Enriching, Editing, and Representing Interlinear Glossed Text

Fei Xia[1], Michael Wayne Goodman[1], Ryan Georgi[1],
Glenn Slayden[1], and William D. Lewis[2]

[1] Linguistics Department, University of Washington, Seattle, WA 98195, USA
{fxia, goodmami, rgeorgi, gslayden}@uw.edu
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
wilewis@microsoft.com

Abstract. The majority of the world's languages have little to no NLP resources or tools. This is due to a lack of training data ("resources") over which tools, such as taggers or parsers, can be trained. In recent years, there have been increasing efforts to apply NLP methods to a much broader swathe of the worlds languages. In many cases this involves bootstrapping the learning process with enriched or partially enriched resources. One promising line of research involves the use of Interlinear Glossed Text (IGT), a very common form of annotated data used in the field of linguistics. Although IGT is generally very richly annotated, and can be enriched even further (e.g., through structural projection), much of the content is not easily consumable by machines since it remains "trapped" in linguistic scholarly documents and in human readable form. In this paper, we introduce several tools that make IGT more accessible and consumable by NLP researchers.

1 Introduction

Of the world's 7,000+ spoken languages, only a very small fraction have text resources substantial enough to allow for the training of NLP tools, such as part-of-speech (POS) taggers and parsers. Developing enriched resources, e.g., treebanks and POS-tagged corpora, which allow supervised training of such tools, is expensive and time-consuming. In recent years, work has been done to bootstrap the development of such resources for resource-poor languages by tapping the enriched content of a better resourced language and, through some form of alignment, "projecting" annotations onto data for the resource-poor language. Some studies have focused on the typological similarity of languages, using cognates and similar word forms in typologically similar languages, to bridge between languages and to build tools and resources [1,2]. Other work has relied on parallel corpora, where one language of a corpus is highly resourced, and annotations are projected onto the lesser resourced language(s) [3,4]. A third line of work is to use linguistically annotated data, specifically Interlinear Glossed Text (IGT), a data format very commonly used in the field of linguistics, to project annotations

from a highly resourced language to one or more under-resourced languages, potentially hundreds at a time [5,6].

Building upon the previous studies on IGT, we see the potential for bootstrapping or training up tools for a larger number of the world's languages. Since IGT is common in the field of linguistics, and because linguists study thousands of the world's languages, the possibility exists to build resources for a sizable percentage of the world's languages. The problem is that IGT is typically locked away in scholarly linguistic papers, and not easily accessible to NLP researchers who might otherwise want access to the data. The Online Database of Interlinear text (ODIN) [7], a database of over 200,000 instances of IGT for more than 1,500 languages, tackles the issue of extracting IGT from scholarly resources, but focuses more on presenting the captured content for human consumption and query. By taking the content of ODIN, enriching it (e.g., through projected annotations), and reformatting it into a machine readable form, enriched IGT becomes a much more useful resource for bootstrapping NLP tools.

In this paper, we first describe the raw (or original) IGT used by linguists and the enriched IGT which is more relevant to the NLP field. Then we outline a data format for representing enriched called *Xigt*. Next, we introduce two packages that we have developed for processing IGT: the first one enriches raw IGT automatically, and the second one is a graphic editor which the annotators can use to edit enrich IGT in the Xigt format. By making these tools, and the resulting data, available to the NLP community, we open the door to a much wider panoply of the world's languages for NLP research.

2 Interlinear Glossed Text

IGT is a common format that linguists use to present language data relevant to a particular analysis. It is most commonly presented in a three-line canonical form, a sample of which is shown in (1). The first line, the *language line*, gives data for the language in question, and is either phonetically encoded or transcribed in the language's native orthography. The second line, the *gloss line*, contains a morpheme-by-morpheme or word-by-word gloss for the data on the language line. The third line, the *translation line*, contains a translation of the first line, often into a resource-rich language such as English. There could be additional lines showing other information such as a citation and a language name and/or code. In Ex (1), *(Bailyn, 2001)* is the source of the IGT instance, referring to [8]; *cym* is the language code for Welsh.

(1) Rhoddodd yr athro lyfr i'r bachgen ddoe
gave-3sg the teacher book to-the boy yesterday
"The teacher gave a book to the boy yesterday" (Bailyn, 2001) [cym]

2.1 Collecting IGT

In linguistics, the practice of presenting language data in interlinear form has a long history, going back at least to the time of the structuralists. IGT is often

used to present data and analysis on a language that the reader may not know much about, and is frequently included in scholarly linguistic documents. ODIN, the Online Database of INterlinear text, is the result of an effort to collect IGT instances in scholarly documents posted to the Web [9,7]. It currently contains approximately 200,000 IGT instances from over 1500 languages.

