

# Appendice C

## Da XML a PDF

Progetto Atena

**Redatto da:**

Mariarosaria Padalino

---

# Sommario

<b>1</b>	<b>Perché Xml e XSLT?</b> .....	<b>3</b>
<b>2</b>	<b>Trasformazione da XML a PDF</b> .....	<b>4</b>
<b>3</b>	<b>Creazione dei file XSD</b> .....	<b>4</b>
<b>4</b>	<b>Creazione dei file XML</b> .....	<b>8</b>
<b>5</b>	<b>Creazione file XSLT</b> .....	<b>10</b>
<b>6</b>	<b>Trasformazione XSLT</b> .....	<b>11</b>

## 1 Perché Xml e XSLT?

Xml è un Metalinguaggio che permette di rappresentare informazioni in un formato testuale che può essere trasmesso con estrema velocità e tra piattaforme diverse.

Al contrario di altri strumenti, Xml presenta le informazioni in modo gerarchico utilizzando dei marcatori (tag) per identificarle, e fornisce delle regole per verificarne l'applicazione (validazione); inoltre è possibile trasformare un documento XML da un formato ad un altro senza troppo sforzo.

Xml separa completamente l'informazione da come sarà presentata, e risulta quindi un mezzo molto flessibile per la rappresentazione degli stessi, rendendo semplice l'aggiunta o la cancellazione di un campo senza troppi sforzi.

Quindi è conveniente utilizzare XML in caso di scambio di informazioni, di gestione dei contenuti separati dal loro layout e di rappresentazione di dati che provengono da un database.

L'eXtensible Stylesheet Language for Transformation (XSLT) è un linguaggio scritto con sintassi XML. Un documento XSLT è XML quindi sicuramente well-formed, e contiene al suo interno elementi e attributi che fanno riferimento alla grammatica XSLT.

Il risultato di una trasformazione XSLT che ricerca i contenuti in un documento XML con regole XPath, può essere un documento XSL-FO che, processato con un apposito parser, produrrà come risultato un documento PDF.

L'eXtensible Stylesheet Language Formatting Objects (XSL-FO) è una grammatica pensata per la formattazione di contenuti XML. Il documento XSL-FO ottenuto dalla trasformazione XSLT è ancora un documento XML well-formed che sarà elaborato da un apposito processore in grado di leggere FO produrre un determinato output, nel nostro caso un file PDF.

Utilizzando XML e XSLT si possono risparmiare tantissime righe di codice e, partendo dalla suddivisione dei dati e dalla formattazione grafica, tutti gli interventi successivi sono molto più immediati con un notevole risparmio di tempo.

Nel sistema Atena è richiesta la stampa di due file PDF, lo statino e il verbale in formati differenti. I dati per la compilazione vengono prelevati da database, e alcuni dei dati vengono utilizzati in entrambi i documenti.

Con l'utilizzo di XML è stato possibile riutilizzare le stesse strutture per i dati comuni utilizzati nella compilazione dei documenti, come per esempio il nominativo dello studente, la data dell'esame, il voto dell'esame etc.

Inoltre per differenziare i formati di stampa del verbale è bastato modificare il template della trasformazione XSLT adattandolo al diverso formato, senza dovere modificare il resto del codice.

## 2 Trasformazione da XML a PDF

La trasformazione per ottenere il documento PDF da un documento XML è quindi una combinazione di XSLT e XSL-FO. XSLT permette di navigare sull'albero del documento di partenza tramite le regole XPath e di applicare ai contenuti XML iniziali le trasformazioni necessarie per ottenere come risultato finale un altro documento XML, interpretabile dal parser FO.

