A Web-framework for ODIN Annotation

Ryan Georgi **Michael Wayne Goodman** **Fei Xia**

University of Washington
Seattle, WA, USA
{rgeorgi,goodmami,fxia}@uw.edu

Abstract

The current release of the ODIN (Online Database of Interlinear Text) database contains over 150,000 linguistic examples, from nearly 1,500 languages, extracted from PDFs found on the web, representing a significant source of data for language research, particularly for low-resource languages. Errors introduced during PDF-to-text conversion or poorly formatted examples can make the task of automatically analyzing the data more difficult, so we aim to clean and normalize the examples in order to maximize accuracy during analysis. In this paper we describe a system that allows users to automatically and manually correct errors in the source data in order to get the best possible analysis of the data. We also describe a RESTful service for managing collections of linguistic examples on the web. All software is distributed under an open-source license.

1 Introduction

The current release of the ODIN (Online Database of INterlinear Text) database contains over 150,000 linguistic examples in the form of interlinear glossed text (IGT), an example of which is shown in Fig. 1. These IGT instances are extracted from PDFs found on the web, representing a significant source of data for computational typology, as well as providing information for resource-poor languages (RPLs). These instances are additionally useful for inducing annotation on RPLs, as demonstrated by Georgi et al. (2014, 2015), in which the relationships between words and glosses are identified and encoded for the purpose of enriching the data with annotations not present in the original examples. However, the PDF-to-text conversion process can introduce noise into the data, and some examples are not formatted well in the original document. These and other issues can decrease the efficacy of the automatic structural analysis.

```
keené    ʔaksí dónq-ine-m
DEM.PLUR dog   five-DEF-ACC
'these five dogs'
```

Figure 1: An IGT instance of Aari [aiw], an Omotic language of Ethiopia. Extracted from Dryer (2007)

To address these issues, we have created a web interface that combines automatic cleaning and normalization procedures with a user-friendly browser-based GUI to enable human annotators to review and improve the data quality of the available IGT instances. Additionally, as editing is meant as one of multiple capabilities of the final system, this browser interface is driven by a RESTful (Fielding, 2000) backend system that will support future interface extensions.

2 Related Work

The system we describe is not the first web-based editor of IGT, but none of the existing systems (for IGT or annotation in general) that we're aware of fit our use case. TypeCraft[1] (Beermann and Mihaylov, 2014) is a wiki-based collaborative IGT editor. As it directly targets IGT, the editor is designed to support tabular annotations and is a useful tool for users creating new IGT. However, it limits the kinds of annotations (morphemes, POS tags, glosses, etc.) and it is not obvious how the ODIN model (see Section 3) would fit, nor how our automated transformation scripts (see Section 4) would be inte-

[1] http://typecraft.org

grated. The *brat rapid annotation tool*[2] (BRAT; Stenetorp et al., 2012), with its RESTful web API, is somewhat similar in implementation to our system, but does not seem to support tabular visualization of hierarchical annotations. The current annotation task for our users is primarily correcting errors in text extracted from PDFs, which is similar in some ways to how RECAPTCHA (Von Ahn et al., 2008) lets users provide clean transcriptions of text in images. But, unlike RECAPTCHA, our task requires some knowledge of IGT structure.

3 ODIN Data

The data that we are seeking to annotate in this paper comes from the ODIN 2.1 data release,[3] which provides the data in two formats: the plain text format, and in Xigt, an extensible format for encoding IGT (Goodman et al., 2014).

3.1 Building ODIN

The ODIN database was constructed in several steps. First, documents were retrieved using a meta-crawling approach, where queries with IGT-like search terms such as {3SG, ACC} were directed to online search engines. The resulting documents were converted to text format using an off-the-shelf PDF-to-text conversion tool. This text output was subsequently used to extract features to detect the IGT instances within the text, as well as build a language identification system. The full details of the construction of the ODIN database can be found in Lewis and Xia (2010).

3.2 Extracted Textual IGTs

The first format for representing IGT is intended to be human-readable while maintaining the appropriate metadata for processing. An example of this format can be seen in Fig. 2. This format includes the textual content of the IGT, as well as whether a line belongs to the language line (L), gloss line (G) or translation line (T) of the instance, or whether it is metadata (M). In addition, secondary tags exist for more fine-grained categorization, such as for corrupted instances (CR), language name metadata (LN), etc. Furthermore, a doc_id is provided for reference to the original source document, as well as line numbers referring to the lines from the pdftotext[4] output of the PDF document.