2.2 Enriching IGT

The unique structure of IGT makes it an extremely rich source of information for resource-poor languages: Implicit in an IGT instance is not only a short bitext between that language and a language of wider communication (almost universally English, but instances of Spanish and German have been discovered as well), but also information encoded in the so-called *gloss line* about the grammatical morphemes in the source language and word-by-word translations to lemmas of the translation language. Thus even small quantities of IGT could be used to bootstrap tools for resource-poor languages through structural projection [3,10]. However, bootstrapping tools often require the raw IGT to be enriched first. The enrichment process normally contains the following two steps.

Cleaning and normalizing IGT instances: The process of collecting IGT from linguistic document may introduce noise. For instance, ODIN uses an off-the-shelf converter to convert pdf documents into text format, and the converter sometimes wrongly splits a language line into two lines. One such an example is Ex (2) from [11], where the language line is incorrectly split into two lines by the converter.

```
(2) Haitian CF (Lefebvre 1998:165)
                  ak
    Jani   pale         lii/j
    John   speak  with   he
    (a) 'John speaks with him' (b) 'John
        speaks with himself'
```

Furthermore, the raw IGT is often not in the three-line canonical form. For instance, an IGT instance often contains other information such as a language name, a citation, and so on. In Ex (2), the first line contains the language name and citation,[1] the third line includes coindexes i and i/j, and the last two lines show two possible translations of the sentence.

The cleaning and normalization step aims at fixing errors that were introduced when IGT was extracted from the linguistic documents, separating out various fields in an IGT, normalizing each field, and storing the results in a uniform data structure. Ex (3) shows the resulting IGT after this step. Noticing that coindexes i and j are removed from the language line and stored in a separate field, and the wrongly split language lines are merged back.

[1] *CF* here stands for French-lexified creole.

```
(3) Language: Haitian CF
    Citation: (Lefebvre 1998:165)
    L:        Jan   pale   ak     li
    Coindx:   (Jan, i), (li, i/j)
    G:        John  speak  with   he
    T1:       John speaks with   him
    T2:       John speaks with himself
```

Adding word alignment and syntactic structure: After IGT has been cleaned and normalized, the next step is to add word alignment and syntactic structure. For word alignment, if the IGT instance is clean, the alignment between the language line and the gloss line is implicit from the layout (i.e., the i-th tokens in the two lines align to each other). The alignment between the gloss line and the translation line can be obtained by running automatic word aligner such as GIZA++ [12] or using some heuristics (e.g., aligning words with the same spelling or stem). The common way for getting syntactic structure is to parse the translation line with an English parser, and then project that parse tree to the language line via the word alignment [10]. Given the IGT in Ex (1), the algorithm will produce the word alignment in Fig 1, the syntactic structures in Fig 2.

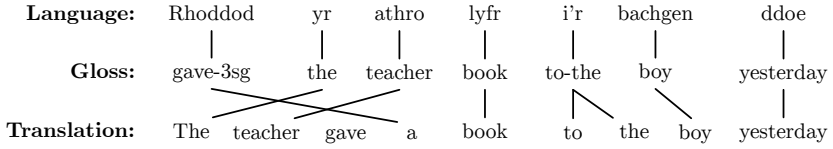

Fig. 1. Aligning the three lines in an IGT instance

2.3 Using Enriched IGT

Enriched IGT can help linguistic studies and NLP in many ways. For instance, linguists can search an IGT database for sentences with certain linguistic constructions (e.g., passives, conditionals, double object structures). Enriched IGT also allows discovery of computationally relevant typological features (e.g., word order or the presence or absence of particular grammatical markers), and does so with high accuracy [13,14]. Furthermore, enriched IGT can also be used to bootstrap NLP tools for resource-poor languages; for instance, adding features extracted from projected syntactic structures to a statistical parser provided a significant boost to parser performance [5].

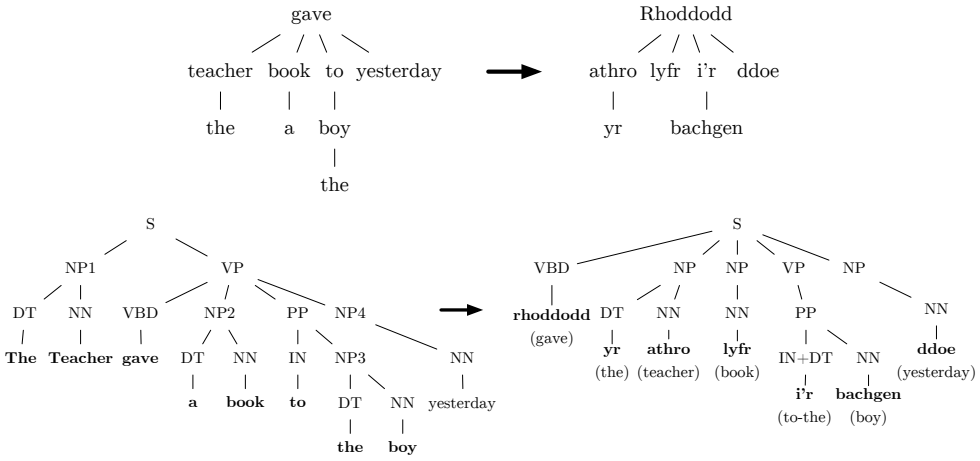

Fig. 2. Projecting dependency and phrase structure from the translation line to the language line

