Noname manuscript No.
(will be inserted by the editor)

Bridging the Gaps

Interoperability for Language Engineering Architectures Using GrAF

Nancy Ide · Keith Suderman

Received: date / Accepted: date

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 et al, 2002) and the Unstructured Information Management Architecture (UIMA) (Ferrucci and Lally, 2004). 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.

Keywords Linguistic annotation · Standards · Language resources · Annotation processing software

1 Introduction

Linguistically annotated corpora are required to develop sophisticated language models that can be used to improve natural language understanding capabilities. It has long been recognized that resource creation is time-consuming and costly, and there have been consistent calls within the field for resource reusability to offset some of those costs. One very basic requirement for reusability of linguistic annotations is their representation in a format that is processable by different software programs. While this could be accomplished by universal adoption of a single standard format for linguistic corpora and annotations, there is growing recognition that *interoperability* among formats, rather than universal use of a single representation format, is more suited to

Nancy Ide
Department of Computer Science, Vassar College
Tel.: +1 845 437 5988
Fax: +1 845 437 7498
E-mail: ide@cs.vassar.edu

Keith Suderman
Department of Computer Science, Vassar College

the needs of the community and language technology research in general. Interoperability is achieved when there is *conversion transitivity* between formats, as defined in Ide and Bunt (2010); that is, when transduction from one format to another can be accomplished automatically without information loss.

This paper explores interoperability for data represented using the Graph Annotation Format (GrAF) (Ide and Suderman, 2007) and the data formats utilized by two general-purpose annotation systems: the General Architecture for Text Engineering (GATE) (Cunningham et al, 2002) and the Unstructured Information Management Architecture (UIMA) (Ferrucci and Lally, 2004). GrAF is an XML format for representing language data and standoff annotations that was developed in ISO TC37 SC4 as a part of the Linguistic Annotation Framework (LAF) (**?**). GrAF is intended to serve as a "pivot" in order to facilitate interoperability among different formats for data and linguistics annotations and the systems that create and exploit them. UIMA and GATE are commonly-used frameworks that enable users to define pipelines of prefabricated software components that annotate language data, each of which uses a different internal representation for annotations over data. For GrAF to serve as a liaison between these two systems, conversion transitivity must hold between these internal formats and GrAF. In this paper, we first provide a general overview of GrAF and then describe the steps required to perform a round-trip rendering from GrAF to GATE and GrAF to UIMA 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 in turn feed future development.

2 Background

2.1 GrAF

GrAF has been developed by the International Standards Organization (ISO)'s TC37 SC4, as a part of the Linguistic Annotation Framework (LAF) (ISO, 2008). GrAF is the XML serialization of the LAF abstract model, which is based on a generic, abstract model consisting of a graph decorated with feature structures. GrAF is intended to serve primarily as a "pivot" for transducing among user-defined and tool input formats. As such, GrAF functions in much the same way as an *interlingua* in machine translation: as a common, abstract 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. Figure 1 shows the overall idea for six different user annotation formats (labeled A to F), which requires only two mappings for each scheme – one into and one out of the GrAF pivot format. The maximum number of mappings among schemes is therefore $2n$, vs. $n^2 - n$ mutual mappings without the pivot. GrAF is currently an ISO Candidate Draft.

Two of the most commonly-used platforms for generating automatic and manual annotations for language data are GATE (Cunningham et al, 2002), (Bontcheva et al, 2004) and UIMA (Ferrucci and Lally, 2004). Each of these systems uses a different model for representing data internally as well as for "dumping" these representations in a system-specific XML-based format. Given the widespread use of these two systems, means to transduce annotations from one representation to the other is desirable. We

describe below the internal model used by each of these systems, and then go on in the next sections to consider transducing GATE-produced annotations to UIMA format and vice versa, using GrAF as the intermediary.

2.2 GATE and Annotation Graphs

GATE (Cunningham et al, 2002), (Bontcheva et al, 2004) is an infrastructure for language processing developed at the University of Sheffield, first introduced in 1996. GATE uses a modified form of the representation format developed in the TIPSTER project (Grishman, 1997), later formalized as Annotation Graphs (AG) (Bird and Liberman, 2001). 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. 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. Formally, an Annotation Graph over a set of annotation labels L and timeline T is a 3-tuple $< N, A, t >$, where

- N is a set of nodes,
- A is a set of edges labeled with elements of L, and
- t is partial function from N to T satisfying the following conditions:
  1. $< N, A >$ is acyclic, with no nodes of degree zero, and
  2. for any path from node n_1 to n_2, if $t(n1)$ and $t(n_2)$ are defined, then $t(n_1) \leq t(n_2)$.

