Un programma di computer fa quello che gli dici,
non quello che vuoi.

Legge di Greer
(Leggi di Murphy applicate all'informatica)

Tecniche Multimediali

Corso di Laurea in «Informatica» - aa 2010-2011

Prof. Giorgio Poletti – giorgio.poletti@unife.it

Rappresentazione dei dati

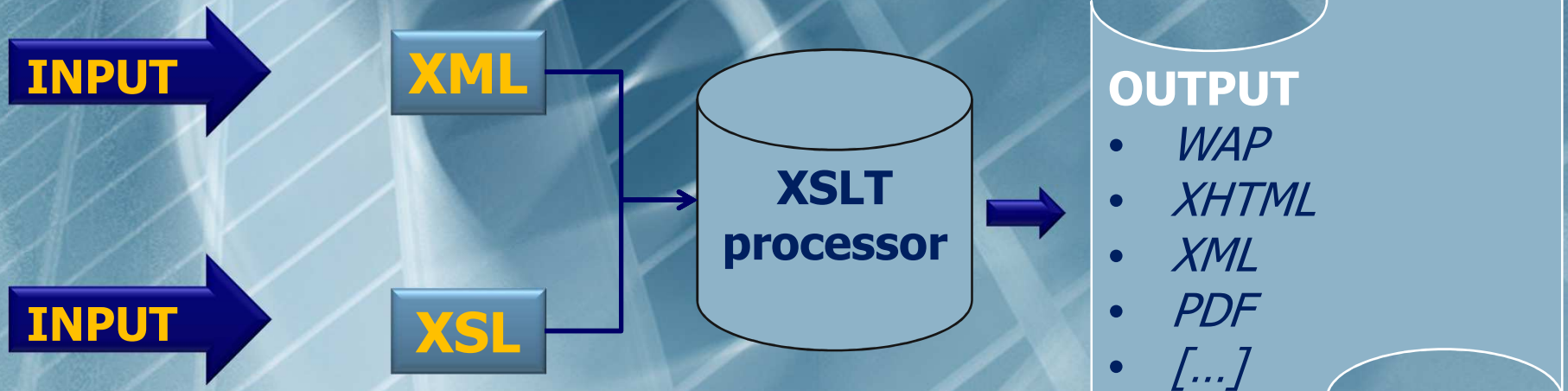
- CSS (*Cascading Style Sheets*)
  - Linguaggio **descrittivo**
    - assegna a ogni nodo del file XML un **formato**
    - non fa modifiche di struttura
- XSL (*eXtensible Stylesheet Language*)
  - assegna a ogni nodo del file XML una **rappresentazione**
  - permette la riorganizzazione della struttura

XSL – *Linguaggio di Stylesheet*	
XML → XML (XSLT)	VOCABOLARIO (XSL-FO o XSL) *(semantica di trasformazione)*
XSLT (recommendation 1999)	XSL-FO (Formattin Object)(recommendation 2001)

XSL e XSLT

- XSLT (*XSL Transformation*)
  - Estensione del concetto di foglio di stile
  - Manipolazione della struttura del documento
  - Ha le caratteristiche di un linguaggio «*dichiarativo*»
  - Si basa anche su **XPath**
- XSL permette di trasformare
  - XML in un documento XHTML
  - Rappresentare i documenti (attraverso FO) in formati di visualizzazione (senza presupporre linguaggi marcatori)

XSL e XSLT



1. *Modifica dinamica dei dati*
2. *Device Indipindent*
3. *Esportare dati in multi-formato*

XSL – Struttura generale

- XSLT è un documento XML

```
<?xml version= "1.0" encoding="ISO-8854-1"?>
```

- XSLT ha *stylesheet* come nodo radice

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
---------------------------------
</stylesheet>
```

XSL – Template

- XSLT è composto da *template di costruzione*
  - soubroutine di trasformazione
  - individua pattern (schema)
  - usa XPath per filtrare i nodi da elaborare (*pattern matching*)
  - per ogni *radice stylesheet* deve esserci un ***template match="/«***
  - Contiene regole di rappresentazione e gestione della struttura

---

SINTASSI

```
<xsl:template
        name="nome_template"
        match="percoso "
        priority="priorità"
        mode="modo" >
    <!-- azioni e comandi-->
</xsl:template>
```

XSL – Template

SINTASSI