3 Xigt: an XML Representation of the Enriched IGT

A human can read a textual IGT from ODIN and understand the structure of annotations, but a computer only sees strings of characters and spaces. We can—and do, for the enrichment process—write code to interpret the spacing as delimiters for groups of related tokens, but this is unreliable in noisy data, and moreover cannot handle alignments that do not arrange vertically. Once we have initially analyzed the space-delimited structure of a textual IGT, we store the information in a structured data format so a computer can more easily understand the structure for later tasks.

Goodman et al. [15] introduced a new data model for IGT, called Xigt, that allows for complex annotation alignments, which is useful for encoding the enriched IGT.[2] Key features of Xigt include a relatively flat structure (tiers are represented side-by-side, not nested) and the use of IDs and references for annotations. There are only four structural elements: `<xigt-corpus>`, `<igt>`, `<tier>`, and `<item>`. Xigt also introduces a new referencing system called "alignment expressions" that allows annotations to have multiple targets, or to select sub-spans (e.g., character spans) from targets. These features allow for novel or complex annotation types, alternative analyses, and the ability to add new annotations without changing the structure of previous ones.

Xigt's generality and expressiveness allow for rich representations of many kinds of annotations, but at the same time the lack of hard constraints allows the same data to be represented in multiple ways. For example, *words* may be

[2] While Xigt itself is the data model, it has a canonical XML serialization format called XigtXML. For the sake of simplicity, in this paper we will not make such a distinction and instead use the same name for both.

specified as segments of a *phrase* (i.e. a kind of annotation of the phrase) or as primary data (unaligned and explicitly specifying the form of the word); *glosses* may align to *morphemes*, *words*, or *phrases*, depending on the availability of these tiers. Our IGT enrichment package (INTENT) and IGT editor (XigtEdit) need more specifications in order to efficiently process Xigt-encoded corpora, so we establish a set of conventions on top of Xigt that our data abide by.

In this section we outline the conventions for our data. The Xigt project includes an API for programmatically interacting with corpora, so we also cover the API functions that are useful for our purposes.

3.1 Representing Enriched IGT in Xigt

To represent the enriched IGT, as described in Section 2.2, in Xigt, we need to extend Xigt in several ways. Figure 3 shows an enriched IGT in this extended format. First, we define some new tier types and constrain how the default ones are used. Collectively, the tiers can be divided into three groups according to the source of information: (1) Original text, (2) Inferred structure, and (3) Additional enrichment.

Group (1): Stores the original text from ODIN IGT so that structural information is encoded as stand-off annotation of it. This is useful, in part, in case the process of enriching IGT changes and we need to regenerate the IGT in Xigt. This group has only one tier type, called *odin*, and each `<item>` on such a tier contains a line from the original IGT. In Figure 3, lines 7–11 encode the raw text, while lines 12–16 encode the text after normalization. The *odin* tier is also used for storing cleaned text (not shown here as it is identical to the raw IGT for this particularly clean example). The `state` attribute specifies the level of processing undergone by each *odin* tier.

Group (2): Encodes the structural annotations that are implicit in the textual IGT. This group only uses the default tier types in Xigt, although we specify some constraints on their usage to aid INTENT and XigtEdit in their processing, and includes:

- *phrases*: representing the language line (lines 17–19)
- *words*: showing the segmentation of the language or translation line (lines 20–25 and 41–45)
- *morphemes*: marking the morpheme boundary within a word (lines 26–32)
- *glosses*: providing word-to-word or morpheme-to-morpheme glosses (lines 33–37)
- *translations*: providing the translation of the language line (lines 38–40)

