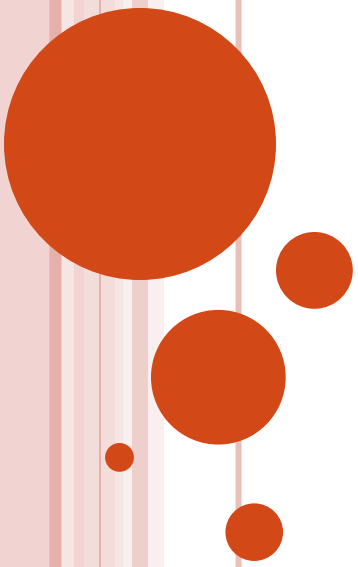
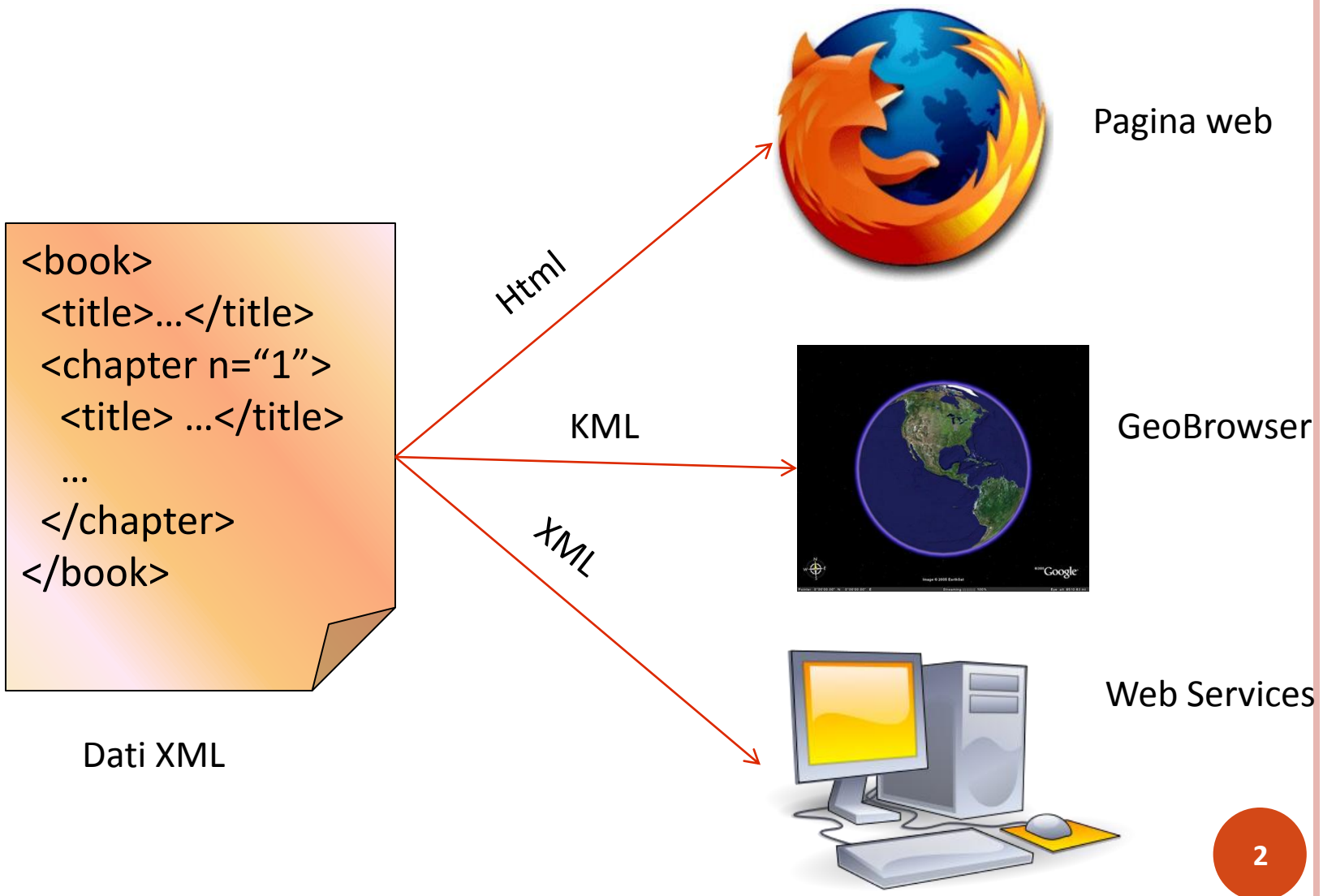
XSLT
Extensible Stylesheet Language Transformation