Under this definition, multiple annotations over the data produce multiple arcs; there is no provision for arcs associating one annotation with another. As a result, hierarchical structures such as syntax trees are difficult to represent using AGs. An *ad hoc* mechanism to represent hierarchy with AGs by including some of the structural information in arc labels has been developed (Cotton and Bird, 2002), but the resulting structure is not a "true" graph that is, for example, able to be traversed using standard graph traversal algorithms.

Following the AG Model, vertices of the GATE-internal AG are anchored in the document content; annotations label the arcs in the graph, each of which has a start node and an end node, an identifier, a type, and a set of simple feature-value pairs providing the annotation content. Instead of referring to timestamps, nodes have pointers into the content, e.g. character offsets for text, milliseconds for audio-visual content, etc. As such, the GATE internal model of annotations, like AGs, does not allow for associating annotations with other annotations and is therefore limited in its capacity to represent annotation hierarchies.

2.3 UIMA CAS

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[1]. Apache UIMA[2] is an

[1] http://www.oasis-open.org/committees/uima/
[2] http://incubator.apache.org/uima/index.html

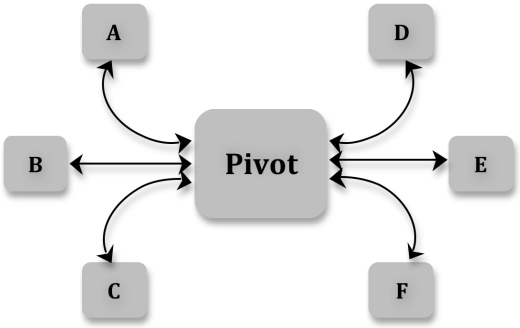

Fig. 1 GrAF as a pivot for six different annotation formats

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 medium, upon which processing modules ("annotators") operate. 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 isomorphic 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. As in GrAF, annotation information is expressed using (recursive) feature structures. UIMA provides mechanisms for specifying an annotation *type system* and a set of *type priorities*, which are described below in section 3.

The LAF abstract data model consists of a referential structure for associating stand-off annotations with primary data instantiated as a directed graph, and a feature structure representation for annotation content. In the LAF model, an annotation forms a directed graph referencing n-dimensional regions of primary data as well as other annotations, in which nodes (and possibly edges) are labeled with feature structures providing the annotation content. Formally, the LAF data model for annotations is defined as follows:

> A graph of annotations G is a set of vertices $V(G)$ and a set of edges $E(G)$. Vertices and edges may be labeled with one or more features. A feature consists of a quadruple $(G\prime, VE, K, V)$ where, $G\prime$ is a graph, VE is a vertex or edge in $G\prime$, K is the name of the feature and V is the feature value. Terminal nodes of the graph are associated with a set of one or more regions in primary data, which may provide the *base segmentation* for an annotation or several layers of annotation.

LAF has adopted the graph model for annotations for several reasons: first, graph theory provides a well-understood model for representing objects that can be viewed as a connected set of more elementary sub-objects, together with a wealth of graph-analytic algorithms for information extraction and analysis. As a result, the generic graph model has recently gained ground as a natural and flexible model for linguistic annotations that can represent all annotation varieties, even those that were not originally designed with the graph model as a basis (see for example Ide and Suderman

(2007)). Trees, which are restricted graphs, have long been used to describe syntactic annotations. As noted above, Annotation Graphs use multiple graphs over primary data to define data regions associated with annotations. More recently, the Penn Discourse TreeBank released its annotations of the Penn TreeBank as a graph, accompanied by an API that provides a set of standard graph-handling functions for query and access, and there is an increasing amount of work that treats linguistic annotations as graphs in order to identify, for example, measures of semantic similarity based on common subgraphs (for example, Cui et al (2005); Bunescu and Mooney (2007); Nguyen et al (2007); Gabrilovich and Markovitch (2007)).

A GrAF document represents the referential structure of an annotation with two XML elements: <node> and <edge>. Both <node> and <edge> elements may be labeled with associated annotation information. According to the LAF specification, an annotation is itself a graph representing a feature structure. In GrAF, feature structures are encoded in XML according to the specifications of ISO TC37 SC4 document ISO 24610. Note that the ISO specifications implement the full power of feature structures and define inheritance, unification, and subsumption mechanisms over the structures, thus enabling the representation of linguistic information at any level of complexity. The specifications also provide a concise format for representing simple feature-value pairs that suffices to represent many annotations, and which, because it is sufficient to represent the vast majority of annotation information, we use in our examples.

