# EXCOTATE: An Add-on to MMAX2 for Inspection and Exchange of Annotated Data

*Tobias STADTFELD* *Tibor KISS*
Sprachwissenschaftliches Institut
Ruhr-Universität Bochum
stadtfeld@linguistics.rub.de, kiss@linguistics.rub.de

ABSTRACT

In this paper, we present an add-on called EXCOTATE for the annotation tool MMAX2. The add-on interacts with annotated data stored in and spread over different MMAX2 projects. The data can be inspected, revised, and analyzed in a tabular format, and will be reintegrated into MMAX2 projects afterwards. It is based on Microsoft Excel with extensive usage of the script language Visual Basic for Applications.

KEYWORDS : ANNOTATION, INSTANCE-BASED REPRESENTATION, MMAX2, EXCEL, ADD-ON

## 1 The Scenario

Annotation mining sometimes requires the manual annotation of large data sets, such as the annotation of preposition senses (cf. A. Müller et al., 2011). These annotations have been carried out with the annotation tool MMAX2 (C. Müller and Strube 2006). It allows the declaration of multiple layers and stores data in an xml-standoff format. As the annotation scheme for preposition senses is itself subject to evolutionary development, already annotated data require manual re-annotation or correction. It is thus necessary to inspect, and quite often also to re-annotate already processed sentences containing prepositions under investigation.

Due to the design of MMAX2, the search for particular annotations in the vast amount of data is cumbersome. In some applications, searching becomes practically impossible since the data has to be spread over various MMAX2 projects to allow efficient processing within MMAX2.

To overcome this inherent problem of MMAX2 and to allow inspection and analysis from a different – tabular – perspective, we have developed an annotation add-on to MMAX2 based on *Microsoft Excel* (Microsoft Corporation, 2010). It is used to search for and correct instances that require a do-over of their current annotation. The name of the add-on – EXCOTATE – reflects its basis as well as its purpose.

## 2 General Requirements and Advantages of Microsoft Excel

With MMAX2 at hand, we did not intend to create a new stand-alone annotation tool, and hence introduce an additional data format. Such a format would require the synchronization of existing MMAX2-project files, as well as the duplication of the scheme files that describe the encoding rules in an MMAX2 project. Due to the frequency by which changes in the annotation scheme are conducted, this would have resulted in high maintenance and was therefore considered unacceptable.

The choice of Microsoft Excel as a basis for an add-on was guided by several considerations. Using commercial spreadsheet software might be considered controversial, but it offers various advantages over a "from the scratch" approach. Many recent tools in computational linguistics have been implemented and maintained by (PhD) students. This has the major disadvantage that support, bug fixing and the development of extensions come to a rapid end when the students in question move on to different tasks. This disadvantage can be at least partially circumvented, if one chooses popular and widely common software as a foundation and reduces further custom modifications to a minimum.

Besides these rather general thoughts, Microsoft Excel offers several technical advantages making it almost ideal for annotating instance-based data:

*Data validation* is a core functionality of Excel, ensuring that an annotator can only set values for attributes that have been defined legitimate.

*Sorting* and *filtering* are also core functions and quite helpful if one wants to investigate the data in more detail; a task which in most annotation tools can only be fulfilled, if at all, when using a search query with its own syntax to be learned upfront.

Further, *exporting* data from a spreadsheet to a data-mining tool is almost trivial, as all popular tools provide CSV-support.

Finally, Microsoft Excel offers the possibility to encode hierarchical graphs using the script language *Visual Basic for Applications* (Microsoft Dynamics, 2006). With the help of dynamically created VBA code, changes to one attribute lead to automatic changes of dependent attributes.

## 3 Description of the Data and Scripts

In the following, we will offer descriptions of the input data, the scripts that are used to automatically build EXCOTATE instances from the data and the functionality of EXCOTATE itself.

### 3.1 MMAX2 Projects and Scheme Files

In light of performance limitations and experienced drawbacks in lucidity when opening too many sentences within the annotation tool, we decided to restrict individual MMAX2 projects to a maximum number of 50 sentences. All data are stored as MMAX2 projects after a completed processing step; all files created besides MMAX2 projects are mere temporary files.

The meaning of the preposition is the key-feature in the annotation task. Hence, it forms the basis for an instance-based representation of the data in a shallow spreadsheet. Every MMAX2 project contains an xml file that declares all tokens contained in this particular project along with an id. All additional layers of information are optional. We declared several layers, two of which are the sentence and the preposition-meaning layer. Each layer is stored in an individual xml file, in which the *ids* of the words spanned by this particular layer are being named along with attributes and their values. Figure 1 exemplifies this for the preposition-meaning layer.

```
<markable id="markable_1" span="word_298" mmax_level="prep-meaning" modal="+" local ="na" (…) />
<markable id="markable_2" span="word_1373" mmax_level="prep-meaning" modal="na" local ="na" (…) />
```

FIGURE 1 – Sample of two markables in a preposition-meaning-markable file.