Corso di Basi di Dati II mod. B
A.A. 2009/2010
Docente: Francesco Cutugno
Slide a cura di: Enza Leano

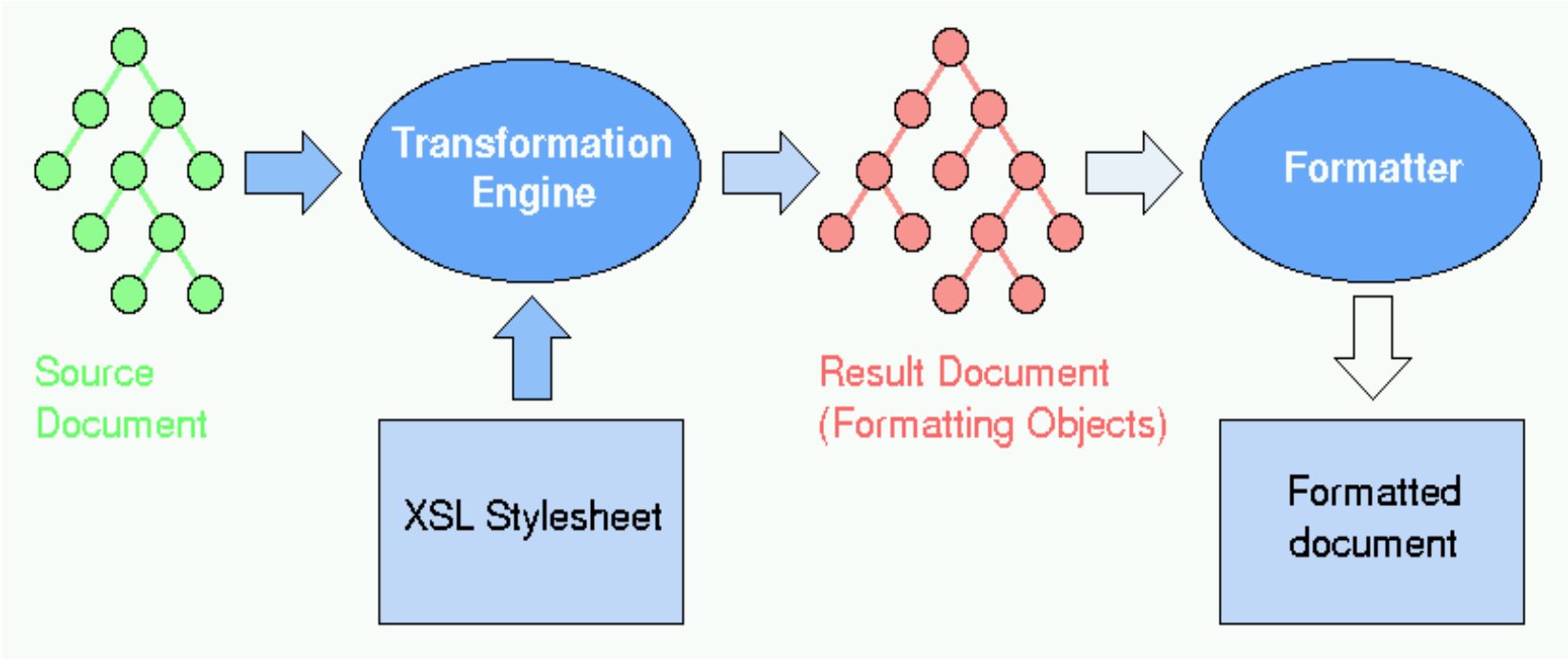
Introduzione



XSL :Trasformazione documenti XML

- **XSL (eXtensible Stylesheet Language Family)** è una famiglia di raccomandazioni W3C per definire la trasformazione e la presentazione di documenti XML. Consiste di 3 linguaggi:
  1. XSL Transformations (XSLT): linguaggio che definisce le regole di trasformazione;
  2. XML Path Language (Xpath): linguaggio di espressioni utilizato da XSLT per accedere al (o a parti del) documento
  3. XSL Formatting Objects (XSL-FO): linguaggio che formatta l'output della trasformazione.

Architettura XSL



XSL - FO

- XSL-FO (Formatting Objects) è un linguaggio XML per descrivere il layout fisico di testi
- E’ stato pensato originariamente per interagire con l’output di XSLT
- Ad oggi non è supportato da nessun browser
- E’ molto utilizzato nell’industria grafica come standard di scambio tra i diversi formati grafici

XSLT TRANSFORMATION

- Una trasformazione in linguaggio XSLT è espressa come un documento XML ben formato.
- Esprime regole per trasformare un source tree in un result tree.
- E' anche chiamata stylesheet.
- Uno stylesheet è composto da una serie di template rules. Ognuna di queste regole ha due parti:
  - Un pattern, che identifica i nodi del source tree interessati nella trasformazione
  - Una regola di template, che trasforma i nodi selezionati in nodi dell'albero di destinazione.

