La produzione di una rivista: ArsTEXnica

Massimiliano Dominici

Sommario

Nell'ultimo anno è stata intrapresa la riorganizzazione della rivista ArsTEXnica. Dopo la ristrutturazione dei compiti della redazione, è la volta degli strumenti di produzione ad essere oggetto di revisione.

Abstract

During the past year ArsTEXnica has gone through a process of reorganization. After the structure of editorial board it's now the turn of the build process to be revised.

1 Introduzione

Nell'ultimo numero di ArsTEXnica è stato annunciato un programma di rinnovamento della rivista (Beccari *et al.*, 2014) e ne sono state indicate le tappe:

1. ristrutturazione della redazione (già completata al momento dell'annuncio);
2. riscrittura degli strumenti per la produzione della rivista;
3. rivisitazione della veste grafica.

In questo articolo ci occuperemo del secondo punto che, benché ancora in fase di sperimentazione, può dirsi sostanzialmente compiuto. Il terzo punto è ancora in fase di elaborazione.

Nell'articolo citato venivano indicati i problemi del sistema di produzione fino allora impiegato e individuate alcune linee guida per ovviare ad essi. È opportuno riportare qui di seguito una sintesi di quell'esposizione.

1.1 Portabilità

Il primo obbiettivo che la redazione si è posta nel ridisegno del sistema di produzione della rivista è stato il massimo grado di portabilità e la facilità di installazione, possibilmente senza dover fare ricorso a niente altro che alla propria distribuzione di TEX.

Il vecchio sistema si basava su alcuni script per la shell `bash`. Era stato utilizzato su sistemi GNU/Linux e probabilmente era in grado di funzionare su sistemi operativi MacOS X e Windows in ambiente Cygwin. Un pregio di questo sistema consisteva nel fatto che, tranne per il programma `barcode` necessario per generare il codice a barre sulla quarta di copertina, le dipendenze erano ridotte alla presenza di `bash` e delle utilità di base dei sistemi Unix. Difetto principale: la necessità di installare l'ambiente Cygwin su Windows.

Nelle linee guida si affermava la necessità di sostituire agli script `bash` l'uso di un linguaggio di programmazione vero e proprio, con il duplice vantaggio di avere un sistema funzionante su ogni sistema operativo e un ambiente di programmazione più confortevole. I candidati erano stati individuati in Lua, Python, Ruby. Alla fine la scelta è caduta su Lua per le considerazioni fatte nel primo capoverso di questo paragrafo: una distribuzione TEX, infatti, contiene già un interprete Lua sotto forma del programma `texlua`. Rimane `barcode` come dipendenza esterna ed è quindi allo studio la possibilità di sostituirlo con il pacchetto pst-barcode.

1.2 Efficienza

Il vecchio script non usava un vero e proprio sistema di automatizzazione dello sviluppo. L'intero script veniva eseguito per intero ogni volta che veniva lanciato, ricompilando anche gli oggetti che non erano stati modificati. Nel caso dei singoli articoli la compilazione seguiva l'intero ciclo (`pdflatex`, `bibtex`, `pdflatex` di nuovo, due volte).

La soluzione individuata comportava l'uso di un sistema di automatizzazione dello sviluppo (scritto nello stesso linguaggio di programmazione dello script principale, ovvero `lake` per Lua, SCons per Python o `rake` per Ruby) abbinato a `latexmk` come programma di automazione dedicato per LATEX.

Successive sperimentazioni hanno condotto l'autore alla considerazione che `latexmk` era sufficiente agli scopi, senza ricorrere a un sistema di automazione generale. Infatti molti dei compiti che nello script originale venivano eseguiti ogni volta, potevano o essere eseguiti una sola volta in fase di inizializzazione della cartella di lavoro o essere separati e selezionati con una precisa opzione da passare allo script di compilazione. Questa decisione, inoltre, prosegue nella direzione di ridurre al minimo le dipendenze esterne al sistema TEX già installato.

