Bridging the Gaps: Interoperability for GrAF, GATE, and UIMA

Nancy Ide
Department of Computer Science
Vassar College
Poughkeepsie, New York USA
ide@cs.vassar.edu

Keith Suderman
Department of Computer Science
Vassar College
Poughkeepsie, New York USA
suderman@anc.org

Abstract

This paper explores interoperability for data represented using the Graph Annotation Framework (GrAF) (Ide and Suderman, 2007) and the data formats utilized by two general-purpose annotation systems: the General Architecture for Text Engineering (GATE) (Cunningham, 2002) and the Unstructured Information Management Architecture (UIMA). GrAF is intended to serve as a "pivot" to enable interoperability among different formats, and both GATE and UIMA are at least implicitly designed with an eye toward interoperability with other formats and tools. We describe the steps required to perform a round-trip rendering from GrAF to GATE and GrAF to UIMA CAS and back again, and outline the commonalities as well as the differences and gaps that came to light in the process.

1 Introduction

The creation of language data and linguistic annotations remains a fundamental activity in the field of language technology, in order to develop increasingly sophisticated understanding and generation capabilities for the world's languages. Substantial effort has been devoted to the creation of resources for major languages, and new projects are developing similar resources for less widely-used languages; the cost and effort of resource creation, as well as the possibilities for linking multilingual and multi-modal language data, demands that resources and tools are reusable as well as compatible in terms of their representation. Various representation standards and annotation tools have emerged over the past decade and have contributed to some convergence in practice, but at the same time, there has been growing recognition that *interoperability* among formats and tools, rather than universal use of a single representation format, is more suited to the needs of the community and language technology research in general.

This paper explores interoperability for data represented using the Graph Annotation Framework (GrAF) (Ide and Suderman, 2007) and the data formats utilized by two general-purpose annotation systems: the General Architecture for Text Engineering (GATE) (Cunningham, 2002) and the Unstructured Information Management Architecture (UIMA)[1]. UIMA and GATE are similar in design and purpose: both represent documents as text plus annotations and allow users to define pipelines of processes that manipulate the document. However, there are some differences in implementation and representation format that prohibit direct exchange of data and annotations between the two.

The Graph Annotation Framework (GrAF) (Ide and Suderman, 2007) is intended to serve as a "pivot" to enable interoperability among different formats for data and linguistics annotations and the systems that create and exploit them. In this paper, we describe the steps required to perform a round-trip rendering from GrAF to GATE and GrAF to UIMA CAS and back again, and outline the commonalities as well as the differences and gaps that came to light in the process. In doing so, we hope to shed some light on the design and implementation choices that either contribute to or impede progress toward interoperability, which can feed future development.

2 Background

A handful of formats for linguistic data and annotations have been proposed as standards over the past ten years, including Annotation Graphs (AG) (Bird and Liberman, 2001), and,

[1] http://www.oasis-open.org/committees/uima/

most recently, the Graph Annotation Framework (GrAF) (Ide and Suderman, 2007). UIMA's Common Analysis System (CAS) also provides a "common" way to represent annotations so that they can be shared and reused among UIMA annotator components.

Annotation Graphs were introduced primarily as a means to handle time-stamped speech data, in large part to overcome the problem of overlapping annotations that violate the strict tree structure of XML-based schemes. However, AGs are limited by the inability to represent hierarchical relations among annotations (as, for instance, in a syntax tree). AGs are used in GATE to represent standoff annotations.

GrAF has been developed by the International Standards Organization (ISO)'s TC37 SC4, as a part of the Linguistic Annotation Framework (International Standards Organization, 2008). GrAF provides an XML serialization of an abstract data model for annotations that is intended to serve as a "pivot" for transducing among user-defined and tool input annotation formats. GrAF is intended to function in much the same way as an *interlingua* in machine translation: a common, abstract conceptual representation into and out of which user- and tool-specific formats are transduced, so that a transduction of any specific format into and out of GrAF accomplishes the transduction between it and any number of other GrAF-conformant formats. GrAF is currently an ISO Candidate Draft.

The UIMA framework is a data management system that supports pipelined applications over unstructured data. UIMA was originally developed by IBM and is currently under further development by an OASIS technical committee[2]. Apache UIMA[3] is an Apache-licensed open source implementation of the UIMA specification being developed as an Apache incubator project. UIMA's Common Analysis System (CAS) is used to describe typed objects (annotations) associated with a given text or other media, upon which processing modules ("annotators") operate.

