XML-TEI e Tagged PDF

Luigi Scarso

Sommario

In questo articolo mostriamo come gestire in ConTEXt un documento in formato XML per produrre una rappresentazione in formato PDF e come l'utilizzo dei tag permetta di strutturare il documento PDF in modo analogo al documento XML. Viene presentato un originale foglio di stile per il completo embedding del documento XML sorgente e come analizzare il PDF tramite luaTEX per estrarre il documento.

Abstract

In this paper we show how to manage an XML document in ConTEXt to produce a PDF output and how the tags of the PDF can be used to give a structure to the PDF document. We also show a novel stylesheet for the complete embedding of the XML source into the PDF and how to analyze the final PDF with luaTEX to extract the document.

1 Introduzione

Con il termine *workflow* si intende una descrizione di un processo che, possibilmente per passi successivi, trasforma uno o più *input* in uno o più *output*. La definizione è evidentemente adattabile a numerosi contesti, e nell'ambito della tipografia digitale è solitamente usato per descrivere come ottenere un documento finale (che nella maggior parte delle volte è in formato PDF) a partire da uno o più documenti sorgenti, attraverso varie fasi che trasformano i sorgenti in forme intermedie via via simili a quella finale. Ad esempio, nel caso di una rivista, i documenti sorgenti possono essere gli articoli, le foto e le inserzioni pubblicitarie ed il workflow descriverebbe allora come trasformare i testi per il software di impaginazione, la corretta dimensione delle immagini e delle inserzioni, e come marcare i componenti intermedi così ottenuti per il corretto ordine nella rivista finale. In ambito umanista il caso tipico è quello che presenta un solo documento sorgente (normalmente una edizione cartacea) e un solo output, anche in questo caso PDF in quanto, oltre ad offrire elevate qualità tipografiche, supporta l'archiviazione a lungo termine ed è usufruibile su una gran numero di dispositivi. Il problema in questo caso è come passare dal documento cartaceo al PDF: la scansione delle pagine è solamente una parte della soluzione perché, benché sia il metodo più economico per preservare le informazioni (anche tipografiche), ha l'evidente svantaggio di non rendere fruibile il documento sorgente per attività di elaborazione del testo. Una alternativa è tradurre il testo originale in un formato elettronico che sia facilmente adattabile ad un generico workflow, e questo è precisamente il compito della *Text Encoding Initiative* (TEI), un consorzio che comprende università ed enti di ricerca ed il cui compito è mantenere una *applicazione* XML per la codifica di testi (specialmente dell'area umanista, scienze sociali e linguistica) in forma elettronica. In breve, in Anonymous (2014) TEI specifica e descrive un formato XML per la *struttura* del testo e alcune metainformazioni (ad esempio le revisioni, i dati relativi all'edizione originale ecc.) ma *non* le proprietà tipografiche — in sostanza è il duale della scansione. Un workflow che ha come ingresso un formato XML quasi sempre prevede dei passi intermedi che utilizzano XSLT o XQUERY (altre applicazioni XML, cfr. (Clark, 1999), (Kay, 2007) e Florescu *et al.* (2010)) per trasformare il documento in una forma adatta; in questo caso è lo stesso TEI che mette a disposizione presso `http://www.tei-c.org/Tools/Stylesheets/index.xml` i file XSLT 2.0 (noti come "fogli di stile" o *stylesheets*) per trasformare un documento XML-TEI in una moltitudine di output: csv, docbook, docx, dtd, epub3, epub, fo, html5, html, ibooks, json, latex, lite, oddhtml, odt, p4, rdf, relaxng, tcp, txt, e xlsx, seguendo quanto scritto in `http://www.tei-c.org/release/doc/tei-xsl`. Si noti che non è direttamente previsto il formato PDF. In pratica esiste un solo programma (in questo ambito chiamato *processore*) open source in grado di utilizzare i fogli di stile XSLT 2.0: Saxon (disponibile presso `http://saxon.sourceforge.net`), che è un programma Java e richiede il runtime adeguato.