Valutazione di uno Stylesheet

- Le valutazioni delle regole in uno stylesheet vengono fatte a partire da un contesto corrente.
- Il parser costruisce una lista di nodi correnti e per default inizia dal nodo radice.
  - Ad ogni nodo del documento di origine viene applicata una regola, se non è stata definita nello stylesheet ne viene applicata una di default.
- Seleziona tutti i template applicabili al contesto corrente, se ve ne è più d'uno viene selezionato il più specifico.
- Il template viene applicato al nodo corrente creando frammenti dell'albero di destinazione, invocando ricorsivamente altri template e inserendo altri nodi nella lista di nodi correnti.
- Il ciclo si ripete fino ad esaurimento della lista di nodi correnti.

XSL:TEMPLATE

```
<xsl:template
 match = pattern
 name = qname
 priority = number
 mode = qname>
 <!-- Content:
(xsl:param*, template
) -->
</xsl:template>
```

- `name`: usato per chiamare i template come delle funzioni
- `mode`: usato per restringere i template candidati, permesso solo se match è presente
- `priority`: priorità della regola

Pattern

- Un *pattern è un espressione XPath ristretta*
  - È un unione di path expressions (*patExp1* | *patExp2* |…)
  - ogni path expression contiene un numero di passi separati da / o //
  - Ogni passo può usare solo l'axis child o l'axis attribute
  - E' possibile utilizzare funzioni Xpath (*text()*…)
- Un pattern è il *match di un nodo se* a partire da qualche nodo del documento di origine, il nodo dato è contenuto nella nodelist risultato:
  - A pattern P matches node A ⇔ A ∈ P(someNode)

Risoluzioni di conflitti tra regole

- E' possibile che più regole siano il match di un nodo:
  - XSLT processor seleziona il **best match**
- Regole di precedenza:
  - Se il pattern contiene più alternative separate da |, ognuna di esse è trattata alla pari, come se ci fosse una regola separata per ogni alternativa.
  - Un pattern specifico ha una priorità più alta di uno che contiene informazioni generali.
    - `quotelist/quote/body` è più specifico di `body`, e ha precedenza
  - Un pattern che contiene un * è più generale e ha una priorità minore rispetto a un pattern specifico.
    - Il pattern `stuff/cruft` batte il pattern `stuff/*`
  - A pattern con una test expression racchiusa tra [] sovrascrive un pattern senza test expression
    - `bobo[@role="clown"]` ha maggiore priorità di `bobo`

Default rules

- Le default rules sono applicate a quei nodi per i quali lo styelesheet non ha previsto una regola di template
- Le default rules hanno la priorità più bassa.
- Ogni tipo di nodo ha una sua regola di default

Default rules

- **Root ed elementi:**

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

invoca il template che è un match del figlio del nodo corrente.

- **Nodi testo e attributo**

```
<xsl:template match=" text() | @* ">
      <xsl:value-of select="."/>
  </xsl:template>
```

Si limita a copiarne il valore

- **Processing instruction o commenti:**

```
<xsl:template match="processing-instruction()
  |comment()"/>
```

Nessuna azione

- **Nodi di namespace:**
  - La regola di default è quella di non eseguire nessuna azione
  - Poiché non ci sono pattern che sono il match di un nodo di namespace, la regola di default è l'unica ad essere applicata

Esempio

```
<?xml-stylesheet type="text/xsl"
  href="businesscard.xsl"?>
  <card xmlns="http://businesscard.org">
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.inc</email>
  <phone>(202) 555-1414</phone>
  <logo uri="widget.gif"/>
</card>
```

StyleSheet Esempio (1/2)

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:b="http://businesscard.org"
    xmlns="http://www.w3.org/1999/xhtml">
    <xsl:template match="b:card">
      <html>
      <head>
      <title><xsl:value-of select="b:name/text()"/></title>
      </head>
     <body bgcolor="#ffffff">
      <table border="3">
     <tr>
    <td>
              <xsl:apply-templates select="b:name"/><br/>
              <xsl:apply-templates select="b:title"/><p/>
    <tt>   <xsl:apply-templates select="b:email"/></tt><br/>
```

StyleSheet Esempio (2/2)

