Un uso insolito di arara

Claudio Beccari, Paulo Roberto Massa Cereda

Sommario

Si presenta un uso insolito di arara che richiede la definizione di nuove regole con l'uso del linguaggio di espressione `mvel` per Java. L'uso insolito consiste nel chiedere ad arara di impostare certi parametri in linguaggio LaTeX per produrre documenti in formati diversi ma con lo stesso contenuto.

Abstract

We present an uncommon use of arara, that requires the definition of new rules implying the use of the expression language `mvel` for Java. The unusual application consists in requiring arara to set some LaTeX parameters in order to typeset documents in different formats but with the same contents.

Premessa

Va subito premesso che dei due autori, Beccari è quello cha ha avuto bisogno di usare arara in modo insolito. La parte di articolo che si riferisce a questo uso insolito riguarda Beccari.

Il secondo autore ha creato il programma Java arara. In un passato meeting del GUIT Massa Cereda (2015) ha presentato l'ultima versione di questo programma nella sua versione 4.0; però, purtroppo, non è ancora dotato della necessaria documentazione e quindi non fa ancora parte delle distribuzioni del sistema TeX, le quali contengono la versione 3.x adeguatamente documentata.

In questo articolo, quindi, si mostra un uso insolito di arara in versione 3.x. La futura versione 4.0 di arara, già disponibile su `git`, ed entro il prossimo giugno su CTAN, permetterà probabilmente di affrontare l'uso insolito descritto in questo articolo senza bisogno di ricorrere all'uso di `mvel`.

Il secondo autore ha promesso che nella primavera del 2018 metterà finalmente mano alla documentazione e caricherà la versione 4.0 su CTAN.

1 Il problema insolito da affrontare

Beccari scrive documentazione per l'uso di LaTeX mediante i tre programmi pdfLaTeX, XeLaTeX e LuaLaTeX e desidera poterli comporre in tre formati: A4, A5 e B5. Il primo è il formato standard che può essere usato in diversi modi: può venire letto a schermo oppure può essere stampato solo in bianca per poi mettere i fogli in un raccoglitore ad anelli beneficiando, nella volta di ogni foglio, di ampio spazio per le annotazioni. Il formato B5 è più comodo da leggere a schermo, specialmente se si tratta di quello di un laptop. Il formato A5 può essere letto a schermo, ma può essere anche impaginato in segnature dove i fogli A4 sono stampati in bianca e volta, con due pagine su ogni facciata. Se i fogli complessivamente non superano la quindicina, essi possono essere cuciti a quaderno con una pinzatrice "qualunque" e conservati in poco spazio, ma comodi da leggere senza fatica quando se ne avesse bisogno.

La differente larghezza del foglio nei tre formati richiede una impostazione diversa a seconda di quale si sta usando per la composizione, e richiede molta attenzione quando, come succede con le guide, ci sono molti brani di testo, specialmente quando è formato da codice, che devono venire impaginati diversamente. Sono quindi necessari degli switch (delle variabili booleane) che permettano di distinguere sia il formato della carta da specificare fra le opzioni dello statement `\documentclass`, sia le diverse impaginazioni che qua e là sono necessarie.

Per la manutenzione delle guide sopra menzionate è importante che ci sia un solo file da gestire e che le informazioni relative allo specifico file compaiano anche nel nome del file PDF di uscita, e in quelle che possono venire inserite nel frontespizio o comunque usate in altri punti del testo.

Inizialmente Beccari aveva creato tre diversi main file per ciascuno dei tre formati che impostavano il valore delle variabili booleane a cui si è accennato sopra. Tutti e tre importavano lo stesso file contenente il corpo del documento; questo non cambiava da un formato all'altro se non usando gli stati degli switch booleani per aggiustare le impaginazioni generalmente dei formati più piccoli. Ben presto si è accorto che questa operazione era piuttosto delicata e fonte di dimenticanze o di errori, quindi cercava un metodo per potersi liberare da questi problemi. Il risultato è stato il ricorso ad arara.

2 Qualche nozione elementare su arara

La documentazione di arara versione 3.x specifica che ogni esecuzione di arara opera su stringhe che contengono nomi di regole; queste a loro volta possono fare uso di parametri, ma a questi bisogna assegnare un valore prima di invocare la regola.

