This is a repository copy of Web-based Annotation Tool for Inflectional Language Resources.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/127399/

Version: Accepted Version

**Proceedings Paper:**

**Reuse**
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

**Takedown**
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.
Web-based Annotation Tool for Inflectional Language Resources

Abdulrahman Alosaimy, Eric Atwell
School of Computing, University of Leeds
EC Stoner Building, Leeds LS2 9JT
{scama, e.s.atwell}@leeds.ac.uk

Abstract
We present Wasim, a web-based tool for semi-automatic morphosyntactic annotation of inflectional languages resources. The tool features high flexibility in segmenting tokens, editing, diacritizing, and labelling tokens and segments. Text annotation of highly inflectional languages (including Arabic) requires key functionality which we could not see in a survey of existing tools. Wasim integrates with morphological analysers to speed up the annotation process by selecting one from its proposed analyses. It integrates as well with external POS taggers for kick-start annotation and adaptive predicting based on annotations made so far. It aims to speed up the annotation by completely relying on keyboard, with no mouse interaction required. Wasim has been tested on four case studies and these features proved to be useful. The source code is released under the MIT license1.

Keywords: annotation, inflectional language, morphosyntactic, Arabic

1. Introduction
POS tagging text in inflectional languages is usually harder. A typical problem is substantial lexical data sparseness due the high number of possible inflections of a single word. To reduce sparseness and number of Out-of-Vocabulary (OOV) words, inflected words are often segmented prior or in parallel with POS tagging. However, the segmentation process is prone to errors. An inflection boundaries are often not marked which increases the number of homographs (two or more words spelled in the same form but has different POS tag or pronunciation (e.g. diacritization)). Some orthographical changes are caused by an inflection, making it hard to recover original word form. As a result, segmentation process sometimes fail to detect morphemes.

Wasim is a web-based tool for semi-automatic annotation of text for the purpose of gold standard corpus production. It was developed for the annotation of Sunnah Arabic Corpus (SAC) (A Alosaimy & Atwell, 2017), a collection of classical Arabic prophet sayings. It has also been tested in four case studies.

For the project, we analysed the required set of features needed for annotating SAC and used this as a criteria in a survey of existing tools. In our search of current available tools, we considered tools that 1. Is Web-based: to integrate it with other systems, and to allow easier access through browsers. 2. Annotates text tokens with morpho-syntactic tags in CoNLL-U v.2 format (Nivre & Agic, 2017)2. 3. Supports right-to-left languages. 4. Is available to download for research purposes.

Morphosyntactic annotation of SAC (and highly inflectional languages as well) requires additional specialized functionality:

1. Segmentation of one word into a set of segments
2. Addition of orthographical accents or diacritics
3. Listing a set of solutions from a lexicon dictionary (internally or externally using a morphological analyser)
4. Consistency validation and integrating annotation guidelines (e.g. homographs).

In the next section, we provide a detailed overview of major related tools for annotating corpora, with a tabular comparison with Wasim.

2. Related Work
We limit our literature review to tools that meet our four conditions, which results in five tools. Brat (Stenetor et al., 2012) is a widely-used visualization and annotation tool that is mainly for syntactic annotation in addition to morpho-syntactic annotation. WebAnno (Yimam, Gurevych, de Castilho, & Biemann, 2013) is Java-based set of well-documented tools for multiple annotation tasks. Arborator (Gerdes, 2013) a dependency annotation tool, that supports RTL languages natively. Sequence Annotation Web Tool (Samih, Maier, & Kallmeyer, 2016) is a basic web-based tool for the annotation of token sequences with an arbitrary set of labels (e.g. POS tags). The author promised to publish the code on github, but we could not find a link to it, so we exclude it from the table comparison. CorA (Marcel Bollmann, Florian Petran, Stefanie Dipper, 2014) is a web-based tool for morpho-syntactic annotation of non-standard texts.

Although these tools did not meet all of our requirements, we must say that they support different features (e.g. syntactic annotation) that are not needed in our project.

