LAF/GrAF-grounded Representation of Dependency Structures

Yoshihiko Hayashi[§], **Thierry Declerck**[¶], **Chiharu Narawa**[§]

[§]Graduate School of Language and Culture, Osaka University
1-8 Machikaneyama, Toyonaka, 5600043, Japan
{hayashi, narawa}@lang.osaka-u.ac.jp
[¶]DFKI GmbH, Language Technology Lab
Stuhlsatzenhausweg, 3, D-66123 Saarbrücken, Germany
declerck@dfki.de

Abstract

This paper shows that a LAF/GrAF-based annotation schema can be used for the adequate representation of syntactic dependency structures in many languages. We first argue that there are at least two types of textual units that can be annotated with dependency information: words/tokens and chunks/phrases. Based on this consideration, we discuss a sub-typing of GrAF to represent the corresponding dependency structures. We then describe a wrapper program that, as a proof of concept, converts output data from different dependency parsers in proprietary XML formats to the GrAF-compliant XML representation.

1. Introduction

Dependency structures attract a great deal of attention as a representation of natural language utterances, which is offering a good support for semantic annotation in the context of the Semantic Web and other communities. This interest is underlined by a recent increase of the publicized language resources based on the notion of dependency, including text corpora (Brants et al., 2002; Kurohashi and Nagao, 2003) as well as parsing tools (Kudo and Matsumoto, 2002; Marneffe and Manning, 2008; Kübler et al., 2009).

In order to ensure the reusability and interoperability of these resources, a standardized annotation of dependency structures that can be applied to a variety of languages should be used. LAF/GrAF (Ide and Romary, 2006; Ide and Suderman, 2007) and SynAF (Declerck, 2008) are on-going ISO TC37/SC4 standardization activities[1], which deal respectively with a generic meta-model for linguistic annotation and with a meta-model for syntactic annotation, including dependency structures.

This paper presents concrete experiments done with those annotation models, which also provided feedback to the editorial committees of the corresponding ISO groups. We focus on the basic units of dependency relations that can be detected in the analysis of texts in many languages. We argue, in line with the SynAF initiative, that there should be at least two types of textual units that can be annotated with dependency information: words/tokens and chunks/phrases. Based on this consideration, we discuss a sub-typing of GrAF to represent the corresponding dependency structures. Finally we describe wrapper programs that, as a proof of concept, convert output data from different dependency parsers in proprietary XML formats to the GrAF-conformant XML representation, which is also giving the base for the SynAF representation format.

[1]`http://www.tc37sc4.org/`

2. Representation of Dependency Structures

2.1. Definition of Dependency Structure

Dependency structure is a way of representing the syntactic properties of a natural language utterance by focusing on directed binary relations (dependencies) between two elements of the utterance: One element is called the *head/governor* and the other the *modifier/dependent*. A dependency relation can be labeled with a specific type of grammatical/functional relation and annotated by some additional information such as the estimated probability of occurrence. A dependency structure of an utterance can be formally modeled by a graph, where a node in the graph represents a linguistic unit, and a (directed) edge linking two nodes in the graph is denoting the dependency relation between a head/governor and a modifier/dependent.

2.2. Basic Units of Dependency

(Kübler et al., 2009) and other literatures state that the basic unit of dependency in general is a *word*. We however argue that there are at least two types of linguistic units that can be annotated with dependency information: *words/tokens* and *chunks/phrases*. We motivate the necessity of the latter type as follows.

2.2.1. Japanese dependency structure

Japanese is a head-final language and the syntactic structure of a sentence can be represented with so-called *Kakari-Uke* structure. A Kakari-Uke structure is described by a set of non-projective relations, in which a phrasal unit (or Bunsetsu chunk; hereafter *B-chunk*) modifies another one (modifee) that is somewhere right to the modifier. Many Japanese syntactic parsers, for example CaboCha (Kudo and Matsumoto, 2002)[2], make use of this property. Note that a B-chunk consists of one or more content word and zero or more succeeding functional words. Thus a B-chunk could be aligned, in English, to a minimum noun/prepositional phrase, adverb phrase, or verb phrase.