2.1 Annotation models

Each of the formats described above is based on some model of annotations and their relation to the data they describe. The AG model consists of sets of arcs defined over nodes corresponding to timestamps in primary data, each of which is labeled with an arbitrary linguistic description that applies to that region. Multiple annotations over the data produce multiple arcs; there is no provision for arcs associating annotations.

GrAF defines the *regions* to be annotated in primary data as the area bounded by two or more *anchors*. The definition of anchor and the number of anchors needed to define a region depends on the medium being annotated. The only assumption that GrAF makes is that anchors have a natural ordering. For textual data GrAF uses character offsets for anchors, and two anchors bound each region. Regions serve as the leaf nodes of a directed acyclic graph. Annotations in the form of feature structures are associated with nodes in the graph, including nodes associated with both regions and other annotations, via edges in the graph. GrAF can represent common annotation types such as hierarchical syntax trees by allowing, for example, a sentence annotation to have edges to constituent annotations such as NP, VP, etc. As opposed to AGs, annotations typically label nodes rather than edges in GrAF, although labeled edges are allowed, and the information comprising the annotations is represented using feature structures rather than simple labels.

The underlying model of UIMA CAS is similar to GrAF's, due to its hierarchical type system and the use of feature structures to represent annotation information. In fact, the GrAF model, consisting of a directed acyclic graph whose nodes are labeled with feature structures, provides the relevant abstraction underlying UIMA CAS. In principle, then, annotations represented in GrAF and UIMA CAS are trivially mappable to one another. The same is not true for AGs: in GrAF, annotations can be directly linked to other annotations, but in the AG model annotations are effectively independent layers linked to the primary data. As a result, while it is possible to "flatten" a GrAF representation so that it can be represented as an AG, it is not possible to take the round trip back into GrAF without losing information about relations among annotations. An AG can, of course, always be represented in GrAF, since independent graphs layered over data (possibly with shared anchors in the data) are valid GrAF structures.

[2]http://www.oasis-open.org/committees/uima/

[3]http://incubator.apache.org/uima/index.html

3 GrAF → UIMA → GrAF

Conversion of a GrAF data structure into UIMA involves generating (1) a UIMA data structure (a *CAS*), (2) a UIMA *type system*, and a specification of *type priorities*.

The CAS consists of a *subject of analysis* (sofa), which is the data (in our examples here, a text) itself, together with its annotations. The CAS XML representation of the annotations is very similar to the GrAF XML representation: each annotation is identified by its start and end location in the data expressed in terms of virtual nodes between each character in the data, where the position before the first character is node 0. The conversion of GrAF anchors to UIMA indexes is therefore trivial.

3.1 UIMA Type Systems

A UIMA type system specifies the type of data that can be manipulated by annotator components. A type system defines two kinds of objects; types and features. The *type* defines the kinds of data that can be manipulated in a CAS, arranged in an inheritance hierarchy. A *feature* defines a field, or slot, within a type. Each CAS type specifies a single supertype and a list of features that may be associated with that type. A type inherits all of the features from its supertype, so the features that can be associated with a type is the union of all features defined by all supertypes in the inheritance tree. A feature is a name/value pair where the value can be one of UIMA's built in primitive types (boolean, char, int, etc.) or a reference to another UIMA object. UIMA also allows feature values to be arrays of either primitive types or arrays of references to other objects.

UIMA defines a top level type *uima.cas.TOP* which contains no features and serves as the root of the UIMA type system inheritance tree. The root type *uima.cas.TOP* is the supertype of *uima.cas.AnnotationBase*, which is the supertype of *uima.tcas.Annotation*, which in turn is the supertype for *org.xces.graf.uima.Annotation*. All UIMA annotations generated by GrAF use *org.xces.graf.uima.Annotation* as their supertype. Note that the UIMA type hierarchy is strictly an *is-a* hierarchy; for example, there may be an annotation type *pos* with subtypes *penn_pos*, *claws_pos*, etc., indicating that each of these annotations are a kind of part of speech annotation. The hierarchy does not reflect other kinds of relations such as the relation between a "lemma" annotation and a "pos" annotation (i.e., a lemma and a pos are typically companion parts of a morpho-syntactic description, but neither one *is* a morpho-syntactic description), or constituency relations in syntactic annotation schemes.

