

Apache FOP

Un esempio pratico

Introduzione

FOP è una libreria Java creata dalla Apache Foundation, che permette di generare diverse tipologie di documenti (PDF, POSTSCRIPT etc...). La libreria si basa sulla tecnologia XSLT, ed in particolare sul linguaggio XSL-FO. L'utilizzo di FOP quindi, richiede la conoscenza seppur parziale di diversi linguaggi. Oltre a Java, bisogna almeno conoscere i rudimenti di XML, XSL, XSL-FO e XPATH.

Facciamo un esempio: se volessimo generare un file PDF con FOP, dovremmo creare del codice Java, che caricato un file (o comunque una fonte qualsiasi) XML, lo trasforma tramite un file (o fonte qualsiasi) XSL, in file PDF. Ovviamente il file XML deve contenere i dati da visualizzare nel documento PDF, mentre il file XSL deve contenere le regole per la formattazione del documento stesso. Il file XSL, deve essere scritto utilizzando anche il linguaggio XSL-FO (Formatting Objects) che serve proprio per formattare tipologie di file come PDF. Ovviamente l'XSL, richiede in qualche modo anche l'utilizzo del linguaggio XPATH, per la ricerca dei nodi XML.

Il flusso di lavoro da seguire per l'utilizzo di FOP è quindi il seguente:

- 1) Progettare un file XML capace di ospitare tutti i dati da visualizzare nel PDF
- 2) Creare del codice Java che formatta dinamicamente il file XML
- 3) Creare un file XSL che sfruttando XSL-FO (e XPATH) riesce a formattare il file PDF visualizzando tutti i dati presenti nel file XML
- 4) Creare il codice Java che sfruttando le classi messe a disposizione da FOP, presi in input il file XML e il file XSL genera il file PDF

Esempio

Supponiamo di voler creare dinamicamente un semplice curriculum come quello mostrato in figura 1.

Dott. Mario Rossi – Java Developer

Curriculum Vitae
Mario Rossi

Informazioni Personali Nome: Mario Rossi
Nato a: Verona, 30/01/1978
Cellulare: 3300000000
Telefono: 0123456789
Fax: 0987654321
E-Mail: mario@sitodimariorossi.com
Internet: <http://www.sitodimariorossi.com>

Istruzione e Formazione Titolo di Studio: 2001, Università di Pisa - Laurea in Matematica
Specializzazione: 2002, Master in Tecnologie Web presso il Politecnico di Milano
Certificazione Professionale: 2004, Java SCJP 1.4

Esperienze Professionali QWERTY Electronics SRL: 2003, Programmatore Java
ABCDE Solutions SRL: 2003 - 2004, Programmatore Java e Web Master
Pinco Pallino Software SPA: 2004 a oggi, Analista programmatore con ottima conoscenza Java-J2EE, XML, SQL. Buona conoscenza Javascript e PHP

Internet: <http://www.sitodimariorossi.com>; E-mail: mario@sitodimariorossi.com

Figura 1: un esempio di curriculum

Come è possibile osservare dalla Figura 1, il template del curriculum può essere suddiviso in diverse aree. La Figura 2 prova a schematizzare il layout del curriculum individuando delle aree ben precise.

<hr/> <hr/>		1
Dott. Mario Rossi – Java Developer		
Curriculum Vitae Mario Rossi		3
4	Informazioni Personali Nome: Mario Rossi Nato a: Verona, 30/01/1978 Cellulare: 3300000000 Telefono: 0123456789 Fax: 0987654321 E-Mail: mario@sitodimariorossi.com Internet: http://www.sitodimariorossi.com	
5	Istruzione e Formazione Titolo di Studio: 2001, Università di Pisa - Laurea in Matematica Specializzazione: 2002, Master in Tecnologie Web presso il Politecnico di Milano Certificazione Professionale: 2004, Java SCJP 1.4	
6	Esperienze Professionali QWERTY Electronics SRL: 2003, Programmatore Java ABCDE Solutions SRL: 2003 - 2004, Programmatore Java e Web Master Pinco Pallino Software SPA: 2004 a oggi, Analista programmatore con ottima conoscenza Java-J2EE, XML, SQL. Buona conoscenza Javascript e PHP	
7		
<hr/> <hr/>		2
Internet: http://www.sitodimariorossi.com ; E-mail: mario@sitodimariorossi.com		