Considerando l'ambito TEX, praticamente esistono due tipi di workflow:

- il primo, delineato sopra, utilizza un processore XSLT per tradurre un documento XML-TEI in formato pdfLATEX o XELATEX e quindi usa il motore di impaginazione corrispondente per il PDF. In ambito LATEX probabilmente è l'unico workflow usato;
- il secondo utilizza un processore scritto in TEX (non necessariamente completamente XSLT compatibile) per elaborare direttamente il file XML.

È da notare che il set di elementi/attributi descritti nelle *guidelines* è veramente molto ricco: è naturale che un autore utilizzi un sottoinsieme piuttosto ristretto (ad esempio, solo gli elementi

relativi ad una edizione critica) e che quindi il relativo foglio di stile implementi un workflow XML che traduca solo questi elementi. Si tratta in sostanza di distinguere tra workflow che accetta un *qualsiasi* XML-TEI contro un workflow specifico per un preciso testo, il quale normalmente risulta più gestibile rispetto ad uno generico.

Ecco un frammento dell'opera di William Shakespeare *Romeo and Juliet* (di cui più avanti daremo i dettagli) in formato XML-TEI; sono le prime battute del primo atto:

```
<div1 type="act" n="1" org="uniform"
                    sample="complete">
  <head>ACT I</head><lb ed="F1" n="2" />
<div2 type="scene" n="1" org="uniform"
                     sample="complete">
<head>SCENE I</head>
<stage type="setting">
Verona. A public place.
</stage>
<lb ed="F1" n="3" />
<stage type="entrance">
Enter SAMPSON and GREGORY,
<lb ed="F1" n="4" />
of the house of Capulet,
armed with swords and bucklers.</stage>
<lb ed="G" /><lb ed="F1" n="5" />
<sp who="sam.">
<speaker>Sam.</speaker><p>
Gregory, o' my word, we'll not
<lb ed="G" />carry coals.
<lb ed="G" />
<lb ed="F1" n="6" /></p></sp>
<sp who="gre.">
<speaker>Gre.</speaker>
<p>No, for then we should be colliers.
<lb ed="G" />
<lb ed="F1" n="7" /></p></sp>
<sp who="sam.">
<speaker>Sam.</speaker>
<p>I mean, an we be in choler, we'll
<lb ed="G" />draw.
<lb ed="G" />
<lb ed="F1" n="8" /></p></sp>
<sp who="gre.">
<speaker>Gre.</speaker><p>
Ay, while you live, draw your neck
<lb ed="G" />
<lb ed="F1" n="9" />out o' the collar.
<lb ed="G" />
<lb ed="F1" n="10" />
</p></sp><sp who="sam.">
<speaker>Sam.</speaker>
<p>I strike quickly, being moved.
<lb ed="G" />
<lb ed="F1" n="11" />
</p></sp><sp who="gre.">
<speaker>Gre.</speaker>
<p>But thou art not quickly moved to
<lb ed="G" />strike.
<lb ed="G" n="10" />
<lb ed="F1" n="12" /></p></sp>
<sp who="sam.">
<speaker>Sam.</speaker>
<p>A dog of the house of Montague moves me.
<lb ed="G" /><lb ed="F1" n="13" /></p></sp>
<sp who="gre.">
<speaker>Gre.</speaker>
<p>To move is to stir; and to be valiant
<lb ed="G" />is to stand:
<lb ed="F1" n="14" />
therefore, if thou art moved, thou
<lb ed="G" />runn'st away.
<lb ed="G" /><lb ed="F1" n="15" />
</p></sp><sp who="sam.">
<speaker>Sam.</speaker>
<p>A dog of that house shall move me
<lb ed="G" />to stand: <lb ed="F1" n="16" />
I will take the wall of any man or
<lb ed="G" />maid of Montague's.
```

Nella figura 1 c'è una possibile presentazione.

Infine, nella quasi totalità dei casi non interessa se il workflow è *reversibile*, ovvero se dall'uscita sia possibile riscostruire l'ingresso ed in quale misura. Infatti nei PDF prodotti con pdfLaTeX o XeLaTeX la struttura logica appare evidente all'utente ma non è codificata *dentro* il PDF in modo da poter essere utilizzabile da altri programmi (i *bookmark* contengono soltanto le informazioni di struttura, non tabelle, liste ecc).

