DK-Clarin-WP2-format

Dokumentation af processeringspipeline

Fællesdokument for WP2.2 og WP2.6

Dorte Haltrup Hansen, CST, KU

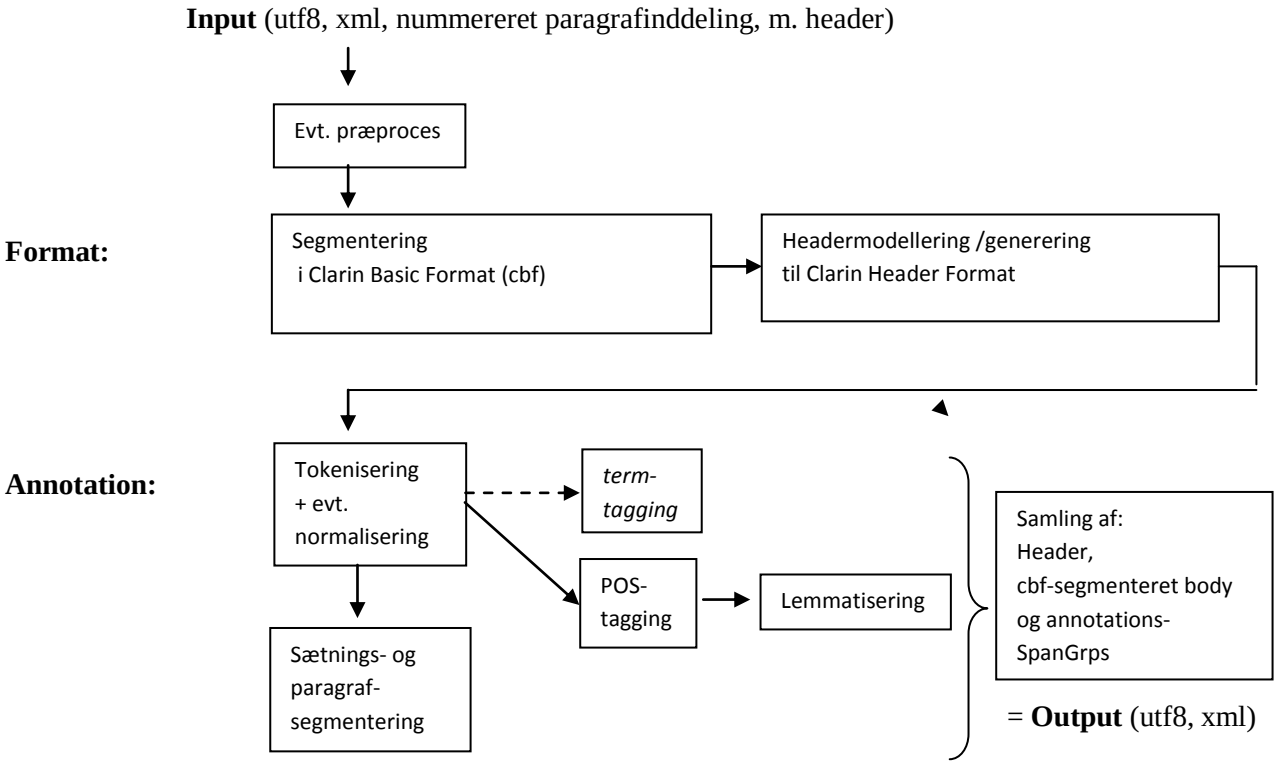
25/7 2011

1 Generelt om processen

I DK-Clarin WP2 har vi vedtaget et fælles basisformat der skal favne de forskellige måder at tokenisere tekst på i arbejdspakkens undergrupper. Vi har også vedtaget et fælles headerformat der bygger på TEIP5 (TEIP5DKCLARIN).
I arbejdspakkerne WP2.2 og WP2.6 har vi desuden skabt en fælles processeringspipeline hvor segmentering, annotering og til dels headermodellering er processer der kan genbruges.

De enkelte dele i processeringspipelinen bliver beskrevet i detaljer i dette dokument.

2 Processen


Input (utf8, xml, nummereret paragrafinddeling, m. header)
Evt. præproces
Format:
Segmentering
i Clarin Basic Format (cbf)
Headermodellering /generering
til Clarin Header Format
Annotation:
Tokenisering
+ evt.
normalisering
term-
tagging
POS-
tagging
Lemmatisering
Sætnings- og
paragraf-
segmentering
Samling af:
Header,
cbf-segmenteret body
og annotations-
SpanGrps
= **Output** (utf8, xml)

Input er en tekstfil i xml, utf8 med fortløbende nummererede paragrafinddelinger og en paragraf på hver linje. Hvis en af delene ikke er tilfældet, skal der laves en præproces som indsættes allerførst i pipelinen. Som præproces kan man fx også konvertere html eller rette uregelmæssigheder i inputteksten.
Bemærk at der pt. arbejdes med 3 xml-entiteter: <, >, & hvilket vil sige at hvis <, >, & optræder i inputtet bliver det transformeret til <, >,&amp.

I cbf-segmenteringen splittes inputteksten i:

- interpunktionstegn,
- spaces og i
- *andet* (stort set svarende til ord).

Da det kun er strenge der indledes af <p el. <head som segmenteres, er det vigtigt at præprocesseringen er i orden. Hvert cbf-element forsynes med et unikt id der kan refereres til i senere processer.

Dernæst modelleres inputtekstens header så den kan validere mod det fastlagte skema. Hvis der endnu ikke er lavet en TEIP5DKCLARIN-header til teksten, skal den genereres på dette tidspunkt. Dette modul varierer derfor efter inputtets headerformat.

Efter det basale Clarinformat er skabt, kan annotationslagene lægges på. Hver annotation lægges i en spangroup ved siden af teksten, og hvert led i de forskellige annotationslag refererer tilbage til et unikt id:

- tokeniseringen refererer til cbf- id,
- pos-annotation, lemmatisering og term-annotation refererer til de unikke token-id.

