeting"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="greeting">
    <h1><xsl:value-of select="."/></h1>
    <b>Basta che rispetti le regole ... di XML </b>
  </xsl:template>

</xsl:stylesheet>
```

- **greetingPlusStylesheet2.xml**

Una lista di saluti (32/42)

- **Consideriamo un documento XML**
-

```
<?xml version="1.0" ?>
<greetings>
   <greeting>Hello, World!</greeting>
   <greeting>I'm fine!</greeting>
</greetings>
```

- **Vogliamo ottenere in output sempre un documento HTML**
-

```
<html>
   <body>
      <h1>Greeting's List</h1>
      <ol>
         <li>Hello, World!</li>
         <li>I'm fine!</li>
     </ol>
   </body>
</html>
```

- **Stylesheet XSLT**
- **Verifica Output**

Hello World in tutti i formati (33/42)

- **Scalable Vector Graphics (SVG)**
- **XSL-FO**
- **Programma Java**
- **Virtual Reality Modelling Language (VRML)**

SVG (34/42)

- **Stylesheet**
-

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

  <xsl:output method="xml"
    doctype-public="-//W3C//DTD SVG 20001102//EN"
    doctype-system="http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd"/>

  <xsl:template match="/">
    <svg width="10cm" height="4cm">
      <g>
        <defs>
          <radialGradient id="MyGradient"
            cx="4cm" cy="2cm" r="3cm" fx="4cm" fy="2cm">
            <stop offset="0%" style="stop-color:red"/>
            <stop offset="50%" style="stop-color:blue"/>
            <stop offset="100%" style="stop-color:red"/>
          </radialGradient>
        </defs>
        <rect style="fill:url(#MyGradient); stroke:black"
          x="1cm" y="1cm" width="8cm" height="2cm"/>
        <text x="5.05cm" y="2.25cm" text-anchor="middle"
          style="font-family:Verdana; font-size:24;
          font-weight:bold; fill:black">
          <xsl:apply-templates select="greeting"/>
        </text>
        <text x="5cm" y="2.2cm" text-anchor="middle"
          style="font-family:Verdana; font-size:24;
          font-weight:bold; fill:white">
          <xsl:apply-templates select="greeting"/>
        </text>
      </g>
    </svg>
  </xsl:template>

  <xsl:template match="greeting">
    <xsl:value-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

- **Output**
-

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
                     "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
<svg height="4cm" width="8cm">
   <g>
      <defs>
         <radialGradient fy="2cm" fx="4cm" r="3cm" cy="2cm" cx="4cm" id="MyGradient">
            <stop style="stop-color:red" offset="0%"/>
            <stop style="stop-color:blue" offset="50%"/>
            <stop style="stop-color:red" offset="100%"/>
         </radialGradient>
      </defs>

      <rect height="2cm" width="6cm" y="1cm" x="1cm" style="fill:url(#MyGradient); stroke:black"/>
      <text style="font-family:Verdana; font-size:24; font-weight:bold; fill:black"
                  text-anchor="middle" y="2.2cm" x="4cm">
         Hello, World!
      </text>
   </g>
</svg>
```

- **Demo**

XSL-FO trasformato in PDF (35/42)

- **Stylesheet**
-

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

  <xsl:output method="xml"/>

  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master margin-right="75pt" margin-left="75pt"
          page-height="11in" page-width="8.5in"
          margin-bottom="25pt" margin-top="25pt" master-name="main">
          <fo:region-before extent="25pt"/>
          <fo:region-body margin-top="50pt" margin-bottom="50pt"/>
          <fo:region-after extent="25pt"/>
        </fo:simple-page-master>
        <fo:page-sequence-master master-name="standard">
          <fo:repeatable-page-master-alternatives>
            <fo:conditional-page-master-reference master-name="main" odd-or-even="any"/>
          </fo:repeatable-page-master-alternatives>
        </fo:page-sequence-master>
      </fo:layout-master-set>

      <fo:page-sequence master-name="standard">
        <fo:flow flow-name="xsl-region-body">
          <xsl:apply-templates select="greeting"/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>

  <xsl:template match="greeting">
    <fo:block line-height="76pt" font-size="72pt" text-align="center">
      <xsl:value-of select="."/>
    </fo:block>
  </xsl:template>