```
      <xsl:if test="b:phone">
          Phone: <xsl:apply-templates select="b:phone"/><br/>
      </xsl:if>
     </td>
     <td>
      <xsl:if test="b:logo">
              <img src="{b:logo/@uri}"/>
     </xsl:if>
     </td>
     </tr>
  </table>
</body>
 </html>
 </xsl:template>
     <xsl:template match="b:name|b:title|b:email|b:phone">
         <xsl:value-of select="text()"/>
     </xsl:template>
</xsl:stylesheet>
```

Corpo di un template

- Il corpo di un template puo' contenere due tipi di istruzioni:
  - Istruzioni che creano e modificano 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>
  - Istruzioni che modificano la lista dei nodi correnti
    - <xsl:apply-templates>
    - <xsl:if>
    - <xsl:sort>
    - <xsl:for-each>
    - <xsl:choose>

Istruzioni di costruzione Albero di destinazione

Costruzione dell'albero di destinazione

1. Costruttori letterali
   - Sequenza di nodi che contengono dati carattere e nodi elemento che non appartengono al namespace XSLT
2. Costruttori specifici
   - `<xsl:element name="">`
   - `<xsl:attribute name="">`: l'attributo creato fa riferimento all'elemento padre del tag (un xsl:element o un tag non appartenente al namespace xslt)
   - `<xsl:text>`: gli elementi all'interno del tag preservano gli spazi.
   - `<xsl:processing-instruction name="">`
   - `<xsl:comment select="">`

ESEMPIO: COSTRUTTORI LETTERALI

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
   <html>
    <head>
     <title>Hello World</title>
    </head>
    <body>
     <xsl:attribute name="bgcolor" select="'green'"/>
     <b>Hello World</b>
    </body>
   </html>
  </xsl:template>
</xsl:stylesheet>
```

Esempio Costruttori specifici

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
    <xsl:element name="html">
     <xsl:element name="head">
       <xsl:element name="title">Hello World</xsl:element>
     </xsl:element>
     <xsl:element name="body">
      <xsl:attribute name="bgcolor" select="'green'"/>
     <xsl:element name="b">Hello World</xsl:element>
     </xsl:element>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

XSLT value-of Element

```
<xsl:value-of
  select = string-expression
  disable-output-escaping= "yes"|"no"/>
```

- Valuta l'espressione e mostra il risultato
- Se applicata a una nodelist l'espressione viene applicata al primo nodo
- "." seleziona il text value del nodo corrente
- `disable-output-escaping`: se vale yes sono consentiti i valori "<" e "&"

I VALORI PER OGNI TIPO DI NODO

- Root:
  - Il nodo radice eredita il valore del document element
- Element
  - Il testo dell'elemento insieme a tutti I suoi figli di tipo testo o elemento
- Attribute
  - Il valore dell'attributo con gli spazzi normalizzati
- Text
  - Tutto il testo nel nodo
- Processing instruction
  - Tutto all'interno del delimitatore eccetto il nome
- Comment
  - Testo all'interno del delimitatore del commento
- Namespace
  - L'URI del namespace

Esempio

- Input:

```
<name prefix="Mr.">John Doe</name>
```

- Xslt:

```
<xsl:template match="name">
    Your name is
    <xsl:value-of select="@prefix"/>
    <xsl:value-of select="."/>
</xsl:template>
```

- Output:
  - `Your Name is Mr. John Doe`

Copiare nodi

```
<xsl:copy-of
    select = expression />
```

- Copy-of inserisce il frammento di source tree derivante dalla select nel result tree senza trasformare prima il valore in testo come quando utilizziamo value-of

```
<xsl:copy>
    <!-- Content: template -->
</xsl:copy>
```

- Crea una copia locale del nodo corrente, senza attributi nè figli.

Esempio Copy

- Cambiare il punto degli elenchi di html in un quadrato:

```
<UL>
    <li>…</li>
        …
   <li>…<li>
</UL>
```

```
<xsl:template match="ol|ul">
    <xsl:copy>
          <xsl:attribute name="style"
            select="'list-style-type:square;'"/>
          <xsl:copy-of select="*"/>
   </xsl:copy>
</xsl:template>
```

Istruzioni per modificare la NodeList

Applicazione ricorsiva

```
<xsl:apply-templates
      select=""
      mode=""/>
```

Applica ricorsivamente l'intero foglio di stile ai nodi specificati dall'attributo select.

- Concatena la sequenza risultante
- Il valore di default di select è child::node()
- Utilizzando l'attributo mode vengono prese in considerazione solo le trasformazioni che hanno un mode corrispondente.