1.3 Gestione dei conflitti

Nel vecchio sistema il corpo dei singoli articoli era incluso nel file principale (dopo che erano stati inclusi separatmente i pacchetti caricati e i comandi definiti dall'autore) e compilato all'interno di questo. Per quanto fosse stata posta una particolare attenzione nel tentare di gestire i possibili conflitti

tra articoli, nell'esperienza dei due incaricati della produzione che si sono susseguiti finora (Gianluca Pignalberi e l'autore di questo articolo), non c'è stata una singola uscita in cui non si presentassero incompatibilità, a volte evidenti fino al punto di bloccare la compilazione, a volte più sottili e subdole, introducendo errori di composizione non sempre immediati da individuare. La risoluzione di tali conflitti è stata spesso laboriosa, sempre fastidiosa e costosa in termini di tempo speso nella fase di debug. Nei casi più insidiosi, tale risoluzione non c'è proprio stata, perché l'errore è stato scoperto troppo tardi.

È stato deciso che la soluzione migliore sarebbe stata compilare i singoli articoli separatamente e includerli in un secondo tempo come PDF. Questa soluzione, ispirata da un articolo di Thierry Bouche su TUGboat (Bouche, 2006), era già stata presa in considerazione anni fa e poi scartata perché nel processo si perdevano irrimediabilmente i collegamenti ipertestuali. Fortunatamente nel frattempo è comparso il pacchetto pax (Oberdiek, 2012) che permette, tramite uno script Perl (pdfannotextractor), di salvare e reinserire i collegamenti ipertestuali.

Se da una parte l'uso di pax semplifica notevolmente la gestione degli articoli, dall'altra introduce un paio di problemi. Innanzitutto pax funziona solo con pdflatex (o, usando qualche accorgimento nel preambolo, con lualatex) ma non con xelatex. Questa limitazione, però, è valida solo per il file che deve includere i PDF, non per i singoli PDF da includere, che possono essere prodotti con qualsiasi programma (anche non appartenente al sistema TEX), quindi di fatto non ha alcuna incidenza nel nostro caso. Il secondo inconveniente è che vengono introdotte alcune dipendenze: Perl, Java e una precisa versione della libreria PDFBox. Inoltre potrebbe essere necessario attuare qualche aggiustamento per l'uso su Windows.

Compilare autonomamente i singoli articoli e poi includerli come PDF comporta vantaggi e svantaggi. Ogni problema di compilazione derivante da conflitti nell'uso di pacchetti o nella (ri)definizione di comandi è sostanzialmente eliminato. E questo è un vantaggio. Tuttavia, poiché l'autore virtualmente è libero di fare ciò che vuole senza che questo incida nella compilazione, c'è il rischio, che comunque si correva anche in passato, di introdurre incongruenze stilistiche. Sarà l'esperienza a decidere sull'opportunità di introdurre eventualmente delle "norme editoriali" a cui autori, redattori e revisori di bozze possano fare riferimento.

2 Descrizione del processo di produzione

Come premessa a questo paragrafo va fatto notare che il sistema è ancora in fase di sviluppo e di sperimentazione, sebbene possa reputarsi abbastanza stabile nella sua architettura. Il funzionamento e il nome di alcune macro e funzioni, però, potrebbe essere soggetto a modifiche, in particolare per quanto riguarda la gestione dei metadati.

Nel paragarafo verranno esaminati solo i punti salienti del processo, senza dare una descrizione dettagliata del funzionamento generale della classe.

Il nuovo sistema di produzione è composto da due script lua (AT_new_issue.lua e AT_build.lua), la classe arstexnica e una serie di template per il file master e per gli elementi fissi della rivista: pagine di copertina, colophon, eventualmente altro materiale, se con il tempo si renderà necessario.

2.1 Preparazione di un nuovo numero

Ogni nuova uscita viene inizializzata tramite lo script AT_new_issue.lua (riportato nelle figure 1 e 2), corrispondente grosso modo al vecchio AT_nuovo_numero. A differenza del vecchio script, che leggeva i dati relativi al nuovo numero a partire da un file di configurazione, al nuovo script tali dati devono essere passati tramite opzioni in fase di compilazione: questo garantisce che il sistema funzioni anche in assenza del file di configurazione. Tuttavia è in fase di aggiunta anche la vecchia funzionalità, molto comoda ogni volta che si vuole semplicemente inizializzare il numero successivo all'ultimo compilato.