The GrAF Java API provides a Java class that generates a valid UIMA type system given one or more GrAF objects. The type system is generated by iterating over all the nodes in the graph and creating a new type for each kind of annotation encountered (e.g., token, sentence, POS, etc.). Feature descriptions are generated for each type at the same time.

One drawback of deriving a type system automatically is that some of the power of UIMA type systems is lost in the conversion. For example, in the process of conversion, all feature values are assumed to be strings, even though UIMA allows specification of the type of a feature value. Since in GrAF, feature values have been serialized from the contents of an XML attribute, all feature values are represented internally as strings; to convert a feature value to any other representation would require that GrAF have some external knowledge of the annotation format being deserialized. Therefore, any type checking capability for feature value types in UIMA is lost after automatic generation of the type system. Similarly, it is not possible to determine a supertype for an annotation if it is more specific than *org.xces.graf.uima.Annotation* from the information in the GrAF representation alone, so in effect, it is not possible to derive any meaningful type hierarchy without additional knowledge. For example, it is not possible to include the information in the type system description that *penn_pos* and *claws_pos* are subtypes of *pos* since this information is not represented in the graph. Even in cases where this kind of information is represented in the graph, it is not retrievable; for example, FrameNet annotation includes a *grammaticalFunction* annotation whose children are elements such as `subject`, `object`, etc. However, there is no way to determine what the parent-child relation is between nodes without *a priori* knowledge of the annotation scheme.

Without a source of external knowledge, GrAF does not attempt to make any assumptions about the annotations and features in the graph. However, all of these problems are avoided by providing an XML Schema or other source of information about the GrAF annotations that can be used when generating the type system. The XML schema can specify the type hierarchy, data types and restricted ranges for feature values, etc. (see, for example, the XCES (Ide *et al.*, 2000) schema is used for the data and annotations in the American National Corpus (ANC)[4].)

3.2 UIMA Views and Indexes

A UIMA CAS object may contain more than one view of the artifact being annotated; for example, a CAS may contain an audio stream as one view and the transcribed text as another. Each view contains a copy of the artifact, referred to as the *subject of analysis* (*sofa*), and a set of indexes that UIMA annotators (processing modules) use to access data in the CAS. Each index is associated with one CAS type and indexes that type by its features–that is, the features are the keys for the index.

The indexes are the only way for UIMA annotators to access annotations in the CAS. It is necessary to generate these indexes, which are not provided automatically within UIMA. The GrAF Java API provides a module that generates the indexes at the same time the it generates the type system description. Since we do not know, and make no assumptions about, which annotations might be required by other annotators, all annotations are indexed by all of their features.

3.3 Type Priorities

Type priorities in UIMA are used to determine nesting relations when iterating over collections of annotations. That is, if two annotations have the same start and end offsets, then the order in which they will be presented by an iterator is determined by their type priority; the annotation with the highest priority will be presented first. Type priorities are specified by an ordered listing of annotation types, where order determines priority. In GrAF, annotation nesting is implicit in the graph itself.

To generate an explicit type priority specification for UIMA we must first obtain a list of all annotation types that appear in the graph and then sort the list based on the order they are encountered during a a depth first traversal of the graph. During the depth first traversal a *N x N precedence matrix* is constructed where *N* is the number of annotation types in the graph. If *precedes*[A,B] == *true* then A was encountered as an ancestor of B in the depth first traversal. If *precedes*[A,B] == *precedes*[B,A] == *true* then it is assumed that the annotation types have the same priority. Once the list of annotation types has been collected and the precedence matrix constructed, the matrix can be used to to sort the annotation types:

```
int compare(Annotation A,
            Annotation B,
            PrecedenceMatrix m)
{
   boolean AB = m.precedes(A,B);
   boolean BA = m.precedes(B,A);
   if (AB && BA)
   {
      return 0; // equal
   }
   else if (AB)
   {
      return -1; // A first.
   }
   else if (BA)
   {
      return 1; // B first.
   }
   // Neither AB or BA means A and
   // B are not in connected
   // components.
   return 0;
}
```

Not all nodes in the graph may be reachable in a depth first traversal, particularly if multiple annotations formats have been merged together. Therefore, after the initial traversal has been completed each node is checked to determine if it has been visited. If not, then another traversal is started from that node. This is repeated until all nodes/annotations in the graph have been visited at least once.