Student Data

```
<students>
   <student id="100026">
      <name">Joe Average</name>
      <age>21</age>
      <major>Biology</major>
      <results>
        <result course="Math 101" grade="C-"/>
        <result course="Biology 101" grade="C+"/>
        <result course="Statistics 101" grade="D"/>
      </results>
   </student>
   <student id="100078">
      <name>Jack Doe</name>
      <age>18</age>
      <major>Physics</major>
      <major>XML Science</major>
      <results>
        <result course="Math 101" grade="A"/>
        <result course="XML 101" grade="A-"/>
        <result course="Physics 101" grade="B+"/>
        <result course="XML 102" grade="A"/>
      </results>
   </student>
</students>
```

```
<summary>
  <name id="100026">Joe Average</name>
  <name id="100078">Jack Doe</name>
  <grades id="100026">
    <grade>C-</grade>
    <grade>C+</grade>
    <grade>D</grade>
  </grades>
  <grades id="100078">
    <grade>A</grade>
    <grade>A-</grade>
    <grade>B+</grade>
    <grade>A</grade>
  </grades>
</summary>
```

STYLESHEET PER IL SUMMARY (1/2)

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="students">
    <summary>
      <xsl:apply-templates mode="names" select="student"/>
      <xsl:apply-templates mode="grades" select="student"/>
    </summary>
  </xsl:template>

  <xsl:template mode="names" match="student">
    <name>
      <xsl:attribute name="id" select="@id"/>
      <xsl:value-of select="name"/>
    </name>
  </xsl:template>
```

STYLESHEET PER IL SUMMARY (2/2)

```
<xsl:template mode="grades" match="student">
    <grades>
      <xsl:attribute name="id" select="@id"/>
      <xsl:apply-templates select=".//@grade"/>
    </grades>
  </xsl:template>

  <xsl:template match="@grade">
    <grade>
      <xsl:value-of select="."/>
    </grade>
  </xsl:template>
</xsl:stylesheet>
```

Invocazione di template

- Se un template ha l'attributo name, può essere invocato all'interno dello stylesheet attraverso l'istruzione

```
<xsl:call-template name="">
```

- Utile per sfruttare il principio di modularità.
- Applica una regola al contesto corrente
- Utile per superare il meccanismo del pattern matching.

ITERAZIONE

- E’ possibile iterare sui nodi di una nodelist con

  <xsl:for-each select=“”>

- Calcola la sequenza specificata dall’attributo select e applica la regola specificata nel suo corpo ad ogni elemento della selezione

```
<xsl:template match="student">
  <grades>
   <xsl:attribute name="id" select="@id"/>
   <xsl:for-each select=".//@grade">
     <xsl:call-template name="listgrade"/>
   </xsl:for-each>
  </grades>
</xsl:template>

<xsl:template name="listgrade">
 <grade>
  <xsl:value-of select="."/>
 </grade>
</xsl:template>
```

ISTRUZIONI CONDIZIONALI: XSLT:IF

```
<xsl:if test="expression">
<!-- context --!>
</xsl:if>
```

- Valuta l'espressione se è vera applica la regola
- Se è falsa non produce output

```
<xsl:template match="class">
  <!-- Select the first node in the set. -->
  <xsl:if test="position() = first()">
    <b><xsl:value-of select="."/></b>
  </xsl:if>
 </xsl:template>
```

ISTRUZIONI CONDIZIONALI: XSLT:CHOOSE

```
<xsl:choose>
 <xsl:when test = boolean-expression>
  <!-- Content: template -->
 </xsl:when>
…
 <xsl:otherwise>
    <!-- Content: template -->
  </xsl:otherwise>
</xsl:choose>
```

- Seleziona tra un numero di alternative.
- Viene eseguito il primo `When` il cui test risulta vero
- Se nessun `when` risulta vero viene eseguito `otherwise` se presente

ORDINAMENTO

- ```
  <xsl:sort
    select = string-expression
    lang = { nmtoken }
    data-type = { "text" | "number" | qname-but-not-ncname }
    order = { "ascending" | "descending" }
    case-order = { "upper-first" | "lower-first" } />
  ```
- Ordina le sequenze selezionate dall'attributo select in base ai valori dei parametri
- L'ordine delle sort nel documento definisce la priorità dei principi di ordinamento.
  ```

Esempio di ordinamento

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="students">
    <enrolled>
      <xsl:apply-templates select="student">
        <xsl:sort select="age" data-type="number"
                                order="descending"/>
        <xsl:sort select="name"/>
      </xsl:apply-templates>
    </enrolled>
  </xsl:template>

  <xsl:template match="student">
    <student name="{name}" age="{age}"/>
  </xsl:template>
</xsl:stylesheet>
```

RAGGRUPPAMENTI