Un tipico uso dello script è il seguente:

```
texlua AT_new_issue --year 2014 --issue 18
```

oppure, usando l'alternativa breve per i nomi delle opzioni:

```
texlua AT_new_issue -y 2014 -i 18
```

Le opzioni sono dichiarate tramite la libreria Lua alt_getopt, distribuita con TEX Live.

Lo script compie le seguenti operazioni:

1. crea una cartella usando il numero della rivista come parte del nome (nel nostro esempio Numero_18);
2. crea una struttura all'interno di questa cartella, con sottocartelle preposte a contenere gli elementi del numero da compilare (articoli, materiale aggiuntivo, sommari, voci bibliografiche, PDF compilati);
3. copia nella struttura così ottenuta i template del file principale e delle varie pagine ausiliarie (copertina e colophon);
4. copia un albero TDS con i file latex/bibtex necessari per la compilazione della rivista[1];

1. A rigor di logica non sarebbe necessario, perché tutti i numeri potrebbero attingere a un unico albero centrale, ma non ho ancora deciso se è meglio questa soluzione o assicurarsi che per ogni uscita sia presente la precisa configu-

```
kpse.set_program_name('luatex')

require "alt_getopt"

-- Source (adapted):
-- http://kracekumar.com/post/53685731325/cp-command-implementation-and-benchmark-in-python-go
function cp(source, dest)
    -- body
   lfs.mkdir(dest)
   for filename in lfs.dir(source) do
      if filename ~= '.' and filename ~= '..' then
 local source_path = source .. '/' .. filename
 local attr = lfs.attributes(source_path)
 -- print(attr.mode, path)
 if type(attr) == "table" and attr.mode == "directory" then
    local dest_path = dest .. "/" .. filename
    -- print(dest_path)
    lfs.mkdir(dest_path)
    -- print(lfs.mkdir(dest_path))
    cp(source_path, dest_path)
 else
    local f = io.open(source_path, "rb")
    local content = f:read("*all")
    f:close()
    local w = io.open(dest .. '/' .. filename, "wb")
    -- print(dest .. '/' .. filename, w, f)
    w:write(content)
    w:close()
 end
      end
   end
end

local long_opts = {
   year = "y",
   issue = "i",
}

local short_opts = "y:i:"

local anno_uscita, mese_uscita, numero_uscita

local texmf = "texmf-ars"

optarg,optind = alt_getopt.get_opts (arg, short_opts, long_opts)
for k,v in pairs (optarg) do
   if k == "y" then
      anno_uscita = v
   end
   if k == "i" then
      numero_uscita = v
   end
end

if tonumber(numero_uscita) % 2 == 0 then
   mese_uscita = "Ottobre"
else
   mese_uscita = "Aprile"
end

local nuova_uscita = "Numero_" .. numero_uscita
local config_file = nuova_uscita .. "/arstexnica.cfg"
```

FIGURA 1: `AT_new_issue`, lo script che prepara un nuovo numero (prima parte).

```
local issn_online = 18282369
local issn_stampa = 18282350

local function genera_ean(issn)
   local s1 = string.sub(issn, 1, -2) .. "00"
   local s2 = string.sub(anno_uscita, -1) .. string.format("%04d", numero_uscita)
   local ean = "977" .. s1 .. " " .. s2
   return ean
end

lfs.mkdir(nuova_uscita)
lfs.mkdir(nuova_uscita .. "/articoli")
lfs.mkdir(nuova_uscita .. "/output")
lfs.mkdir(nuova_uscita .. "/output/pdf")
lfs.mkdir(nuova_uscita .. "/output/bib")
lfs.mkdir(nuova_uscita .. "/output/abstracts")
cp("repo/" .. texmf, nuova_uscita .. "/" .. texmf)
local cfg = io.open(config_file, 'w')
cfg:write([[\ATsetup{]] .. "\n")
cfg:write("  year = " .. anno_uscita .. ",\n")
cfg:write("  month = " .. mese_uscita .. ",\n")
cfg:write("  number = " .. numero_uscita .. ",\n")
cfg:write("  mode = online,\n")
cfg:write([[}]])
cfg:close()

print(genera_ean(issn_online))
print(genera_ean(issn_stampa))

os.execute("barcode -E -e ean -u cm -g 4x1.5 -b '" .. genera_ean(issn_online)
           .. "' -o " ..nuova_uscita .. "/issn_online.eps")
os.execute("barcode -E -e ean -u cm -g 4x1.5 -b '" .. genera_ean(issn_stampa)
           .. "' -o " ..nuova_uscita .. "/issn_print.eps")
```

Figura 2: `AT_new_issue`, lo script che prepara un nuovo numero (seconda parte).