<table>
<thead>
<tr>
<th>Features</th>
<th>Brat</th>
<th>WebA</th>
<th>Arb</th>
<th>CorA</th>
<th>Wasim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment one word into segments</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Diacritics</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Suggest a set of solutions from a lexicon dictionary</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Consistency validation</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adaptive predicting based on historical tagging</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

1 http://github.com/aosaimy/wasim
2 CoNLL-U format has been used in Universal Dependencies project and well described in http://universaldependencies.org/
3. Major features

The annotation of text in a highly inflectional language is usually harder because:

1. Words are highly ambiguous, which results in many homographs (i.e. more need of a lexicon),
2. Words need to be segmented into a set of morphemes, and
3. As a result, tagger’s performance is usually poorer, and mostly rely on a lexicon or a morphological analyser to improve the accuracy.

Semi-automatic annotation should help removing the ambiguity of words as it should be able to correct tagger’s errors. Many times, these errors are in the ranking of the solution set. Therefore, the most needed feature is the integration of a morphological analyser, which allows the annotator to re-select the proper analysis. In addition, an efficient way to segment words into a set of morphemes is a necessity.

3.1 Morphological Analyser Integration

Wasim integrates with morphological analysers to speed up the process of annotation. Morphological analysers takes a word as input, and produces a list of possible analyses (which include word’s segmentation, segment’s POS tag and features). By providing a set of possible analyses, Wasim allows annotators to select one analysis. Once a solution is chosen, all its values of POS tag, lemma, segmentation, and morphological analysers will be reflected on the word analysis. The chosen solution can be edited though.

However, this hypothesis of course depends on the quality of the morphological analyser. In our annotation project, if no analyses is correct, we found it more efficient to choose a similar analysis instead of starting from scratch. Annotators have to mark all features though if the analyser returns no results. Once an analysis is saved, it will be saved in the server for possible later requests.

Wasim provide two ways of morphological analyser integration: First, Using an embedded supplementary tool that acts as a simple lexicon memory: It reads the annotated part of the corpus and index words with their annotations. Then, it allows HTTP requests to be made from Wasim, and it will return all possible solutions of the token in hand.

Second is using external morphological analyser. However, since the output is not standard between analysers, analyser outputs should be converted into CoNLL-U like format. A mapping between its tagset and the project tagset may be required, and this can be easily defined in the configuration. If the mapping result in an ambiguous tag in the project’s tagset, Wasim will duplicate the analysis for each possible tag. For example, if NOUN is mapped to PN and N, two analyses will be presented to the annotator.

3.2 Consistency Reinforcement

Consistency (a.k.a. “stability” when measuring consistency of one annotator alone over time) of corpus annotation process is important to ensure that all annotators in all texts follow the same procedure of annotation over time. High consistency means too little disagreement in the annotation, and this helps training machine learning algorithms successfully.

To increase the consistency of the segmentation and tagging of a corpus, Wasim followed three approaches: First, it allows the use of automatic POS tagger. Second, it integrates with morphological analysers. Third, it generates periodically a list of common homographs. Homographs are associated with their possible POS tags and segmentation. Possible segmentations are only shown when the token in hand is a homograph.

Usually in annotation guidelines, there is some guides of specific words, usually homographs. However, in highly inflectional languages, those homographs are overwhelming, and such offline guideline may miss some homographs, or guidelines will be lengthy. This ingenerated serves as an online guideline for annotators, which is automatically built up.

In the segmentation layer, Wasim warns the annotator when a segmentation of a word differ from previous segmentations of the same word. If the annotator insists, its new segmentation will be added. Similar pattern is happening for morphological tagging.

The list is generated periodically from the annotated part of the corpus, and the possible segmentations/POS taggings of homographs are kept. Each homograph will have a set of examples in context for each sense. Moderators can add guideline notes of tagging such cases. The list will appear in Wasim with its notes, when selecting a word in the list.
### 3.3 POS tagging Integration

![Figure 2: A screenshot that shows Wasim in a browser. The middle part represents one sentence where each box is a token (with its XPOS tag). Tokens of inflected word are linked by + symbol. The left side shows feature annotation. The top bar represents actions such “save file” and “undo last action”. On the right side, CoNLL-U synchronized representation of the sentences.](image)

Instead of starting the annotation process of a corpus from scratch, Wasim integrates with UDPipe (Straka & Straková, 2017) to kick start the annotation process. UDPipe provides trained models for more than 60 languages that tokenize, tag, lemmatize and dependency parse raw text and save results in CoNLL-U formatted files. Wasim use UDPipe improve prediction model by periodically adding instances of the corpus that has been annotated so far.

Other tools can be used as long as they generate CoNLL-U formatted files. For example, SAWAREF toolkit can be used for Arabic and the translation from popular POS tagger into CoNLL-U format can be done using one of its tools.

