XML e Tagged PDF

I «motori» TEX: pdfTEX, XƎTEX, LuaTEX

Il `tex` è un linguaggio di programmazione tipografica interpretato. Il costrutto principale è la *macro* — la macro fondamentale è il carattere:

```
\relax Ciao\end
```

sono `\relax` | | `C` | `i` | `a` | `o` | `\end` = 7 macro. L'output di un programma `tex` è una sequenza di istruzioni per posizionare e disegnare caratteri ed in generale elementi grafici — in pratica esistono due tipi di output: DVI e PDF.
Un interprete `tex` è chiamato *motore* e ne esistono diversi:

- TEX: `tex` ⟶ DVI. Font: METAFONT.
- eTEX: `tex` ⟶ DVI. Font: METAFONT
- pdfTEX: `tex` ⟶ DVI/PDF. Font: METAFONT, Type1, TrueType.
- XƎTEX: `tex` ⟶ extended DVI; PDF tramite conversione. Font: METAFONT, Type1, TrueType, OpenType.
- LuaTEX: `lua,tex` ⟶ PDF. Font: Type1, TrueType, OpenType.
  (LuajitTEX (L. Scarso, 2014) è una variante LuaTEX – LuaJIT al posto di Lua)

Grosso modo accettono tutti lo stesso input `tex`.

I «formati» TEX: plain, LATEX, ConTEXt...

Un motore esegue compiti «elementari»: per compiti complessi è necessario usare numerose macro «elementari» in modo corretto. Le macro si possono organizzare in un insieme detto *formato* ottimizzato per il caricamento.
Alcuni tra i formati più diffusi:

- plain: tutti i motori
- eplain: tutti i motori
- LATEX: tutti i motori, anche se il motore di riferimento è pdfTEX.
- $\mathcal{AMS}$-TEX, $\mathcal{AMS}$-LATEX (American Mathematical Society) : TEX, pdfTEX, XƎTEX;
- ConTEXt: pdfTEX e LuaTEX, il motore di riferimento è LuaTEX.

I formati *non* sono completamente compatibili, quindi un utente ha il proprio file `tex` specifico per la coppia `{motore,formato}`. La «coppia» più diffusa in ambiente accademico è {pdfTEX,LATEX} — i.e. pdfLATEX. Anche {XƎTEX,LATEX} — XƎLATEX— ha un buon seguito, mentre plain ed eplain sono poco usati.

LuaTEX, ConTEXt-MKII, ConTEXt-MKIV

LuaTEX ha al suo interno un interprete Lua, un linguaggio di scripting simile al Basic/Pascal — più familiare ad un programmatore medio rispetto al `tex`. Per costruire un formato è possibile usare sia Lua che `tex`; inoltre ha l'output nativo in PDF (come pdfTEX) e supporta i font OpenType (come XETEX).

ConTEXt è il formato sviluppato da H. Hagen per la propria azienda PRAGMA e disponibile con la distribuzione TEXLive e derivate.

- MKII: usa il motore pdfTEX e parzialmente XETEX. Il formato è *frozen*: solo correzioni di bugs.

- MKIV: usa solo LuaTEX ed è in costante sviluppo.

In pratica lo sviluppo di LuaTEX e ConTEXt procede di pari passo; inoltre Lua è stato progettato per interfacciarsi con le librerie C (ma anche C++, JAVA e C#) che espande notevolmente la possibilità di LuaTEX di integrarsi con altri sistemi (SWIGLIB project, L. Scarso 2013-).
Per contro, il formato non gode attualmente di una diffusione come pdfLATEX o XELATEX, almeno in ambiente accademico.

Informatica umanistica

«

• *Formare profili professionali implicati nella gestione di contenuti di carattere culturale con strumenti informatici, spaziando tra i diversi ambiti in cui il connubio tra informatica e discipline umanistiche si può realizzare ;*
• *Fornire ai neo-laureati in Informatica Umanistica la capacità di valutare criticamente le innovazioni tecnologiche e di adeguare le loro competenze alle nuove opportunità, grazie anche all'apertura mentale nata dall'aver lavorato a contatto con settori disciplinari così diversi.*

Informatica Umanistica è TRIENNALE e MAGISTRALE

»

Informatica Umanistica - Università degli Studi di Pisa
infouma.di.unipi.it/

Informatica umanistica

Un problema «classico»:
la traduzione di un testo in un formato digitale che

- preservi la struttura logica in modo rendere possibili ulteriori elaborazioni;
- preservi le informazioni grafiche e tipografiche;
- sia ampiamente accettato, possibilmente «standard»;
- sia stabile nel tempo;
- sia a costo minimo.