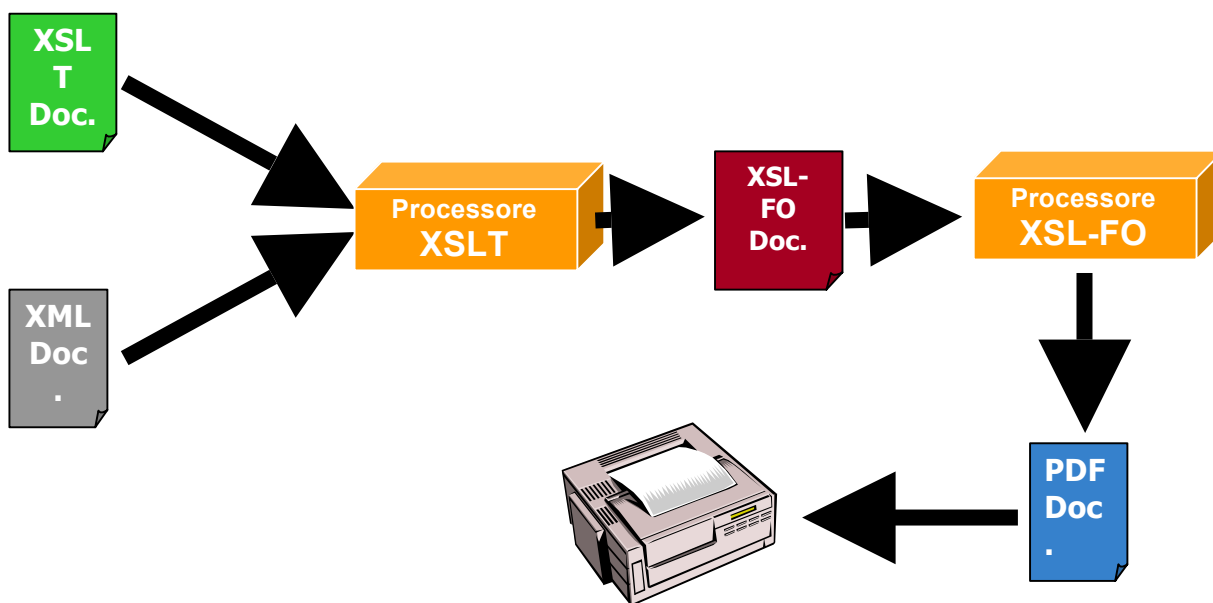


Figura 1 Processo completo della trasformazione da XML a PDF

## 3 Creazione dei file XSD

Per prima cosa sono stati creati i file XSD contenenti tutte le strutture dati necessarie da inserire nei documenti XML.

Queste strutture sono state utilizzate per fare il data-binding con le classi Java che rappresentano i vincoli contenuti nella grammatica definita nel file XSD (validazione).

Si possono allora usare queste classi per generare i documenti XML, per leggere i documenti XML e per convalidarli seguendo la grammatica data.

Vengono adesso riportati tre strutture dati create, FullName e PersonalData e VerbaleFilling.