### 4. Date Representation

Wasim follows the Universal Dependencies v 2.0 (UD) (Nivre & Agic, 2017) in the way it represents sentence segmentation, POS tagging, morphological features, segmentation, and lemmatization. All annotation is stored as CoNLL-U files, which can be downloaded anytime. Since Wasim does not annotate syntactic relationships, related columns are marked as missing.

Unlike some other representations, CoNLL-U is morpheme-based tagging with the ability to recover the original word form prior to segmentation. In addition, each morpheme has two POS tags; one from coarse universal tagset (UPOS) and one from author’s defined fine-grained tagset (XPOS). This enable sharing and comparing of cross-linguistically consistent grammatical annotation of more than 100 treebanks available in UD project. CoNLL-U format serves two purposes; a well-formed structure for saving annotations (like XML) and as a high-level guidelines for morphological tagging.

UD do not have a standard for diacritization. We followed our project’s representation of diacritization of Arabic³. Wasim allows users to enforce such representation by performing a series of transformation using “regex” expressions. Moderators can enforce similar approach for diacritization in the comments part of the sentence.

### 5. Tool Description

Wasim tool has mainly two components: a front-end interface which allows interact with annotator and provide warnings and feedback, and a back-end server that manage sessions and storage.

The front-end web-based tool is built using Ionic framework using Typescript/Javascript programming language. The main screen contains four sections: toolbar at the top is used for warnings, helpful shortcuts, and for a glance of shortcuts. The rest is separated into three columns. The middle column shows the words in small boxes (with its tag beneath it) with the current word in process highlighted in a different colour. Words show its morphemes linked by a “+” symbol. Instead of displaying words in a tabular format (like in CorA, SAWT), we display words in natural paragraph flow; allowing annotator to easily examine word’s context. The left column shows key-value pairs of lemma, and morphological features and the right column shows the synchronized Conll-U format of current tagging. Closed features are a dropdown list with an auto complete feature. In addition, two useful subviews are displayed on demand: A. a list of other alternative solutions can be shown by pressing “Enter” key. Once opened, one solution can be chosen from the list. B. a tabular format of morphological features and possible values. The front-end of Wasim can be seen as an Conll-U file editor: it parse the file, validate the syntax and visualized the sentences with a synced side by side view of Conll-U file. Figure 2 shows a screenshot that shows the main components of Wasim.

The back-end is a server operated using Node.js Express server, and is responsible of authentication and managing annotated and raw files. A connection with server using WebSocket is established for the purpose of several requests, such as morphological analyser requests, sessions, diacritization requests, and temporary session backup.

Each project is a folder in the system that contains document files, configuration files, a database of homographs and a file of the corpus lexicon. It manages the versioning of files using the famous Git version control system. Git system tracks all the changes that are made to files, and allow multiple operations, e.g. *diff* to show changes to a file in colorful interface.

All annotations are stored in CoNLL-U format as plain text files. Accessing one file from an annotator will grab a copy of that file; however, this might allow other annotators to work on the same file. To prevent that, we maintain a simple lock system where a file is locked while a connection is maintained with the server (using websocket). We only release the lock if the annotator accessed another file or the connection is closed.

Wasim is designed to be configurable to support preferences and project related setup. Project setup includes its name, language, remote Git repository, UDPipe model, morphological analyser path and several other preferences. Projects must define their own fine-grained tagset (unless UD tagset is used), with their morphological features. Wasim allows custom key-binding for actions. The configuration files are saved in the project level as JSON files.

The annotation process can be fully manual or semi-manual. In case of semi-manual, the corpus is first tagged using UDPipe models. Automatically generated tags can be then checked and manually edited using Wasim. In the next section, we will describe the supported morphosyntactic layer in more details.

### 6. Morphosyntactic tasks

Wasim provides easy interface for the annotation of six tasks. While these tasks can be processed sequentially, we allow annotators to annotate at any of the tasks at the same time. Tasks are interrelated, e.g. if the automatic tagger produced the wrong POS tag, it might as well produced the wrong morphological segmentation/lemma ...etc. In addition, since Wasim uses morphological analysers, if the annotator chose one solution, it will affect multiple tasks at the same time. Therefore, we allow annotator to edit previous tasks without leaving the screen.