Esempio: l'edizione critica di un testo antico dove la forma del carattere può giocare un ruolo decisivo nella datazione del testo. Quanto è possibile «digitalizzare» questo aspetto e quali risorse richiede ?

Informatica umanistica

Una soluzione ampiamente adottata è la seguente:

- il testo viente tradotto in XML, un linguaggio di marcatura generale (i.e. a semantica non predefinita) che permette ls codifica della struttura logica scelta. XML offre con UNICODE un limitato supporto all'aspetto grafico e tipografico, ma è possibile utilizzare le *entità* per codificare queste informazioni; può essere direttamente gestito in database XML nativi, oppure trasformato e gestito in database relazionali o key-value; è uno standard de-facto (una raccomandazione W3C), a basso costo, e gli strumenti di sviluppo basilari sono presenti su quasi tutte le piattaforme hw/sw.

  Esiste un linguaggio per la trasformazione di documenti XML, XSLT , ed uno per l'interrogazione di documenti, XQUERY. Entrambi sono applicazioni XML — ovvero sono linguaggi di marcatura tipo XML, ma la semantica è predefinita. In breve, XML è ampiamente usato per lo scambio di dati specialmente se in forma narrativa.

Informatica umanistica

- un documento XML difficilmente ha una buona *presentazione* — in pratica il suo contenuto semantico è poco leggibile. Normalmente viene transformato in un formato tipografico, tipicamente HTML (un altro linguaggio a marcatori, ma con una semantica predefinita) o PDF. Il formato PDF è quello che garantisce una maggiore flessibilità ma il divario tra i due formati tende a ridursi: ad esempio il PDF utilizza la migliore tecnologia per quanto riguarda i font — OpenType — ma WOFF è una raccomandazione W3C che offre le stesse caratteristiche di OpenType. Le differenze diventano percettibili nell'aspetto di impaginazione e sicuramente fondamentali nella stampa su carta, dove la tecnologia PDF/POSTSCRIPT ha un ruolo quasi monopolistico. Infine, PDF offre diversi sotto-formati per l'archiviazione digitale che sono standard ISO mentre le raccomandazioni W3C per HTML non sempre hanno una implementazione completa ed uniforme.

Il problema quindi è trovare una applicazione XML — i.e. stabilire una semantica di un linguaggio XML adeguata per questo problema.

TEI — Text Encoding Initiative

“La Text Encoding Initiative (TEI) è un consorzio di istituzioni internazionali, di ambito linguistico e letterario, che ha sviluppato uno standard per la rappresentazione dei testi in forma digitale. La missione della TEI è quella di sviluppare e mantenere una serie di linee guida di alta qualità per la codifica di testi umanistici e per sostenere il loro uso da parte di comunità di progetti, istituzioni e singoli individui. Il consorzio ha sede presso l’Institute for Advanced Technology in the Humanities,... University of Virginia.

:

Attraverso le cosiddette Guidelines for Electronic Text Encoding and Interchange, la TEI definisce un linguaggio di markup (in XML) per la digitalizzazione dei testi, utile in particolar modo per coloro che intendono costituire archivi e banche dati testuali.

:

Oggi il TEI è riconosciuto come uno strumento di fondamentale importanza al livello internazionale, sia per la conservazione a lungo termine dei dati elettronici, che in altri ambiti disciplinari. È lo schema di codifica scelto per la creazione di documenti come critiche, testi scientifici e letterari, e per la gestione e produzione di metadati dettagliati associati a testi elettronici.”

http://it.wikipedia.org/wiki/Text_Encoding_Initiative

TEI — Text Encoding Initiative

TEI offre anche un set di fogli di stile XSLT sviluppati da Sebastian Rahtz (tra i fondatori della TEXLive) per trasformare documenti XML-TEI in HTML e LATEX (e non solo). La versione dei fogli di stile è XSLT 2.0 attualmente solo il programma SAXON (JAVA) gestisce questa versione (esiste una versione free).

Il sito `http://www.tei-c.org/release/doc/tei-xsl/##intro` consiglia inoltre l'editor XML oXygen, un programma proprietario non gratuito ma con prezzi particolari per università. Integra un processore XSLT 2.0 e facilita notevolmente la gestione di documenti XML.