```
<xsl:template
        name="nome_template"
        match="percoso "
        priority="priorità"
        mode="modo" >
    <!-- azioni e comandi-->
</xsl:template>
```

Attributo	Significato
name	nome del template; se l'attributo **name** è presente può, ma non necessariamente deve, esserci l'attributo **match**
match	***pattern*** (schema) che identifica uno o più nodi di origine a cui applicare le regole, l'attributo **match** è obbligatorio a meno che non ci sia l'attributo **name**
priority	priorità del modello; tutte le regole del modello con una priorità più bassa rispetto alle regole di modello non vengono tenute in considerazione
mode	valore del **modo**, consente di elaborare più volte un elemento, producendo ogni volta un risultato diverso; mode non deve esserci se non c'è un attributo **match;** se un elemento ***<xsl:apply-templates>*** ha un attributo mode, può essere applicato solo alle regole di modello da elementi **<xsl:template>** che hanno un attributo **mode** con lo stesso valore; un elemento <xsl:apply-templates> che non ha un attributo mode, può essere applicato solo alle regole di modello da elementi <xsl:template> che non hanno un attributo mode.

XSL – Template

SINTASSI

```
<xsl:template
        name="nome_template"
        match="percoso "
        priority="priorità"
        mode="modo" >
    <!-- azioni e comandi-->
</xsl:template>
```

Elemento	Valori
percorso	• **item** *seleziona tutti gli elementi denominati item*; • * seleziona qualsiasi elemento • **parent/child** seleziona tutti gli elementi denominati *child* il cui padre si chiama *parent* • **/** seleziona la radice dell'albero • **text()** seleziona nodi di tipo testo • **@att** seleziona l'attributo di nome *att* • **item[@name=``foo'']** seleziona gli elementi denominati *item* che hanno un attributo chiamato *name*, il cui valore è *foo* • **@*** seleziona qualsiasi attributo

XSL – Template

ESEMPIO 1 (template)

Cap. 1 <titolo>La storia</titolo> ***(elemento di un documento)***

```
<xsl:template  match= "titolo">

    <html:b>
        <xsl:apply-templates/>
     </html:b>

</xsl:template>
```

Il **template** fa match con **titolo** e scrive un elemento **b** (HTML) e inserisce tutti i nodi figli nella lista dei nodi correnti; XPATH è relativo (child::titolo), restituisce un nodeSet.

XSL – Template

ESEMPIO 2 (apply-template)

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

Viene applicato il **template** che fa match con il nodeSet selezionati da **nodeA** e con un attributo **mode** istanziato (con un valore) **modeA**.

Modalità utile per

- *gestire documenti a struttura complessa (il parser applica il template più opportuno in funzione delle regole espresse)*
- *inserire qualunque nodo, in qualunque ordine e in qualunque molteplicità durante il processing (***non ci sono problemi di stack***)*
- *elaborare, in fase di processing, in modo ricorsivo i figli dopo la sospensione del template principale*

XSL – Template

ESEMPIO 3 (apply-template)

```
<xsl:templates match="Paragrafo"/>

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

<xsl:apply-templates/>
```

Trasforma il nodo Paragrafo nel tag **<P>** di **HTML**

XSL – Template

ESEMPIO 4 (creare elementi)

```
<xsl:templates match="Paragrafo"/>

  <P>Testo del Paragrafo</P>

<xsl:apply-templates/>
```

Scrive direttamente un elemento nel file risultante

XSL – Template

ESEMPIO 5 (creare elementi)

INPUT

```
<testo tipo="nota" contenuto= "testo nota " />
```

```
<xsl:templates match="testo"/>
   <xsl:element name= "{@tipo}">
    <xsl:value-of select="@conbtenuto"/>
   </xsl:element>
<xsl:apply-templates/>
```

OUTPUT

```
<testo nota="testo nota " />
```

Trasforma il valore di un attributo nel documento di partenza nel nome di un nodo; utile in elementi complessi o calcolati

XSL – Template

ESEMPIO 6 (creare attributi)

INPUT

```
<testo notaN="1" contenuto= "testo nota " />
```

```
<xsl:templates match="testo"/>
   <A><xsl:attribute name= "HREF">
    <xsl:value-of select="@notaN"/>.txt
   </xsl:attribute>
    <xsl:value-of select="@contenuto"/>
<xsl:apply-templates/>
```

OUTPUT