[2]`http://sourceforge.net/projects/cabocha/`

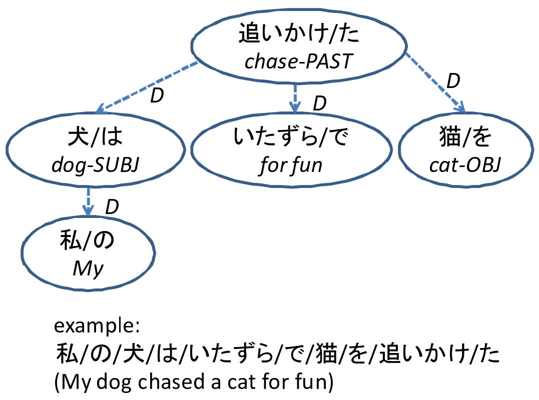


Figure 1: An example of Japanese dependency structure.

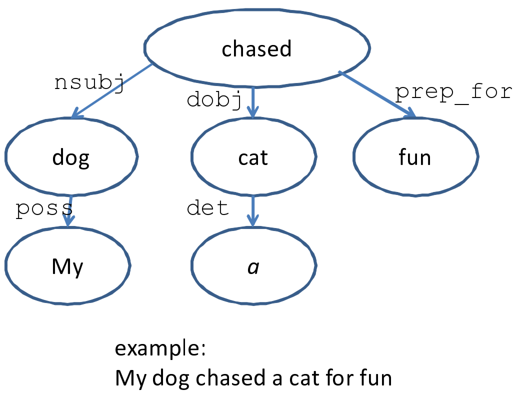


Figure 2: An example of English collapsed dependency structure.

Figure 1 exemplifies a Japanese Kakari-Uke structure, where each node represents a B-chunk and each directed edge represents a dependency relation between B-chunks. Note also in Fig. 1 that a B-chunk consists of more than one words, whose boundaries are indicated by ”/” for convenience.

For the syntactic analysis of Japanese sentences, dependency parsing has been excessively utilized since the 1980’s rule-based machine translation era, since the syntactic behavior of B-chunks is much clearer (more specified) than that of words (Kurohashi and Nagao, 2003), and the structure is very suitable for semantic analysis that utilizes, for example, the case frame of the main verb. Another reason lies in the fact that Japanese allows word order variations (or *scrambling*) in the B-chunk level.

2.2.2. Collapsed representation of English dependency structure

Marneffe and Manning (2008) propose a collapsed representation of English dependency structure, where a node for certain types of words, typically prepositions in English, is collapsed and turned to a part of the dependency relation label. Figure 2 illustrates an example of the collapsing representation, as generated by the Stanford parser[3]. Each node corresponds to a word in the sentence, and each edge, except the one labeled as `prep_for`, represents a dependency relation between words. In this representation, there exists no node for the word ”for”; instead, a dependency label `prep_for` is introduced for the prepositional phrase “ for fun. ”Notice here that this representation is quite similar to the Japanese dependency structure as shown in Fig. 1, where each node is essentially devoted to a content word. This type of representation scheme has been introduced by carefully examining the trade-off between the linguistic fidelity and the usability of the representation, arguing that the collapsed representation is more useful for semantic tasks such as relation extraction. The collapsed representation could be further collapsed towards a higher semantic representation level, which could be more adequate when we are to align dependency structures cross-linguistically.

2.3. Relevant Work on Standardization of Annotation of Dependency Structures

As mentioned earlier, dependency structures can be represented by the kind of graphs described in the ongoing ISO LAF/GrAF initiative. LAF (Linguistic Annotation Framework) (Ide and Romary, 2006) provides a general framework for representing linguistic annotations. The data model is based on directed graph, where a node represents a linguistic unit, and a edge indicates a relation between the linguistic units. These nodes and edges may be labeled by feature structures. LAF does not contribute to semantic interoperability of linguistic annotations, as it does not specify anything about the linguistic categories in annotations. It however provides us with a solid conceptual foundation for the syntactic interoperability of linguistic annotations. GrAF (Ide and Suderman, 2007) is an XML serialization of the generic graph structure of linguistic annotations specified by LAF. The presented work has utilized the XML schema obtained from the Web site[4].

As the LAF/GrAF is a highly generic framework for representing unconstrained/untyped linguistic annotations, we need to go into more details for proposing a graph-based representation scheme, which is appropriate for dependencies. In this regard, we can base on the on-going SynAF (Syntactic Annotation Framework) (Declerck, 2008) activity which is dealing with the multi-layered annotation of both constituency and dependency structures. The work we describe in this paper is actually proposing in a way a bridge between SynAF and LAF/GrAF. Since in SynAF no concrete annotation examples are available at this stage, we want to show the validity of the GrAF approach for the particular dependency annotation task.