<edge> elements may also be labeled (i.e., associated with a feature structure), but this information is typically not an annotation *per se*, but rather information concerning the meaning, or role, of the link itself. For example, in PropBank, when there is more than one target of an annotation (i.e., a node containing an annotation has two or more outgoing edges), the targets may be either co-referents or a split argument whose constituents are not contiguous, in which case the edges collect an ordered list of constituents. In other cases, the outgoing edges may point to a set of alternatives. To differentiate the role of edges in such cases, the edge may be annotated. Unlabeled edges default to pointing to an ordered list of constituents.

A base segmentation is an annotation that contains only <region> elements (i.e., nodes with no outgoing edges). It is possible to define multiple base segmentations over the same data, where desired; each annotation is associated with one and only one base segmentation. GrAF defines *regions* in primary data as the area bounded by two or more *anchors*, which are first-class objects in the model. 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.

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.

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*. In principle, because they are based on the same model, annotations represented in GrAF and UIMA CAS are trivially mappable to one another. This is true in terms of the model, but there are a few details of the UIMA-internal implementation that require some additional steps.

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 performing a depth-first traversal of 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 used for the data and annotations in the American National Corpus (ANC)[3]).

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 that 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

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

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 generated 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 2.

```
<a label="token" as="PENN" ref="n0">
    <fs>
        <f name="msd" value="NN"/>
    </fs>
</a>
<a label="token" as="CLAWS5" ref="n0">
    <fs>
        <f name="msd" value="NN"/>
    </fs>
</a>
```

Fig. 2 GrAF representation of alternative POS annotations

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

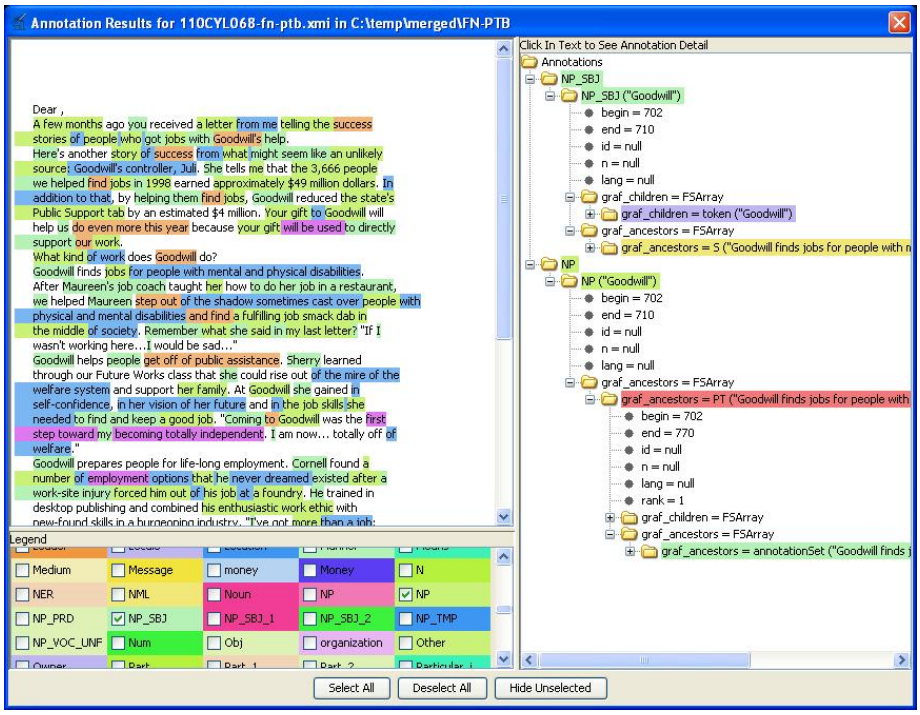

Fig. 3 UIMA rendering of GrAF annotations

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 annotation is expected to have a start and end pointer into the document content, and annotations are independent layers linked to the primary data. In GrAF, annotations can be directly linked to other annotations, and a node may have multiple edges to other nodes that cover (possibly) disjoint regions of text. For example, the FrameNet[4] annotation for a given verb typically includes edges to the associated role

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

fillers (e.g., agent, theme, instrument, etc.), each of which is an annotation itself, and all of which are rarely contiguous in the document. While it is always possible to "flatten" a GrAF representation so that it can be represented in GATE's internal model, it is not possible to take the round trip back into GrAF without losing information about relations among annotations, unless special metadata is provided on the edges. 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. This solution is roughly similar to the *ad hoc* strategy used to enable AGs to represent hierarchy.