The attributes for annotation in a MMAX2 project are declared within the correspondent scheme file of a layer. Here, the availability of specific attributes can be defined as being dependent on another attribute showing a particular value. This results in the provision of decision trees in which the annotator may choose the next steps for a precise interpretation of a preposition (cf. A. Müller et al., 2011). For example, *he is working at a school* would be a candidate to be classified as *local* (mother node) in general and also as (localized at an) *institution* (child node). We currently use sub-trees with a depth of one to a total depth of over seven chained attributes and their respective values in one tree (cf. A. Müller 2012).

### 3.2 Creation of an EXCOTATE Instance

The creation of an EXCOTATE instance containing the information of several MMAX2 projects should be automated. It therefore requires some sort of script, which acts as a transformation between both programs. To create Excel files from within a script, we decided to use the Perl module *Spreadsheet::WriteExcel*[1]. This module offers support for all necessary functions within Excel, as data validation, auto filtering and the automated insertion of VBA code into an Excel worksheet, and the ability to declare write-protected cells. We create *xls* as well as *xlsx* versions.[2]

[1] Available on www.cpan.org
[2] See module *Excel::Writer::XLSX* for xlsx support.

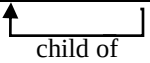
To create an EXCOTATE instance, the first step is to generate a list of all MMAX2 projects to be included in such. Successively, every project is processed by the following steps.

We decided to annotate exactly one preposition per sentence to allow for an instance-based representation of the data. Therefore, the first information to be written into every row into the first of two sheets is the string of the preposition and also a simple continuous number as an id. In addition, the path to the root directory of the MMAX2 project the data originated from is also stored. This step obviously differs in minor details depending on the data to be annotated but the remaining script is generic. If additional layers from a MMAX2 project shall be included into EXCOTATE, these layers have to be declared as parameters and the respective scheme files need to be parsed.

As the scheme files include the declaration of all possible attributes, it is a straightforward task to initialize each attribute as a column and read the respective value of this attribute in the affiliated MMAX2 file. If the attribute is not set, a default value (*na = not applicable*) is used. Figure 2 gives a simplified overview of the first rows of an EXCOTATE worksheet.

| *Basic information* | | | *meta-information layer* | | *preposition-meaning layer* | | |
|---|---|---|---|---|---|---|---|
| id | mmax_path | preposition | annotation status | comment | local | local_ extension | modal |
| 1 | über\match_1-50 | über | done | interesting | na | na | + |
| 2 | über\match_1-50 | über | problem | idiomatic | local_ext | institution | na |

FIGURE 2 – Sample of first worksheet.




In our annotation task the layer *meta-information* is used to mark the status of the annotation and to set a comment if needed. The comment attribute is declared as a free text attribute in the scheme file and as such does not possess a default value. The *preposition-meaning* layer lists all attributes one can use for the meaning of the preposition in a given context. We decided to output the context in which the preposition occurred as a string value in an additional attribute column named *context* in the basic information section (not included in Figure 2), while marking the pertinent preposition in the string by an unique symbol (**).

Annotators err during annotation, especially when annotating vast amounts of data. Therefore, we make use of the data validation functionality of Excel by setting a list of legitimate values for every column (attribute). The list can easily be obtained from the already parsed scheme file of the layer the attribute belongs to. Free text attributes, such as the comment attribute, are automatically recognized as such and therefore excluded from this functionality.

The data validation function adds a comfortable drop-down menu to all attributes with more than one value, allowing an easy and error resistant adjustment of the values by an annotator. The first row of the Excel sheet has to be write-protected, as the header should never be changed. In addition, we add the functionality of an *autofilter* to the header, allowing an annotator to only show rows that contain specifically chosen values for an attribute. The *annotation status* attribute is frequently used in this fashion to display only those cases in which the annotation is marked as problematic and therefore should be revised.

As we want to reintegrate all layers included in the sheet back into the original MMAX2 projects, we also need to store the position of the *markables* defined in the MMAX2 files. To do so, we store the information on every span of every *markable* in a second worksheet, but in the same cell the attribute of this *markable* is saved. An ongoing id in both sheets is needed due to the

possibility of sorting the rows of the first sheet and is therefore used to keep both sheets synchronized. As the basic information is used for other purposes than storing annotations, we do not need to store any values on their position. The information on the position of a markable can contain only one token or span over multiple words. To maintain the functionality, the second worksheet is never to be changed and is declared write protected as a whole to hinder accidental changes.

Data validation, auto filtering etc. are functions directly amenable from the utilized Perl module. However, when it comes to coding dependencies between specific attributes and their values, these functions reach their limits. Dependencies between attributes occur frequently in our project, since some of our defined meanings show a hierarchical structure. If a most specific branch (which corresponds most closely to specific senses of a preposition) in a decision tree is changed, its mother(s) may change as well. Hence, we would like Excel to automatically adjust all influenced attributes to their corresponding values. So to speak, child nodes should rectify their parents' values if changed. Similarly, changes at a mother node will influence daughter nodes, and this should be reflected as well.