```
<xsl:for-each-group select="" group-by="">
    <!- context --!>
</xsl:for-each group>
```

- Il contesto può contenere due funzioni:
  - `current-grouping-key()`: restituisce la chiave di raggruppamento corrente
  - `current-group()`: restituisce la sequenza di componenti con quella data chiave
- Valutazione:
  - Raggruppa gli elementi di una selezione in base a una chiave.
  - L'espressione di select viene valutata-> popolazione
  - Valutazione per ogni elemento della popolazione dell'espressione di group-by
  - Confronto dei valori risultanti-> chiavi di raggruppamento distine
  - Valutazione del contesto della regola per ogni chiave
  - Concatenazione risultati

Esempio

```
<xsl:stylesheet version="2.0"
     xmlns:rcp="http://www.brics.dk/ixwt/recipes"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="rcp:collection">
      <uses>
        <xsl:for-each-group select="//rcp:ingredient"
                            group-by="@name">
           <use name="{current-grouping-key()}"
                count="{count(current-group())}"/>
        </xsl:for-each-group>
      </uses>
   </xsl:template>
</xsl:stylesheet>
```

Output

```
<uses>
  <use name="beef cube steak" count="1"/>
  <use name="onion, sliced into thin rings" count="1"/>
  <use name="green bell pepper, sliced in rings" count="1"/>
  <use name="Italian seasoned bread crumbs" count="1"/>
  <use name="grated Parmesan cheese" count="1"/>
  <use name="olive oil" count="2"/>
  <use name="spaghetti sauce" count="1"/>
  <use name="shredded mozzarella cheese" count="1"/>
  <use name="angel hair pasta" count="1"/>
  <use name="minced garlic" count="3"/>
  ...
</uses>
```

Funzioni

```
<xsl:function name="">
    <!- context --!>
</xsl:function>
```

- Facilitano il riuso e la modularizzazione
- Riceve i parametri attraverso la dichiarazione di elementi param
- Una volta definita la funzione può essere richiamata all'interno di funzioni XPath

Esempio

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:local="http://www.w3.org/2004/07/xquery-local-functions">
  <xsl:function name="local:fib">
    <xsl:param name="n"/>
    <xsl:value-of select="if ($n le 1)
                          then 1
                          else local:fib($n -1)+local:fib($n -2)"/>
  </xsl:function>

  <xsl:template match="/">
    <xsl:value-of select="local:fib(10)"/>
  </xsl:template>
</xsl:stylesheet>
```

Struttura di un documento XSLT

STRUTTURA DOCUMENTO XSLT

- Root:
  - <xsl:stylesheet
    xmlns:xsl=http://www.w3.org/1999/XSL/Transform
    version = *number*>
      <!-- Content: (xsl:import*, *top-level-elements*) -->
    </xsl:stylesheet>

  - <xsl:transform
      xmlns:xsl=http://www.w3.org/1999/XSL/Transform
    version = *number*>
      <!-- Content: (xsl:import*, *top-level-elements*) -->
    </xsl:transform>

Top Level Elements

- xsl:import
- xsl:include
- xsl:strip-space
- xsl:preserve-space
- xsl:output
- xsl:key
- xsl:decimal-format
- xsl:namespace-alias
- xsl:attribute-set
- xsl:variable
- xsl:param
- xsl:template

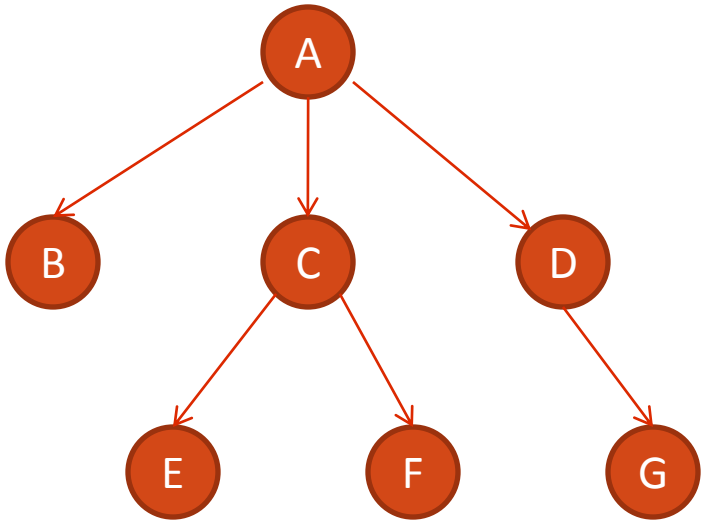
IMPORT E INCLUDE

- `<xsl:import href="uri-reference" />`
- `<xsl:include href="uri-reference" />`

- Servono per la modularizzazione dei file xsl
- Effetto: i file dichiarati vengono inseriti nel file di origine al posto del tag
- Include:
  - errore se ci sono riferimenti circolari o se si tenta di redifinire una regola.
- Import:
  - deve comparire prima di ogni altra regola
  - consente l'overriding delle regole importate

Import tree

- Gli elementi incontrati mentre si processa un'istruzione di import vanno a formare l'import tree.
- Ogni include viene risolto prima di costruire l'albero
- Ogni import incontrato viene definito come figlio di stylesheet
- Se i fogli di stile importati al loro interno hanno altri import, saranno figli di quel nodo
- Le priorità delle regole, dalla più bassa alla più alta, è data dalla visita post order dell'albero.