⟶ In questo ambito i fogli di stile XSLT 2.0 di TEI e l'editor oXygen sono piuttosti diffusi.

È quindi possibile descrivere un *flusso di lavoro* (*workflow*) per la digitalizzazione di un testo letterario.

LaTeX workflow

- il testo viene tradotto in un documento XML-TEI ⟶ UNICODE, entità XML per la codifica degli aspetti tipografici ;
- tramite un foglio di stile XSL ed processore XSLT si trasforma il documento in LaTeX⟶ XSTL 2.0 vs XSLT 1.0 (più diffuso, con processori completamente gratuiti), XSL TEI vs sfogli di stili ad hoc;
- il documento LaTeX viene trasformato in PDF da un motore TeX⟶ pdfLaTeX vs XeLaTeX; disponibilità di font OpenType adeguati; necessità di scrivere un foglio di stile in LaTeX specifico per il testo; PDF/A per l'archiviazione digitale.

Esempio: workflow con foglio di stile XSL TEI, processore SAXON, editor oXygen, XeLaTeX. PDF conforme a PDF/A-1b (cfr. http://www.guitex.org/home/it/guide-per-iniziare/la-guida-guit)
Pro: utilizzo di strumenti diffusi in ambito umanistico, processore XeTeX adeguato.
Contro: strumenti proprietari, XeTeX meno diffuso di pdfTeX.

ConTEXt-MKIV workflow

ConTEXt-MKIV ha un approccio alternativo:

- il testo viene tradotto in un documento XML-TEI ⟶ UNICODE, entità XML per la codifica degli aspetti tipografici (esattamente come LATEX);
- il formato include un processore (scritto in Lua) simile ad processore XSLT: ogni elemento XML viene mappato in una macro TEX e quindi in oggetti PDF — in pratica è come se l'input fosse XML. E' possibile selezionare il formato PDF/A-1b oppure PDF/A-1a.

Pro: nessuna dipendenza da strumenti esterni.
Contro: il processore XML *non* è conforme alla raccomandazione XSLT; diffusione limitata del formato ConTEXt.

ConTEXt-MKIV workflow

Vediamo brevemente un esempio. Cominciamo con un frammento di XML TEI (l' opera of W. Shakespeare *Romeo and Juliet* cfr. http://www.perseus.tufts.edu/hopper/text?doc=Perseus:text:1999.03.0053)

```
<?xml version="1.0" encoding="utf-8"?>
<TEI>
  <teiHeader type="text" >
:
 </teiHeader>
<text xml:lang="en">
 <body>
:
<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>
:
 </body>
</text>
</TEI>
```

ConTEXt-MKIV workflow

Associazione degli elementi XML a macro TEX (1/3):

```
\startxmlsetups xml:tei:Perseus:text:1999:03:0053-setups
      \xmlsetsetup{#1}{*}{xml:tei:*}
\stopxmlsetups
\xmlregistersetup{xml:tei:Perseus:text:1999:03:0053-setups}
% TEI Header
\startxmlsetups xml:tei:teiHeader
  \xmlfunction{#1}{tei:teiHeader}
\stopxmlsetups
\startxmlsetups xml:tei:fileDesc
  \xmlfunction{#1}{tei:fileDesc}
\stopxmlsetups
```

ConTEXt-MKIV workflow

Associazione degli elementi XML a macro TEX (2/3):

```
xml_func["tei:TEI"]  = TEI
xml_func["tei:teiHeader"] = header
xml_func["tei:fileDesc"]  = filedesc
xml_func["tei:titleStmt"] = titlestmt
xml_func["tei:title"]     = title
xml_func["tei:author"]    = author
xml_func["tei:editor"]    = editor
xml_func["tei:sponsor"]   = sponsor
xml_func["tei:principal"] = principal
xml_func["tei:respStmt"]  = respStmt
xml_func["tei:resp"]      = resp
xml_func["tei:name"]      = name
xml_func["tei:funder"]    = funder
```

ConTEXt-MKIV workflow

Associazione degli elementi XML a macro TEX (3/3):

```
local function TEI(t)
   local att = {[’elementname’]=’TEI’}
   context.startelement({"document"})
   context.setupelementuserproperties({’document’},att)
   lxml.flush(t)
   context.stopelement()
end

local function header(t)
   local att = {[’elementname’]=’teiHeader’}
   att.type=t.at.type
   context.startelement({"metadata"})
   context.setupelementuserproperties({’metadata’}, att)
   context([[{\sc]])
   lxml.flush(t)
   context([[}]])
   context.blank()
   context.stopelement()
end
```

ConTEXt-MKIV workflow

Risultato:

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.