Le istruzioni per arara vanno messe nel file `.tex` da compilare né più né meno come le righe magiche (commenti di autoconfigurazione). Rispet-

to alle righe magiche, i comandi arara sono più performanti, perché non sarà uno dei tre motori di composizione che agirà sul file `.tex` direttamente, ma questo file sarà dato in pasto al programma arara, che lo leggerà tutto per estrarre i comandi arara da eseguire. Poi arara stesso lo darà in pasto ai vari programmi che ne faranno l'uso che i comandi arara specificano.

Una serie di comandi arara potrebbe, per esempio, eseguire tutto quello che fa anche lo script `latexmk`, ma secondo noi arara lo fa molto meglio; nel senso che durante la lavorazione di un documento un certo numero di comandi arara possono venire commentati (come si vedrà più avanti) in modo da limitare le compilazioni allo stretto necessario senza ripetere compilazioni non necessarie, per esempio senza rigenerare la bibliografia se non è necessario. Una delle applicazioni già presenti nella dotazione di regole preconfezionate, contributo di Gregorio (2013), è la gestione del frontespizio delle tesi facendo uso del pacchetto frontespizio.

Nella fattispecie del problema insolito descritto sopra, quello che si chiede ad arara sono i passi seguenti.

1. Ricevere dal comando il valore di un opportuno parametro per scegliere il formato della carta.
2. Generare le opportune istruzioni LaTeX da trasmettere al programma di composizione perché imposti i giusti valori degli switch booleani.
3. Lanciare la composizione con uno dei tre programmi citati all'inizio specificando le opportune opzioni.
4. Solo quando il file `.bib` della bibliografia è pronto, presumibilmente verso la fine della lavorazione del documento, si può decommentare la riga di arara che lancia l'elaboratore della bibliografia per estrarre i riferimenti citati e formattarli in linguaggio LaTeX per la loro composizione.
5. Dopo aver eseguito il passo precedente arara può lanciare il programma di composizione quelle due o tre volte necessarie perché il tutto finisca con tutti i riferimenti bibliografici corretti e che gli altri riferimenti incrociati siano tutti definiti e nel file `.log` non compaiano errori.
6. Alla fine, a lavorazione completata, si può attivare l'ultimo comando arara destinato a cambiare il nome del file di uscita PDF, in modo che contenga nel suo nome anche la sigla del formato della carta; in modo più chiaro, se il testo da comporre si chiama `guida.tex`, il nome `guida`, viene conservato nelle viscere del programma di compilazione come contenuto della macro `\jobname`, ma questo nome non può essere cambiato lavorando all'interno di LaTeX. Tocca all'utente, quindi ad arara, copiare il file `guida.pdf` ottenuto, per esempio, avendo specificato il formato B5, nel file `guida-B5.pdf`. È meglio copiare che non rinominare il file esistente, perché i collegamenti fra il file `.pdf` e il file `.tex` restano attivi fra le due finestre del programma di editing in modo da consentire sia la forward sia l'inverse search così da semplificare la vita dell'utente.

Evidentemente durante la lavorazione del documento non sarà ancora necessario comporre anche la bibliografia con le relative citazioni, per cui gli ultimi tre passi potranno essere indicati nel file sorgente `.tex` in modo che non vengano eseguiti; in questo modo, anche con documenti semplici, si risparmia molto tempo.

3 Impostare gli switcth logici per trasmetterli al file `.tex`

È poco noto che l'affermazione presente in quasi tutte le guide di LaTeX, secondo cui non si può scrivere nessun comando prima di `\documentclass`, è una raccomandazione e non un divieto assoluto. Tant'è che persino il manuale originale di Lamport (1994), il creatore di LaTeX, raccomanda di usare l'ambiente filecontents (con o senza asterisco) prima di `\documentclass`. Ecco: questo è il punto cruciale della procedura descritta in questo articolo. Basta iniziare il file sorgente `.tex` con queste righe:

```
\newif\ifacinque \acinquefalse
\newif\ifbcinque \bcinquefalse
\newif\ifaquattro \aquattrofalse
\input{\jobname.cfg}

\documentclass[\ifacinque
  a5paper\else\ifbcinque
    b5paper\else
      a4paper\fi\fi
  ,⟨altre opzioni⟩]{⟨classe⟩}
\ProvidesFile{guida.tex}[2018/02/08
  v.1.1.7 Guida breve]
```