```
<A HREF="1.txt">testo nota</A>
```

Specifica di attributi, utilizzando come il nome dell'attributo come espressione

XSL – Template

ESEMPIO 6 (leggere i valori) - <xsl:value-of select="nodo" />

INPUT

```
<testo notaN="1" contenuto= "testo nota " />
```

```
<xsl:templates match="testo"/>
   <P> <xsl:value-of select="@notaN"/>
          <xsl:text> </xsl:text>
          <xsl:value-of select="@contenuto"/>
   </P>
<xsl:apply-templates/>
```

OUTPUT

```
<P> 1 testo nota</P>
```

L'attributo **select** è obbligatorio, **xsl:text** inserisce il testo nel documento

XSL – Template

ESEMPIO 7 (leggere i valori) – uso delle parentesi grafe {}

INPUT

```
<file nome= "filename" tipo= "pdf" />
```

```
<xsl:templates match="linkPDF"/>
   <A><xsl:attribute name= "HREF">
      <xsl:value-of select="{@nome}.{@tipo}"/>.txt
        </xsl:attribute>
    <xsl:value-of select="@testo"/>
<xsl:apply-templates/>
```

OUTPUT

```
<A HREF="filename.pdf">testo</A>
```

Uso frequente per la conversione di valori in **tag** per il **markUP**

XSL – Template

Strutturare l'albero del documento finale– Alcuni comandi

<xsl:comment> </xsl:comment>

Inserimento di un commento nel documento finale

<xsl:number> </xsl:number>

Inserimento di numeri formattati nel documento finale.

<xsl:text> </xsl:text>

Inserimento esplicito del testo contenuto dentro al documento. Rispetto all'inserimento diretto di testo:

- rispetta gli spazi (blank)
- con l'attributo *disable-output-escaping="yes"* rispetta i caratteri speciali ("&" e "<")

<xsl:processing-instruction> </xsl:processing-instruction**>**

Permette di inserire le processing instruction nel foglio di stile che altrimenti vengono ignorate;

Esempio

INPUT

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

OUTPUT

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

XSL – Template

ESEMPIO 8 (leggere i valori) – l'iterazione <xsl:for-each>

Uso del template

```
<xsl:template match= "BODY">
   <xsl:apply-templates select="H1"/>
</xsl:template>

<xsl:template match="H1">
   <P><xsl:value-of select="."/></P>
</xsl:template>
```

Uso dell'iterazione

```
<xsl:template match="BODY">
   <xsl:for-each select="H1">
      <P><xsl:value-of select=".”/></P>
   </xsl:for-each>
</xsl:template>
```

XSL – Template

ESEMPIO 9 (istruzioni condizionate - <xsl:if>)

<xsl:if> *esegue azioni in funzione della verità di un XPath test*

Colorare di giallo (in RGB: FFFF00) lo sfondo di una riga ogni due di una tabella HTML

```
<xsl:template match="riga">
  <tr>
    <xsl:if test="position() mod 2 = 0">
       <xsl:attribute name="bgcolor">
         #FFFF00
       </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </tr>
</xsl:template>
```

XSL – Template

ESEMPIO 9 (istruzioni condizionate - <xsl:choose>)

<xsl:choose> *esegue azioni in funzione di diversi test di verità di un XPath test*

Colorare di giallo, blu e rosso (in RGB: FFFF00, 0000FF, FF0000) alternativamente le righe di una tabella HTML

```
<xsl:template match="riga">
 <tr>
  <xsl:choose>
   <xsl:when test="position() mod 3 = 0">
    <xsl:attribute name="bgcolor">#FFFF00</xsl:attribute>
   </xsl:when>
   <xsl:when test="position() mod 3 = 1">
    <xsl:attribute name="bgcolor">#0000FF</xsl:attribute>
   </xsl:when>
  <xsl:otherwise>
    <xsl:attribute name="bgcolor">red</xsl:attribute>
   </xsl:otherwise>
  </xsl:choose>
  <xsl:apply-templates/>
 </tr>
</xsl:template>
```

XSL – Template

ESEMPIO 9 (ordinamento - <xsl:sort>)

<xsl:sort> *ordina i contenuti dei nodi della lista corrente, è figlio solamente di <xsl:for-each> o <xsl:apply-templates>; si possonno iterare e annidare gli elementi <xsl:sort>*