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. Plugins for GATE to input and/or output annotations in GrAF format and a "CAS Consumer" to enable using GrAF annotations in UIMA are available at http://www.anc.org. We also provide a corpus reader for importing MASC data and annotations into NLTK.

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 3 shows a rendering of a Penn Treebank annotation (bracketed format) and a FrameNet annotation (XML) that have been transduced to GrAF, merged, and then transduced for use in UIMA. The same data is shown rendered in GATE in Figure 4. 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.

Annotations from different sources (singly or merged in any combination) can also be converted to several other formats. We provide a web service (Ide et al, 2010b)

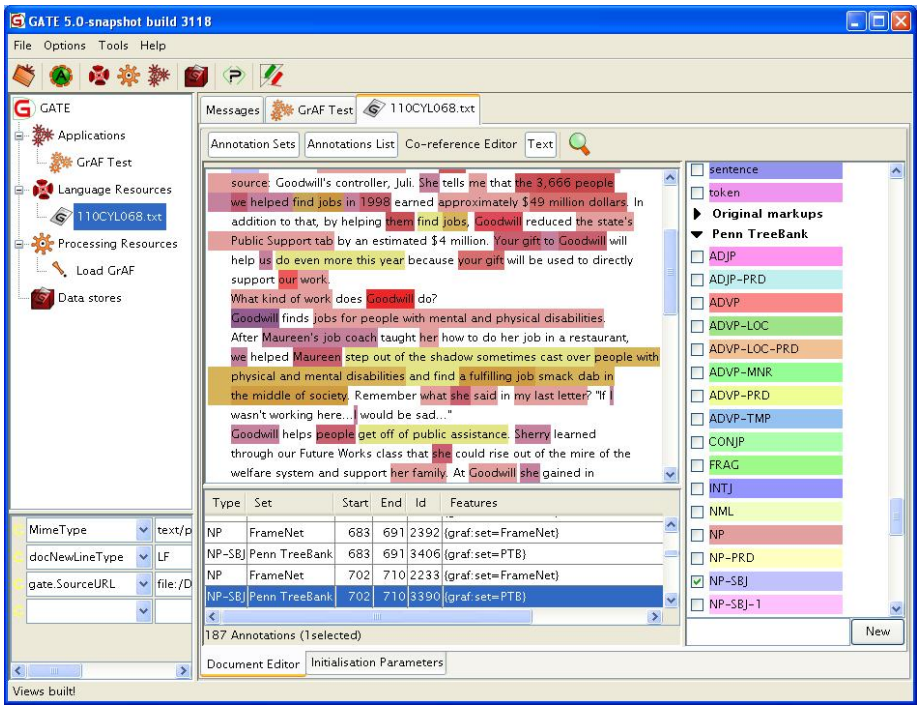

Fig. 4 GATE rendering of GrAF annotations

that allows users to select some or all parts of the Open American National Corpus (OANC)[5] and the Manually Annotated Sub-Corpus (MASC) (Ide et al, 2010a)–both of which are represented using GrAF–and choose among the available annotations. The service then generates a corpus and annotation "bundle" that is made available to the user for download. The following output formats are currently available:

1. in-line XML (XCES[6]), suitable for use with the BNCs XAIRA search and access interface[7] and other XML-aware software;
2. token / part of speech, a common input format for general-purpose concordance software such as MonoConc[8], as well as the Natural Language Toolkit (NLTK) (Bird et al, 2009);
3. CONLL IOB format, used in the Conference on Natural Language Learning shared tasks.[9]

We also provide a GrAF Java API[10] that can be used to access and manipulate GrAF annotations directly from Java programs, and render GrAF annotations in a

[5] http://www.anc.org
[6] XML Corpus Encoding Standard, http://www.xces.org
[7] http://xaira.sourceforge.net/
[8] http://www.athel.com/mono.html
[9] http://ifarm.nl/signll/conll
[10] http://www.anc.org/graf-api

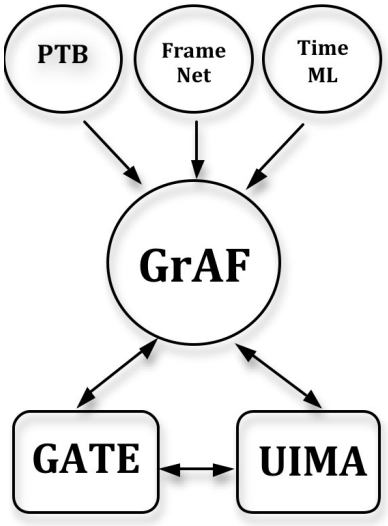

Fig. 5 Conversion capabilities

format suitable for input to other software such as the open source GraphViz[11] graph visualization application.