3.3 Xigt-encoded IGTs

The Xigt format (Goodman et al., 2014) encodes all of this information in a model that is better suited for computational work. The Xigt package provides codes for either XML or JSON, and enables standoff annotation that is capable of preserving the original text format while adding multiple annotation layers (Xia et al., 2016). This is the format used for storing additional annotation, including the syntactic enrichment found in Xia et al. (2016), as well as metadata such as annotator comments. This standoff annotation is implemented as XML elements we call *tiers* that refer back to the elements they annotate.

4 Automatic Processing

The data in ODIN was assembled using an approach that combined metacrawling for IGT-containing documents with a classifier trained to detect IGT-formatted instances (Lewis and Xia, 2010). The resulting data can look like that in Fig. 2; with a variety of corruption and non-linguistically relevant data. To speed up annotation, we run several automated cleaning and normalization steps before handing the instance off to human annotators.

4.1 Cleaning

In the first step, we attempt to clean any artifacts introduced by the PDF-to-text conversion process. First, invalid characters for XML data, such as the form feed control character U+000C, are automatically replaced with the Unicode replacement character U+FFFD. Line and character corruption are addressed next. The instance in Fig. 2 exhibits both line corruption and character corruption. For line corruption, we merge two lines of the same type (e.g., L) if they are adjacent, one or both has the corruption tag CR, and any characters in the lines line up with whitespace in the other. In this example, the 'ak' on line 875 would be combined with the rest of the text on 876 as the two lines are merged into one. The output of the cleaning process is output to a new *cleaned* Xigt tier. The cleaning process also removes blank lines (if any) and superfluous initial columns of whitespace (consistently across lines to avoid altering column alignments). Currently we do not attempt to automatically fix character corruption (e.g., when a character's diacritics are erroneously extracted as a separate character), but instead allow users to cor-

[2] http://brat.nlplab.org

[3] http://depts.washington.edu/uwcl/odin

[4] http://www.foolabs.com/xpdf

```
doc_id=1482 874 878 M+AC+LN L+CR L+SY+CR G T+DB
language: Haitian (hat)
line=874 tag=M+AC+LN:   (25) Haitian CF (Lefebvre     1998:165)
line=875 tag=L+CR   :                         ak
line=876 tag=L+SY+CR:          Jani    pale            lii/j
line=877 tag=G      :         (John speak with         he)
line=878 tag=T+DB   :         (a) ’John speaks with   him’, (b) ’John speaks with    himself’
```

Figure 2: A text-format ODIN IGT instance exhibiting line corruption, character corruption, and language names and parentheticals, extracted from Heine (2001).

rect the corrupted characters (including Unicode), and we make the original PDF available to the user for consultation, if it is necessary. We are also investigating an alternative PDF extractor that more accurately extracts Unicode characters, diacritics, combined ligatures, etc.

4.2 Normalization

The second automated step we perform relates to information that is either non-linguistic or meta-linguistic in nature. In Fig. 2, such information includes the instance numbering (`25`), the language name (`Haitian`), author citation (Lefebvre), and quotation marks (on line 878). In the instance in Fig. 2, these elements have been placed on a line above the language line, which the IGT detection algorithm has tagged as non-IGT data (`tag=M`). Other times, this data occurs on the language line and thus instance numbering on the language line are removed with regular expressions. Other information, such as the language name or linguistic construction, are detected and placed on a separate `M` line. However, not all data can be reliably automatically extracted, such as the co-indexation variables `i` and `j` in line 876 which could be interpreted as being part of the word.

4.3 Enrichment

In addition to cleaning and normalizing, through the use of the INterlinear Text ENrichment Toolkit (INTENT) (Georgi, 2016; Xia et al., 2016), we automatically generate word alignments and part-of-speech tags for the different lines of IGT. Currently, this is visualized in the editor, as seen in Fig. 5, and will be able to be corrected by the annotators for subsequent tasks.

5 A RESTful IGT Server

While our immediate needs for the editor are fairly simple, we anticipate expansion for future tasks that may be required by the R*i*PL*es* (*i*nformation *e*ngineering and *s*ynthesis for Resource-Poor Languages) project (Xia et al., 2016). In order to facilitate such expansion, we created the backend for the editor as a general-purpose RESTful IGT server with the HTTP methods listed in Fig. 4.