I tokeniseringen samles de atomare cbf-segmenter i tokens, dvs. at interpunktionstegn og ord holdes adskilt, mens fx punktummer i forkortelser og komma i tal samles med de størrelser hvor de hører til. Hvis man ønsker at normalisere teksten fx ved at ændre stavefejl, transformere store bogstaver el.lign. kan man indsætte et script i pipelinen her.

På baggrund af tokeniseringen identificeres sætningspunktummer og teksten opdeles i nummererede sætningssegmenter og paragrafsegmenter.

Næste led i annotationen er POS-tagging. Pt. kræver taggeren at input er markeret med de enheder der skal tagges, fx sætninger eller paragraffer. Output ligger i en spangroup med pos-tagget 'klistret' til tokenet. Der er derfor brug for en postproces som fjerner selve tokenet så kun POS-tag samt referencer lades tilbage i spangruppen. Samme gør sig gældende for lemmatiseringen hvor hvert token er blevet annoteret med ordets grundform, lemmaet. For både POS-tagger og lemmatiser ligger de forskellige optioner til programmerne i optionsfiler der er datomærket.

Domænespecifikke tekster (fra WP2.2 og årsrapporter fra WP2.6) annoteres med en termhoodvægtning.

Sidst i processen samles alle spangroups med cbf-formatet og headeren (se eksempel på en samlet fil i afsnit 5).

3 Kørsel af processen

Pipelinen styres af et RUN-script (perl) der varierer en smule alt efter om det er engelske, danske, WP22-, WP26-, Acquis- eller andre tekster der køres.

Pt. ligger alle scripts/programmer på serveren IDA :
/opt/clarin/wp2-tools/wp26-specific (senere kaldet $wp26dir/)
/opt/clarin/wp2-tools/wp22-specific (senere kaldet $wp22dir/)
/opt/clarin/wp2-tools/tools (fællesscripts, senere kaldet $tooldir/)
/opt/WP2_2/KONVERTERINGSSCRIPTS
/var/csttools/bin/csttaggerXML
/var/csttools/bin/cstlemma

Pipelinen (for danske Acquis-tekster) køres med kommandoen: perl RUN_daProcesAcquis.pl.
Der er på nuværende tidspunkt implementeret flere RUN-scripts, se afsnit 6.

Flg. information er **hard-coded** og ændres derfor i RUN-scriptet:
Inputfilnavn (listen af filer som skal processeres, filnavne er *uden* stinavne)
Inputdirektorier
Outputdirektorier
OBS! Man kan på et senere tidspunkt implementere de hard-coded informationer som input-parametre.

Underkataloger med "Source year", som er en del af Acquis-input/output-katalogerne beregnes ud fra Acquis-filnavnene. Dette er ikke tilfældet for andre ikke-Acquis-tekster.

Input:
1) en liste af filnavne på de dokumenter der skal processeres (u. stinavn, som er hardcoded i RUN-scriptet)
Format: txtfil, utf8, et filnavn på hver linje. Filnavne må ikke indeholde #

Eks. fra filen *da-Acquis-files*

```
jrc31958Q1101-da.xml
jrc31958R0001-da.xml
jrc31958R0003_01-da.xml
jrc31960D0511-da.xml
```

2) selve dokumenterne der skal processeres (listet i inputfilen fra 1.)
Format: xml, utf8, fortløbende nummererede paragraffer, en paragraf på hver linje, filen forsynet m. header. Filnavnet *skal* ende på .xml.
Bemærk at hvis inputtet ikke har fortløbende paragrafnummerering, skal det laves i en præproces, fordi både segmentering og tokenisering bruger nummereringen i processering af teksterne. Endvidere skal paragraf*teksten* starte på samme linje som paragrafnummereringen

fx: TILLÆGSAFTALE til aftalen …..

da det kun er strenge der indledes af <p el. <head som segmenteres.

Eks. fra filen *jrc21970A0720_01-da.xml*

```
<teiHeader>… </teiHeader>
<text> <body>
 <head n="1">Tillægsaftale til aftalen vedrørende urmagervarer mellem Det europæiske … </head>
```

TILLÆGSAFTALE til aftalen vedrørende urmagervarer mellem Det …
</body></text>

Output:

Fil(er) med ekstensionen FILNAVN.*unite.xml* (fx *jrc21970A0720_01-da-unite.xml*)
Outputdirektoriet er defineret i RUN-scriptet

fx: *$acquisdir /OUTPUT/da/acquis-unite*
($acquisdir = */opt/clarin/wp2-tools/wp26-specific/acquis*)

I processen genereres en række direktorier med midlertidige outputfiler. Disse slettes til sidst når alle output-spangroups er samlet i FILNAVN.*unite.xml*. Bemærk at Acquis-filerne også her er anderledes end de andre filer der processeres, fordi der skabes "sourceyear"-underdirektorier til outputtet. "Sourceyear" genereres automatisk fra filnavnet.

Format: xml, utf8, der overholder TEI P5 formatet og validerer mod et Clarin-WP2-skabt xml-skema:

<TEI xmlns=http://www.tei-c.org/ns/1.0
xmlns:schemaLocation="http://www.tei-c.org/ns/1.0
http://dkclarin.dk/schemas/WP2/TEIDKCLARINv1.0/xml.xsd">

4 Fejlsøgning

Her gives eksempler på potentielle fejlkilder i processen. Opdater gerne listen efterhånden som nye fejlkilder opstår.

a) Source year
   Det er kun ved Acquis-filerne at der beregnes og skabes underdirektorier for *source year*. Hvis man modellerer sit RUN-script over *RUN_daProcesAcquis.pl* og ikke retter den del af programmet kommer der intet output.