The structural annotations in Group (2) can be inferred by examining tokens in the tiers from word segmentation (i.e., by spaces) or morpheme segmentation (i.e., first by spaces, then by hyphens or other morpheme delimiters). By definition, in a clean IGT the i^{th} token from a morpheme-segmented gloss line will align to the i^{th} token of a morpheme-segmented language line. In a less clean

```
<?xml version="1.0" encoding="utf-8"?>
<xigt-corpus alignment-method="auto" xml:lang="en">
 <metadata xmlns:olac="http://www.language-archives.org/OLAC/1.1/" ...>
   ...
 </metadata>
 <igt id="i1" doc-id="397" line-range="959 961" tag-types="L G T">
  <tier type="odin" state="raw" id="r">
   <item id="r1" line="959" tag="L"
        >(1) Nay-ka ai-eykey pap-ul mek-i-ess-ta</item>
   <item id="r2" line="960" tag="G"
        >    I-Nom child-Dat rice-Acc eat-Caus-Pst-Dec</item>
   <item id="r3" line="961" tag="T"
        >    'I made the child eat rice.'</item>
  </tier>
  <tier type="odin" state="normalized" id="n" alignment="r">
   <item id="n1" alignment="r1" line="959" tag="L"
        >Nay-ka ai-eykey pap-ul mek-i-ess-ta</item>
   <item id="n2" alignment="r2" line="960" tag="G"
        >I-Nom child-Dat rice-Acc eat-Caus-Pst-Dec</item>
   <item id="n3" alignment="r3" line="961" tag="T"
        >I made the child eat rice.</item>
  </tier>
  <tier type="phrases" id="p" content="n" xml:lang="ko">
   <item id="p1" content="n1"/>
  </tier>
  <tier type="words" id="w" segmentation="p" xml:lang="ko">
   <item id="w1" segmentation="p1[0:6]"/>
   <item id="w2" segmentation="p1[7:15]"/>
   <item id="w3" segmentation="p1[16:22]"/>
   <item id="w4" segmentation="p1[23:35]"/>
  </tier>
  <tier type="morphemes" id="m" segmentation="w" xml:lang="ko">
   <item id="m1.1" segmentation="w1[0:3]"/>
   <item id="m1.2" segmentation="w1[4:6]"/>
   <item id="m2.1" segmentation="w2[0:2]"/>
   <item id="m2.2" segmentation="w2[3:8]"/>
   ...
  </tier>
  <tier type="glosses" id="g" alignment="m" content="n">
   <item id="g1.1" alignment="m1.1" content="n2[0:1]"/>
   <item id="g1.2" alignment="m1.2" content="n2[2:5]"/>
   ...
  </tier>
  <tier type="translations" id="t" alignment="p" content="n">
   <item id="t1" alignment="p1" content="n3"/>
  </tier>
  <tier type="words" id="tw" segmentation="t">
   <item id="tw1" segmentation="t1[0:1]"/>
   <item id="tw2" segmentation="t1[2:6]"/>
   ...
  </tier>
  <tier type="bilingual-alignments" id="a" source="tw" target="g">
   <item id="a1" source="tw1" target="g1.1"/>
   <item id="a2" source="tw2" target="g4.2,g4.3"/>
   ...
  </tier>
  <tier type="dependencies" id="dt" dep="tw" head="tw">
   <item id="dt1" dep="tw1" head="tw2">nsubj</item>
   <item id="dt2" dep="tw2">root</item>
   <item id="dt3" dep="tw3" head="tw4">det</item>
   ...
  </tier>
  ...
 </igt>
 ...
</xigt-corpus>
```

Fig. 3. The Xigt representation of an enriched IGT example

IGT, these lines may not have the same number of tokens, in which case we back off to aligning word-segmented gloss and language lines. Again, in the case that this doesn't work, we align the unsegmented gloss line to the language line.

Group (3): Encodes new information (i.e., information not present in the original IGT) obtained through manual annotation or by running the IGT through NLP systems. If needed, a tier can appear multiple times in an IGT, representing alternative analyses.[3] This group includes:

- *bilingual-alignments*: showing word alignment between the gloss line and the translation line (lines 46–50)
- *dependencies*: showing syntactic dependencies (lines 51–56)
- *phrase-structure*: showing the syntactic phrase structure
- *pos*: providing POS tags for the words in a *words* tier

Group (3) can be extended further if new tier types are needed to present new type of information (e.g., co-reference for the words in the language line as in Ex (2)).

Other extensions: Besides defining three groups of tiers, we extend Xigt in other ways. Most importantly, we provide a detailed specification about what information should be represented in which tier and how. For instance, the word alignment between the language line and the gloss line is represented in the *alignment* field in the *glosses* tier, whereas the alignment between the gloss line and the translation line is shown in the *bilingual-alignments* tier. We distinguish these two types of alignment because it is more likely that users would want to store alternative analyses for the second type than the first type. In that case, they can simply include multiple *bilingual-alignments* tiers without repeating the segmentation of the gloss or translation line. We also define the conventions for naming tier IDs and item IDs so that those IDs can be generated automatically and systematically. Furthermore, we conventionalize a partial order between tiers so that a tier can only refer to itself or tiers that precedes it. This partial order is crucial when XigtEdit determines how editing in one tier affects other tiers (see Section 5.2).

3.2 Processing Documents with the Xigt API

To make it easier for the researchers to access IGT data in the Xigt format, Xigt provides an application-program interface (API), with a reference implementation in Python, for interacting with Xigt-encoded corpora computationally. The API provides the following functionalities:

- Serialize/deserialize Xigt documents to in-memory data structures
- Iterate over data collections (corpora, IGT, tiers)

[3] It is worth noting that multiple tiers for alternative analyses can be used on the tiers in Groups (1) and (2) as well.