Toward the standardization of dependency structures annotation, we also need to develop a reasonable inventory of dependency relation labels together with their proper definition. Some work has already started in this direction at ISO in the context of the release of the ISOcat platform[5], also as a complement to the SynAF annotation model.

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

[4]`http://www.xces.org/ns/GrAF/0.99/`

[5]`http://www.isocat.org/`

3. Sub-typing GrAF for Representing Dependency Structures

We describe an informative sub-typing of GrAF to adequately represent the two types of basic units of dependency relations we suggested above. Sub-typing GrAF in this context means introducing sub-types of GrAF node and edge. Figure 3 overviews our proposal on sub-typing GrAF, where the thick dotted arrow represents a dependency between the chunks.

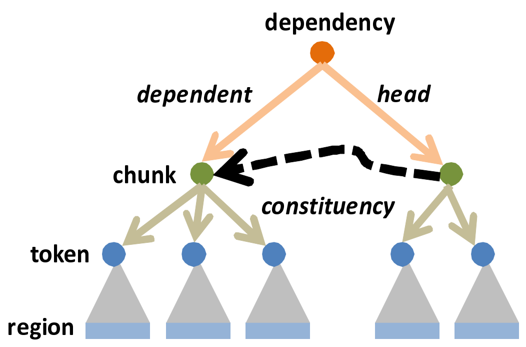


Figure 3: Overview of the Sub-typing of GrAF.

3.1. Sub-typing of GrAF Node and Edge

In order to represent the two basic units of dependency relations, we need two node types: a node type for words/tokens and a node type for phrases/chunks. According to the GrAF specification, these types of nodes can be annotated by attaching feature structures. To a word/token node we assign a feature structure describing its morpho-syntactic information, while to a phrase/chunk node we assign a feature structure describing its syntactic category. Every token node is associated with the corresponding span (or region) in text by a `<link>` element: GrAF specification allows a `<node>` element to have `<link>` elements as the immediate daughters. In addition, we further introduce another type of node for representing dependency. A dependency node carries information on a dependency relation, such as its label and other information such as the estimated probability of occurrence.

For edges, we introduce two types: one is to represent a chunk constituency (chunk constituency edge) and the other is to represent a dependency relation (dependency element edge). The chunk constituency edge links a chunk node to its constituent token nodes; or to other chunk nodes if the chunk is complex. As the SynAF strongly encourages, we allow a complex chunk that includes nested smaller chunks[6]. A dependency element edge denotes a part of a dependency relation: it links a dependency node with a token node or a chunk node, depending on the type of the dependency structure. The dependency element edge can be further divided into two sub-types: one marks the head of a dependency relation, while the other marks the dependent.

[6]To represent Japanese Kakari-Uke structures, however, no nested chunks are necessary.

The sub-typing of GrAF edge can be summarized as follows, where the values of `from` and `to` attributes of `<edge>` element are constrained.

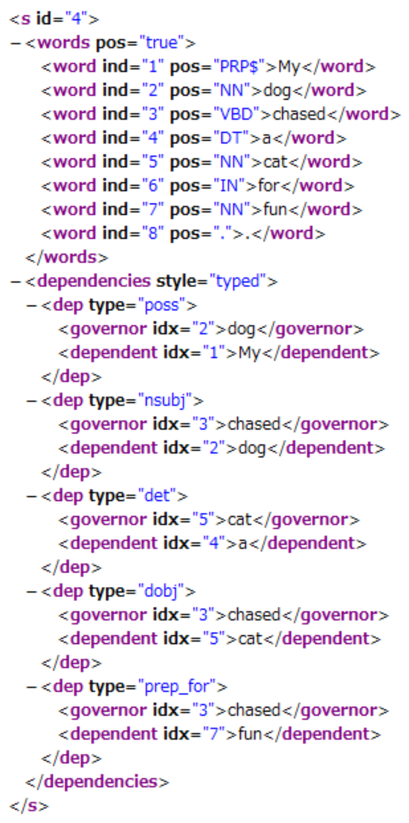
edge type = chunk constituency

- **`from`:** chunk node
- **`to`:** token node, chunk node (for complex chunk)
- **`type` in `<as> element`:** ”chunk_const”

edge type = dependency element

- **`from`:** dependency node
- **`to`:** token node (for token-based dependency), chunk node (for chunk-based dependency)
- **`type` in `<as> element`:** ”dep_head” (for head), ”dep_dependent” (for dependent)

There exists an obvious alternative to represent a dependency structure, in which an edge directly encodes the dependency. With this rendering, it turned out that `from` and `to` attributes carry special connotation: `from` denotes a head, while `to` denotes a dependent. This may not be adequate, because `from/to` attribute of a GrAF edge should simply designate the source/destination, and has nothing to do with the semantics of a dependency structure.