Figura 2: aree del curriculum

Dove:

- 1) L'intestazione della pagina: viene utilizzata per includere il nome e la professione, ed è presente su ogni pagina del curriculum.
- 2) Piè di pagina: viene utilizzata per includere altre informazioni personali come il numero di telefono e l'indirizzo e-mail, ed è presente su ogni pagina del curriculum.
- 3) Titolo: dove appare il la stringa fissa "Curriculum Vitae" e sotto il nome del candidato.
- 4) Informazioni Personali: sezione dedicata alle informazioni personali. Le informazioni personali immesse possono variare da candidato a candidato in numero oltre che in contenuti. Per esempio un candidato potrebbe voler esplicitare il suo sito personale mentre un altro potrebbe voler segnalare le sue cinque e-mail.
- 5) Istruzione e Formazione: sezione dedicata alle informazioni sull'istruzione e la formazione del candidato. Di solito oltre al titolo di studio il candidato può esplicitare anche le certificazioni professionali acquisite ed eventuali corsi di formazione seguiti.

- 6) Esperienze lavorative: sezione dedicata alle esperienze lavorative. Ovviamente questa sezione potrebbe essere particolarmente differente in lunghezza (oltre che ovviamente in contenuti) tra candidati differenti.
- 7) Ulteriori Informazioni: sezione dedicata all'inserimento di qualsiasi altra informazione il candidato ritiene di esplicitare nel suo curriculum.

Per semplificare l'apprendimento al lettore cercheremo di costruire il nostro curriculum gradualmente sezione dopo sezione.

Essendo solo un esempio non creeremo il codice Java che crea il file XML dinamicamente (punto 2), ma semplicemente definiremo un file XML statico.

Preparazione

Prima di iniziare, listiamo tutti gli strumenti di cui abbiamo bisogno. Per prima cosa abbiamo bisogno di un Java Development Kit ed anche se non strettamente necessario di un editor per poter scrivere il nostro codice Java. Nel nostro caso abbiamo utilizzato un JDK versione 1.5.0_04, e come editor EJE versione 2.5 (scaricabile gratuitamente da <http://www.claudiodesio.com/eje.htm>). Abbiamo bisogno ovviamente di FOP. È possibile scaricare FOP versione 0.20.5 da questo indirizzo: <http://xmlgraphics.apache.org/fop/>. In realtà a noi occorrono solo le librerie di FOP, e non tutto lo strumento per sviluppare. Quindi una volta scaricato FOP, fate in modo che il vostro editor punti con la variabile classpath al file "fop.jar" che si trova nella cartella "build" di FOP, o altrimenti copiate tale file nella cartella "JDK/jre/lib/ext".

È probabile che vi sia utile anche un editor per XML. Un'ottima scelta, ma a pagamento, è XMLSpy (su <http://www.altova.com> è possibile scaricare una demo).

Infine ovviamente occorre che sulla vostra macchina sia installato un PDF Reader come Adobe Acrobat Reader (scaricabile gratuitamente da <http://www.adobe.com>).

Applicazione Java

Ma iniziamo dall'applicazione Java che sfruttando le librerie messe a disposizione da FOP, esegue la trasformazione XSLT che permette di formattare il nostro curriculum in formato PDF. A tale scopo, come già asserito, essa ha bisogno di utilizzare una fonte da cui prelevare i dati, formattata in formato XML, e una fonte XSL che definisce il layout del curriculum, e che utilizza il linguaggio XSL-FO per creare un file PDF. Nel nostro caso, entrambe queste fonti saranno semplici file, ma con poche modifiche sarebbe possibile per esempio creare il file XML dinamicamente.