b) Inputfilnavn
   Kontroller at inputfilen (listen af filnavne) har samme filnavn som i RUN-scriptet:
   open(FILES,filnavn)

c) Inputfilformat
   Filerne skal være utf8.
   Kun linjer der startes af <p og <head processeres

d) Stinavn
   Pt. er stinavne for filer der skal processeres, hard-coded i RUN-scriptet.
   Inputfilen (listen af filnavne) skal derfor ikke indeholde stinavne.

e) Outputdirektorier
   Ouputdirektorierne burde genereres under kørslen, hvor er defineret i RUN-scriptsene.
   Af en eller anden grund lykkes det ikke altid; men så kan de skabes manuelt inden kørslen.

f) Paragrafnumre
   Inputfilerne (filerne der skal processeres) skal være opmærket med *nummererede* paragraffer. Og da det kun er linjer indledt af <p eller <head der processeres, skal der stå en paragraf med flg. Syntaks på hver linje:

   ```
   <p n="2">TILLÆGSAFTALE til aftalen vedrørende urmagervarer mellem Det …</p>
   ```

g) Intet tagger/lemmatiser output
   Kontroler at stinavne til tagger/lemmatiser-koden er korrekt i optionsfilerne.

h) perl version
   Der benyttes perl v5.10.0 som ligger: /opt/clarintools/localperl/bin/perl

5 Eksempeloutput fra processering

Eksemplet er fra WP2.2-filen: 1.pdf, fra Region Hovenstaden.
POS-taggeren *cstClarintaggerXML* er pt. kun brugt i WP2.6-filerne fra Rapid samt i dette eksempel.

```
<TEI xmlns="http://www.tei-c.org/ns/1.0" xmlns:schemaLocation="http://www.tei-c.org/ns/1.0
http://dkclarin.dk/schemas/WP2/TEIDKCLARINv1.0/xml.xsd">
<teiHeader type="text">
                                        ….
  </teiHeader>
<text>
 <body>
<p n="1">
                    <w xml:id="i1.1">Rapport</w>
                    <c xml:id="i1.2" type="s"/>
                    <w xml:id="i1.3">fra</w>
                    <c xml:id="i1.4" type="s"/>
                    <w xml:id="i1.5">specialegruppen</w>
                    <c xml:id="i1.6" type="s"/>
                    <w xml:id="i1.7">i</w>
                    <c xml:id="i1.8" type="p">:</c>
</p>
<p n="2">
                    <w xml:id="i2.1">Allergologi</w>
</p>
<p n="3">
                    <w xml:id="i3.1">19</w>
                    <c xml:id="i3.2" type="p">.</c>
                    <w xml:id="i3.3">1</w>
                    <c xml:id="i3.4" type="p">.</c>
                    <w xml:id="i3.5">2006</w>
</p>
<p n="4">
                    <w xml:id="i4.1">Specialegruppens</w>
                    <c xml:id="i4.2" type="s"/>
                    <w xml:id="i4.3">medlemmer</w>
                    <c xml:id="i4.4" type="p">:</c>
                    <c xml:id="i4.5" type="s"/>
                    <w xml:id="i4.6">Sygeplejedirektør</w>
                    <c xml:id="i4.7" type="s"/>
                    <w xml:id="i4.8">Kirsten</w>
                    <c xml:id="i4.9" type="s"/>
                    <w xml:id="i4.10">Poulsen</w>
                                        ….
</p>
</body>
<spanGrp ana="#CstClarinDaTokeniser">
                    <span xml:id="t1" from="#i1.1">Rapport</span>
                    <span xml:id="t2" from="#i1.3">fra</span>
                    <span xml:id="t3" from="#i1.5">specialegruppen</span>
                    <span xml:id="t4" from="#i1.7">i</span>
                    <span xml:id="t5" from="#i1.8">:</span>
                    <span xml:id="t6" from="#i2.1">Allergologi</span>
                    <span xml:id="t7" from="#i3.1" to="#i3.5">19.1.2006</span>
```

```
                <span xml:id="t8" from="#i4.1">Specialegruppens</span>
                <span xml:id="t9" from="#i4.3">medlemmer</span>
                <span xml:id="t10" from="#i4.4">:</span>
                <span xml:id="t11" from="#i4.6">Sygeplejedirektør</span>
                <span xml:id="t12" from="#i4.8">Kirsten</span>
                <span xml:id="t13" from="#i4.10">Poulsen</span>
                                ….
</spanGrp>
<spanGrp ana="#CstClarinSentenceSegmenter">
                <span xml:id="S1" type="Sseg" from="#t1" to="#t5"/>
                <span xml:id="S2" type="Sseg" from="#t6" to="#t6"/>
                <span xml:id="S3" type="Sseg" from="#t7" to="#t7"/>
                <span xml:id="S4" type="Sseg" from="#t8" to="#t53"/>
                                ….
</spanGrp>
<spanGrp ana="#CstClarinParagraphSegmenter">
                <span xml:id="P1" type="Pseg" from="#t1" to="#t5"/>
                <span xml:id="P2" type="Pseg" from="#t6" to="#t6"/>
                <span xml:id="P3" type="Pseg" from="#t7" to="#t7"/>
                <span xml:id="P4" type="Pseg" from="#t8" to="#t53"/>                                    ….
</spanGrp>
<spanGrp ana="#csttaggerXML">
                <span xml:id="p1" from="#t1">N_INDEF_SING</span>
                <span xml:id="p2" from="#t2">PRÆP</span>
                <span xml:id="p3" from="#t3">N_DEF_SING</span>
                <span xml:id="p4" from="#t4">PRÆP</span>
                <span xml:id="p5" from="#t5">TEGN</span>
                <span xml:id="p6" from="#t6">N_INDEF_SING</span>
                <span xml:id="p7" from="#t7">NUM</span>
                <span xml:id="p8" from="#t8">N_DEF_SING_GEN</span>
                <span xml:id="p9" from="#t9">N_INDEF_PLU</span>
                <span xml:id="p10" from="#t10">TEGN</span>
                <span xml:id="p11" from="#t11">V_INF</span>
                <span xml:id="p12" from="#t12">EGEN</span>
                <span xml:id="p13" from="#t13">EGEN</span>
                                ….
</spanGrp>
<spanGrp ana="#cstClarintaggerXML">
                <span xml:id="pc1" from="t1"> NN_COM_SING_INDEF </span>
                <span xml:id="pc2" from="t2"> PREP </span>
                <span xml:id="pc3" from="t3"> NN_COM_SING_DEF </span>
                <span xml:id="pc4" from="t4"> PREP </span>
                <span xml:id="pc5" from="t5"> RESID_SIGN </span>
                <span xml:id="pc6" from="t6"> NPROP </span>
                <span xml:id="pc7" from="t7"> NPROP </span>
                <span xml:id="pc8" from="t8"> NN_COM_SING_GEN_DEF </span>
                <span xml:id="pc9" from="t9"> NN_NEUT_PLU_INDEF </span>
                <span xml:id="pc10" from="t10"> RESID_SIGN </span>
                <span xml:id="pc11" from="t11"> NN_COM_SING_INDEF </span>
                <span xml:id="pc12" from="t12"> NPROP </span>
                <span xml:id="pc13" from="t13"> NPROP </span>
                                ….
</spanGrp>
<spanGrp ana="#cstlemma">
                <span xml:id="l1" from="#t1" >Rapport</span>
                <span xml:id="l2" from="#t2" >fra</span>
```

```
                <span xml:id="l3" from="#t3" >specialegruppe</span>
                <span xml:id="l4" from="#t4" >i</span>
                <span xml:id="l5" from="#t5" >:</span>
                <span xml:id="l6" from="#t6" >Allergologi</span>
                <span xml:id="l7" from="#t7" >19.1.2006</span>
                <span xml:id="l8" from="#t8" >Specialegruppe</span>
                <span xml:id="l9" from="#t9" >medlem</span>
                <span xml:id="l10" from="#t10" >:</span>
                <span xml:id="l11" from="#t11" >Sygeplejedirektør</span>
                <span xml:id="l12" from="#t12" >Kirsten</span>
                <span xml:id="l13" from="#t13" >Poulsen</span>
                                ….
</spanGrp>
<spanGrp ana="#DsnClarinDaTermTaggerLogLikelihood">
                <span xml:id="th1" from="#t1" >2.985</span>
                <span xml:id="th2" from="#t2" >-4.467</span>
                <span xml:id="th3" from="#t3" >26.735</span>
                <span xml:id="th4" from="#t4" >0</span>
                <span xml:id="th5" from="#t5" >0</span>
                <span xml:id="th6" from="#t6" >240.644</span>
                <span xml:id="th7" from="#t7" >0</span>
                <span xml:id="th8" from="#t8" >26.735</span>
                <span xml:id="th9" from="#t9" >3.172</span>
                <span xml:id="th10" from="#t10" >0</span>
                <span xml:id="th11" from="#t11" >18.491</span>
                <span xml:id="th12" from="#t12" >1.638</span>
                <span xml:id="th13" from="#t13" >3.990</span>
                                ….
</spanGrp>
</text>
</TEI>
```