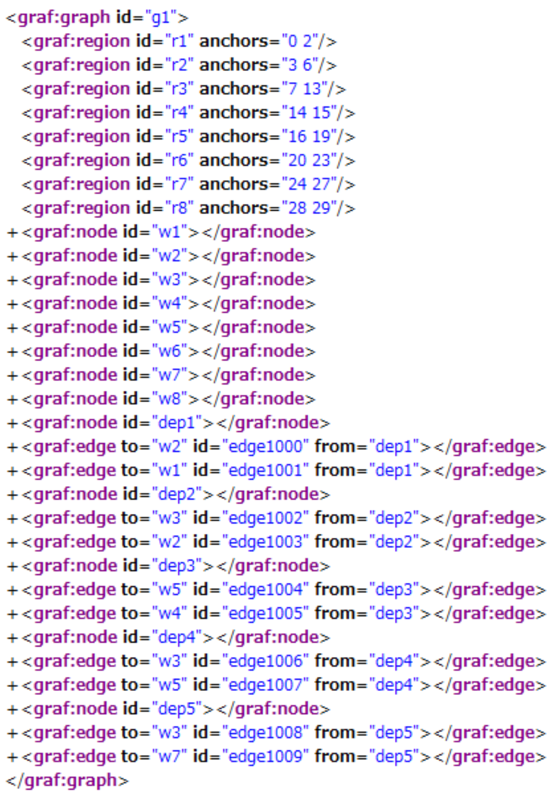


```
<s id="4">
- <words pos="true">
    <word ind="1" pos="PRP$">My</word>
    <word ind="2" pos="NN">dog</word>
    <word ind="3" pos="VBD">chased</word>
    <word ind="4" pos="DT">a</word>
    <word ind="5" pos="NN">cat</word>
    <word ind="6" pos="IN">for</word>
    <word ind="7" pos="NN">fun</word>
    <word ind="8" pos=".">.</word>
  </words>
- <dependencies style="typed">
  - <dep type="poss">
      <governor idx="2">dog</governor>
      <dependent idx="1">My</dependent>
    </dep>
  - <dep type="nsubj">
      <governor idx="3">chased</governor>
      <dependent idx="2">dog</dependent>
    </dep>
  - <dep type="det">
      <governor idx="5">cat</governor>
      <dependent idx="4">a</dependent>
    </dep>
  - <dep type="dobj">
      <governor idx="3">chased</governor>
      <dependent idx="5">cat</dependent>
    </dep>
  - <dep type="prep_for">
      <governor idx="3">chased</governor>
      <dependent idx="7">fun</dependent>
    </dep>
  </dependencies>
</s>
```

Figure 4: English dependency structure in the Stanford Parser XML format.

3.2. Working Examples

Here we show examples from the Stanford English parser and the Japanese dependency parser CaboCha.
Given the example sentence shown in Fig 2, the Stanford parser returns the analysis result in its proprietary XML format as shown in Fig. 4. As seen in the figure: only the part-of-speech information is assigned to each word; a dependency relation is encoded by using `<dep>` element, in which the head (`<governor>`) and the dependent (`<dependent>`) are designated by word indices (`idx` attribute); the dependency type is given as the value of `type` attribute.
This XML document is converted into the GrAF-compliant XML document as shown in Fig. 5 by the dependency parser wrapper described in the next section. Although each top-level XML element is folded in the figure[7], some of them are unfolded and shown in Fig. 6 (token nodes) and in Fig. 7 (a dependency node and the corresponding dependency element edges). These together represent a `nsubj`-type dependency whose head is the word ”chased” (`w3`) and the dependent is the word ”dog” (`w2`).