– Retrieve object attributes, metadata, and content
– Retrieve the parent (i.e. container) of some object, such as a tier from an item
– Resolve the content, or the targeted items/tiers, of alignment expressions
– Construct new in-memory data structures

These functions allow users to easily build more complicated functions for their data, such as for counting statistics (e.g., finding the most frequent word), forming complex queries of data (e.g., "what are all the morphemes appearing on words marked as verbs?"), or augmenting a corpus with new analyses (e.g., creating a word-sense tier by looking up each word and its context in an external ontology and aligning the result to the word it came from). The API also enables users to construct new corpora in-memory (e.g., by converting or analyzing some other data) which can then be serialized to disk.

We make use of this API for serializing the ODIN textual data into Xigt and for the subsequent enrichment of the data, as described in Section 4.

4 INTENT: a Package for Creating Enriched IGT

In the previous sections, we described what type of information is in enriched IGT and how it is represented in Xigt. Because manually creating enriched IGT is time consuming and error-prone, we have developed a package, the INterlinear Text ENrichment Toolkit (INTENT), which takes an original IGT file as the input, and produces the enriched IGT in the Xigt format as the output. This output can then be corrected by a human annotator using XigtEdit, or be used to train an NLP system such as a POS tagger or a parser.

4.1 Toolkit Components

Figure 4 shows a typical enrichment workflow in INTENT. The input to INTENT is a file with the original IGT in either plain text format or in Xigt. INTENT first cleans and normalizes the IGT by some simple heuristic rules. It then generates the second group of tiers including *words*, *morphemes* (if the morpheme boundary is present in the IGT), *glosses*, and the like. After that, the third group of tiers are created by running the following modules.

Word Alignment: In our previous study [10], we proposed two methods for aligning the gloss line and the translation line. The first method ran a morphological analyzer on the translation line, and then aligned the words in the two lines if they had the same stems. The second method used GIZA++ [12], a statistical word aligner. Experimental results showed that the performances of the two methods were similar and combining them yielded a small boost. INTENT currently re-implemented those methods, showing F_1 scores of around 0.84 for the heuristic approach and 0.86 for the statistical approach [16]. We are enhancing the heuristic method by taking advantage of the POS tags in the enriched IGT.

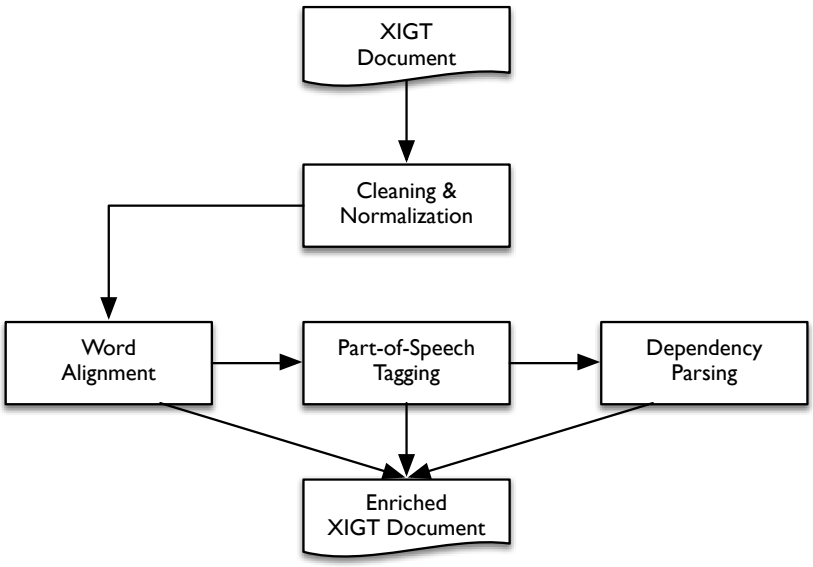

Fig. 4. A typical enrichment workflow in INTENT

Part-of-Speech Tagging: INTENT tags the translation line by running Stanford's English POS tagger [17], trained on the English Penn Treebank [18].[4] As for the language line, while one can simply project the POS tags from the translation line, the quality of the resulting tags is often low due to word alignment errors and translation divergence [19]. Instead, INTENT takes advantage of the annotation on the gloss line; for instance, grammatical markers such as *-Nom* (nominative case marker) and *-Dec* (declarative marker) are good cues for predicting the POS tags of the corresponding words in the language line. We can also find the POS tags of most morphemes in the gloss line using an English dictionary even if those morphemes are not aligned to the words in the translation line. We built a classifier using those features and trained it with a small amount of labeled gloss line data from multiple languages.[5] Evaluation of the classifier yielded a 90% POS tagging accuracy for the gloss line tokens in IGT, compared with 69.5% for the heuristic projection-based approach [16].