We have found that UIMA type priorities impose some limitations because they cannot represent context sensitive annotation orderings. For example, given

<!ELEMENT E1 (A,B)>
<!ELEMENT E2 (B,A)>

The order of `A` and `B` differs depending on whether the parent annotation is `E1` or `E2`. This type of relationship cannot be expressed by a simple ordering of annotations.

3.4 Naming Conflicts

The annotation type names used when generating the UIMA type system are derived automatically based on the annotation names used in the graph. Annotations in GrAF may also be grouped into named annotation sets and the gen-

[4]http://www.anc.org

```
<as type="POS">
   <a label="token">
       <fsr:fs type="PENN">
          <fsr:f name="msd" fVal="NN"/>
       </fsr:fs>
       <fsr:fs type="CLAWS5">
          <fsr:f name="msd" fVal="NN"/>
       </fsr:fs>
   </a>
</as>
```

Figure 1: GrAF representation of alternative POS annotations

erated UIMA type name consists of a concatenation of the nested annotation set names with the annotation label appended. For example, multiple part of speech annotations may be represented in different annotation sets, as shown in Figure 1.[5]

For the above example, two types will be generated: `POS_token_PENN` and `POS_token_CLAWS5`. However, GrAF places no restrictions on the names used for annotation set names, annotation labels, or feature structure types. Therefore, it is possible that the derived type name is not a valid UIMA identifier, which are required to follow Java naming conventions. For example, `Part-Of-Speech` is a valid name for an annotation label in GrAF, but because of the hyphen it is not a valid Java identifier and therefore not valid in UIMA.

To avoid the naming problem, a derived name is converted into a valid UIMA identifier before creating the UIMA type description. To permit round trip engineering, that is, ensuring a GrAF → UIMA → GrAF transformation results in the same GrAF representation as the original, a `NameMap` file is produced that maps a generated name to the compatible UIMA name. NameMaps can be used in a UIMA → GrAF conversion to ensure the GrAF annotations and annotation sets created are given the same names as they had in the original GrAF representation.

3.5 Preserving the Graph Structure

While UIMA does not have any graph-specific functionality, the value of a UIMA feature can be an array of annotations, or more specifically, an array of references to other annotations. In this way, annotations can effectively "point" to other annotations in UIMA. We exploit this capability to preserve the structure of the original graph in the UIMA representation, by adding two features to each annotation: `graf_children` and `graf_ancestors`. This information can be used to recreate the GrAF representation, should that ever be desired. It can also be used by UIMA annotators that have been designed to use and/or manipulate this information.

Although rarely used, GrAF permits edges in the graph to be annotated in the same way that nodes are. For UIMA conversion, if a graph contains labeled edges it must be converted into an equivalent graph without labeled edges. A graph with labeled edges can be converted into an equivalent graph without labeled edges, where a node replaces the original edge. To preserve the original graph structure, an attribute indicating that the node is represented as a a labeled edge in GrAF is included.