Come si vede, prima di `\documentclass` compaiono alcune definizioni di switch booleani, impostati tutti tre al valore `false` e l'inserimento di un file di configurazione con lo stesso nome del main file, grazie all'uso di `\jobname`, nel quale si troverà l'informazione di quale switch, unico fra i tre definiti, viene posto uguale a `true`. Sarà compito di arara scrivere tale file con il valore specificato nell'apposito comando arara. Quelle quattro righe fanno tutta la "magia" necessaria per ottenere dal solo file `guida.tex` un file `guida.pdf` ed eventualmente, grazie ad arara, i tre file `guida-A4.pdf`, `guida-A5.pdf` e `guida-B5.pdf` composte su carta virtuale dei tre formati diversi.

Va però da sé che la geometria della pagina, e le strutture diverse da usare per i tre formati devono essere assoggettati agli stati dei tre switch non diversamente da come si è fatto per scegliere l'opzione di formato da inserire fra le opzioni del comando `\documentclass`.

4 I comandi arara

Quello che bisogna fare ora è mettere le righe magiche per arara, in teoria in qualunque parte del file sorgente, ma è decisamente meglio inserirle all'inizio, dopo le righe magiche di autoconfigurazione dell'editor. Queste righe magiche contengono i comandi arara con i loro argomenti; devono essere scritte ciascuna su una sola riga; qui le spezzeremo per la ristretta giustezza della colonna, ma si deve intendere che le righe che iniziano con un rientro siano la prosecuzione della riga precedente.

```
% !TEX TS-program = arara
% !TEX encoding = UTF-8 Unicode
% arara: writeconfig: { suffix: B5 }
% !arara: pdflatex
% !arara: bibtex
% !arara: pdflatex
% !arara: pdflatex
% arara: pdflatex: { synctex: true }
% arara: rename
```

Alcuni commenti sono necessari.

1. La prima riga non specifica come programma uno di quelli del sistema TEX, ma specifica di usare `arara`, come motore per gestire la composizione; in sostanza si dice all'editor che deve gestire il file da comporre con `arara`, non con pdfLATEX o XeLATEX o LuaLATEX. In questo modo `arara` legge l'intero file, ne estrae le informazioni che concernono i suoi comandi, e quando li ha letti tutti, li esegue nell'ordine in cui sono scritti.
2. La seconda riga, come al solito, dice all'editor di gestire il file sorgente con la codifica `utf8`, oggi quella da preferire sempre. Non riguarda i comandi arara, ma fa parte delle righe magiche di autoconfigurazione dell'editor, righe che non possono essere messe dovunque, ma solo fra le prime 20 del file.
3. Successivamente compaiono 6 righe di comandi arara; come si vede tutte cominciano con la parola chiave `arara`; se questa parola chiave è preceduta da un punto esclamativo, la riga viene totalmente ignorata dal programma `arara`. I punti esclamativi potranno essere tutti tolti quando si vorranno eseguire anche i comandi che si sono "commentati" agli effetti di arara. Si noti che l'ultimo contiene anche l'opzione per la "sincronizzazione" del file sorgente con il file composto, in modo che viene conservata la possibilità che l'editor possa eseguire la forward e inverse search.
4. I nomi delle regole sono autoesplicativi, ma le regole `writeconfig` e `rename` non corrispondono a nessuna fra le varie decine di regole già predefinite e descritte nella documentazione di `arara`; quindi bisogna definirle apposta. Si noti che mentre `rename` apparentemente non ha bisogno di argomenti, `writeconfig` deve sapere cosa scrivere nel file di configurazione, ed è ciò che appare dopo i due punti fra parentesi graffe; il qualificatore `suffix` servirà per i comandi arara affinché ne facciano l'uso appropriato; si vedrà più avanti meglio quando si descriveranno le due regole `writeconfig` e `rename`.
5. Non è immediatamente chiaro, invece, dove i codici di queste regole debbano venire memorizzati. La documentazione di arara lo spiega in modo abbastanza chiaro; esse vanno conservate una cartella residente nella propria "home"; questo concetto di "home" è chiarissimo con i sistemi operativi Linux (cartella indicata con `~/`) e per il sistema operativo Max OS X (cartella `~/Library/`); per le piattaforme Windows il concetto è chiaro solo per le versioni più recenti, dove per "home" si intende la cartella `C:\Users\`⟨*nome dell'utente*⟩; per le altre versioni si consulti bene la documentazione di arara. Chi scrive sta lavorando su un Mac, quindi ha creato la cartella `~/Library/myararararules`; egli ha messo anche un piccolo file con estensione `.yaml`, nella cartella `~/` dove si indica il path completo della cartella `myararararules`. Di questo si parlerà con maggiore dettaglio nel prossimo paragrafo
6. Va da sé che nel momento in cui `arara` legge il main file, prende nota automaticamente del suo nome e lo memorizza in una sua variabile interna.