Dependency Parsing: The dependency structure for the translation line is produced in three steps. First, the dependency structure for the language line is produced by running the Stanford Parser [20,21].[6] Second, INTENT projects the dependency structure to the language line following the heuristic algorithm

[4] http://nlp.stanford.edu/software/tagger.shtml

[5] We use a classifier, not a sequence labeller, because the word order in the gloss line will be language-dependent, and the training and test data of our POS tagger can come from different languages.

[6] http://nlp.stanford.edu/software/lex-parser.shtml

in [10]. Third, from a small amount of dependency tree pairs, INTENT learns common divergence patterns automatically and applies them to the dependency structure produced in the second step. Our experiment shows that adding the third step results in an average of 25% error reduction over using the heuristic projection algorithm alone, when tested on eight different languages [5].

While the workflow in Figure 4 shows a pipeline approach, we are expanding the package to allow feedback loops among the modules. For instance, INTENT runs the word aligner first to get the initial alignment, which will be used by the classifier-based POS tagger. The output of that POS tagger can then be fed back to the word aligner to improve word alignment; for instance, two words unaligned during the first pass of word alignment is more likely to be linked together in the second pass if they have the same POS tags after POS tagging. The improved word alignment can in turn improve the next round of POS tagging.

4.2 Implementation of INTENT

INTENT is written in Python 3 and uses the Xigt API to interface with the serialized documents. INTENT also supplements the Xigt API's internal representations with a number of convenience subclasses for performing tasks such as tokenization and word alignment. Each type of enrichment can be run individually or in sequence.

5 XigtEdit: a GUI Editor for Enriched IGT

As Xigt is an XML-based format, it is nominally human-readable and thus editable with any text editor which is compatible with the desired or appropriate text encodings. However, using existing text editors to edit enriched IGT in the Xigt format is not convenient due to the special properties of Xigt:

- Xigt standoff annotation requires each tier and each tier item to have a unique ID, which is used for cross-reference within an IGT instance. Assigning unique IDs manually is tedious and error-prone.
- Some alignment expressions (e.g., *segmentation* and *alignment* fields in many tiers) require precise computation of string offsets, which are tedious to manually derive.
- Phrase-structure and dependency-structure views are inherently graphical views that do not lend themselves to convenient text-based editing.
- Because tiers can refer to one another, editing one tier could affect the validity of annotation in its cross-referenced tiers. Manually keeping track of the ripple effect of such editing is challenging.

In order to address these issues, we developed a graphical Xigt editor, *XigtEdit*, which facilitates the creation, editing and manipulation of Xigt files.