Consideriamo quindi la seguente classe:

```
1. import java.io.*;
2. import org.apache.fop.apps.*;
3. import org.apache.fop.messaging.*;
4. import org.xml.sax.*;
5.
6. public class CurriculumCreator {
7.     private static final String XML_FILE = "C:/Curriculum.xml";
8.     private static final String XSL_FILE = "C:/Curriculum.xsl";
9.     private static final String PDF_FILE = "C:\\Curriculum.pdf";
10.
11.     public static void main(String []args) throws Exception {
12.         CurriculumCreator curriculumCreator = new CurriculumCreator();
13.         curriculumCreator.printCurriculum();
14.     }
15.
16.     public void printCurriculum() throws Exception {
17.         ByteArrayOutputStream out = new ByteArrayOutputStream();
18.         FileOutputStream pdfFOS = new FileOutputStream(PDF_FILE);
19.         FileInputStream xslFileIS = null;
20.         FileInputStream xmlFileIS = null;
21.         try {
22.             // Trasferimento del file XML in un array di byte
```

```

23.         File xmlFile = new File(XML_FILE);
24.         xmlFileIS = new FileInputStream(xmlFile);
25.         ByteArrayOutputStream xmlOutput = new ByteArrayOutputStream();
26.         byte xmlBuffer[] = new byte[(int) xmlFile.length()];
27.         for (int count = xmlFileIS.read(xmlBuffer); count > 0;
28.             count = xmlFileIS.read(xmlBuffer)) {
29.             xmlOutput.write(xmlBuffer, 0, count);
30.         }
31.         byte[] xml = xmlOutput.toByteArray();
32.         // Trasferimento del file XSL in un array di byte
33.         File xslFile = new File(XSL_FILE);
34.         xslFileIS = new FileInputStream(xslFile);
35.         ByteArrayOutputStream xslOutput = new ByteArrayOutputStream();
36.         byte xslBuffer[] = new byte[(int) xslFile.length()];
37.         for (int count = xslFileIS.read(xslBuffer); count > 0;
38.             count = xslFileIS.read(xslBuffer)) {
39.             xslOutput.write(xslBuffer, 0, count);
40.         }
41.         byte[] xsl = xslOutput.toByteArray();
42.         //Trasformazione XSLT
43.         TraxInputHandler input = new TraxInputHandler(
44.             new InputSource(new ByteArrayInputStream(xml)),
45.             new InputSource(new ByteArrayInputStream(xsl)));
46.         Driver driver = new Driver();
47.         driver.setRenderer(Driver.RENDER_PDF);
48.         driver.setOutputStream(out);
49.         input.run(driver);
50.         // per testare il pdf
51.         byte[] content = out.toByteArray();
52.         pdfFOS.write(content);
53.         pdfFOS.flush();
54.         Runtime runtime = Runtime.getRuntime();
55.         runtime.exec("explorer " + PDF_FILE);
56.     }
57.     catch(Exception exc) {
58.         exc.printStackTrace();
59.     }
60.     finally {
61.         pdfFOS.close();
62.         xslFileIS.close();
63.         xmlFileIS.close();
64.         out.close();
65.     }
66. }

```

In pratica si tratta di una semplice classe eseguibile, che dopo aver importato le necessarie librerie, ed aver dichiarato le tre costanti che rappresentano i file XML, XSL e PDF che saranno coinvolti, non fa altro che lanciare il metodo `printCurriculum()`, dal metodo `main()`. Tale metodo, trasferisce il contenuto dei file XML e XSL in due array di byte chiamati per l'appunto `xml` e `xsl`, per poi realizzare la trasformazione tra le righe 40 e 46. Le righe da 48 a 50 trasferiscono l'array di byte che contiene il risultato della trasformazione in un file PDF. Le righe 51 e 52 sono opzionali e servono solamente per aprire il file appena creato.

File XML

Ora consideriamo il seguente file XML che contiene tutti i dati che vogliamo visualizzare nel nostro curriculum:

```

<?xml version="1.0" encoding="utf-8"?>
<curriculum>
  <header>Dott. Mario Rossi - Java Developer</header>
  <name>Mario Rossi</name>
  <personalInformations>