ConTEXt-MKIV workflow

I workflow visti fino ad ora sono pratiche consolidate per gli utenti dei rispettivi formati.
La parte originale sviluppata in questo talk parte dalle seguenti considerazioni:

- l'opera originale è ora suddisa in struttura (XML) e presentazione (PDF) e tipicamente i due documenti sono divisi anche se dipendenti — è quindi possibile ad esempio cambiare la struttura ma non aggiornare la presentazione con conseguente perdita di coerenza;
- il formato PDF/A-1a consente di memorizzare la struttura del documento TEX sorgente.

È possibile immergere il documento XML nel PDF, in modo da

- mantenere inalterata la presentazione;
- estrarre l'XML dal PDF;
- stabilire una corrispondenza tra elemento XML ed oggetti nel PDF;
- conservare la validità del formato PDF/A-1a.

?

Tagged PDF

Tagged PDf indica un PDF che ha al suo interno informazioni sulla struttura logica del documento originale. Le specifiche del PDF descrivono:

- oggetti per marcare parti di testo (le foglie) con marcatori definiti dall'utente:

  ```
  /sectiontitle <</MCID 0>>BDC
  BT
  /F1 17.21541 Tf 1 0 0 1 353.324 528.9996 Tm
  [<002F0060001C004B001C>30<006900420062006800
  5400320060006200510004D>25<00A4>]TJ
  ET
  EMC
  ```

- oggetti che costituiscono i nodi interni (e la radice) della struttura, anche in queesto caso con nome definito dall'utente:

  ```
  19 0 obj
  << /Type /StructElem /K 18 0 R /S /document /P 17 0 R /A << /P [ << /N (elementname) /V (TEI) >> ]
  /O /UserProperties >> /Pg 15 0 R >>
  endobj
  ```

Tuttavia queste non sono sufficienti per un Tagged PDF.

Tagged PDF

Adobe impone che i nomi dei tag siano direttamente o indirettamente riconducibili ad un insieme prefissato di nomi — e questi non corrispondono ai nomi degli elementi XML TEI. Ad esempio abbiamo nomi come `Span`, `Div`, `Code` …utili per una generica indicazione di presentazione; ConTEXt ha un suo insieme e stabilisce una corrispondenza con i nomi Adobe tramite una `RoleMap`:

```
13 0 obj
<< /Type /StructTreeRoot /ParentTree 14 0 R /K 15956 0 R /RoleMap << /mid /Span /sectioncontent /Div
/p /P /delimited /Quote /item /LI /description /Div /document /Div /mo /Span /sectiontitle /H /math /Div
/section /Sect /label /Span /metavariable /Span /itemgroup /L /division /Div /mrow /Span /mi /Span /ignore
/Span /metadata /Div /mtext /Span /line /Code /mright /Span >> >>
endobj
```

Ad esempio `division` è mappato in `Div`, come pure `description` e `document`. A parte il problema della sovrapposizione, anche in questo caso i nomi non coincidono con XML TEI.

La soluzione trovata è quella di usare un oggetto pdf, `/UserProperties`, che permette di memorizzare un numero arbitrario di coppie (chiave,valore) associate ad un nome visto sopra. Una chiave particolare, `elementname`, identificherà il nome dell'elemento, le altre sono del tipo (nome-attributo, valore-attributo).

Tagged PDF

Vediamo un esempio: si è deciso che l'elemento `<teiHeader type="text">` debba abilitare lo smallcap; non contiene foglie testo, solo nodi intermedi.
Viene creato un elemento ConTEXt `metadata` che corrisponde al tag Adobe `Div` ed ad esso viene associato una `/UserProperties (elementname, teiHeader), (type,tex)`

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

Nel PDF avremo:

```
21 0 obj
<< /Type /StructElem /K 20 0 R /S /metadata /P 19 0 R /A << /P [ << /N (type) /V (text) >> << /N
(elementname) /V (teiHeader) >> ] /O /UserProperties >> /Pg 15 0 R >>
endobj
```

Tagged PDF

Il passo successivo è uno script lua che legga il PDF ed estragga le informazioni relative alla struttura. Ovviamente si richiede una conoscenza piuttosto approfondita del formato PDF, ma ConTEXt ha un set di librerie adatte per questo scopo, quindi anche in questo caso non è necessario alcuno strumento esterno.

