xmLegesMarker

parser.c

- analizza i parametri di esecuzione, configura lo stato del marker (-> *config.c*)
- uniformazione sorgente (html -> *prehtml.lex*, doc -> *antiword*)
- inizializzazione struttura dom (libxml2)
- analisi annessi (-> *annessi.c*)
- post processing (setta attributi id, analizza nodi errore)
- serializzazione dom

annessi.c

- AnnessiAnalizza riceve il testo da analizzare (char *) e un puntatore all'interno del dom a cui agganciare il testo parsato (xmlNodePtr)
- conta gli annessi (principale, allegato1, allegato2, allegato3...)
- per ogni annesso, analizza la struttura (-> *struttura.c*)

struttura.c

- riceve il testo (char *) e il ruolo (principale, allegato)
- creazione dei nodi "esterni" (tipo e nome documento, meta, descrittori...)
- analisi del corpo dell'articolato (-> *articolato.lex*)
- analisi dell'header del testo, tutto ciò che precede l'inizio dell'articolato viene passato a *testa.c* e in seguito al modulo **HeaderParser**
- analisi delle note
- analisi del footer del testo, tutto ciò che è contenuto nell'ultima partizione formale dell'articolato (tipicamente l'ultimo comma) viene passato a *coda.c* e in seguito al modulo **HeaderParser**
- analizza la struttura dom alla ricerca di partizioni modificative ("virgolette") (-> *virgolette.c*)
- analizza la struttura dom aggiustando i nomi delle eventuali partizioni di tipo lista

articolato.lex

- rileva le partizioni formali del testo (capo, articolo, comma, lista...) tramite automa a stati finiti (realizzato con Flex)
- salva il testo nella struttura dom tramite le funzioni di *dom.c*
- controlla le sequenze nelle partizioni formati con l'ausilio di *sequenze.c*

HeaderParser

Crea degli oggetti HMM (vedi modulo HMM) tramite modelli preparati in precedenza. I modelli sono salvati su dei file di configurazione. Nella configurazione è presente anche un mapping degli stati dei vari modelli verso i tag della struttura dom.
Sfrutta l'algoritmo di Viterbi per la scelta del cammino all'interno dell'HMM.

HeaderParser.cpp

- *parseHeader()*- Rileva i pattern relativi all'header, sfrutta vari HMM (intestazione, formulainiziale...)

- *parseFooter()*- Separa l'ultimo comma dal footer, rileva i pattern relativi al footer sfruttando vari HMM (formulafinale, dataeluogo, sottoscrizioni...)

- *parseHeaderGetTipo()*- Cerca di individuare il tipo di documento in base all'intestazione, eseguita solo con la funzione “-t unknown” dell'xmLegesMarker, scrive il tipo di documento rilevato nel file “temp/unknown_type.tmp”, utile per facilitare l'utente di xmLeges nella scelta del tipo di documento da importare e della DTD a cui associarlo. Fornisce un *guess* sul tipo di documento.