Nelle prossime sezioni vedremo come affrontare un workflow in XML con ConTEXt-MkIV, e se le osservazioni precedenti hanno risposta soddisfacente.

2 XML in ConTEXt

ConTEXt-MkIV ((Hagen, a),(Hagen, b)) utilizza esclusivamente luaTEX come motore di impaginazione (*engine*)[1] ed implementa un linguaggio per trasformare documenti XML o più precisamente per selezionare gli elementi di un documento per mapparli poi in macro TEX (Hagen, c). Questo linguaggio *non* è una implementazione di XSLT ed utilizza la libreria interna LPEG (Ierusalimschy, 2009) ed una specifica *Parsing Expression Grammar* (o PEG, cfr. (Grune e Jacobs, 2007)) per il parsing di un generico documento XML. Il parser risultante è piuttosto performante, cosa non insolita se si pensa che deve convertire gli elementi *solo* verso ConTEXt.

Come accennato in precedenza il documento TEI scelto è l'opera di William Shakespeare *Romeo and Juliet*, W. G. Clark, W. Aldis Wright, Ed., The Globe Shakespeare, New York, Nelson Doubleday, Inc. disponibile per il download presso `http://www.perseus.tufts.edu/hopper/text?doc=Perseus:text:1999.03.0053`.

Come primo passo è stato deciso di modificare il file per renderlo compatibile con l'ultima versione

1. Il termine è un po' impreciso in quanto, a volte, in un workflow basato su TEX, con motore di impaginazione si intende ConTEXt o LaTeX, cioè il macro-formato più che l'engine stesso.

ACT I

SCENE I

Verona. A public place.

Enter SAMPSON and GREGORY,
of the house of Capulet, armed with swords and bucklers.

Sam. Gregory, o' my word, we'll not
carry coals.

Gre. No, for then we should be colliers.

Sam. I mean, an we be in choler, we'll
draw.

Gre. Ay, while you live, draw your neck
out o' the collar.

Sam. I strike quickly, being moved.

Gre. But thou art not quickly moved to
strike.

Sam. A dog of the house of Montague moves me.

Gre. To move is to stir; and to be valiant
is to stand:
therefore, if thou art moved, thou
runn'st away.

Sam. A dog of that house shall move me
to stand:
I will take the wall of any man or
maid of Montague's.

5

Figura 1: Le battute iniziali del primo atto dell'opera di W. Shakespeare *Romeo and Juliet*.

P5 delle specifiche TEI (Anonymous, 2014); la modifica più importante probabilmente è stata il corretto nome dell'attributo `id` (deve essere `xml:id`) ma a parte questo pochi altri elementi sono stati modificati. Questo implica una sostanziale robustezza del formato TEI: documenti relativamente "vecchi" possono essere facilmente aggiornati.

Processare un file XML con ConTEXt non differisce molto da usare un processore XSLT: per prima cosa creiamo il file

`Perseus_text_1999.03.0053-2.6.0.tex`

il quale carica il file di stile

`s-tei-Perseus-text-1999-03-0053-style`

e poi elabora il file XML

`Perseus_text_1999.03.0053-2.6.0.xml`

ovvero:

```
\loadmarkfile{%
s-tei-Perseus-text-1999-03-0053-style}
\starttext
\xmlprocessfile{RandJ}
   {Perseus_text_1999.03.0053-2.6.0.xml}{}
\stoptext
```

Lo stylesheet definisce le caratteristiche tipografiche come il font (TeX Gyre Schola), il layout di pagina ed il setup dei due soli livelli di sezionamento usati (in corrispondenza degli analoghi livelli nel documento XML). I file rilevanti per il mapping degli elementi XML sono però