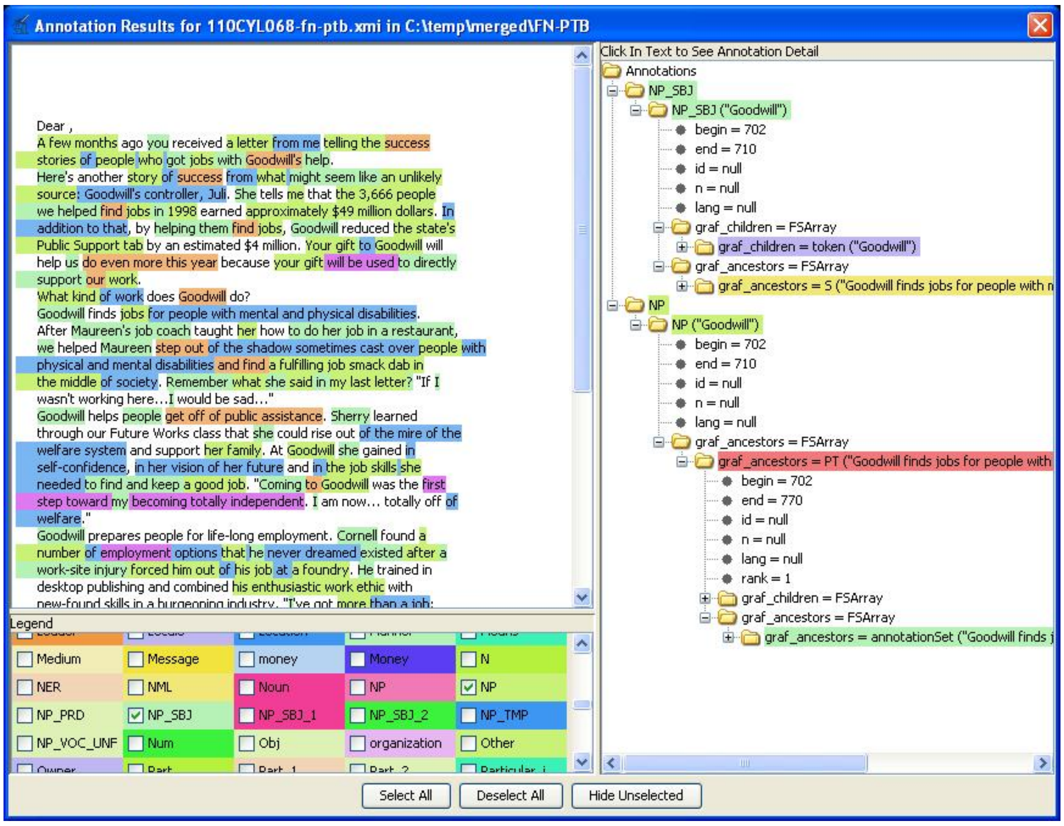
4 GrAF → GATE → GrAF

The conversion to/from GATE is much simpler than conversion to UIMA, since GATE is typeless and does not require the overhead of generating a type system or type priorities list. While GATE does support annotation schemas, they are optional, and annotations and features can be created at will. GATE is also much more lenient on annotation and feature names; names automatically generated by GrAF are typically valid in GATE.

Representing the graph structure in GATE is not as straightforward as it is in UIMA. We have developed a plugin to GATE that loads GrAF standoff annotations into GATE, and a parallel plugin that generates GrAF from GATE's internal format. As noted above, GATE uses annotation graphs to represent annotations, However, because annotation graphs do not provide for annotations of annotations, to transduce from GrAF to the GATE internal format it is necessary to "flatten" the graph so that nodes with edges to other nodes are modified to contain edges directly into the primary data. GATE assigns a unique id value to every annotation, so it is possible to link annotations by creating a special feature and referencing the parent/child annotations by their GATE id values.

The greatest difficulty in a GrAF → GATE conversion arises from the fact that in GATE, every

[5]The use of the `fVal` attribute in this example is subject to change according to revisions of ISO/DIS 24610-1 *Language Resource Management - Feature Structures - Part 1: Feature Structure Representation* (International Standards Organization, 2005), to which the representation of feature structures in GrAF adheres.


Annotation Results for 110CYL068-fn-ptb.xmi in C:\temp\merged\FN-PTB
Legend
Select All
Deselect All
Hide Unselected

Dear ,
A few months ago you received a letter from me telling the success
stories of people who got jobs with Goodwill's help.
Here's another story of success from what might seem like an unlikely
source: Goodwill's controller, Juli. She tells me that the 3,666 people
we helped find jobs in 1998 earned approximately $49 million dollars. In
addition to that, by helping them find jobs, Goodwill reduced the state's
Public Support tab by an estimated $4 million. Your gift to Goodwill will
help us do even more this year because your gift will be used to directly
support our work.
What kind of work does Goodwill do?
Goodwill finds jobs for people with mental and physical disabilities.
After Maureen's job coach taught her how to do her job in a restaurant,
we helped Maureen step out of the shadow sometimes cast over people with
physical and mental disabilities and find a fulfilling job smack dab in
the middle of society. Remember what she said in my last letter? "If I
wasn't working here…I would be sad…"
Goodwill helps people get off of public assistance. Sherry learned
through our Future Works class that she could rise out of the mire of the
welfare system and support her family. At Goodwill she gained in
self-confidence, in her vision of her future and in the job skills she
needed to find and keep a good job. "Coming to Goodwill was the first
step toward my becoming totally independent. I am now… totally off of
welfare."
Goodwill prepares people for life-long employment. Cornell found a
number of employment options that he never dreamed existed after a
work-site injury forced him out of his job at a foundry. He trained in
desktop publishing and combined his enthusiastic work ethic with
new-found skills in a burgeoning industry. "I've got more than a job:

```
Click In Text to See Annotation Detail
Annotations
⊟ NP_SBJ
  ⊟ NP_SBJ ("Goodwill")
      ● begin = 702
      ● end = 710
      ● id = null
      ● n = null
      ● lang = null
    ⊟ graf_children = FSArray
      ⊞ graf_children = token ("Goodwill")
    ⊟ graf_ancestors = FSArray
      ⊞ graf_ancestors = S ("Goodwill finds jobs for people with n
⊟ NP
  ⊟ NP ("Goodwill")
      ● begin = 702
      ● end = 710
      ● id = null
      ● n = null
      ● lang = null
    ⊟ graf_ancestors = FSArray
      ⊟ graf_ancestors = PT ("Goodwill finds jobs for people with
          ● begin = 702
          ● end = 770
          ● id = null
          ● n = null
          ● lang = null
          ● rank = 1
        ⊞ graf_children = FSArray
        ⊟ graf_ancestors = FSArray
          ⊞ graf_ancestors = annotationSet ("Goodwill finds j
```