```
<xsl:template match="elenco">
 <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>
```

XSL – Template

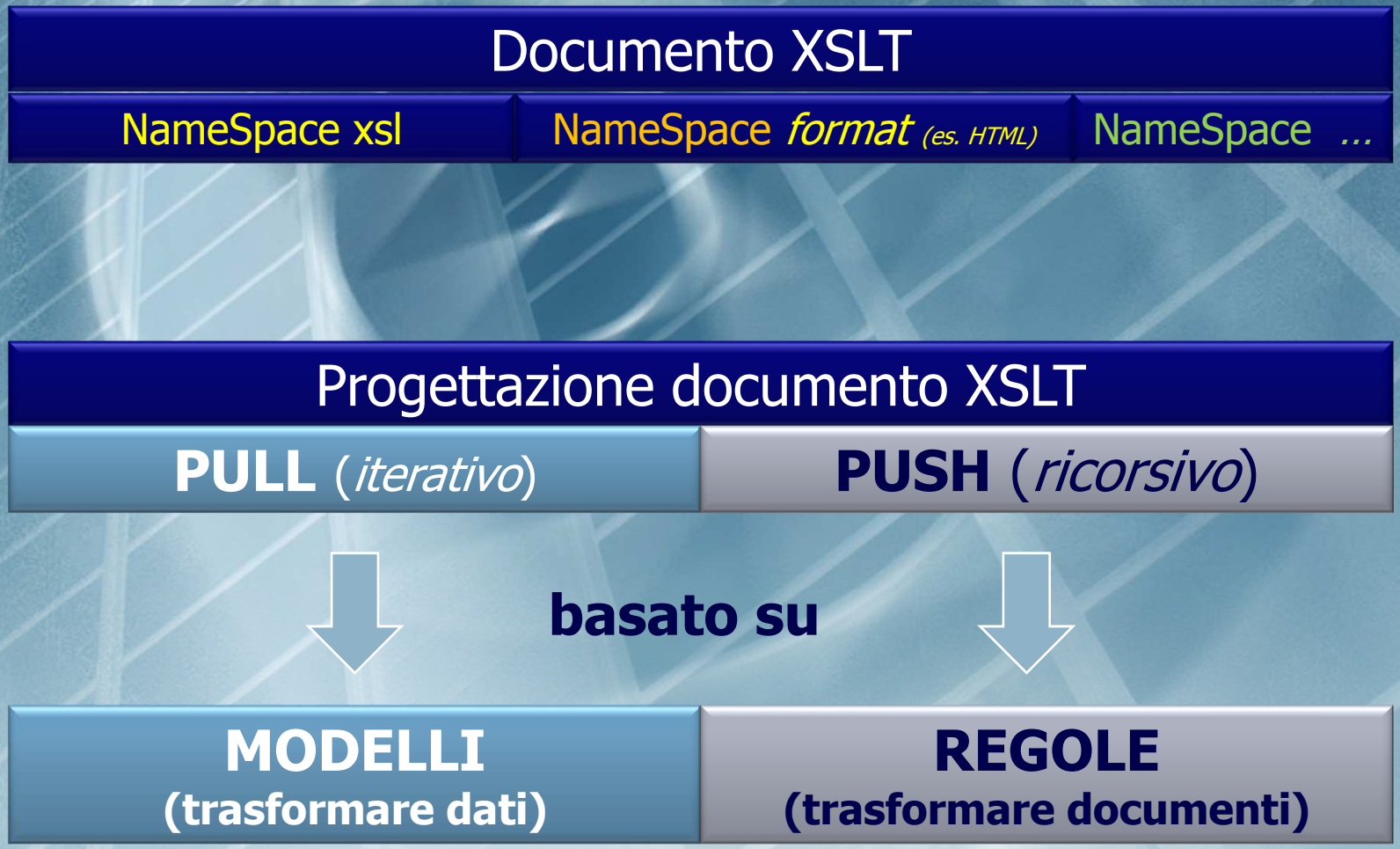
<xsl:sort> - sintassi e attributi

```
<xsl:sort select="expression"
lang="language-code"
data-type="text|number|qname"
order="ascending|descending"
case-order="upper-first|lower-first"/>
```

Attributi	Valori	Descrizione
select	Espressione XPath	**Opzionale**, *specifica il nodo i cui valori devono essere utilizzati per l'ordinamento*
lang	Codice di lingua	**Opzionale**, *specifica quale lingua è utilizzata per l'ordinamento*
data-type	***text|number|qname***	**Opzionale**, *specifica che tipo di dati sono utilizzati per l'ordinamento; di default è* ***text***
order	***ascending|descending***	**Opzionale**, *specifica di che tipo è l'ordinamento; di default è* ***ascending***
case-order	***upper-first|lower-first***	**Opzionale**, *specifica se devono precedere le lettere minuscole o maiuscole; il default dipende dalla lingua*

XSL – Template

Modalità di progettazione



XSL – Template

Modalità PULL (iterativo)

MODELLI
(trasformare dati)

Per documenti a struttura data-base, ripetitivi

XML