6 Programbeskrivelser

6.1 Et styreprogram for pipelinen

Styreprogrammerne har til opgave at kalde alle andre programmer i processen samt definere input- og output direktorier og filnavne. Nedenfor er pipelinen eksemplificeret gennem styreprogrammet for Acquis.

RUN_daProcesAcquis.pl
Håndterer Acquis-formatet, dvs. input organiseret efter *source year* og i xml-format TEI P2. Modellering af Clarin-headeren tager udgangspunkt i Acquis-headeren.
Input: (udførligt beskrevet i Kap. 3 *Kørsel af processen*)
Output: (udførligt beskrevet i Kap. 3 *Kørsel af processen*)

6.1.1 Præprocesser

- $wp26dir/PROG.Acquis_pre_segment.pl:
  transformerer specielle enheder i Acquisfilerne som fx *%Aring%*
- $tooldir/PROG.conv_html_entities.pl*:*
  konverterer html-elementer fx ø → ø

Input: *inputfil* , xml, utf8, nummereret paragrafinddeling
Output: Udskrives ikke men sendes direkte videre til segmentering

Cbf-segmentering:
Bygger på præprossering, xml, utf8 m. fortløbende paragrafnummerering
Segment-id starter m. *i* fx <w xml:id="i1.1">Region</w>

- $tooldir/PROG.segmenter.pl:
  segmenterer i Clarin Basis Format (cbf). Fælles for alle styreprogrammer, beskrives nedenfor.
- $tooldir/PROG.conv_Danish_letters.pl:
  konverterer danske bogstaver fx ae → æ

Input: kommer direkte fra præproces
Output: *fil*-cbf.xml
Efter cbf-segmentering tælles antal ord, dvs. <w> -tags, og summen indsættes i headeren. Fælles for alle RUN–scripter.

Headermodellering:

- PROG.Acquis_header_converter
  Acquis-headeren omformes til Clarin-wp2-header-format således at Acquis headeroplysningerne genbruges. Programmet processerer alle linjer indtil tagget</teiHeader mødes (headeren ender). Det er derfor vigtigt at headeren er af typen <teiHeader>. De segmenterede linjer røres ikke.
  Der sendes flg. parametre med: *source year, word count, language code* (da, en,…). Disse parametre genereres automatisk i RUN-scriptet og skal sendes til programmet i nævnte rækkefølge.
  Oplysningerne der indsættes i headeren kommer altså fra: Acquis-headeren, som inputvariabler eller er hardcoded i scriptet.