Since Wasim allows to annotate text on many levels at the same time, annotator might skip a task accidently. Wasim provides a guide to go through tasks in keyboard mode. It highlight tasks sequentially to grab annotator’s focus on the current task.

However, depending on the corpus annotation goals and preferences, annotator can customize the view; e.g. deactivates one/multiple tasks, or disables CoNLL-U view. Annotator can write post-process rules to check the validity and consistency of different tasks as well as constraints on different tasks.

CoNLL-U representation on the right side is editable at any time, as Wasim synchronize changes. Changes will be validated and errors are reported on error log box. In case of valid changes, such changes is reflected on the Wasim widgets. This should give an option to the annotator to make changes in bulk like copying previous annotations.

Wasim is designed to increase productivity for these particular annotation tasks, while sacrificing some amount of simplicity (many shortcuts/buttons on the screen). While the learning curve (the rate of a person’s progress in gaining experience) is steep, we hypothesized that Wasim features will improve the time required for annotating one word.

#### 6.1 Morphological segmentation

Inflectional languages tend to reflect morphemes to express different grammatical features. Unlike many other annotation tools, we do not assume the text to be tokenized/segmented. Annotators can easily tokenize words by editing their forms. Word can be segmented as well by placing a pointer, and inserting a special character (“)” sign by default). The two generated morphemes will clone the data from the original word except its form which will be divided. The original word form will remain the same though. The original word in the main screen will be replaced by two morphemes linked by “+” symbol. Annotator can remove segmentation by simply hitting the spacebar in one morpheme, and it will merged to the previous morpheme.

With the integration of morphological analysers, annotators should mostly select the proper segmentation/tagging from its provided list. Manually segmenting one word should be sorted to last choice, the case when there is no proper segmentation.

Since we follow CoNLL-U representation, UD representation keep the form of both the word and the token in its two-level indexing scheme. Form of tokens can be rewritten as if they were standing alone. Free morpheme form can be altered because of the inflection, and annotator can recover its original form, e.g. “John’s” can be recovered to either “john+has” or “john+is”. The original form (John+’s) will be written in the MISC (last) column. The result CoNLL-U will be like the following:

<table>
<thead>
<tr>
<th>1-2</th>
<th>John’s_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>NOUN</td>
<td>N</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>2</td>
<td>has</td>
<td>AUX</td>
<td>BE</td>
<td>0</td>
<td>_</td>
<td>_</td>
<td>ORG+’s</td>
</tr>
</tbody>
</table>

#### 6.2 Diacritization

A diacritic (sometimes called accents or short vowels) is an optional small glyph added to letters to change the sound of the letter. Diacritization is the process of adding those glyphs. In our Sunnah project, we asked for this addition as diacritics reduces the ambiguity of words.

This process is tedious as it requires to move the cursor letter by letter to add diacritics. Since the number of possible diacritization pattern is low, we enable the use of morphological analysers to generate the possible diacritization of a word. The annotation process is then eased by only selecting the correctly-diacritized word. The annotator has the ability, though, to edit the form if no one solution is provided.

Additionally, Wasim uses a diacritization tool that borrows more complete diacritization tool of similar context. This method is different than major diacriticizer as it does not “guess” diacritization, but rather “borrow” it if it exists from similar context. Context can be defined in different
ways: e.g. n-word-grams (Abdulrahman Alasaimy & Atwell, 2018).

Wasim allows moderator to enforce some standard on the diacritization. For example, in Arabic, it can be configured to ignore diacritization of letters proceeded by a long vowel. These transformation rules can be enforced using a set of regular expressions (regex). These rules will only be applied to a subset of morpheme/words that conform to certain conditions. For example, in the guidelines of SAC, we require no diacritization on the letter of the definite article Al-. We had a rule that remove such diacritization of the subset morphemes that has a POS tag: DET.

6.3 POS tagging

POS tagging in Wasim is morpheme-based. We assume that the tag set is assignable to any morpheme regardless of its location (e.g. prefix or base). Tags can be easily chosen from a list of POS tags ordered by their frequency or alphabetically. The most common POS tags are shown at the top, and pressing its associated number will assign it to current in hand morpheme.

6.4 Morphological features

Wasim offers a single auto-complete input line for all morphological features. Once the input get the focus of the user, it shows a drop-down list of all possible values of the chosen POS tag. Once a value is selected (e.g. “MASC” for gender), other incompatible values hide as a consequences. The goal is to speed up the annotation by selecting values in one place and ask for relevant tag only.