The data is stored in a custom JSON-based filesystem database so that individual IGTs can be updated without having to reserialize an entire corpus, but the database abstraction layer makes it straightforward to later add support for other databases. Through the `Accept` request header, a user may ask for either the XML or JSON serialization of the data. More information on this interface can be found at the project page at:
`https://github.com/xigt/sleipnir`

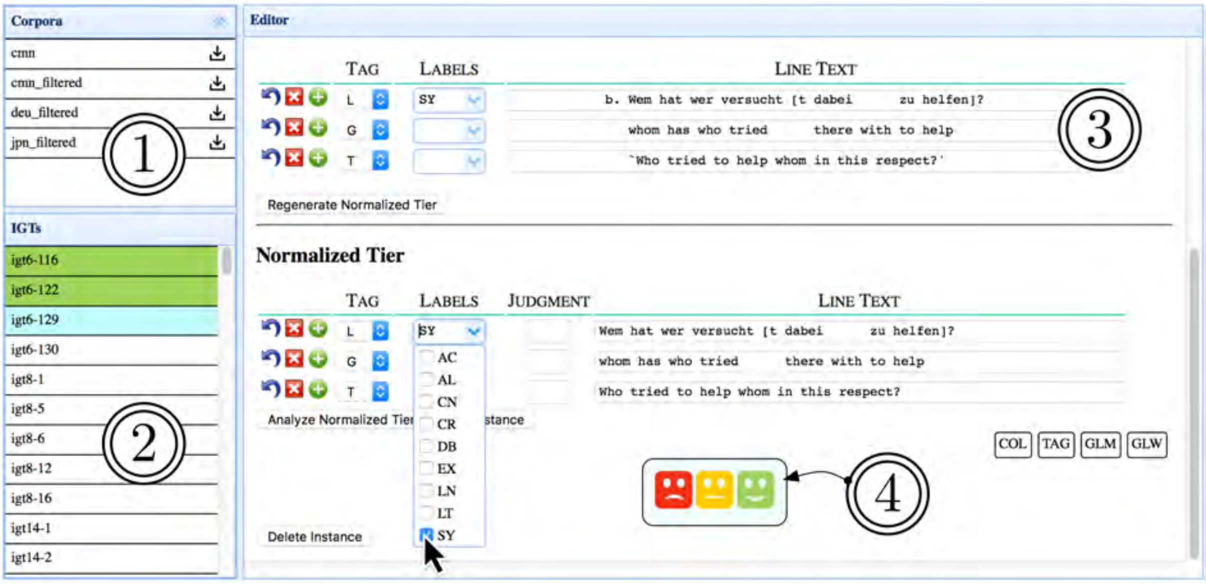
6 Online Editing Environment

The main interface that end-users will experience at this point is the online editing environment, a screenshot of which is provided in Fig. 3. This browser-based interface allows us to invite annotators around the world to participate without needing to install Python or any of the supporting packages required to work with the Xigt-formatted ODIN data.

The interface is contained in three main panes; in Fig. 3, labels (1) and (2) mark the corpus and instance browsers, respectively, while the rest of the window is dedicated to the instance editor.

Loading an Instance To start working on an instance, an annotator first selects the corpus from the corpus browser (1) and then the particular instance from (2). Instances that have been previously annotated are highlighted with the color of their rating, while the currently displayed instance is highlighted in cyan.

Validating an Instance as IGT Once an instance is loaded in the editor, the annotator is presented with an interface showing only the raw text of the instance (not shown), and the rating buttons (4). Since instances in ODIN have been automatically identified, some instances may not, in fact, be IGT, or may be too corrupted to derive what the original content might have been. At this point,



```
b. Wem hat wer versucht [t dabei      zu helfen]?
   whom has who tried       there with to help
   `Who tried to help whom in this respect?'
```

```
Wem hat wer versucht [t dabei      zu helfen]?
whom has who tried       there with to help
Who tried to help whom in this respect?
```

Figure 3: Screenshot of the browser-based editor being used to edit a sentence. (1) and (2) show the corpus and instance browsers, respectively, while (3) labels the instance editing area, and (4) shows the rating system.

`GET /corpora`
retrieve list of available corpora

`GET /corpora/<CID>`
retrieve a corpus by its identifier `<CID>`

`GET /corpora/<CID>/summary`
retrieve a summary of the contents of corpus `<CID>`

`GET /corpora/<CID>/igts`
retrieve the list of IGTs for corpus `<CID>` (parameters exist for filtering this list)

`GET /corpora/<CID>/igts/<IID>`
retrieve a single IGT by its identifier `<IID>` from corpus `<CID>`