`s-tei-Perseus-text-1999-03-0053-basics.lua`

e

`s-tei-Perseus-text-1999-03-0053-basics.mkiv`

```
%D \module
%D [file=s-tei-Perseus-text-1999-03-0053-style,
%D  version=2014.07.01,
%D  title=Romeo and Juliet,
%D  subtitle=,
%D  author=Luigi Scarso,
%D  date=\currentdate,
%D  copyright={Luigi Scarso}]

\registerctxluafile%
{s-tei-Perseus-text-1999-03-0053-basics}
{1.001}

\loadmarkfile%
{s-tei-Perseus-text-1999-03-0053-basics}

\registerctxluafile%
{s-tei-Perseus-text-1999-03-0053-style}
{1.001}

\setuppapersize[A4,landscape][A4,landscape]
\starttypescript[randj]
 \definetypeface [randj]%
           [rm] [serif] [schola] [default]
 \definetypeface [randj]%
           [ss] [sans]  [schola] [default]
 \definetypeface [randj]%
           [tt] [mono]  [schola] [default]
 \definetypeface [randj]%
           [mm] [math]  [modern] [default]
\stoptypescript
\setupbodyfont[randj,10pt]

\setuplayout
  [width=middle,
   height=middle,
   backspace=3cm,
   cutspace=1cm,
   rightmargin=0cm,
   leftmargin=2cm,
   margindistance=0cm,
   footer=1cm,
   header=\zeropoint]
\setuppagenumbering
 [alternative=singlesided,
  location={footer,middle}]
\setuphead[chapter]
          [style=\tfc,
           align=middle,
           number=no,
           expansion=yes,
           before=,
           after=\blank,
           page=yes]
\setuphead[section]
          [style=normal,
           align=middle,
           number=no,
           expansion=yes,
           before={\blank[2*line]},
           after={\blank[line]},
           page=no]
\setupinteraction[state=start,
          color=black,
          contrastcolor=black,
          style=normal,
          title={Romeo and Juliet},
          author={Luigi Scarso}]

\placebookmarks[chapter][all][force=yes]

\mainlanguage[en]
```

Il file MkIV `*basics.mkiv` mostra il meccanismo utilizzato da ConTEXt per l'XML: si definisce un ambiente `setups` che descrive come mappare gli elementi XML in ambienti ConTEXt:

```
\startxmlsetups xml:tei:Perseus:text:1999:03:
0053-setups
\xmlsetsetup{#1}{*}{xml:tei:*}
\stopxmlsetups
```

e lo si registra per attivarlo:

```
\xmlregistersetup{xml:tei:Perseus:text:
1999:03:0053-setups}
```

Il significato della macro `\xmlsetsetup{#1}{*}{xml:tei:*}` è il seguente: l'elemento *e* (`*` significa "tutti gli elementi") del file `#1` associato a questo setup è processato dal `setups` `xml:tei:`*e* — che deve essere definito dall'utente. Ad esempio l'elemento `TEI` è processato da

```
\startxmlsetups xml:tei:TEI
   \xmlfunction{#1}{tei:TEI}
\stopxmlsetups
```

il suo figlio `teiHeader` è:

```
\startxmlsetups xml:tei:teiHeader
   \xmlfunction{#1}{tei:teiHeader}
\stopxmlsetups
```

La macro `\xmlfunction{#1}{#2}` equivale alla funzione Lua `xml.functions["#2"](#1)`, dove `#1` è l'elemento del documento; ad esempio `\xmlfunction{#1}{tei:TEI}` vista prima chiama la funzione `xml.functions["tei:TEI"](#1)` con `#1` l'elemento `TEI` del documento, cioè la radice.

Ogni elemento del documento è mappato in questo modo: in pratica ogni elemento del documento chiama una funzione Lua definita nel file Lua `*basics.lua`. Ecco la sezione del file Lua relativa agli elementi `TEI` e `teiHeader`:

```
local xml_func = xml.functions or {}

local function TEI(t)
   local att = {['elementname']='TEI'}
   context.startelement({"document"},att)
   context.setupelementuserproperties(
                        {'document'},att)
   lxml.flush(t)
   context.stopelement()
end
local function teiHeader(t)
   local att = {['elementname']='teiHeader'}
   context.startelement({"metadata"},att)
   context.setupelementuserproperties(
                       {'metadata'}, att)
   context([[{\sc]])
   lxml.flush(t)
   context([[}]])
   context.blank()
   context.stopelement()
end

xml_func["tei:TEI"]  = TEI
xml_func["tei:teiHeader"] = teiHeader
```