Finally, because the GrAF format is isomorphic to input to many graph-analytic tools, existing graph-analytic software can also be exploited to search and manipulate GrAF annotations. Trivial merging of GrAF-based annotations involves simply combining the graphs for each annotation, after which graph minimization algorithms[12] can be applied to collapse nodes with edges to common subgraphs to identify commonly annotated components. Graph-traversal and graph-coloring algorithms can also be applied in order to identify and generate statistics that could reveal interactions among linguistic phenomena that may have previously been difficult to observe. Other graph-analytic algorithms — including common sub-graph analysis, shortest paths, minimum spanning trees, connectedness, identification of articulation vertices, topological sort, graph partitioning, etc. — may also prove to be useful for mining information from a graph of annotations at multiple linguistic levels.

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 for 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 other applications and systems. Figure 5 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

[11] http://www.graphviz.org/

[12] Efficient algorithms for graph merging exist; see, e.g., Habib et al (2000).

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

Bird S, Liberman M (2001) A formal framework for linguistic annotation. Speech Commun 33(1-2):23–60

Bird S, Klein E, Loper E (2009) Natural Language Processing with Python, 1st edn. O'Reilly Media

Bontcheva K, Tablan V, Maynard D, Cunningham H (2004) Evolving GATE to meet new challenges in language engineering. Natural Language Engineering 10(3-4):349–373

Bunescu RC, Mooney RJ (2007) Extracting relations from text: From word sequences to dependency paths. In: Kao A, Poteet S (eds) Text Mining and Natural Language Processing, Springer, pp 29–44

Cotton S, Bird S (2002) An integrated framework for treebanks and multilayer annotations. In: Proceedings of the Third International Conference on Language Resources and Evaluation

Cui H, Sun R, Li K, yen Kan M, seng Chua T (2005) Question answering passage retrieval using dependency relations. In: In SIGIR 2005, ACM Press, pp 400–407

Cunningham H, Maynard D, Bontcheva K, Tablan V (2002) GATE: A framework and graphical development environment for robust nlp tools and applications. In: Proceedings of ACL'02

Ferrucci D, Lally A (2004) UIMA: An architectural approach to unstructured information processing in the corporate research environment. Natural Language Engineering 10(3-4):327–348

Gabrilovich E, Markovitch S (2007) Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: In Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp 1606–1611

Grishman R (1997) TIPSTER architecture design document version 2.3, technical report, DARPA

Habib M, Paul C, Viennot L (2000) Partition refinement techniques: an interesting algorithmic tool kit. International Journal of Foundations of Computer Science 175

Ide N, Bunt H (2010) Anatomy of annotation schemes: Mapping to GrAF. In: Proceedings of the Fourth Linguistic Annotation Workshop, Association for Computational Linguistics, Uppsala, Sweden, pp 247–255

Ide N, Suderman K (2007) GrAF: A graph-based format for linguistic annotations. In: Proceedings of the Linguistic Annotation Workshop, Association for Computational Linguistics, pp 1–8

Ide N, Bonhomme P, Romary L (2000) XCES: An XML-based encoding standard for linguistic corpora. In: Proceedings of the Second International Language Resources and Evaluation Conference. Paris: European Language Resources Association

Ide N, Baker C, Fellbaum C, Passonneau R (2010a) The manually annotated subcorpus: A community resource for and by the people. In: Proceedings of the ACL 2010 Conference Short Papers, Association for Computational Linguistics, Uppsala, Sweden, pp 68–73

Ide N, Suderman K, Simms B (2010b) ANC2Go: A web application for customized corpus creation. In: Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC), European Language Resources Association, Valletta, Malta

ISO (2008) Language Resource Management - Linguistic Annotation Framework. ISO Document WD 24611

Nguyen DPT, Matsuo Y, Ishizuka M (2007) Exploiting syntactic and semantic information for relation extraction from wikipedia. In: In IJCAI07-TextLinkWS