Sembra complicato? No, è più lungo da scrivere che da fare.

5 Le regole per arara

Veniamo pertanto ai dettagli dei file per arara. Come si è detto, le regole vanno conservate in una cartella con qualunque nome, ma che sia individuabile con precisione da `arara`. Nella "home" `~/` si è messo il file

`araraconfig.yaml`

il cui contenuto è il seguente:

```
!config
paths:
- /Users/claudio/Library/myararararules
```

Il fatto che contenga la parola `paths` al plurale lascia intendere che si possono usare più percorsi per conservare i file delle regole; in generale è piuttosto prudente agire così perché si possano

```
1 identifier: writeconfig
2 name: WriteConfig
3 command: <arara> @{
4   config1 = new java.io.File(getBasename(file) + '.cfg');
5   config2 = new java.io.File('suffix.cfg');
6
7   dict = ['A4':'aquattro', 'A5':'acinque', 'B5':'bcinque' ];
8
9   text = '\\' + dict[suffix] + true;
10
11  java.nio.file.Files.write(config1.toPath(), text.getBytes());
12  java.nio.file.Files.write(config2.toPath(), suffix.getBytes());
13
14  return isWindows('cmd /c echo', 'true') }
15 arguments:
16 - identifier: suffix
17   flag: '@{parameters.suffix}'
18   default: 'A4'
```

Codice 1: Codice di `writeconfig.yaml`

```
1 identifier: rename
2 name: Rename
3 command: <arara> @{
4   config = new java.io.File('suffix.cfg');
5
6   suffix = new java.lang.String(java.nio.file.Files.readAllBytes(config.toPath()));
7   suffix = suffix.replaceAll('\n', '');
8
9   cp = isWindows('cmd /c copy', 'cp');
10  bn = getBasename(file);
11
12  return cp + ' "' + bn + '.pdf" "' + bn + "-" + suffix + '.pdf"' }
13 arguments: []
```

Codice 2: codice della regola `rename.yaml`

conservare regole diverse per far svolgere ad `arara` compiti diversi.

In accordo al contenuto di `araraconfig.yaml`, viene creata la cartella `myarararules` in cui verranno inserite le due regole seguenti: `writeconfig.yaml` e `rename.yaml`. Queste, scritte da Paulo Massa, presentano il contenuto descritto nelle due prossime sezioni.

5.1 La regola `writeconfig.yaml`

Il file `writeconfig.yaml` contiene quanto mostrato nel codice 1. Le parole chiave `identifier` e `name` servono rispettivamente per definire il nome della regola, e la stringa da scrivere durante la sua esecuzione; dalla riga 3 fino alla riga 14 si trova la definizione del comando da eseguire; le righe da 15 a 18 specificano gli argomenti e il valore di default di questi argomenti; in particolare, se al comando non si danno argomenti, si assume che si stia componendo il documento su carta in formato A4.

Si noti che vengono definiti due stream di scrittura; uno per scrivere il file di impostazione con lo stesso nome del file sorgente, che `arara` recupera mediante la sua funzione `getBasename(file)`, e l'atro per specificare il suffisso che l'altra regola `rename` userà per aggiungerlo al nome del file PDF finale.