To fulfill these requirements, we analyzed the hierarchies encoded in the scheme files and stored for every attribute (and its values) its respective parent node. In addition, we internally stored which column position holds which attribute. The next step was to automatically create VBA code, which would capture these hierarchies and could be inserted into the Excel sheet to be used during annotation. A working sample of this code is exemplified in Figure 3.

```
Option Explicit
Private Sub Worksheet_Change(ByVal Target As Range)
  'error handler
  On Error GoTo ErrorHandler
  'disable events
  Application.EnableEvents = False
  'how big is the sheet?
  Dim number_of_rows
  number_of_rows = _
  ActiveSheet.Cells(Rows.Count,1).End(xlUp).Row
  'update the attributes
  'first row is header and therefore excluded
  If Target.Row > 1 And number_of_rows >= Target.Row Then
        hierarchy_update Target
  End If
  ErrorHandler:
      Application.EnableEvents = True   're-enable events
End Sub
```

```
Sub  hierarchy_update(ByVal Target As Range)
    (...)
    If Target.Column = 7 Then
            update_col_7 Target
    End If
    (...)
End Sub

Sub update_col_7(ByVal Target As Range)
    Dim row As Long
    'only rows changed will be checked
    For row = Selection.Row To _
    Target(Target.Cells.Count).Row
      If Not Cells(row, Selection.Column)._
      EntireRow.Hidden Then
          Cells(row, 6).Value = "local_extension"
      End If
    Next row
End Sub
```

FIGURE 3 – Sample of VBA code to handle hierarchical dependencies between attributes.

While the first subroutine (*Worksheet_Change*) is static, the other routines are created dynamically, depending on the hierarchy coded in the scheme files and the overall number of attributes involved. When a change in the Excel sheet is detected, the subroutine of the changed column is called. For all rows, which have been involved in the change, all parent and child nodes of this particular attribute are set to the logical correct value.

### 3.3 EXCOTATE

After creating an Excel file using the aforementioned script, an annotator can directly use it to re-annotate the data. Annotators are allowed to hide columns they do not need during annotation, e.g. preposition meanings never being used with the preposition at hand, but should of course

never delete columns. Also rearranging the columns and the use of copy and paste are problematic. If an annotator wishes to change the value of an attribute on multiple rows, he or she can use the mouse to drag-fill the focused cells.

Even with over 135 columns and several thousands instances, Excel does not show any mentionable performance issues.

### 3.4 Reintegration of EXCOTATE

After correction within EXCOTATE, the sheet needs to be reintegrated into the original MMAX2 projects. Fortunately, reintegration can simply be achieved by completely recreating the layers with the information at hand. Layers not included in the EXCOTATE file, but present in a MMAX2 project remain untouched and are in no need of alteration.

## Conclusion and perspectives

We have described the creation of Excel files from MMAX2 projects in generic terms. Over time extensions have been added to meet specific requirements unique to our project. However, we see EXCOTATE as a quite useful supplement to annotate data in general.

There are some types of annotations that are not as easily transformed from MMAX2 to EXCOTATE and vice versa. Complex annotations, such as syntactic or semantic structures cannot be stored in a flat data representation as it is the case within EXCOTATE. Only with some restrictions and therefore special treatment of these layers, information on layered structures can be processed. Since these are the type of annotations MMAX2 was developed for in the first place, information on these layers are only stored in an EXCOTATE instance when it is crucial input for a statistical analysis.[3]

## References


Kiss, T., Keßelmeier, K., Müller, A., Roch, C., Stadtfeld, T. and Strunk, J. (2010). A logistic regression model of determiner omission in PPs. Paper for *Proceedings of Coling 2010*. Beijing, China.

Microsoft Corporation. (2010). *Microsoft Excel 2010 Product Guide*.

Microsoft Dynamics. (2006). *VBA Developer's Guide. Release 9.0*.

Müller, C. and Strube, M. (2006). Multilevel annotation of linguistic data with MMAX2. In Sabine Braun, Kurt Kohn, and Joybrato Mukherjee, (eds.). *Corpus Technology and Language Pedagogy: New Resources, New Tools, New Methods.* Peter Lang, Frankfurt a.M., pages 197-214.

Müller, A., Roch, C., Stadtfeld, T. and Kiss, T. (2011). Annotating spatial interpretations of German prepositions. In: O'Conner, Lisa (ed.): *2011 Fifth IEEE International Conference on Semantic Computing*, pages 459 - 466. Stanford, CA.

Müller, A. (2012). Location and Path – Annotating Senses of the German Prepositions *auf* and *über*, submitted to Workshop on Semantic Annotation and the Integration and Interoperability of Multimodal Resources and Tools. LREC 2012.


[3] A detailed description of the statistical analysis of the data is to be found in (Kiss et al., 2010).