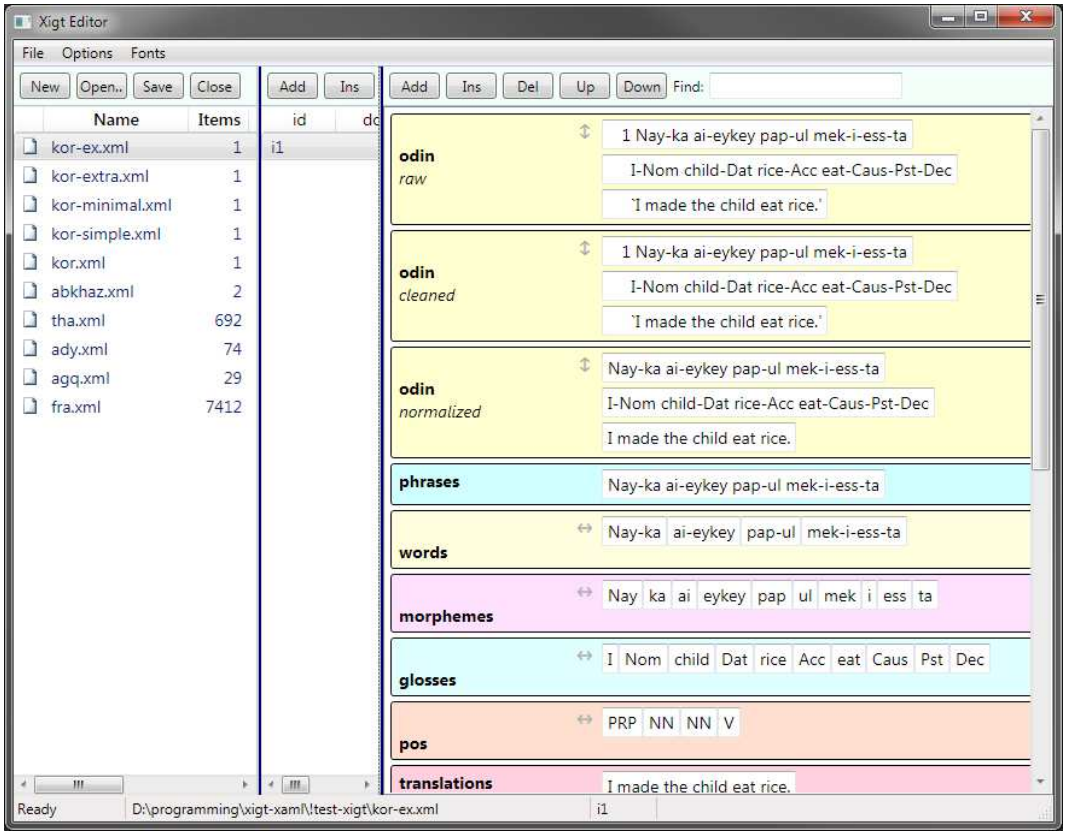

Fig. 5. Main editing interface screen from the XigtEdit application

5.1 Main Functionality of XigtEdit

The fundamental user interface of XigtEdit is a hierarchical structure which closely follows the Xigt abstract data model. Figure 5 shows a screen capture of the XigtEdit application.[7] There are three resizable panels:

- The leftmost panel is a list of the Xigt files that have been loaded. If there are more than one file, exactly one file is *currently selected*.
- The next panel, in the second column, lists the IGT instances which are in the selected file. Again, if the file contains multiple IGT instances, exactly one instance is *currently selected*.
- The rightmost panel is the editing area for the selected IGT instance. Tiers are arranged vertically in this area. For some tiers, items in the tiers are arranged vertically (line-oriented data such as in the *odin* tier) while others have items displayed horizontally (word-oriented data such as in the *words*, *morphemes*, and *glosses* tiers).

At any time during editing, individual Xigt files can be opened, edited, saved, closed (added or removed from the files list), or reverted. Files are read and saved directly in the Xigt format. To enhance annotator productivity, XigtEdit assigns unique IDs to new tiers and tier items based on predefined naming conventions (see Section 3.1). To address the inconvenience of computing and maintaining Xigt alignment expressions (e.g., the *segmentation* field in the *words* tier), XigtEdit allows the text spans for dependent items to be defined automatically. This can be achieved either through automatic tools for segmenting text based on whitespace or other criteria, or manually via intuitive user interfaces for manipulating text ranges. Furthermore, XigtEdit displays dependency or phrase structure tiers as graphical trees which the user can edit with mouse clicks. To support efficient annotation, XigtEdit provides keyboard alternatives to the use of the mouse for most application navigation and editing operations.

5.2 Editing Parent Tiers

As mentioned in Section 3, tiers can refer to other tiers and we define a partial order among tiers such that any tier can only refer to preceding tiers or itself. If a tier C refers to another tier P, we call P a parent of C. A tier can have multiple parents (e.g., a *bilingual-alignments* tier refers to two *words* tiers). Thus, this parent relation among tiers can be represented as a directed acyclic graph, where each node represents a tier, and each link goes from the parent tier to its child tier.

XigtEdit supports the propagation of editing changes from parent to child tiers in an IGT. Consider how editing tier P would affect another tier C that refers to it. The behavior depends on the relation between P and C, and whether other tiers refer to the text region in P that was edited. XigtEdit keeps track

[7] To make Figure 5 more readable, certain attributes (e.g., tier IDs, item IDs, alignment between tiers) are not displayed in the screen capture.

of these relationships and analyzes whether the change would invalidate other tiers. In cases where XigtEdit determines that a change has a deterministic effect on its dependent tiers (and where the *edit-propagation* feature has been enabled in the software), the change can be propagated from the parent tier to its child tiers automatically. Alternatively, if the change has ambiguous effects on other tiers, XigtEdit will prompt the user to choose what action should be taken for dependent tiers.

5.3 Implementation of XigtEdit

XigtEdit is a Windows Presentation Foundation (WPF) application which runs on the .NET Application Framework version 4.5.[8] It will run on any modern version of Microsoft Windows (Vista or later) without requiring any additional software or libraries. We chose to develop XigtEdit in WPF primarily because of the developer productivity benefits that WPF offers, such as the ability to implement the software using a declarative markup notation known as Extensible Application Markup Language (XAML). Also compelling in WPF is the "retained mode" graphics subsystem, which, for example, allows persistent data-bindings to be established between entities in the (abstract) Xigt data model and their on-screen representations. These two-way bindings automatically keep the data model and user-interface in-sync without the need for any procedural code. XigtEdit is open source and licensed under the MIT license.

6 Conclusion

The majority of the world's languages lack large-scale annotated resources over which NLP tools such as POS taggers or parsers can be trained. In recent years, there have been increasing efforts in bootstrapping NLP systems for resource-poor languages. One line of research uses linguistically annotated data, IGT in particular, to project annotations from resource-rich languages to resource-poor ones.

In this paper, we first provide an overview of enriched IGT and outline several ways that enriched IGT can help linguistic studies and NLP research. Second, we extend Xigt, an XML representation for enriched IGT, and provide an API for it. Third, we introduce INTENT, a package that enriches raw IGT automatically by adding word alignment, POS tags, and syntactic structures to IGT. Finally, we describe XigtEdit, a graphic editor for annotating IGT, which overcomes limitations of existing, general-purpose text editors. By making those tools freely available to the public,[9] we hope that NLP researchers will have easier access to the enriched IGT data and can then focus on exploring new methods for bootstrapping NLP tools for thousands of resource-poor languages by taking advantage of rich annotation in IGT.