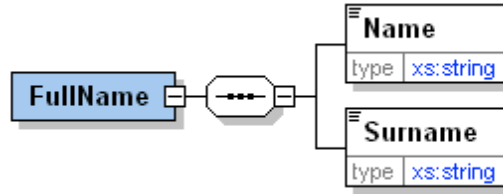


Figura 2 XML Schema Definition della struttura dati FullName

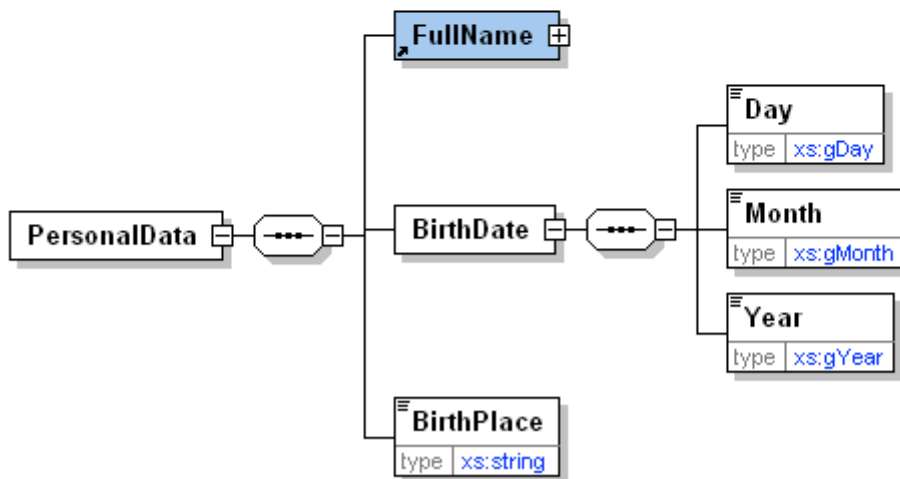


Figura 3 XML Schema Definition della struttura dati PersonalData che riutilizza la struttura dati FullName al suo interno

