Hands-on session #2 – CorpusStudio: Adding features and preparing the database

Lesson content:

1)	CorpusStudio: query adaptation	(15:30 – 15:50)
2)	CorpusStudio: feature types	(15:50 – 16:10)
3)	CorpusStudio: feature calculations	(16:10 – 16:30)
4)	CorpusStudio: over to your query!!	(16:30 – 17:30^{+})
5)	Sharing results	(17:30 – 18:00)

1 CorpusStudio: query adaptation

1) Use the query of this morning
   a) Load the corpus research project (LongdaleTest.crpx) you created this morning (Use File/Recent to get it)
   b) Go to the query tab
   c) We would like to look for more types of complements, so make these changes:

```
  (: make a variable containing $rc-child WHNP, WHPP or WHADVP :)
  let $compl := $rc/child::eTree[ru:matches(@Label, 'WHNP|WHPP|WHADVP')][1]
```

   d) This will yield complementizers *that*, *who*, *in which*, *why*, *where* and so on
   e) We also want to have relative clauses *without* complementizer
      i) How are they characterized? → SBAR only has one child, "S"
      ii) Create a variable that counts the number of SBAR children:

```
  (: Count the number of children of $rc :)
  let $rcChildren := count($rc/child::eTree)
```

      iii) Adapt the "where" condition to include complementizer-less RCs:

```
(: Make sure only the correct ones match :)
 where
 (
   exists($preNP)
   and exists($rc)
   and (
         exists($compl)
         or ($rcChildren = 1)
       )
 )
```

   f) We will now want the subcategorization to identify the types of relative-clause complementizers, so change the `let $cat :=` and `let $cat2 :=` into this:

```
  (: Subcategorization makes use of constituent order :)
  let $cat := if ($rcChildren = 1) then 'Simple'
              else $compl/@Label
```

   g) You should have deleted the line where `$cat2` is defined, which means that the last "return" statement is no longer valid (it refers to an unknown `$cat2`).
      What happens if you try to execute the query anyway?
      i) Try F10
      ii) You should get an error like this:

```
Error executing query temp_npRCtest:
Line [68]: XQuery static error in #...ru:back($search, $db, $cat2) }#:
    Variable $cat2 has not been declared
Code={http://www.w3.org/2005/xqt-errors}XPST0008
```

iii) Note what the error indicates:
   (1) The query it is executing: `temp_npRCtest` – this is a temporary query, consisting of `npRCtest` as well as all definitions that are called from this query. So it includes the definition file `npRCtest_def`. This means the *error* could be either in the query or in the definition file. Well, we know it is in the query!
   (2) The point where the query execution met a problem: the line "`return ru:back…`"
   (3) The problem it encountered: "`Variable $cat2 has not been declared`".

iv) Repair the query by changing `$cat2` into `$cat`.

2) Execute the query (F10)
   a) The results will contain four "subcategories":
      i) Simple → 96 results
      ii) WHADVP → 29 results
      iii) WHNP → 147 results
      iv) WHPP → 11 results

2 CorpusStudio: feature types

There are different kinds of features we may want to get for our queries, and each of these types requires slightly different treatment. The handout has some examples.

Suppose we want to add the following features:

- The determiner (if any) in the `$preNP` constituent
- The number of constituents in the complementizer `$compl`
- The depth of the relative clause `$rc`
- A guess of the function of the whole noun phrase: is it subject, object or something else in the main clause?

1) Find the feature calculation function
   a) Note in tab page "Query editor" the line that calls the feature calculation function:

```
  (: Calculate features :)
  let $db := tb:GetFtnpRCtest($preNP, $rc, $compl)
```

   b) The function `tb:GetFtnpRCtest()` has been created automatically by the query wizard, and is located in the definition tab page under "`npRCtest_def`"
   c) Go to the definition tab page and find the function
   d) It has three arguments:
      i) `$nd_preNP` – a copy of `$preNP`, the `NP` that heads the relative clause
      ii) `$nd_rc` – a copy of `$rc`, the relative clause `SBAR`
      iii) `$nd_compl` – a copy of `$compl`, the `WHNP`/`WHPP`/`WHADVP` under `$rc`

2) Find how the current features are calculated
   a) This is done using the function `ru:NodeText()`

3) How are features returned to the main query?
   a) Features need to be *strings* (words or sentences)
   b) Feature need to be separated by a semicolon
   c) The function returns *one big string* to the main query by joining everything using the Xquery function `concat()`

3 CorpusStudio: feature calculations

Recap. These are the features we want to add:

- PreDet - The determiner (if any) in the `$preNP` constituent
- ComplSize - The number of constituents in the complementizer `$compl`
- RCdepth - The number of "S" layers inside the relative clause
- NPfunction - A guess of the function of the whole noun phrase: is it subject, object or something else in the main clause?

1) Calculating [`$feat_PreDet`]

```
(: Calculate the preNP determiner :)
let $det := $nd_preNP/child::eTree[@Label='DT'][1]
let $feat_PreDet := if (exists($det)) then ru:conv(ru:NodeText($det), 'Lcase')
                    else '(none)'
```

2) Calculating [`$feat_ComplSize`]

```
(: Calculating the size of the complementizer :)
let $feat_ComplSize := ru:words($nd_compl)
```

3) Calculating [`$feat_RCdepth`]

```
(: Calculating the depth of the relative clause :)
let $feat_RCdepth := count($nd_rc/descendant::eTree[@Label='S'])
```

4) Calculating [`$feat_NPfunction`]

```
(: Guessing the function of the NP we are part of :)
(: step 1: get the NP we are part of :)
let $np := $nd_preNP/parent::eTree
(: step 2: check out the parent type of this NP :)
let $npParent := $np/parent::eTree
(: step 3: our guess depends on this parent :)
let $feat_NPfunction :=
  if ($npParent/@Label = 'VP')      then 'obj'
  else if ($npParent/@Label = 'S')  then 'sbj'
  else if ($npParent/@Label = 'PP') then 'PPobj'
  else                                   'other'
```

5) Adapting the "return" statement
   a) The return `concat()` statement needs to contain the four new features:

```
return concat($feat_TextOfpreNP, ';',
       $feat_TextOfrc,     ';',
       $feat_TextOfcompl,  ';',
       $feat_PreDet,       ';',
       $feat_ComplSize,    ';',
       $feat_RCdepth,      ';',
       $feat_NPfunction         )
```

4 Over to your project

Turn to the the corpus research project you have started working at this morning.

1) Feature addition
   a) Think of features you would like to add to your project.
   b) If needed: discuss with one another or with one of the consultants
   c) Try to make features of different *types* (as on the handout)

2) Feature calculation
   a) Add feature calculation code in the definition file for your corpus research project
   b) Adapt the "`return concat()`" statement to include your features

3) Execute the corpus research project
   a) Do Tools/Execute or press F10
   b) Check the returned feature values in the results

5 Sharing results

Time permitting, we will share our results and experiences.