Input: *fil*-cbf.xml
Output: *fil*-cbf-header.xml, *fil*-header.log

6.1.2 Annotation

Tokenisering

Bygger på cbf-segmentering, id'erne i *to* og *from* er af formen: *i+paragrafnr.+ segmentnr.*
Token-id starter altid med *t*

fx Region

▪ $tooldir/PROG.ClarinDaTokeniser.pl:
samler Clarin Basis Format (CBF) til tokens (se beskrivelse nedenfor)
▪ $tooldir/PROG.normaliser.pl:
transformerer fx sætningsinitiale store bogstaver til små. Det er meningen at man med tiden kan tilføje flere typer normalisering. OBS! Fungerer pt. *ikke* (28/10 2010)

Input: *fil*-cbf-header.xml
Output: *fil*-cbf-header-tok.xml

Sætningssegmentering

Bygger på tokenisering, id'erne i *to* og *from* starter derfor altid m. *t*
Sætnings-id starter m. *Sseg* fx ,
Paragraf-id starter m. Pseg fx

▪ $tooldir/PROG.sent-segmenter.pl:
segmenterer i sætninger og paragraffer (se beskrivelse nedenfor).

Segmenteringen bruges pt. ikke i POS-tagging.
Input: *fil*-cbf-header-tok.xml
Output: *fil*-cbf-header-tok-sent.xml og *fil*-cbf-header-tok-para.xml

POS-tagging

Bygger på tokenisering, id'erne i *to* og *from* starter derfor altid med *t*
POS-id starter med *p*

fx N_INDEF_SING

Den nuværende POS-tagger har brug for en markering af de enheder der skal tagges. Indtil sætningssegmenteringen (beskrevet ovenfor) kan anvendes af taggeren, foretages en primitiv segmentering i flg. præproces

▪ $tooldir/PROG.pre-POS.pl:
sætter
 efter hver paragraf som en simulering af sætningssegmentering

Input: *fil*-cbf-header-tok.xml
Output: *fil*-cbf-header-tok-prepos.xml

▪ /var/csttools/bin/csttaggerXML

POS-tagging af spanGrps beskrives nærmere nedenfor. For Rapid-filerne er der 2 taggere i spil: csttaggerXML (kaldet med tagger-options_10052010_da) og cstClarintaggerXML (kaldet med tagger-options_clarintagger_22072011_da)

Input: -i *fil*-cbf-header-tok-prepos.xml, -@ $tooldir/tagger-options_10052010_da
Output: -o *fil*-cbf-header-tok-tmppos.xml

Input: -i *fil*-cbf-header-tok-prepos.xml, -@ $tooldir/ tagger-options_clarintagger_22072011_da
Output: -o *fil*-cbf-header-tok-tmpClarinpos.xml

▪ $tooldir/PROG.post-POS.pl og
$tooldir/PROG.post-ClarinPOS.pl
Postproces til POS der fjerner <br\> igen, omdøber *t*-id til *p*-id, fjerner *i*-referencerne (der er overlevet i *to* og *from* i token-spanGrps) og indsætter reference til *t* (tokenformat) i *to* og *from*, fjerner selve ordene og lader kun POS-tag tilbage.
Processen er samtidig en præproces til lemmatiseren. Her bibeholdes selve ordet og POS-tagget indsættes som et *pos-attribut* og der indsættes et tomt *lemma-attribut.*
Input: *fil*-cbf-header-tok-tmppos.xml
Output: *fil*-cbf-header-tok-pos.xml, *fil*-cbf-header-tok-tmppos-prelemma.xml

Lemmatisering
Bygger på tokenisering, id'erne i *to* og *from* starter derfor altid med *t*
Lemma-id starter med *l*

*f*x region

Præproces er samlet med postprocessen for PoS-taggeren (se beskrivelsen af PROG.post-POS.pl ovenfor)

▪ /var/csttools/bin/cstlemma
Lemmatisering af spanGrps og optionsfil beskrives nærmere nedenfor
Input: *fil*-cbf-header-tok-tmppos-prelemma.xml , -@ lemma-options_14062010_da
Output: *fil*-cbf-header-tok-pos-tmplemma.xml

▪ $tooldir/PROG.post-lemma.pl
Postproces til lemmatiseringen der modellerer den spanGrp (fra POS) som var input. Omdøber *p*-id til *l*-id, fjerner POS-tagget, ordformen og lader kun lemmaet stå tilbage.
Input: *fil*-cbf-header-tok-pos-tmplemma.xml
Output: *fil*-cbf-header-tok-pos-lemma.xml

Samling af spanGrps:
▪ Alle spanGrps med header + cbf samles med unix-commandoen *cat*
Input: *fil*-cbf-header.xml, *fil*-cbf-header-tok.xml, *fil*-cbf-header-tok-sent.xml, *fil*-cbf-header-tok-pos-lemma.xml

Output: *fil*-preunite.xml
▪$tooldir/PROG.change_unite_header_da.pl:
Retter i *fil*-preunite.xml så den kan validere fx slutter med: </text></TEI>. Indsætter information om annotationsværktøjerne i headeren, disse er pt. hardcoded. Indsætter revisionsdato i headeren
Input: *fil*-preunite.xml
Output: *fil*-unite.xml
Sidst slettes midlertidige outputfiler og kun den samlede *fil*-unite.xml beholdes.

TEI-validering:
xmllint --relaxng /var/vhosts/dkclarin.dk/schemas/WP2/TEIDKCLARIN.rng --noout fil-unite.xml

6.2 Andre styreprogrammer fra WP.2.6

RUN_enProcesAcquis.pl: identisk med *RUN_daProcesAcquis.pl* bortset fra at det er de engelske versioner af programmerne der kaldes. Formatet er det samme som i det danske Acquis.

RUN_daProcesRapid.pl: Processerer pressemeddelelser fra EU pressetjeneste Rapid. Præprocesserne er lidt anderledes end for Acquis-filerne fordi Rapid-filerne foreligger i html. Annoteringen er stort set den samme; dog er *cstClarintaggerXML* (POS-tagger) tilføjet.
RUN_enProcesRapid.pl: som RUN_daProcesRapid.pl uden *cstClarintaggerXML*.