Tralasciando per il momento `startelement`, `setupelementuserproperties` e `stopelement` che vedremo nella prossima sezione, la funzione `TEI(t)` con `lxml.flush(t)` semplicemente abilita l'elaborazione degli elementi figli; `teiHeader(t)` è più o meno simile, ma racchiude i figli dentro un gruppo `{\sc ..}` per attivare il maiuscoletto del font. Come si può vedere, quando si è sulla "parte" Lua è possibile comunicare con la "parte" TEX tramite la funzione `context(<tex-string>)`: in ConTEXt è una funzione parecchio usata, ed è specializzata per gestire le macro al punto che è possibile scrivere interi documenti solamente in Lua (ad esempio la funzione Lua `context.blank()` equivale alla macro `\blank`, e così per ogni macro di ConTEXt: naturalmente in alcuni casi TEX rimane più semplice e chiaro rispetto alla controparte Lua).

Tralasciamo gli altri elementi, in quanto riteniamo di aver mostrato le parti principali del processore XML. In Hagen (c) sono descritte le varie tecniche per selezionare/filtrare nodi usando determinate condizioni, ma sostanzialmente la complessità non è differente dal linguaggio XSLT. Anche la relazione tra elemento XML ed elemento tipografico richiede la conoscenza della sintassi di ConTEXt, ma anche in questo caso la complessità è simile a quella di LATEX. Nella prossima sezione vedremo invece una caratteristica peculiare di ConTEXt, i PDF con tags.

3 Tagged PDF

La guida di riferimento del PDF 1.7 (disponibile presso Adobe) dedica il capitolo 10 alla descrizione di come sia possibile inserire informazioni sulla struttura logica del documento sorgente all'interno del file PDF. Il livello più semplice è il *contenuto evidenziato* (Marked Content) dove una serie di operatori PDF associa un tag ad uno stream di contenuto (o parte di esso): ad esempio la coppia `BDC EMC` usa la sintassi *tag properties* `BDC ... EMC` ed è usata in ConTEXt per marcare una parte del testo, ad esempio il titolo di una sezione:

```
/sectiontitle <</MCID 0>>BDC
BT
/F1 17.21541 Tf 1 0 0 1 353.324 528.9996 Tm
[<002F0060001C004B001C>30<006900420062006800
540032006000620051004D>25<00A4>]TJ
ET
EMC
```

Ad un livello superiore esiste la struttura logica, un set gerarchico di elementi struttura ciascuno rappresentato da un dizionario. Gli elementi struttura e i contenuti visibili e associati sono memorizzati in parti differenti del PDF ma ciascuno ha un puntatore verso l'altro, in modo ad esempio da poter associare un titolo ad una pagina. In pratica, la struttura logica permette di definire una struttura ad albero, dove i figli sono terminali (per esempio Marked Content) o nodi non terminali — cioè altri elementi struttura. A questo livello *non* vengono stabiliti nomi degli elementi struttura, per cui l'utente può utilizzare i nomi più convenienti. Adobe ha introdotto un ulteriore livello, il Tagged PDF, che si basa sui due precedenti ma dove fissa il nome degli elementi struttura e la loro semantica (i *tags*). Purtroppo i tags definiscono un documento piuttosto povero dal punto di vista semantico, ma nella struttura logica è previsto un dizionario (*RoleMap*) che mappa il nome degli elementi struttura definito da un utente con i tags del Tagged PDF: questo implica che un utente che fornisca una RoleMap automaticamente definisce un Tagged PDF.

Per utilizzare Tagged PDF in ConTEXt bisogna abilitarli con

```
\setupstructure[state=start,method=auto]
\setuptagging[state=start]
```

in quanto per default sono disabilitati. Una volta abilitati, gli elementi struttura vengono automaticamente aggiunti con i comandi di sezionamento, liste, tabelle e figure.

ConTEXt definisce un proprio insieme di elementi struttura, e fornisce la relativa RoleMap, quindi può produrre un Tagged PDF — in effetti, può produrre PDF/A-1a cioè Tagged PDf validi per l'archiviazione digitale. Anche se questo insieme è migliore di quello definito da Adobe ovviamente non è l'insieme di tags usati per esempio da TEI: ovviamente è possibile modificare il codice ConTEXt per usare i tag di TEI con una opportuna RoleMap, ma qui proponiamo una soluzione differente.