```

```
<title>Personalisti</title>
<items>
  <name>
    <title>Nome</title>
    <value>Mario Rossi</value>
  </name>
  <bornIn>
    <title>Nato a</title>
    <value>Verona, 30/01/1978</value>
  </bornIn>
  <mobile>
    <title>Cellulare</title>
    <value>3300000000</value>
  </mobile>
  <phone>
    <title>Telefono</title>
    <value>0123456789</value>
  </phone>
  <fax>
    <title>Fax</title>
    <value>0987654321</value>
  </fax>
  <email>
    <title>E-Mail</title>
    <value>mario@sitodimariorossi.com</value>
  </email>
  <internet>
    <title>Internet</title>
    <value>http://www.sitodimariorossi.com</value>
  </internet>
</items>
</personalInformations>
<education>
  <title>
    Istruzione e Formazione
  </title>
  <items>
    <studies>
      <title>Titolo di Studio</title>
      <value>2001, Università di Pisa - Laurea in Matematica</value>
    </studies>
    <master>
      <title>Specializzazione</title>
      <value>2002, Master in Tecnologie Web presso il Politecnico di Milano</value>
    </master>
    <certification>
      <title>Certificazione Professionale</title>
      <value>2004, Java SCJP 1.4</value>
    </certification>
  </items>
</education>
<professionalExperience>
  <title>
    Esperienze Professionali
  </title>
  <items>
    <experience>
      <title>QWERTY Electronics SRL</title>
      <value>2003, Programmatore Java</value>
    </experience>
    <experience>
      <title>ABCDE Solutions SRL</title>
      <value>2003 - 2004, Programmatore Java e Web Master</value>
    </experience>
    <experience>
      <title>Pinco Pallino Software SPA</title>
      <value>2004 a oggi, Analista programmatore con ottima conoscenza Java-J2EE, XML,
      SQL. Buona conoscenza Javascript e PHP</value>
    </experience>
  </items>
</professionalExperience>
  Claudio De Sio Cesari --- http://www.claudiodesio.com --- claudio@claudiodesio.com
```

```
</experience>
</items>
</professionalExperience>
<other/>
<footer>Internet: http://www.sitodimariorossi.com; E-mail:mario@sitodimariorossi.com
</footer>
</curriculum>
```

La progettazione di un file XML di questo tipo, richiede tempo, ed è un passo fondamentale che non si può saltare. Inoltre bisogna tenere conto che è probabile che in un sistema reale non si utilizzino file XML statici come il precedente, bensì bisognerà crearli “al volo”, dinamicamente, utilizzando i dati presi da un eventuale database. Quindi non solo sarà difficile progettare tale struttura, ma spesso, è anche difficile implementarne il codice.

Nel nostro caso, abbiamo cercato di rendere le cose più semplici possibili. Il nostro file XML infatti, ha una struttura piuttosto omogenea. Infatti, il tag `curriculum` contiene come figli diretti, dei tag che rappresentano le sezioni principali della struttura del nostro curriculum, come `header`, `name`, `education`, `personalInformation`, `professionalExperience` e `other` (che però un tag vuoto). I campi che possono contenere diverse voci come per esempio le esperienze professionali, hanno uno schema fisso. Contengono un sotto-tag denominato `title`, che rappresenta il titolo della sezione del documento che viene visualizzato sulla sinistra della pagina. A seguire è sempre definito un tag `items`, contenente a sua volta una serie di sotto-tag che rappresentano le voci da visualizzare. Ognuno di questi tag, è a sua volta diviso in due sotto-tag: `title` e `value`. Il tag `title` deve contenere il titolo della voce mentre `value`, deve contenere il dettaglio della voce:

```
<professionalExperience>
  <title>
    Esperienze Professionali
  </title>
  <items>
    <experience>
      <title>QWERTY Electronics SRL</title>
      <value>2003, Programmatore Java</value>
    </experience>
    <experience>
      <title>ABCDE Solutions SRL</title>
      <value>2003 - 2004, Programmatore Java e Web Master</value>
    </experience>
    <experience>
      <title>Pinco Pallino Software SPA</title>
      <value>2004 a oggi, Analista programmatore con ottima conoscenza Java-J2EE, XML,
SQL. Buona conoscenza Javascript e PHP</value>
    </experience>
  </items>
</professionalExperience>
```

La struttura standard dei tag figli del tag principale `curriculum`, ci permetterà di scrivere del codice XSL più semplice.