5. scrive in un file di configurazione i dati del numero da compilare (anno, mese, numero e modalità di compilazione online/stampa);

6. genera il codice a barre della versione online e di quella a stampa e le copia nella cartella di lavoro.

2.2 La compilazione

La compilazione avviene tramite lo script Lua `AT_build.lua`. Lo script, oltre a impostare il percorso in cui il compilatore potrà trovare i file della classe, consente di scegliere tra la compilazione della versione online e di quella a stampa.

Il responsabile dell'impaginazione dovrà collocare, man mano che gli arrivano, gli articoli nell'apposita cartella (fantasiosamente denominata `articoli`), ciascuno nella propria sottocartella. Questi verranno poi inclusi nel file principale tramite il comando `\IncludeArticle` che prende come unico argomento il percorso del file principale dell'articolo (senza estensione) relativo alla cartella `articoli`. Per esempio:

```
\IncludeArticle{Editoriale/editoriale18}
```

Le pagine ausiliarie vanno incluse con il comando `\IncludeSuppl`, sostanzialmente una copia del precedente che esegue qualche compito in meno, ma in compenso permette di specificare, tramite un argomento opzionale, una particolare modalità di compilazione:

```
\IncludeSuppl[prima]{copertina/prima}
```

dove l'argomento opzionale indica che la pagina in questione deve essere compilata in modalità "prima di copertina", attivando quindi una serie di funzionalità che di solito sono inibite e viceversa.

Queste due macro fanno internamente entrambe riferimento alla macro `\AT@build@pdf` che rappresenta il fulcro del sistema di compilazione. Quando, infatti, il compilatore raggiunge questa istruzione, se lanciato con l'opzione `-shell-escape`, entra nella modalità di esecuzione di comandi esterni, si porta nella cartella del file da compilare, lancia `latexmk` e immediatamente dopo

razione con cui è stata compilata. In realtà, per assicurarsi di una cosa del genere sarebbe necessario disporre della precisa versione di tutti i pacchetti usati in quella determinata occasione.

```
kpse.set_program_name('luatex')

local loader_file = "luatexbase.loader.lua"
local loader_path = assert(kpse.find_file(loader_file, "lua"),
                           "File '"..loader_file.."' not found")
require(loader_path)
require "alt_getopt"

local long_opts = {
   mode = "m",
}

local short_opts = "m:"
local prev_mode = ""

local cfg = io.open("arstexnica.cfg", "r")
local lines = {}
for line in cfg:lines() do
   if(string.find(line, "mode")) then
      if (string.find(line, "online")) then
 prev_mode = "online"
      else
 prev_mode = "print"
      end
      break
   end
end

local mode = prev_mode

optarg,optind = alt_getopt.get_opts (arg, short_opts, long_opts)
for k,v in pairs (optarg) do
   if k == "m" then
      mode = v
   end
end

if prev_mode ~= mode then
   local cfg = io.open("arstexnica.cfg", "r")
   local lines = {}
   local restOfFile
   for line in cfg:lines() do
      if(string.find(line, prev_mode)) then
 lines[#lines + 1] = string.gsub(line, prev_mode, mode)
 restOfFile = cfg:read("*a")
 break
      else
 lines[#lines + 1] = line
      end
   end
   cfg:close()

   local cfg = io.open("arstexnica.cfg", "w")
   for i, line in ipairs(lines) do
      cfg:write(line, "\n")
   end
   cfg:write(restOfFile)
   cfg:close()
end


os.execute("export TEXMFHOME=`pwd`/texmf-ars:`kpsewhich -var-value=TEXMFHOME`" ..
      " && echo $TEXMFHOME && " ..
      "pdflatex -shell-escape arstexnica.tex")
```