Al livello di struttura logica per un elemento struttura è definito il dizionario (opzionale) UserProperties il cui contenuto è gestito dall'applicazione utente. Esso contiene un array di coppie chiave/valore che è possibile utilizzare in questo modo:

1. si fissa la chiave `elementname` con il nome dell'elemento XML.
2. se un elemento XML ha un attributo `a` con valore `v` si fissa la chiave `a` con valore `v` — in pratica si copia l'attributo.

Ad esempio l'elemento TEI `sp` che marca il discorso di un attore e che ha come attributo `who` viene mappato nell'elemento `p` di ConTEXt in questo modo:

```
local function sp(t)
   local at = t.at
   local att = {['elementname']='sp'}
   att.who = at.who or nil
   context.startelement({"p"},att)
   context.setupelementuserproperties({'p'},
                                      att )
   lxml.flush(t)
   context.stopelement()
end
```

Ecco ad esempio come viene salvato nel PDF l'elemento `<sp who='chorus'>...</sp>`:

```
209 0 obj
<< /A << /O /UserProperties /P [
<< /V (sp) /N (elementname) >>
<< /V (chorus.) /N (who) >> ] >>
/Type /StructElem /S /p
/Pg 201 0 R /K 208 0 R /P 207 0 R >>
```

È evidente che il nome `elementname` è arbitrario ed in generale va verificato che non sia in conflitto con altri attributi — in questo specifico caso non esistono conflitti.

Il passo finale è il seguente: è possibile verificare la struttura di un Tagged PDF ? La risposta è positiva: con l'attuale versione di LUATEX il modulo Lua `epdf` fornisce un insieme di funzioni di basso livello per leggere un file PDF e quindi per "navigare" attraverso i suoi oggetti. Naturalmente richiede una conoscenza piuttosto approfondita del formato PDF, ma le specifiche ADOBE sono chiare e liberamente disponibili. Inoltre utilizzando funzioni di basso livello il relativo programma sarà meno soggetto alle conseguenze degli aggiornamenti delle librerie. L'autore, tuttavia, essendo parte del team di sviluppo, ha notato che con l'ultimo aggiornamento della libreria `poppler` che è la base di `epdf` sono stati aggiunti nuovi metodi per gli elementi struttura e quindi ha aggiunto tali metodi alla libreria `epdf` (la modifica riguarda per il momento solo la versione sperimentale, ma è probabile che a breve verrà estesa a quella canonica).

Costruire un programma che legga la struttura di un PDF è leggermente più semplice se si dispongono dei nuovi metodi per gli elementi struttura; in questo caso il programma salva le informazioni relative alle UserProperty di un elemento struttura e le presenta in formato XML. Nel caso del documento TEI di *Romeo and Juliet* i primi tag dentro il PDF sono:

```
<TEI>
  <teiHeader type="text" >
   <fileDesc>
    <titleStmt>
     <title>   Romeo and Juliet
     </title>
     <author>  William Shakespeare
     </author>
     <editor role="editor" > W. G. Clark
     </editor>
     <editor role="editor" > W. Aldis Wright
     </editor>
     <sponsor>  Perseus Project,
                Tufts University
     </sponsor>
     <principal>  Gregory Crane
     </principal>
     <respStmt>
      <resp>  Prepared under the
              supervision of
      </resp>
      <name>   Lisa Cerrato
      </name>
      <name>   William Merrill
      </name>
      <name>   Elli Mylonas
      </name>
      <name>   David Smith
      </name>
     </respStmt>
     <funder>  NSF, NEH: Digital
    Libraries Initiative, Phase 2
     </funder>
     <respStmt xml:id="CEW.ed" >
      <name>  CEW
      </name>
      <resp>  ed.
      </resp>
     </respStmt>
```

```
    </titleStmt>
    <publicationStmt>
```

da confrontare con quello originale:

```
<TEI>
 <teiHeader type="text" > <!-- status="new" -->
  <fileDesc>
   <titleStmt>
    <title>Romeo and Juliet</title>
    <author>William Shakespeare</author>
    <editor role="editor">W. G. Clark</editor>
    <editor role="editor">W. Aldis Wright
    </editor>
    <sponsor>Perseus Project, Tufts University
    </sponsor>
     <principal>Gregory Crane</principal>
     <respStmt>
      <resp>Prepared under the supervision of
      </resp>
      <name>Lisa Cerrato</name>
      <name>William Merrill</name>
      <name>Elli Mylonas</name>
      <name>David Smith</name>
     </respStmt>
  <funder n="org:DLI2">NSF, NEH: Digital
    Libraries Initiative, Phase 2</funder>
  <!-- Revision -->
  <respStmt xml:id="CEW.ed"><name>CEW</name>
  <resp>ed.</resp></respStmt>
  </titleStmt>

      <publicationStmt>
```