[8] http://msdn.microsoft.com/en-us/library/aa970268(v=vs.110).aspx
[9] http://depts.washington.edu/uwcl/packages/

As for future work, we are working on improving the performance of INTENT by allowing feedback loops in the workflow. In addition, we plan to create enriched IGT data sets for a dozen resource-poor languages, by first running INTENT on the raw IGTs coming from ODIN and then using XigtEdit for manual correction. The data sets will be released to the public and can be used as training and test data for evaluating NLP systems.

Acknowledgments The work is supported by the National Science Foundation under Grant No. BCS-1160274 and 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 National Science Foundation. We would also like to thank two anonymous reviewers for helpful comments.

References

1. Hana, J., Feldman, A., Amaral, L., Brew, C.: Tagging portuguese with a spanish tagger using cognates. In: Proc. of the Workshop on Cross-language Knowledge Induction, in conjunction with the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006), Trento, Italy (2006)
2. Feldman, A., Hana, J., Brew, C.: A cross-language approach to rapid creation of new morpho-syntactically annotated resources. In: Proc. of the 5th international conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy (2006)
3. Yarowsky, D., Ngai, G.: Inducing Multilingual POS Taggers and NP Bracketers via Robust Projection across Aligned Corpora. In: Proc. of the 2001 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-2001). (2001) 200–207
4. Hwa, R., Resnik, P., Weinberg, A., Cabezas, C., Kolak, O.: Bootstrapping Parsers via Syntactic Projection across Parallel Texts. Special Issue of the Journal of Natural Language Engineering on Parallel Texts (2005) 311–325
5. Georgi, R., Xia, F., Lewis, W.D.: Enhanced and portable dependency projection algorithms using interlinear glossed text. In: Proceedings of ACL 2013 (Volume 2: Short Papers), Sofia, Bulgaria (2013) 306–311
6. Georgi, R., Xia, F., Lewis, W.D.: Capturing divergence in dependency trees to improve syntactic projection. Language Resources and Evaluation **48** (2014) 709–739
7. Lewis, W., Xia, F.: Developing odin: A multilingual repository of annotated language data for hundreds of the world's languages. Journal of Literary and Linguistic Computing (LLC) **25** (2010) 303–319
8. Bailyn, J.F.: Inversion, Dislocation and Optionality in Russian. In Zybatow, G., ed.: Current Issues in Formal Slavic Linguistics. (2001)
9. Lewis, W.D.: Mining and migrating interlinear glossed text. Technical report, Workshop on Digitizing and Annotating Texts and Field Recordings, LSA Institute (2003) http://emeld.org/workshop/2003/papers03.html.
10. Xia, F., Lewis, W.D.: Multilingual structural projection across interlinear text. In: Proc. of the Conference on Human Language Technologies (HLT/NAACL 2007), Rochester, New York (2007) 452–459
11. Lefebvre, C.: Creole Genesis and the Acquisition of Grammar: The case of Haitian Creole. Cambridge University Press, Cambridge, England (1998)

12. Och, F.J., Ney, H.: A systematic comparison of various statistical alignment models. Computational Linguistics **29** (2003) 19–51
13. Lewis, W.D., Xia, F.: Automatically Identifying Computationally Relevant Typological Features. In: Proc. of the Third International Joint Conference on Natural Language Processing (IJCNLP-2008), Hyderabad, India (2008)
14. Bender, E.M., Goodman, M.W., Crowgey, J., Xia, F.: Towards creating precision grammars from interlinear glossed text: Inferring large-scale typological properties. In: Proceedings of the 7th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities, Sofia, Bulgaria (2013) 74–83
15. Goodman, M.W., Crowgey, J., Xia, F., Bender, E.M.: Xigt: extensible interlinear glossed text for natural language processing. Language Resources and Evaluation (2014) 1–31
16. Georgi, R., Xia, F., Lewis, W.D.: Training part-of-speech taggers using interlinear text (2015) Manuscript.
17. Toutanova, K., Klein, D., Manning, C.D., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of HLT-NAACL 2003. (2003) 252–259
18. Marcus, M., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of English: the Penn Treebank. Computational Linguistics **19** (1993) 313–330
19. Dorr, B.J.: Machine translation divergences: a formal description and proposed solution. Computational Linguistics **20** (1994) 597–635
20. Klein, D., Manning, C.D.: Accurate Unlexicalized Parsing. In: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-2003). (2003)
21. de Marneffe, M.C., MacCartney, B., Manning, C.D.: Generating typed dependency parses from phrase structure parses. In: Proc. of LREC 2006. (2006)