☐ Medium	☐ Message	☐ money	☐ Money	☐ N
☐ NER	☐ NML	☐ Noun	☐ NP	✓ NP
☐ NP_PRD	✓ NP_SBJ	☐ NP_SBJ_1	☐ NP_SBJ_2	☐ NP_TMP
☐ NP_VOC_UNF	☐ Num	☐ Obj	☐ organization	☐ Other

Figure 2: UIMA rendering of GrAF annotations

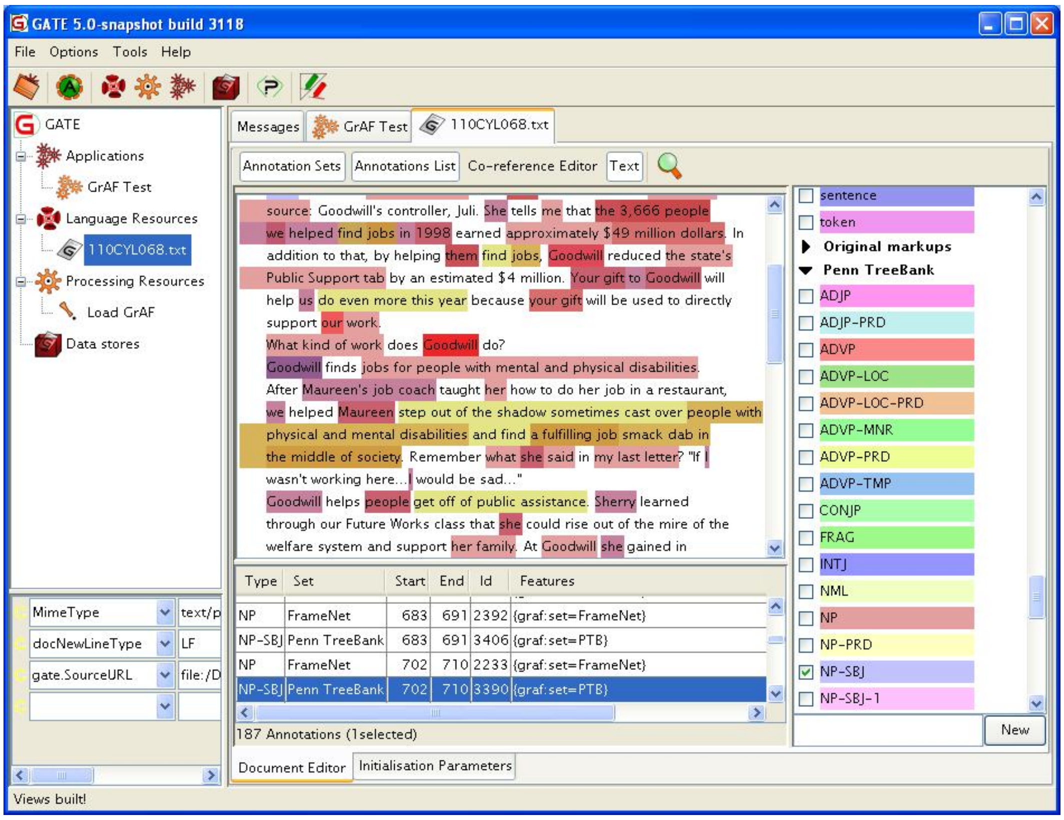
annotation is expected to have a start and end offset. In GrAF, a node may have multiple edges to other nodes that cover disjoint regions of text. For example, the FrameNet[6] annotation for a given verb typically includes edges to the associated role fillers (e.g., agent, theme, instrument, etc.), which are rarely contiguous in the text itself. Our current solution to this problem is to give a start and end offset that covers the smallest region of the text covering the regions associated with all descendants of the annotation, and recording the information concerning the original graph structure in attributes to enable reconversion into the original GrAF representation.