`POST /corpora`
add a new corpus

`POST /corpora/<CID>/igts`
add a new IGT to corpus `<CID>`

`PUT /corpora/<CID>/igts/<IID>`
assign or replace IGT `<IID>` in corpus `<CID>`

`DELETE /corpora/<CID>`
delete corpus `<CID>`

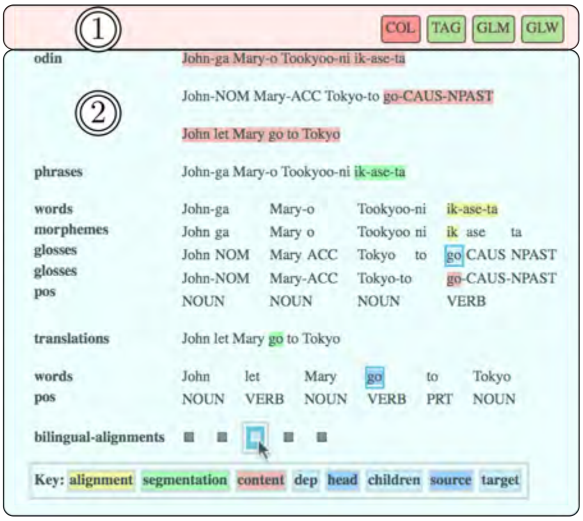
`DELETE /corpora/<CID>/igts/<IID>`
delete IGT `<IID>` in corpus `<CID>`

Figure 4: HTTP methods for the IGT server, where `<CID>` refers to the corpus identifier and `<IID>` the single IGT identifier.

the annotator may click the red “bad quality” rating button to flag that instance. If the instance is IGT and of acceptable quality they may continue onto the next task.

Cleaning After the annotator has verified that an instance is valid, they may click the *Generate Cleaned Tier* button to trigger the automatic cleaning procedure described in Section 4.1. The annotator is then given an opportunity to manually correct any errors made by the automatic corruption removal. The cleaning stage only corrects errors introduced by the PDF-to-text conversion process, so for clean instances there is little to be done here. If the annotator has made a mistake or wishes to start over, they may restore the content of an item to the state after automatically cleaning, or they may regenerate the clean tier entirely by re-invoking the cleaning procedure on the raw data. The raw tier cannot be edited, so the annotator can always get back to the original representation. Once satisfied, the annotator may continue to the normalization step.

Normalization By clicking the *Generate Normalized Tier* button, the annotator triggers the normalization procedure described in Section 4.2. In addition to placing non-IGT information on `M` lines, annotators are also asked to correct spurious or missing whitespace, ensure that there are an equal number of language-line and gloss-line tokens, and, when possible, an equal number of morpheme or clitic boundaries denoted by ‘-’ or ‘=’, following the Leipzig Glossing Rules (Comrie et al., 2015). Just as with the cleaning



```
odin                    John-ga Mary-o Tookyoo-ni ik-ase-ta

                        John-NOM Mary-ACC Tokyo-to go-CAUS-NPAST

                        John let Mary go to Tokyo

phrases                 John-ga Mary-o Tookyoo-ni ik-ase-ta

words                   John-ga    Mary-o     Tookyoo-ni   ik-ase-ta
morphemes               John ga    Mary o     Tookyoo ni   ik  ase    ta
glosses                 John NOM   Mary ACC   Tokyo   to   go CAUS NPAST
glosses                 John-NOM   Mary-ACC   Tokyo-to     go-CAUS-NPAST
pos                     NOUN       NOUN       NOUN         VERB

translations            John let Mary go to Tokyo

words                   John   let    Mary   go     to    Tokyo
pos                     NOUN   VERB   NOUN   VERB   PRT   NOUN

bilingual-alignments    ■   ■   □   ■   ■

Key: alignment segmentation content dep head children source target
```

Figure 5: Normalized tier analysis, with the section labeled (1) showing the annotation indicator labels and (2) showing the enrichment and alignment information. The colors highlighted in (2) are to indicate which elements of the IGT instance are referenced by another element. Here, the alignment between the English *go* and the Japanese *ikaseta*s visualized. The titles on the left refer to the tier type represented in the Xigt representation.[6]

step, the annotator may restore the original normalized content of an item or regenerate the normalized tier entirely. At this point, if the annotator believes they have satisfactorily met the normalization guidelines, they are done editing and continue to the analysis step to ensure the system is able to accurately infer the IGT structure.