RUN_daProcesAnnualReports.pl: Processerer årsrapporter fra store danske virksomheder. Præprocesserne er lidt anderledes end Acquis-filerne fordi udgangspunktet er pdf-filer konverteret til text vha. unix-kommandoen *pdftotext*. Annoteringen er stort set den samme; dog er DSN-termtagger er tilføjet.
RUN_enProcesAnnualReports.pl: som RUN_daProcesAnnualReports.pl: uden DSN-termtagger.

6.3 Styreprogrammer fra WP.2.2

Der er 28 styreprogrammer for filerne i WP2.2. Det høje antal skyldes at der er mange tekstleverandører og flere filformater (pdf og html). Alle scripts er bygget over *RUN_daProcesAcquis.pl* men adskiller sig fra det dels ved præprocesserne og dels ved at alle filer er term-tagget.

Styrerprogrammerne ligger i IDA:/opt/clarin/wp2-tools/wp22-specific.

Pdf-filerne er konverteret til text-format vha. unix-kommandoen *pdftotext*; mens HTML-filerne viste sig i de fleste tilfælde relativt enkle at konvertere til XML, idet værktøjet TagSoup (http://home.ccil.org/~cowan/XML/tagsoup/) muliggør nem konvertering fra (muligvis invalid) HTML til valid XHTML som igen kan konverteres til XML ved hjælp af et XSLT-script og en XSLT-fortolker som Saxon (http://saxon.sourceforge.net/). I nogle tilfælde var det også muligt automatisk at trække alle relevante metadata om teksten ud af det originale HTML. XSLT-scriptsne ligger i */opt/WP2_2/KONVERTERINGSSCRIPTS/*.

Præprocesser:
▪ */opt/WP2_2/KONVERTERINGSSCRIPTS/PROG_texttoxml*. Indsætter <p> og sletter bindestreg over linjeskift.
▪ */opt/WP2_2/KONVERTERINGSSCRIPTS/PROG_modifyRawPdfToText.pl*. Laver velformet xml.
▪ *makeUTF8*. Konverterer input til utf8. Bruges kun til ISO-filer
▪ *$wp22dir/PROG.WP22preproces.pl*. Modellerer selve xml-inputtet og nummererer <p>-tags

Headermodellering:
▪ *$wp22dir/PROG.daWP22convert-header.pl*.Indsætter antal ord og paragraffer i headeren.

WP2.2 styreprogrammer:
- RUN_agrsci_daProcesWP22.pl (fra pdf)
- RUN_AktuelNatur_1_daProcesWP22.pl (fra pdf)

- RUN_AktuelNatur_2_daProcesWP22.pl (fra pdf)
- RUN_DMU_daProcesWP22.pl (fra pdf)
- RUN_erhvervsOgByggestyrelsen_daProcesWP22.pl (fra pdf)
- RUN_erhvervsOgByggestyrelsen_2_daProcesWP22.pl (fra html)
- RUN_erhvervsOgSelskabsstyrelsen_daProcesWP22.pl (fra pdf)
- RUN_finanstilsynet_daProcesWP22.pl (fra pdf)
- RUN_finanstilsynet_2_daProcesWP22.pl (fra html)
- RUN_Hovedland_daProcesWP22.pl (fra pdf)
- RUN_librisIT_daProcesWP22.pl (fra html)
- RUN_librisSundhed_daProcesWP22.pl (fra pdf)
- RUN_muro_daProcesWP22.pl (fra pdf)
- RUN_nano_1_daProcesWP22.pl (fra pdf)
- RUN_nano_2_daProcesWP22.pl (fra pdf)
- RUN_nano_3_daProcesWP22.pl (fra pdf)
- RUN_nano_4_daProcesWP22.pl (fra html)
- RUN_nano_5_daProcesWP22.pl (fra html)
- RUN_nano_6_daProcesWP22.pl (fra html)
- RUN_netpatient_daProcesWP22.pl (fra html)
- RUN_OekRaad_daProcesWP22.pl (fra html)
- RUN_OO_it_daProcesWP22.pl (fra html)
- RUN_regionH_daProcesWP22.pl (fra pdf)
- RUN_SBI_daProcesWP22.pl (fra pdf)
- RUN_SKAT_daProcesWP22.pl (fra html)
- RUN_soefartsstyrelsen_daProcesWP22.pl (fra pdf)
- RUN_SST_daProcesWP22.pl (fra pdf)
- RUN_sundhed_dk_daProcesWP22.pl (fra html)

6.4 Fælles (større) programmer

Nedenfor beskrives de større programmer der kaldes fra alle styreprogrammer. Input, output samt hvor i pipelinen de kaldes er beskrevet i afsnit 6.1, outputformat er beskrevet i Kap 5.

Programmerne ligger alle i: /opt/clarin/wp2-tools/tools

6.4.1 PROG.segmenter.pl

Segmenterer input til Clarin Basic Format. Inputtet indeholder en header; men denne røres ikke på nuværende tidspunkt.
Segmenterne inddeles i:

- *space-agtige* entiter som benævnes *type="s"*:
  (\s \t   )
  Ex. <c xml:id="i1.7" type="s"/>
  Bemærk at den space-agtige entitet ikke er udtrykt!
- *punctuation* entiteter som benævnes *type="p"*:
  [\', \(, \), \[, \], \{, \}, \:, \,, \-, \–, \—, \…, \!, \., \‹ , \›, \«, \», \?, \‘, \’, \“, \”, \;, \\, \/, \"]
  Ex. <c xml:id="i1.8" type="p">:</c>
- *andet* som benævnes *type="w"*:
  Ex. <w xml:id="i2.1">Allergologi </w>

Segmenterne nummereres *i*+ paragrafnr.+ordnr.