5 Exploiting Interoperability

GrAF is intended to serve as the *lingua franca* for data and annotations used in processing systems such as GATE and UIMA. As such, it provides a way for users to take advantage of each framework's strengths, e.g., UIMAs capabilities for deploying analysis engines as services that can be run remotely, and GATE's wide array of processing resources and capabilities for defining regular expressions over annotations (JAPE). It should be noted that GATE provides wrappers to allow a UIMA analysis engine to be used within GATE, and to allow a GATE processing pipeline to be used within UIMA. To share data and annotations between the two systems, it is necessary to construct a *mapping descriptor* to define how to map annotations between the UIMA CAS and the GATE Document, which operate similarly to the converters from and to GrAF from data and annotations described above. However, one advantage of using a GrAF representation as a pivot between the two systems is that when an annotation schema is used with GrAF data, the conversion from GATE to UIMA is more robust, reflecting the true type description and type priority hierarchies.

Using GrAF as a pivot has more general advantages, for example, by allowing annotations to be imported from and exported to a wide variety of formats, and also enabling merging annotations from disparate sources into a single annotation graph. Figure 2 shows a rendering of a Penn Treebank annotation (bracketed format) and a FrameNet annotation (XML) that have been transduced to GrAF, merged, and the transduced

[6]http://framenet.icsi.berkeley.edu/



MimeType	text/p
docNewLineType	LF
gate.SourceURL	file:/D

source: Goodwill's controller, Juli. She tells me that the 3,666 people we helped find jobs in 1998 earned approximately $49 million dollars. In addition to that, by helping them find jobs, Goodwill reduced the state's Public Support tab by an estimated $4 million. Your gift to Goodwill will help us do even more this year because your gift will be used to directly support our work.
What kind of work does Goodwill do?
Goodwill finds jobs for people with mental and physical disabilities.
After Maureen's job coach taught her how to do her job in a restaurant, we helped Maureen step out of the shadow sometimes cast over people with physical and mental disabilities and find a fulfilling job smack dab in the middle of society. Remember what she said in my last letter? "If I wasn't working here...I would be sad..."
Goodwill helps people get off of public assistance. Sherry learned through our Future Works class that she could rise out of the mire of the welfare system and support her family. At Goodwill she gained in

Type	Set	Start	End	Id	Features
NP	FrameNet	683	691	2392	{graf:set=FrameNet}
NP-SBJ	Penn TreeBank	683	691	3406	{graf:set=PTB}
NP	FrameNet	702	710	2233	{graf:set=FrameNet}
NP-SBJ	Penn TreeBank	702	710	3390	{graf:set=PTB}

187 Annotations (1selected)

Figure 3: GATE rendering of GrAF annotations

for use in UIMA. The same data is shown rendered in GATE in Figure 3. The two ”views” of the data consisting of overlaid annotations for each annotation type are visible in each rendering. There are multiple possibilities for exploiting and exploring merged annotations representing a range of annotation types within these two frameworks. For example, a UIMA analysis engine could be developed to identify regions annotated by both schemes, or all FrameNet elements that are annotated as `agent` and also annotated with Penn Treebank `NP-OBJ`, etc. In GATE, JAPE rules could locate patterns in annotations obtained from different sources, or named entity recognition rules could be enhanced with annotation information from data annotated in other formats. It would also be possible to compare multiple annotations of the same type, such as different tokenizations, different POS taggings , etc.