Perciò agendo con `arara` sul file `guida.tex` con la serie di comandi non "commentati" indicati nel paragrafo 4, viene specificato il valore `B5` alla regola `writeconfig` e questa crea il file di configurazione `guida.cfg`, che contiene solo l'impostazione `\bcinquetrue`, e un secondo file `suffix.cfg` che contiene solo il suffisso `B5`; quando il documento finale `guida.pdf` verrà "rinominato" con la regola `rename` esso verrà in realtà copiato nel file `guida-B5.pdf`.

5.2 La regola `rename.yaml`

L'atra regola `rename.yaml`, infatti, contiene quanto riportato nel codice 2; si vede che sfrutta alcune delle cose impostate con la regola `writeconfig`. Infatti recupera il suffisso precedentemente memorizzato nel file `suffix.cfg` e lo usa per generare

il nome del file di destinazione; si vede, infatti che quanto fornito da questa regola è un comando per il sistema operativo (Mac o Linux)

```
cp guida.pdf guida-B5,pdf
```

tanto per continuare ad usare l'esempio descritto nei paragrafi precedenti.

Il codice 2 contiene un test per sapere se si sta componendo su una piattaforma Windows; in questo caso invece di `cp`, che va bene per Mac e Linux, la regola imposta correttamente il comando `copy`. Quindi la regola riportata nel codice 2 può essere usata indifferentemente su tutte le piattaforme più comuni.

6 Commenti

Nel caso specifico, che ha dato origine a questo uso insolito di **arara**, il modo di procedere è il seguente: si inseriscono le righe indicate nel paragrafo 4 in testa al main file del documento da comporre; durante la lavorazione si lasciano commentate le righe con i comandi **arara** preceduti dal punto esclamativo; quando il documento è pronto, si tolgono i punti esclamativi e si esegue l'intera compilazione compresa la bibliografia e le due successive compilazioni in modo da sistemare tutte le citazioni dei riferimenti bibliografici in modo corretto. Si ripetono le stesse operazioni con e senza i punti esclamativi con un diverso valore del formato della carta; nel fare questo si aggiustano in base ai valori degli switch booleani relativi al formato quelle parti dell'impaginazione che presentano dei difetti causati dalla minore giustezza della gabbia del testo. Volendo si potrebbero aggiungere ai comandi **arara** indicati nel paragrafo 4 gli stessi comandi con gli altri valori dei formati della carta, in modo che con un'unica compilazione complessiva del documento in tutti e tre i formati si ottengano i file PDF finali coerenti gli uni con gli altri, nonostante le differenti impaginazioni; chi scrive, tuttavia, preferisce agire su ciascun formato in modo indipendente e dopo ogni correzione del corpo del documento preferisce compilare separatamente i tre file richiesti controllandoli visualmente in ogni dettaglio per essere sicuro che i tre file siano tutti e tre impaginati correttamente.

L'esempio che si è presentato è molto particolare, ma getta una luce sulle infinite possibilità di uso di **arara** per altri casi particolari e insoliti non ancora previsti nella distribuzione del pacchetto **arara**. Fra pochi mesi la distribuzione di questo pacchetto potrebbe essere aggiornata alla versione 4.0, insieme alla sua documentazione; probabilmente per la scrittura di altre regole non sarà necessario ricorrere direttamente ai comandi Java usati nell'esempio riportato in questo articolo; per lo meno dovrebbe essere molto più semplice. Nel frattempo il lettore interessato può farsi venire il desiderio di approfondire **arara** leggendo l'articolo (Massa Cereda, 2015).

Riferimenti bibliografici

Gregorio, E. (2013). «Introduzione a **arara**». PDF document. URL `http://profs.scienze.univr.it/~gregorio/introarara.pdf`.

Lamport, L. (1994). *LaTeX. A Document Preparation System*. Addison-Wesley.

Massa Cereda, P. R. (2015). «Easy TeX automation with **arara**». *ArsTEXnica*, (20). URL `https://www.guitex.org/home/images/ArsTeXnica/AT020/cereda.pdf`.

▷ Claudio Beccari
`claudio dot beccari at gmail dot com`

▷ Paulo Roberto Massa Cereda
Università di San Paolo, Brasile
`paulo dot cereda at usp dot br`