Tagged PDF

XML originale (frammento)

```
<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>
```

Tagged PDF

$> mtxrun –script query-taggedpdf.lua Perseus_text_1999.03.0053-2.6.0.pdf

Tagged PDF

Risultato

```
<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 n="org:DLI2" >NSF, NEH: Digital Libraries Initiative, Phase 2</funder>
<respStmt xml:id="CEW.ed" >
<name>CEW</name>
<resp>ed.</resp></respStmt></titleStmt>
<publicationStmt>
```

Tagged PDF

A questo punto è anche possibile associare la posizione iniziale e finale dell'elemento nel PDF:

```
local function header(t)
  local att = {['elementname']='teiHeader'}
  local uuid = os.uuid()
  local uuid = os.uuid()
  local tag_pos = uuid
  att.type=t.at.type
  context.startelement({"metadata"},att)
  att['ctx:begin_page'] = document.lscarso.MPp("b:"..tag_pos)
  att['ctx:begin_x']    = document.lscarso.MPx("b:"..tag_pos)
  att['ctx:begin_y']    = document.lscarso.MPy("b:"..tag_pos)
  att['ctx:end_page'] = document.lscarso.MPp("e:"..tag_pos)
  att['ctx:end_x']    = document.lscarso.MPx("e:"..tag_pos)
  att['ctx:end_y']    = document.lscarso.MPy("e:"..tag_pos)
  att['ctx:id']    = uuid
  context.setupelementuserproperties({'metadata'}, att)
  context.bpos(tag_pos)
  context([[\input knuth\relax\page\input knuth\relax]])
  context.epos(tag_pos)
  context.stopelement()
end
```

Tagged PDF

<teiHeader ctx:end_y="201.97835" ctx:begin_page="1"
ctx:begin_y="247.78565" ctx:end_x="143.08665" ctx:begin_x="25.00030"
ctx:end_page="2" ctx:id="bb0d9c12-4817-9992-8d97-bddd52747aa9" type="text">
Thus,
I came to the conclusion that the designer of a new system must not
only be the implementer and first large–scale user; the designer
should also write the first user manual.The separation of any of these
four components would have hurt TEX significantly. If I had not
participated fully in all these activities, literally hundreds of
improvements would never have been made, because I would never have
thought of them or perceived why they were important.But a system
cannot be successful if it is too strongly influenced by a single
person. Once the initial design is complete and fairly robust, the
real test begins as people with many different viewpoints undertake
their own experiments.Thus, I came to the conclusion that the designer
of a new system must not only be the implementer and first
large–scale user; the designer should also write the first user
manual.The separation of any of these four components would have hurt
TEX significantly. If I had not participated fully in all these
activities, literally hundreds of improvements would never have been
made, because I would never have thought of them or perceived why they
were important.But a system cannot be successful if it is too strongly
influenced by a single person. Once the initial design is complete and
fairly robust, the real test begins as people with many different
viewpoints undertake their own experiments.</teiHeader>

Conclusioni

- non tutti gli elementi XML sono mappati (ad esempio comment, processing instruction); la gestione delle entità è da verificare, come pure la gestione degli spazi;
- verifica del correttezza del formato PDF/A-1a;
- l'estrazione dell' XML immerso può essere onerosa per documenti lunghi e complessi; la compressione usuale degli oggetti PDF abbassa ulteriormente la performance;
- Tagged PDF *non* è equivalente ad XML, sebbene gli assomigli. Il suo scopo è evidenziare la struttura ma mantenere il contenuto del PDF. Questo implica che il contenuto di un nodo Tagged PDF debba essere analizzato per essere eventualmente converito in un contenuto XML valido. Questo incide sulle performance dell'estrazione.
- ConTEXt ha introdotto stabilmente il supporto per le `/UserProperties` a partire da Settembre 2014, ma la versione di LuaTEX utilizzata ha richiesto alcune patch e quindi non è quella della TEXLive 2014 — sarà però presente nella prossima TEXLive.

That’s all !
Thank you Folks !