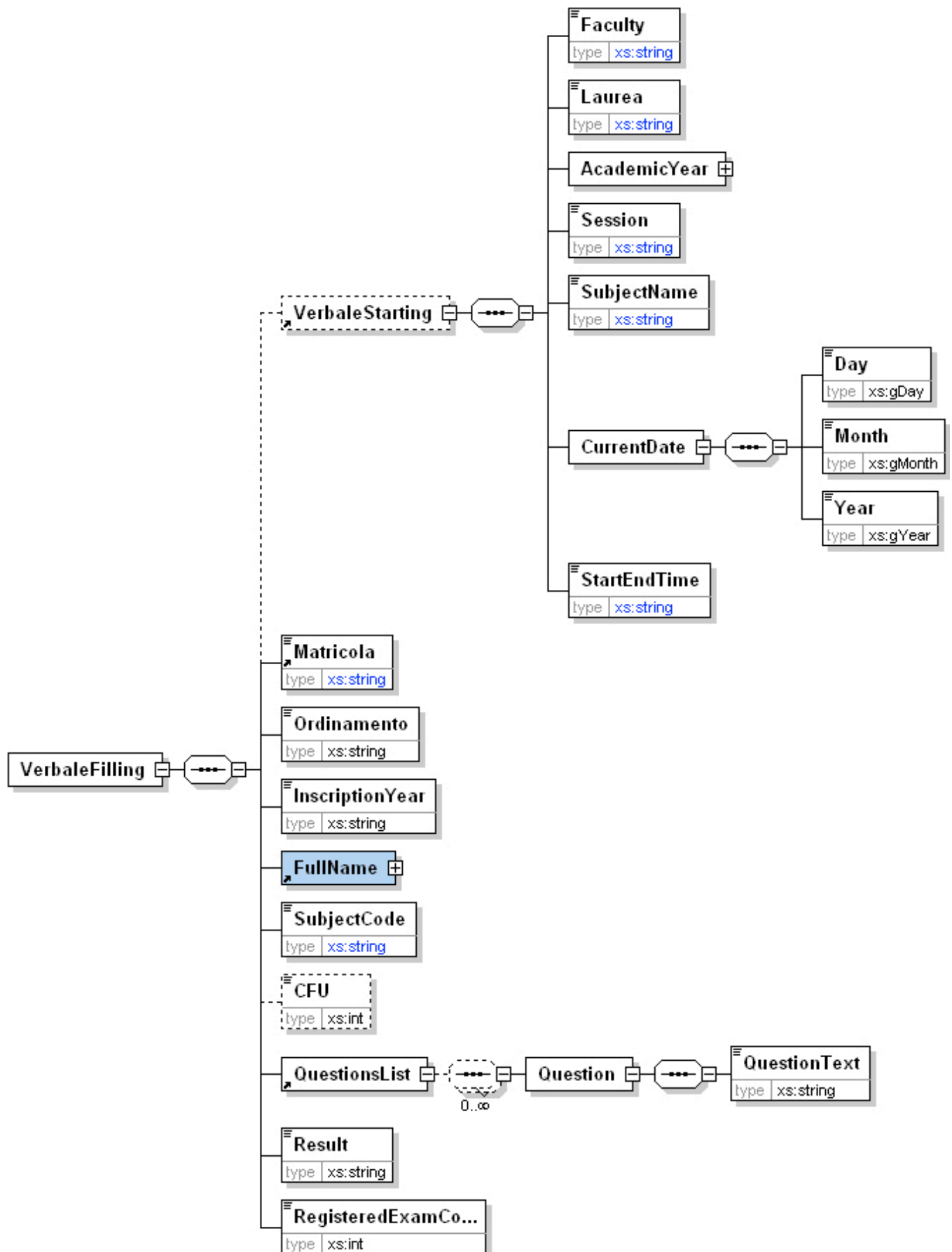


Figura 4 XML Schema Definition della struttura dati VerbaletFilling contenente tutti i campi necessari per la compilazione del verbale.

Questa struttura è composta dai seguenti campi:

- Verbale Starting: struttura dati contenente i dati necessari per l'apertura del verbale quali
  - Faculty di tipo String contenente la facoltà.
  - Laurea di tipo String contenente il corso di laurea
  - AcademicYear è composto da due campi di tipo year:AcademincYearStarting e AcademicYear Ending, e contiene l'anno accademico corrente.
  - Session di tipo String contenente la sessione d'esame
  - SubjectName di tipo String contiene il nome della materia d'esame
  - CurrentDate è composto da tre campi, Day,Month e Year rispettivamente di tipo gDay,gMonth e gYear, contenente la data di apertura del verbale.
  - StartEndTime di tipo String contiene l'ora di apertura del verbale nel caso venga chiamata per l'apertura del verbale e l'ora di chiusura nel caso venga chiamata per la chiusura del verbale.

Questa struttura dati è opzionale. Infatti è presente solo nel caso di apertura del verbale e con registrazione del primo candidato e non per i candidati successivi.

- Matricola di tipo String contenente la matricola.
- Ordinamento di tipo String contenente l'ordinamento(VO o NO).
- InscriptionYear di tipo String contenente l'annodi corso del candidato.
- FullName è composto da due campi di tipo String Name e SurName contenenti rispettivamente il nome e il cognome del candidato.
- SubjectCode di tipo String contenente il codice della materia d'esame.
- CFU di tipo String contenente i crediti formativi. Questa struttura è opzionale perché è utilizzata solo se si tratta di un candidato che ha ordinamento NO.
- QuestionsList contiene una lista di Question che contiene un capo di tipo String contenente il testo della domanda posta al candidato durante l'esame.
- Result di tipo String contenente il voto d'esame.
- RegisteredExamCounter di tipo int contenente il numero di esami svolti.