- Priorità (dalla più bassa alla più alta): B,E,F,C,G,D,A

Strip and preserve space

- `<xsl:strip-sapce elements="`*element list*`" />`
- `<xsl:preserve-sapce elements="`*element list*`" />`

- Gestione degli spazi
- Con strip vengono ignorati gli spazi finali, con preserve vengono conservati

Output

- Determina il formato di output del documento di destinazione
-
```
<xsl:output
  method = "xml" | "html" | "text" | qname-but-not-ncname
  version = nmtoken
  encoding = string
  omit-xml-declaration = "yes" | "no"
  standalone = "yes" | "no"
  doctype-public = string
  doctype-system = string
  cdata-section-elements = qnames
  indent = "yes" | "no"
  media-type = string />
```

KEY

- Dichiarazione di una chiave
- `<xsl:key`
  `name = qname`
  `match = pattern`
  `use = expression />`
- L'istruzione definisce una corrispondenza tra il nome di una chiave e il valore di un nodo
- `name:` nome della chiave
- `match:` nodi obiettivo
- `Use: valore da associare alla chiave,` viene valutato per ogni nodo che soddisfa il pattern
- `Funzione Key:` prende in input il nome e il valore di una chiave e ne restituisce gli elementi corrispondenti.

VARIABILI E PARAMETRI

```
<xsl:param
   name = qname
   select = expression>
   <!-- Content: template -->
</xsl:param>
<xsl:variable
    name = qname
    select = expression>
    <!-- Content: template -->
  </xsl:variable>
```

- Variable definisce una variabile di sola lettura il cui valore può essere assegnato con un'espressione Xpath
- Il campo d'azione della variabile è la parte restante del sequenza in cui è avvenuta la dichiarazione, se è un top element la variabile sarà globale
- Param viene utilizzato per definire parametri formali da utilizzare per esempio nelle istruzioni apply-template o call-template.

Esempio variabili e parametri

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template name="fib">
    <xsl:param name="n"/>
    <xsl:choose>
      <xsl:when test="$n le 1">
        <xsl:value-of select="1"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="f1">
          <xsl:call-template name="fib">
             <xsl:with-param name="n" select="$n -1"/>
          </xsl:call-template>
        </xsl:variable>
        <xsl:variable name="f2">
          <xsl:call-template name="fib">
             <xsl:with-param name="n" select="$n -2"/>
          </xsl:call-template>
        </xsl:variable>
        <xsl:value-of select="$f1+$f2"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

```
<xsl:template match="/">
    <xsl:call-template name="fib">
      <xsl:with-param name="n"
                      select="10"/>
    </xsl:call-template>
  </xsl:template>
</xsl:stylesheet>
```

NAMESPACE-ALIAS

```
<xsl:namespace-alias
    stylesheet-prefix = prefix
    result-prefix = prefix  />
```

- Crea un alias del namespace result prefix
- Utilizzato per esempio per creare uno stylesheet attraverso xsl

Esempio NameSpace-Alias: documento di origine

```
<elements>
  <block>p</block>
  <block>h1</block>
  <block>h2</block>
  <block>h3</block>
</elements>
```

Esempio Namespace alias (1/2)

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:axsl="file://namespace.alias">
<xsl:namespace-alias stylesheet-prefix="axsl"
result-prefix="xsl"/>
<xsl:template match="/">
 <axsl:stylesheet version="2.0">
   <xsl:apply-templates/>
 </axsl:stylesheet>
</xsl:template>
```

ESEMPIO NAMESPACE-ALIAS (2/2)

```
 <xsl:template match="elements">
   <axsl:template match="/">
    <axsl:comment
      select="system-property('xsl:version')"/>
    <axsl:apply-templates/>
   </axsl:template>
 </xsl:template>
 <xsl:template match="block">
   <axsl:template match="{.}">
    <fo:block>
      <axsl:apply-templates/>
    </fo:block>
   </axsl:template>
 </xsl:template>
</xsl:stylesheet>
```

Stylesheet Risultato (1/2)

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <xsl:template match="/">
   <xsl:comment select="system-
  property('xsl:version')"/>
     <xsl:apply-templates/>
  </xsl:template>
<xsl:template match="p">
  <fo:block>
   <xsl:apply-templates/>
</fo:block>
</xsl:template>
```