As a final note, we point out that in addition to conversion to UIMA and GATE, annotations from different sources (singly or merged in any combination) can also be converted to several other formats by using the GrAF Java API. The API allows the user to select from among existing annotations and specify an output format for their merged representation. Currently, in addition to GrAF, the following output formats are supported: XML documents with inline annotations; formats compatible with Monoconc Pro[7] and Wordsmith Tools[8]; NLTK[9]; CONLL (B-I-E) format; and UIMA CAS.[10] So, for example, it is possible to load a collection of standoff annotation files and convert to XML, and then present them to XML-aware applications as XML files with inline annotations. As a result, we are beginning to see possibilities for true interoperability among not only major frameworks like UIMA and GATE, but also applications with more limited functionalities as well as in-house formats. This, in turn, opens up the potential to mix and match among tools for various kinds of processing as appropriate to a given task. In general, the transduction of ”legacy schemes” such as Penn Treebank into GrAF greatly facilitates their use in major systems such as UIMA and GATE, as well as

[7]http://www.athel.com/mono.html

[8]http://www.lexically.net/wordsmith/

[9]http://www.nltk.org/

[10]Note that to render GrAF into GATE, a plugin within the GATE environment is used to perform the conversion.

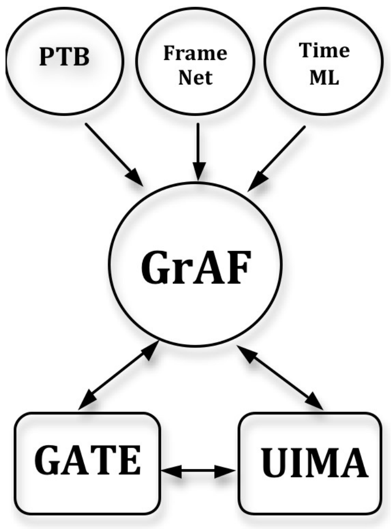


Figure 4: Conversion capabilities

other applications and systems. Figure 4 shows the conversion capabilities among a few annotations schemes, GrAF, and UIMA and GATE.

All of our conversion tools and GATE plugins are freely available for download with no restrictions at http://www.anc.org. The UIMA project has received support to develop a UIMA → GrAF conversion module, which should be available in the near future.

6 Conclusion

Consideration of the transduction from a generic, relatively abstract representation scheme such as GrAF into the formats required for widely adopted frameworks for creating and analyzing linguistically annotated data has several ramifications for interoperability. First, it brings to light the kinds of implementation choices that either contribute to or impede progress toward interoperability, which can feed future development. Second, our work on converting GrAF to the formats supported by UIMA and GATE shows that while minor differences exist, the underlying data models used by the two frameworks are essentially the same, as well as being very similar to the data model underlying GrAF. This is good news for interoperability, since it means that there is at least implicit convergence on the data model best suited for data and annotations; the differences lie primarily in the ways in which the model is serialized internally and as output by different tools. It also means that transduction among the various formats is possible without loss of information.

We have shown that a UIMA → GrAF or GATE → GrAF conversion is fairly straightforward; the expressive power of GrAF can easily represent the data models used by UIMA and GATE. On the other hand, GrAF → UIMA or GrAF → GATE transformations are less straightforward. Both frameworks can represent graphs, but neither provides a standard representation that other components are guaranteed to understand. Given that powerful analysis algorithms for data in graphs are well-established, there may be considerable advantage to using the graph as a general-purpose format for use within various modules and analytic engines. In any case, the generality and flexibility of the GrAF representation has already been shown to be an effective means to exchange linguistic data and annotations that exist in different formats, as well as a model for development of annotation schemes in the future.

Acknowledgments

This work was supported by an IBM UIMA Innovation Award and National Science Foundation grant INT-0753069.

References

Steven Bird and Mark Liberman. 2001. A Formal Framework for Linguistic Annotation. *Speech Communication*, 33:1-2, 23-60.

Nancy Ide and Keith Suderman. 2007. GrAF: A Graph-based Format for Linguistic Annotations. *Proceedings of the First Linguistic Annotation Workshop*, Prague, Czech Republic, June 28-29, 1-8.

International Standards Organization. 2008. Language Resource Management - Linguistic Annotation Framework. ISO Document WD 24611.

International Standards Organization. 2005. Language Resource Management - Feature Structures - Part 1: Feature Structure Representation. ISO Document ISO/DIS 24610-1.

Nancy Ide, Patrice Bonhomme, and Laurent Romary. 2000. XCES: An XML-based Standard for Linguistic Corpora. *Proceedings of the Second Language Resources and Evaluation Conference* (LREC), Athens, Greece, 825-30.

Hamish Cunningham. 2002. GATE, a General Architecture for Text Engineering. Computers and the Humanities, 36:223-254