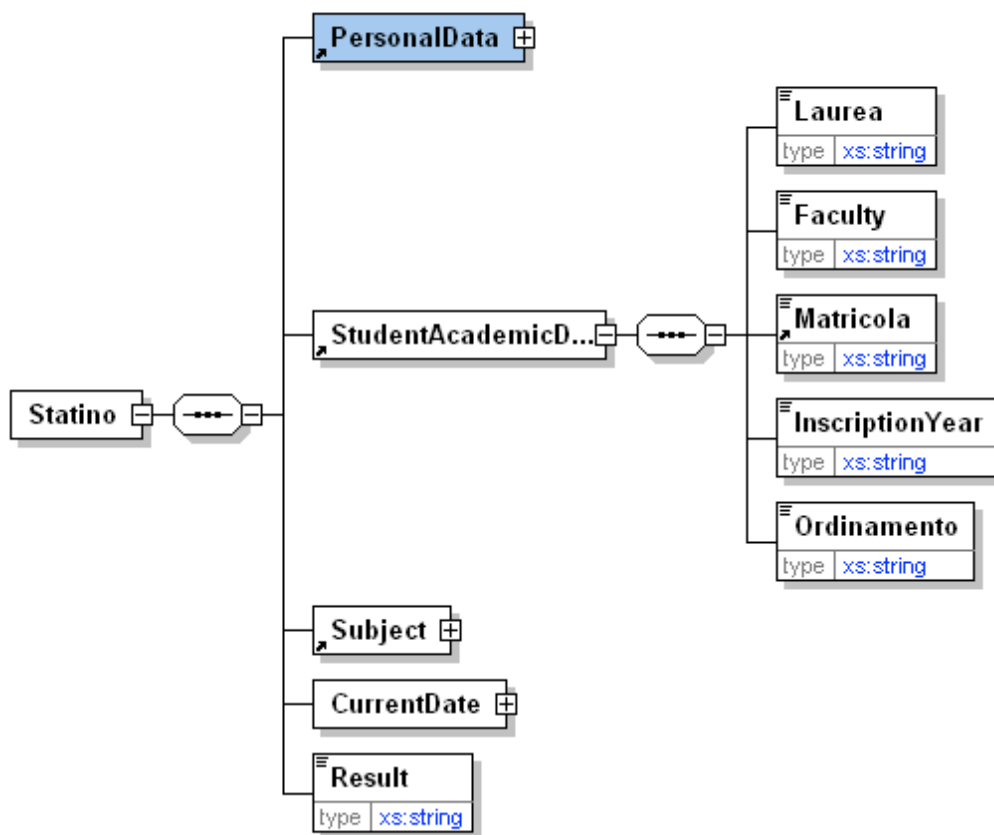


Figura 5 XML Schema Definition della struttura dati Statino contenente tutti i necessari per la compilazione dello statino

Come si può notare dal confronto della Figure 4 e della Figura 5 ci sono molte strutture dati comuni che vengono riutilizzate per le due macro-strutture del verbale e dello statino.

## 4 Creazione dei file XML

Un parser XML è un modulo software che si colloca tra l'applicazione e il documento XML. Esso permette all'applicazione di accedere al contenuto e alla struttura del documento XML.

Esistono due tipi di parser: validanti e non validanti. I primi, oltre a controllare se un documento è ben-formato, cioè che ogni elemento sia racchiuso tra due tag (uno di apertura e uno di chiusura), controlla pure se esso è un documento XML valido, cioè se è fedele alle regole definite nella sua grammatica. I parser non validanti, invece, si preoccupano solo di vedere se un documento è ben formato. La validazione avviene tramite le classi Java che valicano i file XML con i file XSD relativi.

Per interfacciare il parser con l'applicazione è stata usata un'interfaccia object-based. Con l'approccio object-based, il parser costruisce esplicitamente in memoria un albero che contiene tutti gli elementi del documento XML. Rappresentare il contenuto di un documento XML tramite un "in-memory tree" permette di attraversare l'albero ragionando per gradi di parentela (nodi figli, nodo padre, ecc.)