Stylesheet Risultato (2/2)

```
 <xsl:template match="h1">
  <fo:block>
  <xsl:apply-templates/>
  </fo:block>
 </xsl:template>
 <xsl:template match="h2">
  <fo:block>
       <xsl:apply-templates/>
  </fo:block>
 </xsl:template>
 <xsl:template match="h3">
  <fo:block>
       <xsl:apply-templates/>
 </fo:block>
 </xsl:template>
</xsl:stylesheet>
```

Più file di Input

```
<xsl:stylesheet version="2.0"
     xmlns:rcp="http://www.brics.dk/ixwt/recipes"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="rcp:collection">
     <rcp:collection>
        <rcp:title>Selected Recipes</rcp:title>
        <xsl:apply-templates select="rcp:recipe"/>
     </rcp:collection>
  </xsl:template>

  <xsl:template match="rcp:recipe">
    <xsl:variable name="t" select="rcp:title/text()"/>
    <xsl:if test="not(doc('dislikes.xml')//
                      rcp:recipe[rcp:title eq $t])">
      <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Generare più file di Output (1/2)

```
<xsl:stylesheet version="2.0"
     xmlns="http://www.w3.org/1999/xhtml"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="students">
     <xsl:result-document href="names.html">
        <html>
          <head><title>Students</title></head>
          <body>
             <xsl:apply-templates select="student" mode="name"/>
          </body>
        </html>
     </xsl:result-document>
     <xsl:result-document href="grades.html">
        <html>
          <head><title>Grades</title></head>
          <body>
             <xsl:apply-templates select="student" mode="grade"/>
          </body>
        </html>
     </xsl:result-document>
  </xsl:template>
```

Generare più file di Output (2/2)

```
  <xsl:template match="student" mode="name">
    <a href="grades.html#{@id}"><xsl:value-of select="name"/></a>
    <br/>
  </xsl:template>

  <xsl:template match="student" mode="grade">
    <a name="{@id}"/>
    <xsl:value-of select="name"/>
    <ul>
      <xsl:apply-templates select="results/result"/>
    </ul>
  </xsl:template>

   <xsl:template match="result">
      <li>
        <xsl:value-of select="@course"/>:
        <xsl:text> </xsl:text>
        <xsl:value-of select="@grade"/>
      </li>
   </xsl:template>
</xsl:stylesheet>
```

Trasformare i Documenti xml tramite xslt

Trasformazione tramite browser

- Le trasformazioni XSL sono supportate da tutti i browser moderni.
- Un documento xml può contenere una process-instruction che punta a un documento XSLT
- Il browser carica il file ed esegue la trasformazione.

```
<?xml-stylesheet
    type="text/xsl"
    href="path_foglio_xsl"?>
```

Trasformazione tramite Java

- La JDK 1.4 contiene tutti i package necessari alla trasformazione:
  - `javax.xml.transform` package.
- Le versioni precedenti della JDK richiedevano il dowload di un processore XSLT e di un parser SAX

XSLTMANAGER.JAVA

```
public static void main(String[] args) {
  try {
    TransformerFactory tFactory =
TransformerFactory.newInstance();

    Transformer transformer = tFactory.newTransformer
    (new
javax.xml.transform.stream.StreamSource(xslFileName));

    transformer.transform
      (new javax.xml.transform.stream.StreamSource
            (xmlFileName),
       new javax.xml.transform.stream.StreamResult
            ( new FileOutputStream(outputFileName)));
    }
  catch (Exception e) {
    e.printStackTrace( );
    }
}
```

ESERCIZIO

XML

```
<catalog>
 <cd>
   <title>Empire
     Burlesque</title>
   <artist>Bob
     Dylan</artist>
   <country>USA</country>
   <company>Columbia</compan
     y>
   <price>10.90</price>
   <year>1985</year>
   </cd>
   <cd>….</cd>
</catalog>
```

HTML

```
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th><th>Artist</th>
</tr>
<tr>
<td>Empire
   Burlesque</td><td>Bob
   Dylan</td>
</tr>
<tr>….</tr>
</body>
</html>
```

SOLUZIONE (1/2)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Tra
  nsform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
```

Soluzione (2/2)

```
<xsl:for-each select="catalog/cd">
  <tr>
   <td><xsl:value-of select="title"/></td>
   <td><xsl:value-of select="artist"/></td>
   </tr>
</xsl:for-each>
 </table>
 </body>
 </html>
 </xsl:template>
</xsl:stylesheet>
```