Analysis When the annotator clicks the *Analyze Normalized Tier* button, the editor will present an analysis of the IGT such as the one shown in Fig. 5. This analysis includes both a series of indicators (1) to alert the annotator to the aforementioned guidelines, and a visualization of the automatic alignment and enrichment (2) performed by INTENT (see Section 4.3). The enrichment includes both the automatic word, morpheme, and gloss segmentation (**words**, **morphemes**, **glosses**, respectively), as well as word alignment between gloss and translation and part-of-speech tags for each line (**bilingual-alignments**, **pos**). There are currently four indicators:

COL language and gloss tokens are aligned with whitespace into columns

TAG language, gloss, and translation lines all exist and have no extraneous metadata on them

GLM language and gloss lines have the same number of morphological units

GLW language and gloss lines have the same number of whitespace-separated tokens

When an indicator shows red, the annotator should go back to the normalization (or possibly, the cleaning) step and make more corrections, then reanalyze. Occasionally an indicator shows red when there is no error; e.g., a word in the language line might have a hyphen that is not a morphological boundary and is thus glossed with a non-hyphenated token. The visualization of the automatically aligned and enriched IGT illustrates to the annotator how well the system was able to infer the structure of the IGT. Some problems that could not be detected with the indicators may become obvious by the presence of incorrect alignments, and the annotator can use this information to adjust the textual representation until proper alignments are obtained. These two facets of the analysis—indicators and visualization—help the annotator see how usable the instance will be for further processing.

Rating and Saving the Instance Finally, if the annotator has proceeded to the normalization or analysis steps, they may choose to rate the instance as bad (red), unclean (yellow) or clean (green), depending on the level of corruption of the instance. A field is provided to add further comments that will be saved into the IGT file.

User Management Currently, users are identified by a unique 8-character userid string, that also serves as the login. Annotator accounts are created and a backend corpus management script is used to initialize copies of corpus subsections that are assigned to the annotators. Annotators' ratings and comments are saved in these files along with their userid, so that inter-annotator agreement can be quickly and efficiently calculated across selected overlapping corpus segments.

Availability A fully functioning demonstration of the interface containing Chinese, German, and Japanese instances may be accessed at:
`http://editor.xigt.org/user/demo`
The source code is released under the MIT license

[6] See Goodman et al. (2014) for more.

and is available at:
`https://github.com/xigt/yggdrasil`

7 Conclusion and Future Improvements

The system we have presented here greatly streamlines the process of refining and displaying IGT data. Such clean, electronically available IGT data can be of great use to linguists searching for examples of particular phenomena, typologists looking to compare linguistic information over the thousands of languages for which IGT data is available, and computational linguists looking to build NLP tools for resource-poor languages.

In the future, we hope to further expand the editing capabilities of the system to include giving annotators the ability to edit word alignment and POS tag data. Such annotation would provide a high-quality set of data for typological research, as well as evaluation data for the automatic enrichment methods used.

Finally, while the current system is used only for display and editing, we hope to include the ability to search over IGT corpora in a subsequent version of the tool, replacing the current ODIN search capability[7].

Acknowledgments

This work is supported by the National Science Foundation under Grant No. BCS-0748919. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

Dorothee Beermann and Pavel Mihaylov. 2014. TypeCraft collaborative databasing and resource sharing for linguists. *Language resources and evaluation* 48(2):203–225.

Bernard Comrie, Martin Haspelmath, and Balthasar Bickel. 2015. Leipzig glossing rules. `https://www.eva.mpg.de/lingua/pdf/Glossing-Rules.pdf`.

Matthew S Dryer. 2007. Noun phrase structure. In Timothy Shopen, editor, *Language Typology and Syntactic Description*, Language typology and syntactic description, Cambridge, United Kingdom, pages 151–205.

Roy Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architecture*. Ph.D. thesis, University of California, Irvine.

Ryan Georgi. 2016. the INterlinear Text ENrichment Toolkit. `http://intent-project.info/`.

Ryan Georgi, William D Lewis, and Fei Xia. 2014. Capturing divergence in dependency trees to improve syntactic projection. *Language Resources and Evaluation* 48(4):709–739.

Ryan Georgi, Fei Xia, and William D Lewis. 2015. Enriching interlinear text using automatically constructed annotators. *LaTeCH 2015* page 58.

Michael Wayne Goodman, Joshua Crowgey, Fei Xia, and Emily M Bender. 2014. Xigt: extensible interlinear glossed text for natural language processing. *Language Resources and Evaluation* 49(2):455–485.

Bernd Heine. 2001. Accounting for creole reflexive forms. *Pidgins and Creoles Archive* (8).

William D Lewis and Fei Xia. 2010. Developing ODIN: A Multilingual Repository of Annotated Language Data for Hundreds of the World's Languages. *Literary and Linguistic Computing* 25(3):303–319.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. BRAT: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 102–107.

Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. 2008. recaptcha: Human-based character recognition via web security measures. *Science* 321(5895):1465–1468.

Fei Xia, William D Lewis, Michael Wayne Goodman, Glenn Slayden, Ryan Georgi, Joshua Crowgey, and Emily M Bender. 2016. Enriching a massively multilingual database of interlinear glossed text. *Language Resources and Evaluation* pages 1–29.

[7] http://odin.linguistlist.org/