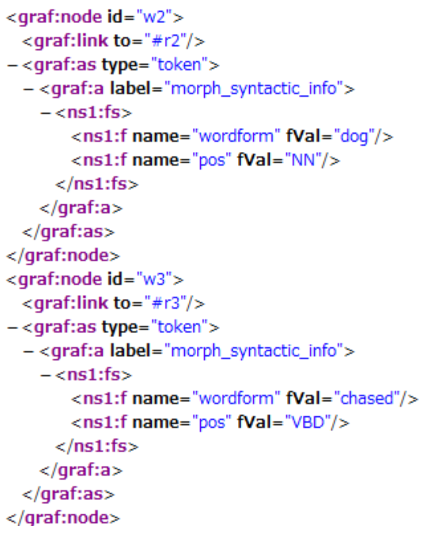


```
<graf:graph id="g1">
  <graf:region id="r1" anchors="0 2"/>
  <graf:region id="r2" anchors="3 6"/>
  <graf:region id="r3" anchors="7 13"/>
  <graf:region id="r4" anchors="14 15"/>
  <graf:region id="r5" anchors="16 19"/>
  <graf:region id="r6" anchors="20 23"/>
  <graf:region id="r7" anchors="24 27"/>
  <graf:region id="r8" anchors="28 29"/>
+ <graf:node id="w1"></graf:node>
+ <graf:node id="w2"></graf:node>
+ <graf:node id="w3"></graf:node>
+ <graf:node id="w4"></graf:node>
+ <graf:node id="w5"></graf:node>
+ <graf:node id="w6"></graf:node>
+ <graf:node id="w7"></graf:node>
+ <graf:node id="w8"></graf:node>
+ <graf:node id="dep1"></graf:node>
+ <graf:edge to="w2" id="edge1000" from="dep1"></graf:edge>
+ <graf:edge to="w1" id="edge1001" from="dep1"></graf:edge>
+ <graf:node id="dep2"></graf:node>
+ <graf:edge to="w3" id="edge1002" from="dep2"></graf:edge>
+ <graf:edge to="w2" id="edge1003" from="dep2"></graf:edge>
+ <graf:node id="dep3"></graf:node>
+ <graf:edge to="w5" id="edge1004" from="dep3"></graf:edge>
+ <graf:edge to="w4" id="edge1005" from="dep3"></graf:edge>
+ <graf:node id="dep4"></graf:node>
+ <graf:edge to="w3" id="edge1006" from="dep4"></graf:edge>
+ <graf:edge to="w5" id="edge1007" from="dep4"></graf:edge>
+ <graf:node id="dep5"></graf:node>
+ <graf:edge to="w3" id="edge1008" from="dep5"></graf:edge>
+ <graf:edge to="w7" id="edge1009" from="dep5"></graf:edge>
</graf:graph>
```

Figure 5: LAF/GrAF compliant XML document for the Stanford parser (folded).

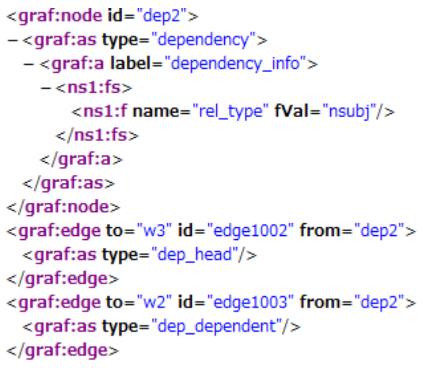
CaboCha returns the analysis result in its proprietary XML format as shown in Fig. 8, given the example sentence shown in Fig 1. A dependency relation is encoded in the `<chunk>` element by using `link` attribute, where the value denotes the id of the head B-chunk. Also notice from this example that morphological information for a token is given with `feature` attribute in the `<tok>` element.

[7]The figure is captured from the Firefox screen.



```
<graf:node id="w2">
  <graf:link to="#r2"/>
– <graf:as type="token">
  – <graf:a label="morph_syntactic_info">
    – <ns1:fs>
        <ns1:f name="wordform" fVal="dog"/>
        <ns1:f name="pos" fVal="NN"/>
      </ns1:fs>
    </graf:a>
  </graf:as>
</graf:node>
<graf:node id="w3">
  <graf:link to="#r3"/>
– <graf:as type="token">
  – <graf:a label="morph_syntactic_info">
    – <ns1:fs>
        <ns1:f name="wordform" fVal="chased"/>
        <ns1:f name="pos" fVal="VBD"/>
      </ns1:fs>
    </graf:a>
  </graf:as>
</graf:node>
```

Figure 6: LAF/GrAF compliant XML document for the Stanford parser (partial; tokens).



```
<graf:node id="dep2">
– <graf:as type="dependency">
  – <graf:a label="dependency_info">
    – <ns1:fs>
        <ns1:f name="rel_type" fVal="nsubj"/>
      </ns1:fs>
    </graf:a>
  </graf:as>
</graf:node>
<graf:edge to="w3" id="edge1002" from="dep2">
  <graf:as type="dep_head"/>
</graf:edge>
<graf:edge to="w2" id="edge1003" from="dep2">
  <graf:as type="dep_dependent"/>
</graf:edge>
```

Figure 7: LAF/GrAF compliant XML document for the Stanford parser (partial; dependency).