File XSL

Non ci resta che creare il file XSL che deve formattare i dati del file XML. Come già asserito utilizzeremo un mix di linguaggi all'interno di esso. Tra le strutture definite dal linguaggio XSL come i template e i cicli `for-each`, si inseriranno le formattazioni definite dal linguaggio XSL-FO, come le `table`. Inoltre XPATH sarà utilizzato per recuperare i tag in maniera efficiente. Segue il listato del file XSL:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0">
<!--=====TEMPLATE PRINCIPALE=====-->
  <xsl:template match="*">
    <fo:root>
      <fo:layout-master-set>
        <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm"
margin-top="10mm" margin-bottom="10mm" margin-left="25mm" margin-right="25mm">
          <fo:region-before region-name="xsl-region-before" extent="4.8cm"/>
          <fo:region-body margin-top="3.2cm"/>
          <fo:region-after region-name="xsl-region-after" extent="1cm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-reference="page">
        <fo:static-content flow-name="xsl-region-before" display-align="auto">
          <xsl:call-template name="Header"/>
        </fo:static-content>
        <fo:static-content flow-name="xsl-region-after" display-align="auto">
          <xsl:call-template name="Footer"/>
        </fo:static-content>
        <fo:flow flow-name="xsl-region-body">
          <xsl:call-template name="Detail"/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
<!--=====TEMPLATE HEADER=====-->
<xsl:template name="Header">
  <fo:block font-size="14pt" text-align="center">
    <fo:table width="100%" font-size="11pt">
      <fo:table-column column-number="1"/>
      <fo:table-body>
        <fo:table-row>
          <fo:table-cell border-bottom-style="solid" border-top="1pt" border-bottom-
color="grey" border-top-color="grey" border-bottom="1pt" border-top-style="solid">
            <fo:block space-before="1mm">
              <xsl:value-of select="/curriculum/header"/>
            </fo:block>
          </fo:table-cell>
        </fo:table-row>
      </fo:table-body>
    </fo:table>
  </fo:block>
  <fo:block font-size="16pt" text-align="center" space-before="10mm">
    Vitae
  </fo:block>
  <fo:block font-size="16pt" text-align="center">
    <xsl:value-of select="/curriculum/name"/>
  </fo:block>
</xsl:template>
<!--=====TEMPLATE FOOTER=====-->
<xsl:template name="Footer">
  <fo:block text-align="center">
    <fo:table width="100%" font-size="12pt">
      <fo:table-column column-number="1"/>
      <fo:table-body>
        <fo:table-row>
          <fo:table-cell border-bottom-style="solid" border-top="1pt" border-bottom-
color="grey" border-top-color="grey" border-bottom="1pt" border-top-style="solid">
            <fo:block space-before="1mm">
              <xsl:value-of select="/curriculum/footer"/>
            </fo:block>
          </fo:table-cell>
        </fo:table-row>
      </fo:table-body>
    </fo:table>
  </fo:block>
</xsl:template>

```

```

<!--=====TEMPLATE DETAIL=====-->
<xsl:template name="Detail">
  <fo:block font-size="14pt" text-align="left">
    <fo:table width="100%" font-size="12pt">
      <fo:table-column column-number="1" column-width="60pt"/>
      <fo:table-column column-number="2"/>
      <fo:table-body>
        <xsl:call-template name="DetailRow">
          <xsl:with-param name="element"
select="/curriculum/personalInformations/items/" />
        </xsl:call-template>
        <xsl:call-template name="VoidRow" />
        <xsl:call-template name="DetailRow">
          <xsl:with-param name="element" select="/curriculum/education/items/" />
        </xsl:call-template>
        <xsl:call-template name="VoidRow" />
        <xsl:call-template name="DetailRow">
          <xsl:with-param name="element"
select="/curriculum/professionalExperience/items/" />
        </xsl:call-template>
        <xsl:call-template name="VoidRow" />
        <xsl:call-template name="DetailRow">
          <xsl:with-param name="element" select="/curriculum/other/items/" />
        </xsl:call-template>
      </fo:table-body>
    </fo:table>
  </fo:block>
</xsl:template>
<!--=====TEMPLATE DETAILROW=====-->
<xsl:template name="DetailRow">
  <xsl:param name="element" />
  <fo:table-row>
    <fo:table-cell column-number="1" font-size="13pt" font-style="italic">
      <fo:block font-weight="bold" space-before="5mm">
        <xsl:value-of select="$element/../../title" />
      </fo:block>
    </fo:table-cell>
    <fo:table-cell column-number="2">
      <xsl:for-each select="$element">
        <fo:block margin-left="10mm" space-before="5mm">
          <xsl:value-of select="title" />: <xsl:value-of select="value" />
        </fo:block>
      </xsl:for-each>
    </fo:table-cell>
  </fo:table-row>
</xsl:template>
<!--=====TEMPLATE VOIDROW=====-->
<xsl:template name="VoidRow">
  <fo:table-row>
    <fo:table-cell number-columns-spanned="2">
      <xsl:call-template name="VoidLine" />
    </fo:table-cell>
  </fo:table-row>
</xsl:template>
<!--=====TEMPLATE VOIDLINE=====-->
<xsl:template name="VoidLine">
  <fo:block white-space-collapse="false">
    <xsl:value-of select="$void" />
  </fo:block>
</xsl:template>
<!--
=====VARIABILI=====-->
<xsl:variable name="void">
  <xsl:text>&#x20;</xsl:text>
</xsl:variable>
</xsl:stylesheet>

```

Il file è comunque dichiarato come un normale file XSL, la differenza sta nel fatto che con il tag:

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

andiamo a dichiarare che utilizzeremo il namespace `xsl` per i tag XSL, e il namespace `fo` per i tag XSL-FO. E' facile notare come la struttura del file in questione sia quindi costituita dai classici template XSL, di cui solo il primo che incontriamo si basa sull'attributo `match`, ed in particolare tale attributo è valorizzato con il valore `"*"`. Quindi è questo il template principale che si applica ad ogni tag del file XML, si tratta quindi una sorta di metodo `main()` del file XSL. Tutti gli altri template del documento sono invece definiti con l'attributo `name`, e sono quindi invocati tramite i tag di tipo `call-template`. Per esempio con l'istruzione:

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

stiamo invocando il seguente template:

```
<xsl:template name="VoidRow">
  <fo:table-row>
    <fo:table-cell number-columns-spanned="2">
      <xsl:call-template name="VoidLine"/>
    </fo:table-cell>
  </fo:table-row>
</xsl:template>
```

Analizziamo quindi il primo template (quello principale da cui partono tutte le invocazioni agli altri template), che riportiamo di seguito nuovamente per rendere più agevole la consultazione:

```
<xsl:template match="*">
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm"
margin-top="10mm" margin-bottom="10mm" margin-left="25mm" margin-right="25mm">
        <fo:region-before region-name="xsl-region-before" extent="4.8cm"/>
        <fo:region-body margin-top="3.2cm"/>
        <fo:region-after region-name="xsl-region-after" extent="1cm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="page">
      <fo:static-content flow-name="xsl-region-before" display-align="auto">
        <xsl:call-template name="Header"/>
      </fo:static-content>
      <fo:static-content flow-name="xsl-region-after" display-align="auto">
        <xsl:call-template name="Footer"/>
      </fo:static-content>
      <fo:flow flow-name="xsl-region-body">
        <xsl:call-template name="Detail"/>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>
```

Si può notare come venga dichiarato il tag `root`, nel quale vengono dichiarati a loro volta i tag `layout-master-set` e `page-sequence`. Il tag `layout-master-set` permette di definire le misure della pagine del documento che sarà generato. Nel nostro caso, con il seguente codice:

```
<fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm" margin-
top="10mm" margin-bottom="10mm" margin-left="25mm" margin-right="25mm">
  <fo:region-before region-name="xsl-region-before" extent="3.2cm"/>
```

```
<fo:region-body margin-top="3.2cm" margin-bottom="2cm" />
<fo:region-after region-name="xsl-region-after" extent="1cm"/>
</fo:simple-page-master>
```

abbiamo stabilito che il nostro documento avrà pagine alte 297 millimetri e larghe 210 millimetri, ovvero il tipico formato A4. Inoltre abbiamo definito che i contenuti della pagina devono rispettare determinati margini dai bordi pagina. Nel nostro caso abbiamo stabilito che i margini verticali siano di 10 millimetri, mentre quelli orizzontali siano di 25 millimetri. Infine abbiamo definito tre aree `region-before` che conterrà l'intestazione della pagina, `region-body` che conterrà il dettaglio del curriculum e `region-after` che conterrà il piè di pagina. Notare che abbiamo definito anche le estensioni in centimetri tramite l'attributo `extent` sia per l'intestazione che per il piè di pagina. Per l'area destinata a contenere il dettaglio del curriculum, invece sono stati definiti i margini entro i quali deve essere definito. In particolare se il dettaglio è troppo grande per entrare all'interno dell'area ad esso destinato, allora verrà creata una nuova pagina del documento automaticamente, ovviamente contenente l'intestazione e il piè di pagina anch'essa. Quindi la gestione di come venga distribuito il contenuto della documento nelle varie pagine, è tutto incluso nel tag `layout-master-set`.

Il successivo tag :