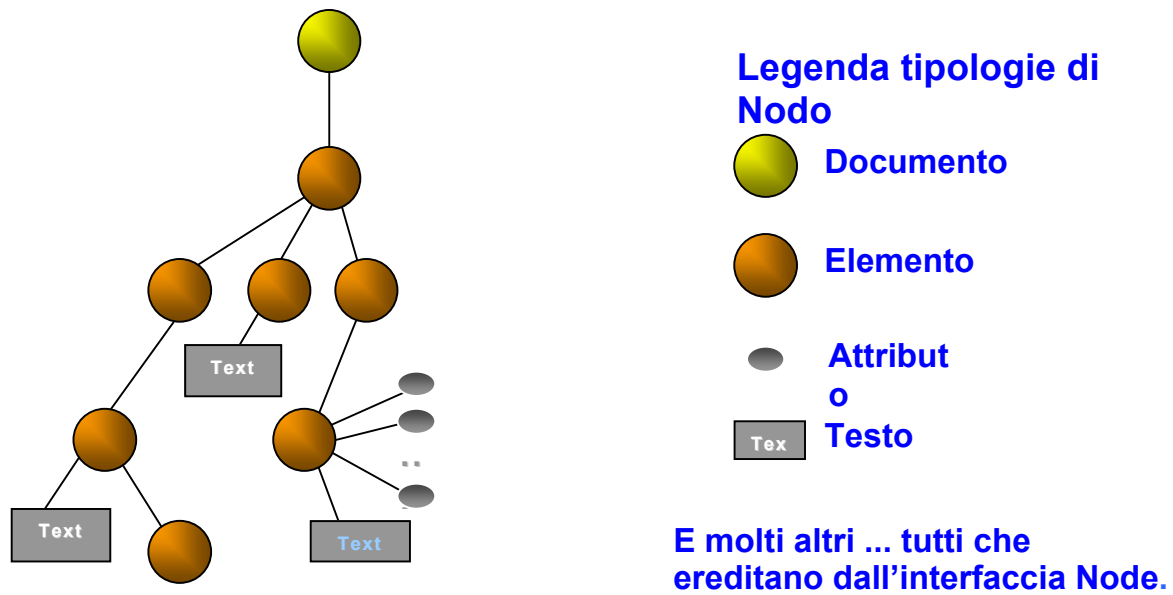


Figura 6 XML in-memory tree

Per la conversione dei dati tra Java e XML è stato usato un marshalling framework. Questo è composto da due parti: marshalling e unmarshalling.

*Marshalling* scrive un documento XML a partire da oggetti Java (Java → XML).

*Unmarshalling* legge un documento XML e lo traduce in oggetti Java (XML → Java).