Parametri

```
$ ./xmLeges-Marker.exe -h

./xmLeges-Marker.exe [opzioni] -f file

Software sviluppato dall'ITTIG/CNR per conto del progetto "Norme in Rete".

Opzioni:
-c file      : file di configurazione
-d dtd       : tipo di dtd (base,completo,flessibile,dl)
-e encoding  : set di caratteri del file di origine
               alcune abbreviazioni:
               'iso'=iso-8859-15, 'utf'=UTF-8, 'win'=windows-1252
-f file      : file da analizzare
-o file      : file di uscita
-i input     : formato del file di input
               'txt'=testo piatto, 'html'=testo formattato in html
               'ted'=formato proprietario TED
-n           : attiva il conteggio dei nodi
-s seq       : controllo sequenza (0/1)
-C n         : struttura del comma
               0=numerati
               1=non numerati con almeno una linea vuota separatrice
               2=non numerati senza linea vuota separatrice
-r n         : tipo rubriche (0/1/9)
               0=rubrica a capo
               1=rubrica adiacente (con separatore : [ -(])
               2=rubrica adiacente senza "Art." (con separatore : [ -(])
               9=testo senza rubrica (nel caso di commi non numerati)
-p []        : disattiva i patterns indicati
-m [at]      : disattiva i moduli indicati (a = annessi, t =tabella)
-t tipo      : tipo di documento analizzato
               l=legge                     lcost=legge costituzionale
               dl=decreto legge            dlgs=decreto legislativo
               lr=legge regionale          rd=regio decreto
               dm=decreto ministeriale     dmNN=decreto ministeriale - non numerato
               dpr=decreto del Presidente della Repubblica
               dprNN=decreto del Presidente della Repubblica - non numerato
               dpcm=decreto del Presidente del Consiglio dei Ministri
               dpcmNN=decreto del Presidente del Consiglio dei Ministri - non
numerato
               aut=atto di Authority       com=comunicato
               ddl=disegno di legge        nir=documento NIR
               reg=regolamento             circ=circolare
               rreg=regolamento regionale  prov=provvedimento
               cnr=provvedimento CNR
-T nomeTipo  : nome tipo di atto da analizzare; valido solo per -t nir
-M <dir>     : directory dei modelli per scansione testa e coda
-v           : livello di log: error, warn, info, debug
-L stderr|   : attiva i log indicati(ripetibile)
   file=nome   Esempio1: -L stderr
               Esempio2: -L stderr -L file=ParserStruttura.log
-V           : visualizzazione del testo con le virgolette soppresse (nessuna
conversione)

Versione 1.7 - [NIR 2.2]
```

Esempi

./xmLeges-Marker.exe -t ddl -d dl -e win -i html -f ddl3509.htm -o ddl3509.xml
./xmLeges-Marker.exe -t l -d flessibile -i txt -f Legge421.txt -o Legge421.xml

Varie

- Attualmente non viene eseguita nessuna operazione di validazione, quindi il marker può restituire documenti non validi (all'interno di xmLeges verrebbero segnalati come non validi). Come verifica della validità si possono usare strumenti esterni come xmllint (“xmllint --noout --valid doc.xml”).
- Le API delle libxml2 si sono rivelate a volte un po' complicate da gestire (questione liste formate da nodi testo e nodi entità e relative API), a volte inefficaci: si trovano, sia nel Marker che nell'HeaderParser varie funzioni che sostituiscono (in maniera brutale, a livello di puntatori) alcune di queste API.

- Aggiornamento dei modelli per l'HeaderParser.
  Modelli e file di configurazioni presenti nella directory “Models”.
  Esistono due vocabolari separati per l'analisi dell'header (header_extractor_model) e del footer (footer_extractor_model). Con i numeri associati alle entries del vocabolario si possono creare file di training “etichettati”, cioè in cui le parole (emissioni) sono associate a un particolare stato. In questo modo si creano i nodi e i collegamenti tra nodi nel modello, si creano inoltre le probabilità di emissione associate a ogni nodo e le probabilità di spostamento da un nodo a un altro.
  I file di training vengono utilizzati con il modulo HMM per creare il modello hmm (./HMM -l -t training.txt -m model.txt). Al modello devono essere poi aggiunti (a mano!) gli stati speciali 0 e -1, perciò andranno anche modificate altre parti del modello (percorsi verso gli stati iniziali e finali...).
  All'interno di parser_config gli stati di ogni modello sono associati ai vari tag del documento xml (è presente anche un valore che regola l'apertura e la chiusura dei tag).
  NOTA
  Anche per quanto riguarda i modelli dell'HeaderParser il più delle volte sono stati fatti degli aggiornamenti e “rattoppi” rispetto a ciò che era presente in origine, creando quindi esempi artificiali affinchè il nuovo modello prodotto includesse i nuovi casi. Quando è stato necessario creare nuovi modelli da zero (per i disegni di legge e per i provvedimenti CNR) si sono creati file di training con tutti esempi artificiali dal momento che in entrambi i casi gli header erano piuttosto rigidi, con una struttura ben precisa conoscendo il tipo di documento.