FIGURA 3: `AT_build`, lo script di compilazione.

```
\newcommand\AT@build@pdf[2][]{%
  \AT@divide@path#2|
  \AT@build@hook
  \immediate\write18{cd \AT@subdirectory/\AT@article@path &&
                     latexmk -pdf -pdflatex='pdflatex -shell-escape
                        '\%O' \string\\PassOptionsToClass{#1}{arstexnica}\string\\input{'\%S'}'
                     \AT@article@name &&
                     pdfannotextractor \AT@article@name.pdf}%
}
%
\newcommand\IncludeSuppl[2][]{%
  \def\AT@build@hook{}
  \def\AT@subdirectory{suppl}
  \AT@build@pdf[#1]{#2}
  \IfFileExists{\AT@subdirectory/#2.pdf}{%
    \includepdf[pages=-, pagecommand={},]
                {\AT@subdirectory/#2.pdf}
  }{\typeout{Articolo #2: Qualcosa \'e andato storto...}}
}
%
\newcommand\IncludeArticle[1]{%
  \def\AT@build@hook{
    \immediate\write\AT@matedata{["\AT@article@name"]=\AT@open@group}
    \immediate\write\AT@matedata{["startpage"]=\thepage,}
  }
  \def\AT@subdirectory{articoli}
  \AT@build@pdf[injournal]{#1}
  \xdef\@currentHref{title.\theHtitle}%
  \makeatletter
  \InputIfFileExists{\AT@subdirectory/#1.lst}{%
    \typeout{Carico articolo #1}%
  }{\typeout{Articolo #1: Qualcosa \'e andato storto...}}
  \makeatother
  \thispagestyle{plain}
  \IfFileExists{\AT@subdirectory/#1.pdf}{%
    \includepdf[pages=-, pagecommand={},
                addtotoc={1,ATtitle,-1,\AT@toc@header,ATtitle-\theATtitle}]
                {\AT@subdirectory/#1.pdf}
  }{\typeout{Articolo #1: Qualcosa \'e andato storto...}}
```

Figura 4: Le macro usate per la compilazione e l'inclusione dei singoli articoli.

`pdfannotextractor`. Al termine di queste operazioni `\IncludeArticle` o `\IncludeSuppl` includono il PDF ottenuto, se tutto è andato a buon fine, usando la macro `\includepdf` del pacchetto pdfpages (Matthias, 2013). Nel caso in cui sia necessario anche generare una voce per l'indice, questa viene specificata tramite l'opzione `addtotoc` di `\includepdf`.

È da notare che `latexmk` entra in funzione realmente solo se il file in questione è stato modificato (o un altro file da cui esso dipende, per esempio la classe stessa). Se non ci sono state modifiche avviene soltanto l'inclusione del PDF con un notevole risparmio di compilazioni. Il codice delle tre macro in questione è riportato nella figura 4.

Questo sistema però comporta un problema. Il PDF così compilato contiene le proprie testatine e i piè di pagina. Teoricamente sarebbe possibile comunicare al momento della compilazione il numero di pagina iniziale (e i dati per le testatine), ma questo vanificherebbe un po' l'uso di `latexmk` perché costringerebbe a ricompilare porzioni del documento che, a parte il numero di pagina, non hanno subito modifiche. Molto più semplice fare in modo che i singoli articoli siano compilati con testatine e piè di pagina in bianco e questi siano riapplicati poi dal documento generale. A questo scopo, in fase di compilazione del singolo articolo viene passata l'opzione injournal (che attiva anche il codice per scrivere i metadati) e poi, al momento dell'inclusione, viene disattivata l'opzione di `\includepdf` che inibisce la scrittura di testatina e piè di pagina. Ovvero riportiamo il controllo di questi due elementi al documento generale togliendolo a quello incluso.

2.3 Metadati

ArsTEXnica non è prodotta solo ed esclusivamente per la stampa, ma anche per la diffusione online tramite il sito web del Gruppo Utilizzatori Italiani

di TEX, dove compaiono i dati principali dei singoli articoli (titoli, autori, sommari) e un collegamento ai PDF di ciascun articolo e dell'intera rivista. Inoltre lo stesso sito distribuisce un database bibliografico degli articoli comparsi sulla rivista e periodicamente vengono inviati alla redazione di TUGboat i dati del contenuto della rivista per la pubblicazione dei sommari.