6.5 Lemmatization

Wasim offer a simple interface for lemmatization. If it is integrated with a morphological analyser, the lemma of the chosen solution will be assigned. The lemma however can edited manually.

6.6 Sentence Segmentation layer

Wasim provides the ability to alter the text and separate one sentence into two. By convention, ConLL-U format leave an empty line as an indicator of sentence start/end.

7. Case Studies

We provide four case studies to show the use of four languages. In each case, we evaluate one major feature and the effect of that feature on the speed and accuracy.

In each case, we annotate a couple of sentences depending on the case. While the text size is a small and might not clearly show the improvement, these experiments are for illustration purposes rather than to actually measure the difference. In addition, the annotator who has done these four experiments is the author of the tool, therefore, we cancel the effect of the learning curve.

For each case, the text is divided into two halves, H1 and H2, and the both halves are tagged twice (two rounds). In all cases and for both rounds, the annotator is the same person. Both halves are tagged with feature enabled (F=True) and then disabled (F=False) but in different order for each half. The steps are {H1F=True,H2F=False,H1F=True,H2F=True}, and first two steps are first round. In the last two steps, the annotator already knows the texts and should annotate it faster. However, results between step 3 and 4 is comparable as the word counts is similar.

In Arabic cases, we used Quranic Arabic Corpus and asked the annotation to follow its annotation guidelines and the annotator well understand its tagset. UDPipe is trained as well on Quranic Arabic Corpus (Dukes & Habash, 2010) (converted to CoNLL-U by the author and available here). The morphological analysis used here is MADAMIRA and its results is parsed and converted to CoNLL-U format using Sawaref toolkit. A manual mapping from MADAMIRA tagset to QAC is defined and used.

Time and speed are used as metric for efficiency. Speed is reported in morpheme/sec. The Intra-rater reliability is high in all cases which shows that using features does not affect the accuracy. Mismatches between the two rounds are reviewed and corrected in a third round. The accuracy in terms of fraction of correctly annotated words are then evaluated for the two rounds compared with the gold standard (third round). More metrics is reported per case requirement. In all cases, we only evaluate the accuracy of segmentation and POS tagging, although all tasks are done. Diacratization, lemmatization, and other features accuracy are not included.

At the end we show a brief statistics on our Sunnah Arabic Corpus Annotation.

7.1 Modern Standard Arabic and Morphological Analyser

In this case, the annotator used the morphological analyser to select one candidate analysis from a list of proposed analyses. “Uses of MA” report the case of annotators selecting an analysis even though such analysis was corrected later. We report the number of time annotator used and the number he edited the proposed analysis. Clearly the results shows that using MA is helpful in speed and accuracy, but in most cases it is prone to errors. Using MA improved the annotation accuracy and speed significantly.

<table>
<thead>
<tr>
<th></th>
<th>Using MA</th>
<th>Without</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>round 1</td>
<td>round 2</td>
</tr>
<tr>
<td>Word count</td>
<td>50</td>
<td>51</td>
</tr>
<tr>
<td>Morphs count</td>
<td>72</td>
<td>70</td>
</tr>
<tr>
<td>Accuracy</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>Time (secs)</td>
<td>1358</td>
<td>635</td>
</tr>
<tr>
<td>Time (s/m)</td>
<td>18.86</td>
<td>9.07</td>
</tr>
<tr>
<td>Uses of MA</td>
<td>39</td>
<td>43</td>
</tr>
<tr>
<td>Number of edits</td>
<td>30</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 1: Comparison between using with and without MA in accuracy and speed.

7.2 Quranic Arabic and Consistency Reinforcement (CR)

In this case, we show how the warning and helper guidelines help improving the accuracy. Consistency Reinforcement feature used the whole QAC corpus to build

---

4 A regular expression, or regex is famous way to define a search and replace pattern.

5 https://github.com/aosaimy/qac.conllu
We present Wasim, an open-source web-based tool efficiency-oriented for semi-automatic annotation of inflectional languages resources. It supports multiple tasks including segmenting tokens, diacritizing, and labelling tokens and segments. It integrates UDPipe toolkit to kick start the annotation process and can be integrated with a morphological analyser to speed up the annotation process. We reported the improvement in accuracy and time in four cases with different genres and languages.

For future work, we might add support for additional layers for syntax, co-referencing, and named entities.

### 9 Bibliographical References