</xsl:stylesheet>
```

- **Output XSL-FO**
-

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <fo:layout-master-set>
      <fo:simple-page-master master-name="main" margin-top="25pt" margin-bottom="25pt"
                page-width="8.5in" page-height="11in" margin-left="75pt" margin-right="75pt">
         <fo:region-before extent="25pt"/>
         <fo:region-body margin-bottom="50pt" margin-top="50pt"/>
         <fo:region-after extent="25pt"/>
      </fo:simple-page-master>

      <fo:page-sequence-master master-name="standard">
         <fo:repeatable-page-master-alternatives>
            <fo:conditional-page-master-reference odd-or-even="any" master-name="main"/>
         </fo:repeatable-page-master-alternatives>
      </fo:page-sequence-master>
   </fo:layout-master-set>

   <fo:page-sequence master-name="standard">
      <fo:flow flow-name="xsl-region-body">
         <fo:block text-align="center" font-size="72pt" line-height="76pt">
            Hello, World!
         </fo:block>
      </fo:flow>
   </fo:page-sequence>
```

```
</fo:root>
```

- **Si deve trasformare XSL-FO in PDF**
- **Demo**

Java Code (36/42)

- **Stylesheet**
-

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

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>
public class Greeting
{
  public static void main(String[] argv)
  {
    </xsl:text>
    <xsl:apply-templates select="greeting"/>
    <xsl:text>
  }
}
    </xsl:text>
  </xsl:template>

  <xsl:template match="greeting">
    <xsl:text>System.out.println("</xsl:text>
    <xsl:value-of select="normalize-space()"/>
    <xsl:text>");</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

- **Output**
-

```
public class Greeting
{
   public static void main(String[] argv)
   {
      System.out.println("Hello, World!");
   }
}
```

- **Demo**

VRML (37/42)

- **Stylesheet**
-

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

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>#VRML V2.0 utf8

Shape
{
  geometry ElevationGrid
  {
    xDimension 9
    zDimension 9
    xSpacing 1
    zSpacing 1
    height
    [
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
    ]
    colorPerVertex FALSE
    color Color
    {
      color
      [
        0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
        1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
        0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
        1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
        0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
        1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
        0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
        1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
      ]
    }
  }
}

Transform
{
  translation 4.5 1 4
  children
  [
    Shape
    {
      geometry Text
      {
    </xsl:text>
    <xsl:apply-templates select="greeting"/>
    <xsl:text>
        fontStyle FontStyle
        {
          justify "MIDDLE"
          style "BOLD"
        }
      }
    }
  ]
}

NavigationInfo
{
```

```
  type ["EXAMINE","ANY"]
}

Viewpoint
{
  position 4 1 10
}
    </xsl:text>
  </xsl:template>

  <xsl:template match="greeting">
    <xsl:text>string"</xsl:text>
    <xsl:value-of select="normalize-space()"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

- **Output**
-
- **Demo**

Espressioni (38/42)

- **XSLT Processor seleziona un nodo dal documento sorgente e gli associa un template**
- **Inizialmente XSLT Processor parte dal nodo radice applica il template relativo che è sempre presente**
- **Le istruzioni che consentono di iterare questo funzionamento sono due:**
- ***<xsl:apply-templates select="select expression">***
  - Una *select expression* seleziona una lista di nodi che costituiranno il nuovo contesto da cui estrarre il nodo corrente
  - Questa istruzione consente di selezionare i nodi dell'albero sorgente che si intende elaborare
- ***<xsl:template match="pattern expression">***
  - Una *pattern expression* permette di selezionare il template che soddisfa il nodo corrente
  - Questa istruzione dice come elaborare i nodi dell'albero sorgente
- **Pattern e Select expression sono sottoinsiemi delle espressioni XPath**

Metodo tradizionale: una sola regola (39/42)

- **Si può continuare a scrivere gli stylesheet in modo deterministico**
- **Si definisce una sola regola apply-templates select="/"**
-

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

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <body>
        <h1>
          <xsl:value-of select="greeting"/>
        </h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Xslt Processor (40/42)

- ***Xalan*** **- Progetto Apache xml.apache.org/xalan-j/**
- ***MsXml*** **- Microsoft www.microsoft.com/xml**
- ***XT*** **- James Clark www.jclark.com/xml/xt.html**
- ***Saxon*** **- Michael Kay saxon.sourceforge.net**
- ***Oracle*** **- Oracletechnet.oracle.com/tech/xml**

Editor Stylesheet XSLT (41/42)

- **Xselerator www.marrowsoft.com**
- **Stylus**

Libri (42/42)

- **XSLT - Mastering XML trasformations - O'Reilly - 2001**
- **XSLT Cookbook - O'Reilly - 2002**
- **The XSL Companion - Addison-Wesley - 2000**
- **W3C Recommendation - novembre 1999**