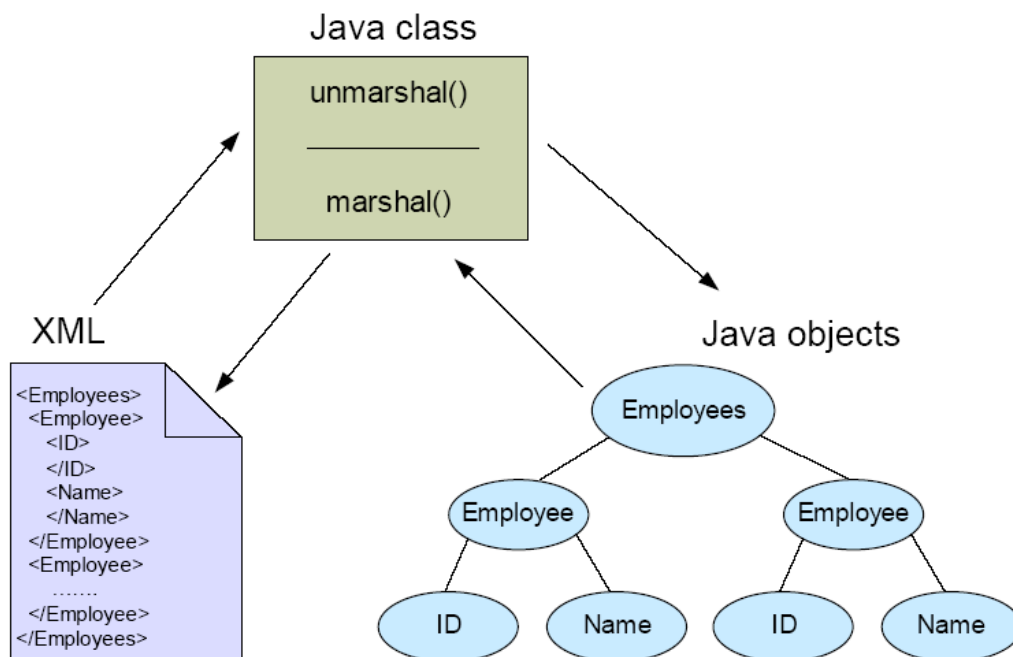


Figura 7 Marshalling framework

A seguire viene riportato un esempio di file XML creato per la compilazione dello statino.

```
<?xml version="1.0" encoding="UTF-8"?>
<Statino>
  <PersonalData>
    <FullName>
      <Name>Mario</Name>
      <Surname>Rossi</Surname>
    </FullName>
    <BirthDate>
      <Day>5</Day>
      <Month>1</Month>
      <Year>1985</Year>
    </BirthDate>
    <BirthPlace>Palermo</BirthPlace>
  </PersonalData>
  <StudentAcademicData>
    <Laurea>Ingegneria Informatica</Laurea>
    <Faculty>Ingegneria</Faculty>
    <Matricola>0311111</Matricola>
    <InscriptionYear>2</InscriptionYear>
    <Ordinamento>NO</Ordinamento>
  </StudentAcademicData>
  <Subject>
    <Name>Ingegneria del software</Name>
    <Code>56215</Code>
    <CFU>6</CFU>
  </Subject>
  <CurrentDate>
    <Day>26</Day>
    <Month>2</Month>
    <Year>2005</Year>
  </CurrentDate>
  <Result>ventotto</Result>
</Statino>
```

## 5 Creazione file XSLT

Per effettuare la trasformazione è stato creato un documento XSLT, di cui fatta la trasformazione si ottiene un output di tipo XSL-FO che definisce la grammatica per impaginare il file XML e che viene processato per ottenere l'output di tipo PDF.

Nel caso in esame, le trasformazioni XSLT sono rappresentate dalla definizione dei *template* per "/" e per gli elementi composti. All'interno delle sezioni "template" vengono posizionati i tag del namespace FO per la definizione delle regole di impaginazione.

```
<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" encoding="UTF-8"/>
  <xsl:template match="/">
```

In questo caso la trasformazione FO si compone di un layout di pagina, `simple-page-master`, posizionato nel modello di trasformazione relativo a `"/`"; qui viene settato l'aspetto grafico quindi vengono settate le dimensioni della foglio e viene inserito lo sfondo

```
<fo:layout-master-set>
  <fo:simple-page-master master-name="PaginaStatino" page-height="214mm" page-width="157mm"
    margin-top="0.0cm" margin-bottom="0.0cm" margin-left="0.0cm" margin-right="0.0cm">
    <fo:region-body background-image="url('./Statino.gif)"
      background-repeat="no-repeat"
      background-attachment="fixed"
      margin-top="0.0cm" margin-bottom="0.0cm" margin-left="0.0cm" margin-right="0.0cm">
      </fo:region-body>
    <fo:region-before extent="0cm"/>
    <fo:region-after extent="0cm"/>
  </fo:simple-page-master>
</fo:layout-master-set>
```

Successivamente alla definizione dell'aspetto grafico della pagina, viene descrittoli contenuto delle singole istanze della pagina. Queste informazioni si forniscono della sezione `page-sequence`

```
<fo:page-sequence master-reference="PaginaStatino">
  <fo:flow flow-name="xsl-region-body">
    <fo:block font-size="20pt" color="black" letter-spacing="8px" space-before="42pt" margin-left="305pt">
      <xsl:value-of select="./Statino/StudentAcademicData/Matricola"/>
    </fo:block>
  </fo:flow>
</fo:page-sequence>
```

## 6 Trasformazione XSLT

Per processare il file XML creato e convertirlo in un file PDF sono state usate le librerie JAXP (XSLT) e FOP (XSL:FO) che permettono l'accesso alle classi che definiscono i processori XSLT e che lavorano su oggetti `StreamSource` come input della trasformazione e su oggetti di tipo `SAXResult` come output della trasformazione.

```
Source input = new StreamSource(xmlFile);
SAXResult output = new SAXResult(driver.getContentHandler());
OutputStream pdfout = new java.io.FileOutputStream(pdfFile);
```

Settati input e output inizia la trasformazione e il processo FOP:

```
TransformerFactory factory = TransformerFactory.newInstance();
Transformer transformer = factory.newTransformer(new StreamSource(xsltFile));
transformer.transform(input, output);
```

infine viene salvato l'output nel file PDF.