```
<magazzino>
  <stock anno="1999">
    <art>bullone</art>
    <cod>10005</cod>
    <prezzo>12,5</prezzo>
  </stock>
 <stock anno="1999">
    <art>vite</art>
    <cod>10012</cod>
    <prezzo>2,5</prezzo>
  </stock>
 <stock anno="2009">
    <art>dado</art>
    <cod>15005</cod>
    <prezzo>16,5</prezzo>
  </stock>
</magazzino>
```

XSL

```
<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=«magazzino/stock">
    <TR>
      <TD><xsl:value-of select="art"/></TD>
      <TD><xsl:value-of select="cod"/></TD>
      <TD><xsl:value-of select="prezzo"/></TD>
    </TR>
   </xsl:for-each>
  </TABLE>
 </BODY>
</HTML>
```

XSL – Template

Modalità PUSH (ricorsivo)

PUSH (*ricorsivo*)

Per documenti a struttura notevolmente diversificata

```
<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>
```

XML

```
<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">
    <I><xsl:apply-templates /></I>
  </xsl:template>
</xsl:stylesheet>
```

XSL

XSL – Template

Le variabili

*Le **variabili** XSL sono **costanti** derivanti dalla valutazione di una espressione; usata nel sottoalbero di definizione e identificata da {} e $*

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

XSL – Template

Altri comandi e potenzialità - MERGING

<xsl:import>, sintassi <xsl:import href="*uri-reference*" />
href è **obblicatorio** e identifica XSLT da importare (modifica la priorità)

ESEMPIO

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

<xsl:import href="article.xsl"/>
<xsl:import href="bigfont.xsl"/>

<xsl:attribute-set name="note-style">
  <xsl:attribute name="font-style">italic</xsl:attribute>
</xsl:attribute-set>
</xsl:stylesheet>
```

XSL – Template

Altri comandi e potenzialità - MERGING

<xsl:include>, sintassi <xsl:include href="*uri-reference*" />
href è **obblicatorio** e identifica XSLT da importare (mantiene la priorità)

ESEMPIO

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" omit-xml-declaration="yes"/>
<xsl:template match="/">
  <xsl:for-each select="COLLECTION/BOOK">
    <xsl:apply-templates select="TITLE"/>
    <BR/> </xsl:for-each>
</xsl:template>
  <xsl:include href="xslincludefile.xsl" />
</xsl:stylesheet>
```

XSL – Template

Altri comandi e potenzialità - OUTPUT

<xsl:output>, definisce le modalità di output

SINTASSI

<**xsl:output**
method="xml|html|text|name" version="string" encoding="string"
omit-xml-declaration="yes|no" standalone="yes|no" doctype-public="string"
doctype-system="string" cdata-section-elements="namelist" indent="yes|no"
media-type="string"**/**>

Attributi	Valori	Descrizione
method	**xml | html | text | name**	**Opzionale**, *specifica il formato di output; di default è **xml***
version	string	**Opzionale**, *specifica la versione del formato di output (solo con html e xml)*
encoding	*string*	**Opzionale**, *specifica il set di caratteri del formato di output*
omit-xml-declaration	**yes | no**	**Opzionale**, *specifica se yes che in output non devono visualizzarsi le PI <?xml … ?>*
standalone	**yes | no**	**Opzionale**, *specifica che in output ci sarà una dichiarazione standalone, di default è "no"*
doctype-public	string	**Opzionale**, *definisce PUBLIC il documento di output*
doctype-system	string	**Opzionale**, *definisce SYSTEM il documento di output*
cdata-section-elements	namelist	**Opzionale**, un elenco di elementi testo separati da spazi deve essere scritto come sezione CDATA
indent	**yes | no**	**Opzionale**, *"yes" specifica che in output il testo sarà indentato come indicato dalla gerarchia*
media-type	string	**Opzionale**, *specifica il tipo MIME (Multipurpose Internet Mail Extensions) di output; di default è **text/xml***