Il codice è disponibile presso: `https://github.com/mlgdominici/TEI-TeX/tree/master/ctx/Perseus_text_1999.03.0053-2.6.0`

4 Conclusioni

L'elaborazione di file XML è piuttosto diretta in ConTEXt e non si appoggia a processori esterni. Questo significa ridurre la dipendenza da componenti esterne, utilizzando però un linguaggio di trasformazione che non è XSLT, sicuramente più diffuso e per il quale sono disponibili processori solidi relativamente a buon mercato. La scelta quindi dipende da altri vincoli imposti dal workflow utilizzato. Invece sono interessanti le prospettive che si aprono con i Tagged PDF, in particolare la possibilità di interrogare i PDF come se fossero documenti XML. Va detto che normalmente un PDF presenta diverse parti compresse per ottimizzare lo spazio, quindi l'interrogazione nel caso di un documento lungo diventa inefficiente se prima non si salva l'XML "embedded". Nel caso di PDF tipo "forms" che presentino i dati in forma tabellare in genere su 2 o 3 pagine, rinunciare alla compresssione può essere fattibile e questo approccio ai Tagged PDF risulta più promettente: ad esempio nel caso di estrazioni di report da un database, un Tagged PDF con queste caratteristiche si presta bene ad essere utilizzato da programmi che analizzano report — magari per pubblicare dati sul Web con una presentazione completamente differente. Un altro ambito interessante è la scheda tecnica di un prodotto industriale: le caratteristiche sono per lo più in forma tabellare e non è particolarmente difficile tradurle in XML — probabilmente in questi casi il problema pricipale è scegliere un XML adeguato.

Riferimenti bibliografici

Adobe. «Document Management – Portable Document Format – Part 1: PDF 1.7, First Edition». `http://www.adobe.com/devnet/pdf/pdf_reference.html`.

Anonymous (2014). *TEI P5: Guidelines for Electronic Text Encoding and Interchange. Version 2.6.0. Last updated on 20th January 2014, revision 12802.* TEI Consortium, eds. URL `http://www.tei-c.org/Guidelines/P5/`.

Clark, J. (1999). «XSL transformations (XSLT) version 1.0». W3C recommendation, W3C. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

Florescu, D., Simeon, J., Boag, S., Chamberlin, D., Fernandez, M. e Robie, J. (2010). «XQuery 1.0: An XML query language (second edition)». W3C recommendation, W3C. `http://www.w3.org/TR/2010/REC-xquery-20101214/`.

Grune, D. e Jacobs, C. (2007). *Parsing Techniques: A Practical Guide*. Monographs in Computer Science. Springer. URL `http://books.google.it/books?id=05xA_d5dSwAC`.

Hagen, H. (a). «? context». `http://www.pragma-ade.nl/general/manuals/what-is-context.pdf`.

— (b). «CONTEXT MKII CONTEXT MKIV». `http://www.pragma-ade.nl/general/manuals/mk.pdf`.

— (c). «Dealing with XML in ConTEXt-MkIV». `http://www.pragma-ade.nl/general/manuals/xml-mkiv.pdf`.

Ierusalimschy, R. (2009). «A text pattern-matching tool based on parsing expression grammars». *Softw. Pract. Exper.*, **39** (3), pp. 221–258. URL `http://dx.doi.org/10.1002/spe.v39:3`.

Kay, M. (2007). «XSL transformations (XSLT) version 2.0». W3C recommendation, W3C. Http://www.w3.org/TR/2007/REC-xslt20-20070123/.

▷ Luigi Scarso
`luigi dot scarso at gmail dot com`