Kun linjer der starter med <p eller <head i inputtet processeres da det er her den relevante tekst forventes. Det er derfor vigtigt at inputtet har det korrekte format.
Eks. på inputformat:

fx: Rapport fra specialegruppen i: …..

Eks. på outputformat:

```
<w xml:id="i1.1">Rapport</w>
<c xml:id="i1.2" type="s"/>
<w xml:id="i1.3">fra</w>
<c xml:id="i1.4" type="s"/>
<w xml:id="i1.5">specialegruppen</w>
<c xml:id="i1.6" type="s"/>
<w xml:id="i1.7">i</w>
<c xml:id="i1.8" type="p">:</c>
```

Desuden fjernes flere efterfølgende spaces, initialt og finalt space og <, >, & omformes til til xml-valide entiteter (<, >, &).

6.4.2 PROG.ClarinDaTokeniser.pl

CstClarinDaTokeniser version 1.0
Tokeniseren samler CBF-elementer til relevante tokens. Paragrafnummereringen bruges således at de samlede dele af et token skal findes inden for samme paragraf (der ca. svare til en sætning i Acquis). Der gøres brug af en liste af forkortelse uden punktum *fork_short-utf8* og en liste af interpunktionstegn *punctuation*.

Følgende tilfælde behandles:

1) Punktum
    1.1 Forkortelsespunktummer
    1.2 Uautoriserede forkortelser - hvor der ikke kommer ord med stort eller tal efter punktummet
    1.3 Ordenstal <100 og datoer m. evt. årstal samlet m. punktummer
    1.4 Inde midt i ord, fx i url og tal
2) Semikolon
    2.1 Semikolon som del af ' midt i et ord
    2.2 Semikolon som del af & midt i et ord
3) Komma
    3.1 Komma mellem 2 tal, også hvis der er tusindtalseparator i
4) Bindestreg
    4.1 Bindestreg mellem 2 tal eller ord
    4.2 Bindestreg i slutningen af et ord/tal (*Er endnu ikke implementeret!!*)
5) Skråstreg /
    5.1 Skråstreg midt i et ord
6) Backslash
    6.1 Backslash midt i et ord
7) Apostrof '
    7.1 Hvis der er 'noget' (tal eller bogstaver) på begge sider af apostrof('), samles det (det gælder både genitiv 's, 'er, 'erne osv.)
8) ’ en slags apostrof ...
    8.1 Hvis der er 'noget' (tal eller bogstaver) på begge sider af apostrof(’), samles det (det gælder både genitiv ’s, ’er, ’erne osv.)

6.4.3 PROG.ClarinEnTokeniser.pl

CstClarinEnTokeniser version 1.0
Programmet er modelleret over PROG.ClarinDaTokeniser.pl, derfor er beskrivelsen nærmest identisk. Det er specielt ved *apostrof* at de to tokenisere er forskellige.

Tokeniseren samler CBF-elementer til relevante tokens. Paragrafnummereringen bruges således at de samlede dele af et token skal findes inden for samme paragraf (ca. en sætning).
Der gøres brug af en liste af forkortelser uden punktum *en-abbr* og en liste af interpunktionstegn *punctuation*.
Outputtet er en spanGrp uden header: <spanGrp ana="# CstClarinEnTokeniser"> …. </spanGrp>
Følgende tilfælde behandles:

1. Punktum
    1.1 Forkortelsespunktummer
    1.2 Uautoriserede forkortelser- der stater m stort og indeholder max 3 tegn
    1.3 Uautoriserede forkortelser - hvor der ikke kommer ord med stort efter
    1.4 Ordenstal <100 og datoer m. evt. årstal samlet m. punktummer
    1.5 Inde midt i ord, fx i url og tal
2. Semikolon
    2.1 Semikolon som del af ' midt i et ord
    2.2 Apostrof (')mellem n og t i fx aren't

2.3 Semikolon som del af & midt i et ord

3. Komma
    3.1 Komma mellem 2 tal, også hvis der er tusindtalsseparator i
4. Bindestreg
    4.1 Bindestreg mellem 2 tal eller ord
    4.2 Bindestreg i slutningen af et ord/tal (*Er endnu ikke implementeret!!*)
5. Skråstreg /
    5.1 Skråstreg midt i et ord
6. Backslash
    6.1 Backslash midt i et ord
7. Apostrof '
    7.1 Apostrof mellem n og t i fx aren't (aren't => are n't, aren(id=1)'(id=2)t(id=3)) => are(id=1) n't(id: from=2 to=3))
    7.2 Apostrof før m,s,d
    7.3 Apostrof før re, ve, ll

6.4.4 PROG.sent-segmenter.pl

CstClarinSentenceAndParagraphSegmenter version 1.0
Markerer teksten med udstrækning af sætninger og paragraffer. Udstrækning af paragraffer er kodet som første tal i det tokeniserede inputs *from* attribut (fx from="i4.1). Udstrækningen af sætninger beregnes på baggrund af sætningspunktum (identificeret af tokeniseren), udråbtegn og spørgsmålstegn (.!?).
Der laves to outputfiler: én for sætningssegmentering og én for paragrafsegmentering
Eks. tokeniseret input:

```
<span xml:id="t1" from="#i1.1">Rapport</span>
<span xml:id="t2" from="#i1.3">fra</span>
<span xml:id="t3" from="#i1.5">specialegruppen</span>
<span xml:id="t4" from="#i1.7">i</span>
<span xml:id="t5" from="#i1.8">:</span>
<span xml:id="t6" from="#i2.1">Allergologi</span>
```

Eks. spanGrp-output:

```
<span xml:id="S1" type="Sseg" from="#t1" to="#t5"/>
<span xml:id="S2" type="Sseg" from="#t6" to="#t6"/>
                    ….
<span xml:id="P1" type="Pseg" from="#t1" to="#t5"/>
<span xml:id="P2" type="Pseg" from="#t6" to="#t6"/>
```

6.4.5 csttaggerXML

csttaggerXML version 3.1 (Ida: /var/csttools/bin/csttaggerXML)
CST's videreudvikling af Brills tagger så den kan fungere på xml og de spanGrp vi definerer i Clarin-WP2, ud over det er funktionaliten som den almindelige tagger. Taggeren kræver at det tokeniserede input er markeret ift. de størrelser der skal tagges (fx sætninger eller paragraffer). Pt. gøres det ved at indsætte
 via en præproces, men det er tanken at sætningssegmenteringen skal bruges.
Outputtet er også en spanGrp, hvor en postproces skal fjerne
 og token, så filen kun indeholder info om POS-tags.