```
<fo:page-sequence master-reference="page">
  <fo:static-content flow-name="xsl-region-before" display-align="auto">
    <xsl:call-template name="Header"/>
  </fo:static-content>
  <fo:static-content flow-name="xsl-region-after" display-align="auto">
    <xsl:call-template name="Footer"/>
  </fo:static-content>
  <fo:flow flow-name="xsl-region-body">
    <xsl:call-template name="Detail"/>
  </fo:flow>
</fo:page-sequence>
```

associa le aree definite nel tag precedente ai contenuti. È abbastanza semplice e intuitivo capire il significato del codice precedente. Vengono invocati i template `Header` e `Footer` come contenuti statici delle aree `region-before` e `region-after`. Viene invece definito come contenuto di tipo flusso (flow) il template `Detail`.

Possiamo quindi andare ora ad analizzare proprio questi template. Partiamo dal template `Header`:

```
<xsl:template name="Header">
  <fo:table width="100%" text-align="center" font-size="11pt">
    <fo:table-column column-number="1"/>
    <fo:table-body>
      <fo:table-row>
        <fo:table-cell border-bottom-style="solid" border-top="1pt" border-bottom-color="grey" border-top-color="grey" border-bottom="1pt" border-top-style="solid">
          <fo:block space-before="1mm">
            <xsl:value-of select="/curriculum/header"/>
          </fo:block>
        </fo:table-cell>
      </fo:table-row>
    </fo:table-body>
  </fo:table>
  <fo:block font-size="16pt" text-align="center" space-before="10mm">
    Curriculum Vitae
  </fo:block>
  <fo:block font-size="16pt" text-align="center">
    <xsl:value-of select="/curriculum/name"/>
  </fo:block>
</xsl:template>
```

In questo template viene dichiarata una tabella con allineamento centrato, larghezza massima e dimensione del font di 11 pixel.

N.B. : con il linguaggio XSL-FO è necessario utilizzare molto spesso le tabelle per definire le posizioni degli elementi contenuti.

Subito dopo la dichiarazione della tabella è necessario dichiarare da quante colonne essa è formata, ed assegnare ad ogni colonna un numero con il tag:

```
<fo:table-column column-number="1"/>
```

Subito viene dichiarato il body della tabella (`table-body`), un riga della tabella (`table-row`), e una cella (`table-cell`) della riga appena dichiarata. Tramite gli attributi del tag `table-cell`, abbiamo poi dichiarato che i bordi superiore e inferiore di tale cella debbano avere un spessore di 1 pixel di colore grigio e stile “`solid`”, Ogni elemento che deve diventare visibile sul documento deve essere contenuto in un tag di tipo `block`. Inoltre con l’attributo `space-before`, abbiamo garantito al contenuto del blocco una distanza dal bordo superiore di 1 millimetro. Poi con la classica istruzione XSL:

```
<xsl:value-of select="/curriculum/name"/>
```

Abbiamo prelevato il valore del tag `name` che si trova sotto il tag `curriculum` del file XML. Il valore dell’attributo `select` è un esempio di espressione XPATH. In realtà questa espressione, essendo banale, non rende l’idea della potenza di XPATH. Ma per ora è importante sottolineare che stiamo utilizzando anche questo terzo linguaggio.

È semplice comprendere che i seguenti due tag di tipo `block` vanno a stampare la scritta “Curriculum Vitae” centrata, con il nome del candidato subito dopo, prelevato dal file XML. In figura 3 possiamo notare il risultato della trasformazione XSL per quanto riguarda la parte dell’intestazione della pagina e del nome del candidato.

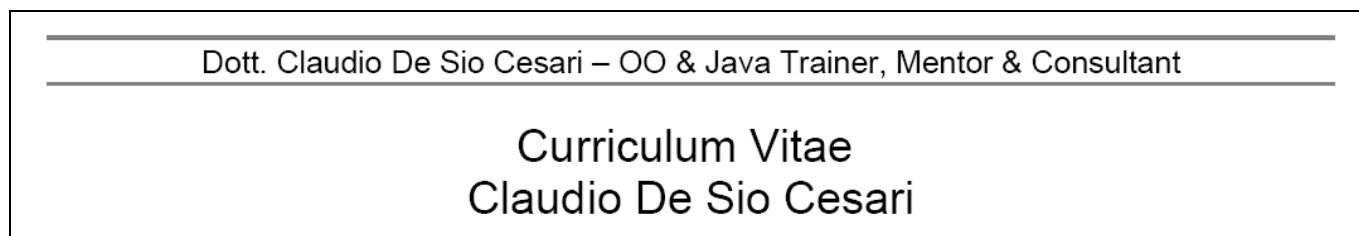


Figura 3: l’intestazione

Il lettore potrà facilmente analizzare il template `Footer` essendo esso molto simile al template `Header`.

Concentriamoci piuttosto sul template `Detail`. Questo template non fa altro che organizzare una tabella all’interno della quale vengono chiamati per ogni riga i template `VoidRow` e `DetailRow`, Il template `VoidRow` non fa altro che creare una riga vuota. Più interessante è il template `DetailRow`, che è praticamente il cuore del dettaglio del documento e che riportiamo di seguito:

```
<xsl:template name="DetailRow">
  <xsl:param name="element"/>
  <fo:table-row>
    <fo:table-cell column-number="1" font-size="13pt" font-style="italic">
      <fo:block font-weight="bold" space-before="5mm">
        <xsl:value-of select="$element/../../title"/>
      </fo:block>
    </fo:table-cell>
    <fo:table-cell column-number="2">
      <xsl:for-each select="$element">
```

```
<fo:block margin-left="10mm" space-before="5mm">
  <xsl:value-of select="title"/>: <xsl:value-of select="value"/>
</fo:block>
</xsl:for-each>
</fo:table-cell>
</fo:table-row>
</xsl:template>
```

Si tratta di un template che definisce un parametro che viene chiamato `element`. Questo assumerà il valore passatogli dal tag `with-param`. Per esempio la prima volta questo template viene chiamato nel seguente modo:

```
<xsl:call-template name="DetailRow">
  <xsl:with-param name="element" select="/curriculum/personalInformations/items/*"/>
</xsl:call-template>
```

In questo caso il parametro `element` avrà come valore

```
/curriculum/personalInformations/items/.*
```

Tale espressione per la sintassi XPATH, identifica tutti i sotto-tag diretti del tag `items` (che è sotto-tag di `personalInformation`, che è a sua volta sotto-tag di `curriculum`). Riguardando il file XML, in pratica stiamo individuando i tag `name`, `bornIn`, `mobile`, `phone`, `fax`, `email`, e `internet`. Tutti questi tag hanno una struttura identica, ovvero definiscono due sotto-tag fissi: `title` e `value`.

Il template `DetailRow` non fa altro che creare una riga con due celle. Nella prima cella, viene messo il titolo della sezione del curriculum, utilizzando la seguente espressione XPATH:

```
$element/../../title
```

Per la sintassi XPATH il valore di un parametro viene prelevato antepoendo il simbolo `$` al nome del parametro. Come per l'usuale modo di indicare i path delle directory sui vari sistemi DOS e UNIX, viene utilizzato il simbolo `..`, per risalire i tag del file XML. Quindi nel caso il parametro `element` valga

```
/curriculum/personalInformations/items/.*
```

Allora il valore finale dell'espressione è:

```
/curriculum/personalInformations/items/*/../../title
```

Ovvero

```
/curriculum/personalInformations/title
```

Quindi sarà stampato sul documento nella prima colonna con uno stile corsivo, e una dimensione di font di 13 pixel, la scritta "Informazioni Personali".

Per quanto riguarda la seconda cella, nel template `DetailRow` viene sfruttato un ciclo `for-each`. Infatti, avendo selezionato con l'espressione XPATH, una serie di tag che hanno la stessa struttura, possiamo ciclare su di essi ignorando i loro nomi, e andando a stampare il valore del tag `title`, seguito da quello del tag `value` nel seguente modo:

```
<xsl:for-each select="$element">
  <fo:block margin-left="10mm" space-before="5mm">
    <xsl:value-of select="title"/>: <xsl:value-of select="value"/>
  </fo:block>
```

```
</xsl:for-each>
```

Siccome il template si basa sul valore del parametro `element`, viene risfruttato per le altre sezioni del documento al variare del valore del parametro.

Il lettore può facilmente verificare il modo in cui vengono create tutte le altre sezioni del documento.