Tutto ciò fa sì che sia estremamente conveniente avere un sistema che, almeno parzialmente, estragga questi dati automaticamente. Il vecchio script lo faceva in maniera abbastanza efficiente, ma alcune cose andavano ripensate. In particolare sarebbe bene immagazzinare i metadati in un formato che ne permetta usi differenti. Per fare un esempio, con il vecchio script i nomi degli autori erano disponibili solo nel file `bibtex`, scritto direttamente a partire da istruzioni della classe arstexnica e non potevano essere usati direttamente per la creazione della pagina web.

Nel nuovo sistema, che può giovarsi di Lua come linguaggio di programmazione, è stato naturale organizzare questi dati in una tabella. Così sarà possibile scrivere funzioni ausiliarie che accedano a questa tabella e usino i dati per i loro scopi.

Le funzioni di scrittura dei metadati usano le primitive di TEX per la gestione di input e output (o le macro del nucleo di LATEX che su di esse si basano). Di solito questo non comporta grossi problemi, ma nel nostro caso c'è un ostacolo: una parte dei dati da scrivere non è immediatamente a disposizione del documento generale, ma solo dei singoli articoli che non comunicano direttamente con il processo di compilazione principale. La soluzione trovata è stata quella di fare in modo che i singoli articoli scrivano su un file esterno la propria porzione di dati; questo file è poi letto dal processo principale di compilazione e i dati sono inseriti nella tabella generale dei metadati.

2.4 Compiti ausiliari

Lo script di compilazione conterrà anche delle funzioni ausiliarie per eseguire compiti di manipolazione dei metadati (creazione delle voci bibliografiche, creazione di un file che contenga tutti i sommari pronti per la consegna al TUGboat, generazione di frammenti di HTML per il sito web) e riorganizzazione dei prodotti di compilazione (copia dei PDF in una apposita cartella). Queste funzioni non sono state ancora implementate, ma esistono già le fondamenta per poterlo fare in maniera abbastanza rapida e semplice.

3 Sviluppi futuri

La costruzione del nuovo sistema di produzione di ArsTEXnica è giunta a buon punto, ma non è completata, infatti il numero di ArsTEXnica che state leggendo è ancora prodotto con il vecchio sistema. Probabilmente sarà l'ultimo. In generale sul versante degli script Lua c'è ancora da lavorare.

- Alcune funzioni accessorie del vecchio script non sono state ancora reimplementate, altre andranno probabilmente riviste.
- Mancano ancora i dovuti controlli in caso di corruzione del file dei metadati.
- Deve essere possibile indicare a `latexmk` che la compilazione va effettuata anche se tutti i file sono aggiornati (flag `-g`) e possibilmente a livello di singolo articolo oltre che globalmente.

Sul versante della classe, invece, un elenco parziale dei compiti da eseguire è il seguente.

- Rendere personalizzabile la stringa che controlla la compilazione dei singoli articoli (permettendo ad esempio di cambiare il motore di composizione o le altre opzioni passate a `latexmk`).
- Ripulire e riorganizzare il codice, tenendo conto anche dell'obbiettivo finale di rendere disponibile su CTAN (e quindi eventualmente su TEX Live e MikTEX) almeno la porzione dedicata agli autori. Questo comporta che si rifletta su quale grado di modularità conferire al codice nello strutturare il `dtx`.

Nei prossimi mesi, quindi, oltre a completare le parti mancanti e a testare il sistema in maniera intensiva, si cercherà di individuare nuove funzionalità da aggiungere e soprattutto si comincerà a rivedere il layout della rivista, a partire dal formato. Una volta prese le debite decisioni su come organizzare il progetto, lo sviluppo del codice avverrà sul repository Github del GUIT: `https://github.com/GuITeX`.

Riferimenti bibliografici

Beccari, C., Dominici, M. e Pignalberi, G. (2014). «Ars si rinnova». *ArsTEXnica*, (17), pp. 12–17.

Bouche, T. (2006). «A pdfLATEX-based automated journal production system». *TUGboat*, **27** (1), pp. 45–50. Proceedings of EuroTEX 2006.

Matthias, A. (2013). *The pdfpages Package*. TUG. Leggibile con `texdoc pdfpages` nella distribuzione TEX Live.

Oberdiek, H. (2012). *README for project pax*. TUG. Leggibile con `texdoc pax` nella distribuzione TEX Live.

▷ Massimiliano Dominici
`mlgdominici at gmail dot com`