6.4.6 Optionsfiler for csttaggerXML

I optionsfilerne er defineret hvilke ordbøger, regler, defaulttags, sætningsmarkør og inputformat.
Den danske version af POS-taggeren bruger dels *CSTtags* (50 tags som brugt i MOSES-projektet) og dels *Clarintags*(103 tags defineret i DK-Clarin). I den engelske version bruges *PennTreebank-tags*. Disse oplysninger er ikke eksplicit udtrykt, i optionsfilerne; men ligger implicit i hvilke ordbøger og regler der bruges. Bemærk derfor at både STO-, PoS-, PAROLE- og lemmatiserens tags (de morfosyntaktiske oplysningstyper) skal ændres hvis man ændrer i POS-tagsættet.

- tagger-options_10052010_da definerer *CSTtags*,
- tagger-options_clarintagger_22072011_da definerer *Clarintags.*
- tagger-options_12052010_en definerer *PennTreebank-tags.*

6.4.7 cststlemma

cstlemma version 4.0 (Ida: /var/csttools/bin/cstlemma)
Som taggeren er lemmatiseren en videreudvikling af den eksisterende lemmatiser, så den kan håndtere xml og spanGrps. I inputtet til lemmatiseren kræves at selve ordet bibeholdes, at POS-tagget indsættes som et *pos-attribut* og der indsættes et tomt *lemma-attribut*. Det gøres i en præproces. En postproces omformer id og fjerner ordformerne og POS-tags igen.

6.4.8 Optionsfiler for cstlemma

I optionsfilerne defineres input og outputformat samt ordbøger og regler som lemmatiseren skal bruge. Lemmatiseren er trænet på STO og det tagsæt der bruges til beregning af lemmaerne er det samme som POS-taggeren bruger. Derfor skal både STO-, PoS-, PAROLE- og lemmatiserens tags (de morfosyntaktiske oplysningstyper) ændres hvis man ændrer i POS-tagsættet.

- lemma-options_14062010_da definerer dansk lemmatisering.
- lemma-options_16062010_en definerer engelsk lemmatisering.

6.4.9 DSN termtagger (termtagger_loglikelihood_TEIP5.pl)

Genererer en spangroup med termhood-værdier for samtlige tokens i input. Termhood-værdierne beregnes på basis af frekvensoplysninger i et almensprogligt referencekorpus sammenholdt med frekvensen af den givne token lokalt i inputdokumentet. Værdierne beregnes med den statistiske metode log-likelihood.

Input: dokumentets token-lag og termlag (sidstnævnte indeholder dummy-værdier)
Output: dokumentets termlag (nu med de aktuelle termhood-værdier)

Valg af referencekorpus: Referencekorpusset med de almensproglige ordfrekvenser er en stor samling nyhedsartikler fra 2007 (i alt ca. 90 mio. løbende ord) som er leveret til Dansk Sprognævn af Infomedia. Årsagen til at vælge dette korpus frem for fx Korpus 2000 er at de almensproglige tekster helst skulle stamme fra samme periode som de fagsproglige tekster.

Normalisering: alle tokens i input omformes til lower-case. Det sker da der ikke skelnes mellem store og små bogstaver i MySQL-databasen med de almensproglige ordfrekvenser, og da det erfaringsmæssigt giver mere

pålidelige resultater at slå varianterne sammen. Årsagen til at arbejde med tokenlaget i stedet for lemmalaget er at referencekorpusset ikke er lemmatiseret og at der ved lemmatisering erfaringsmæssigt kan gå vigtige fagsproglige karakteristika tabt. Fx har passivformer generelt en større hyppighed i fagsproglige tekster end i almensproglige tekster, og fagsproget kan i visse tilfælde have ”monopol” på passivformen af et givet verbum.

Udfaldsrum for termhood: -∞ - ∞

Valg af statistisk metode for termhood: kollokationsstatistik med såkaldte ”contingency tables” (jf. http://collocations.de/AM/index.html) anvendes ofte i den datatmatiske terminologi til automatisk termgenkendelse (ATR). Vi har valgt formlen log-likelihood, da denne formel er mere ”konservativ” og tillægger lav-frekvente fænomener lidt mindre vægt end fx den alternative formel log-odds. Log-likelihood inddrager både de *forventede* frekvenser og de *observerede* frekvenser i reference- kontra analysekorpusset, mens log-odds kun tager de observerede frekvenser i betragtning. Erfaringsmæssigt giver log-odds derfor lidt for meget støj i form af fx stavefejl. En fremtidig version af term-taggeren burde imidlertid generere termlag med begge formler, da de kan supplere hinanden.

Negativ termhood: Hvis den relative frekvens af en given token er større i det almensproglige korpus end i det fagsproglige korpus ganges termhood med -1. Dermed kan termhood både bruges til at identificere tokens som er særligt karakteristiske for de fagsproglige tekster (høje, positive termhood-værdier) og tokens som er særligt ukarakteristiske (underrepræsenterede) for de fagsproglige tekster (høje, negative termhood-værdier).

Neutrale tokens: Termtaggeren tildeler alle tokens der har en længde på ét eller to tegn termhood-værdien 0. Talord og ikke-ord (ord som alene indeholder ikke-alfanumeriske tegn) tildeles ligeledes termhood-værdien 0.