As in the Stanford parser example, the dependency parser wrapper converts the XML document into the GrAF-compliant XML document. Figure 9 focuses on three parts in the resulted XML document: `chunk2`, shown in (b), (consists of `tok3` and `tok4`; both shown in (a)) is the head of the dependency relation `dep1` whose dependent is `chunk1` as shown in (c).
Let us remind again here that the English dependency structure is word-based, while the Japanese dependency structure is chunk-based.

4. Dependency Parser Wrapper

As a proof of concept, we have developed wrapper programs that convert data in proprietary format coming from an actual dependency parser to the data compliant to the proposed LAF/GrAF-based XML schema. Figure 10 schematizes the dependency parser wrapper configuration. These wrappers are currently implemented with handcrafted XSL stylesheets and a generic XSLT processor. The definition of XML schema good for both types of dependency structure is rather straightforward, and the resulted XML schema can be utilized to validate XML data gener-

```
<sentence>
- <chunk id="0" link="1" rel="D" score="1.246175" head="0" func="1">
    <tok id="0" feature="名詞,代名詞,一般,*,*,*,私,ワタシ,ワタシ" ne="O">私</tok>
    <tok id="1" feature="助詞,連体化,*,*,*,*,の,ノ,ノ" ne="O">の</tok>
  </chunk>
- <chunk id="1" link="4" rel="D" score="0.000000" head="2" func="3">
    <tok id="2" feature="名詞,一般,*,*,*,*,犬,イヌ,イヌ" ne="O">犬</tok>
    <tok id="3" feature="助詞,係助詞,*,*,*,*,は,ハ,ワ" ne="O">は</tok>
  </chunk>
- <chunk id="2" link="4" rel="D" score="0.000000" head="4" func="5">
    <tok id="4" feature="名詞,サ変接続,*,*,*,*,いたずら,イタズラ,イタズラ" ne="O">いたずら</tok>
    <tok id="5" feature="助詞,格助詞,一般,*,*,*,で,デ,デ" ne="O">で</tok>
  </chunk>
- <chunk id="3" link="4" rel="D" score="0.000000" head="6" func="7">
    <tok id="6" feature="名詞,一般,*,*,*,*,猫,ネコ,ネコ" ne="O">猫</tok>
    <tok id="7" feature="助詞,格助詞,一般,*,*,*,を,ヲ,ヲ" ne="O">を</tok>
  </chunk>
- <chunk id="4" link="-1" rel="D" score="0.000000" head="8" func="9">
    <tok id="8" feature="動詞,自立,*,*,一段,連用形,追いかける,オイカケ,オイカケ" ne="O">追いかけ</tok>
    <tok id="9" feature="助動詞,*,*,*,特殊・タ,基本形,た,タ,タ" ne="O">た</tok>
  </chunk>
</sentence>
```

Figure 8: Japanese dependency structure in CaboCha XML format.

a

```
<graf:node id="tok1">
  <graf:link to="#r1"/>
- <graf:as type="token">
  - <graf:a label="morpho_syntactic_info">
    - <ns1:fs>
        <ns1:f name="wordform" fVal="私"/>
        <ns1:f name="g_feature" fVal="名詞,代名詞,一般,*,*,*,私,ワタシ,ワタシ"/>
      </ns1:fs>
    </graf:a>
  - <graf:a label="named_entity_info">
    - <ns1:fs>
        <ns1:f name="ne_tag" fVal="O"/>
      </ns1:fs>
    </graf:a>
  </graf:as>
</graf:node>
<graf:node id="tok2">
  <graf:link to="#r2"/>
- <graf:as type="token">
  - <graf:a label="morpho_syntactic_info">
    - <ns1:fs>
        <ns1:f name="wordform" fVal="の"/>
        <ns1:f name="g_feature" fVal="助詞,連体化,*,*,*,*,の,ノ,ノ"/>
      </ns1:fs>
    </graf:a>
  - <graf:a label="named_entity_info">
    - <ns1:fs>
        <ns1:f name="ne_tag" fVal="O"/>
      </ns1:fs>
    </graf:a>
  </graf:as>
</graf:node>
```

b

```
<graf:edge to="tok1" id="e100" from="chunk1">
  <graf:as type="chunk_const"/>
</graf:edge>
<graf:edge to="tok2" id="e101" from="chunk1">
  <graf:as type="chunk_const"/>
</graf:edge>
<graf:node id="chunk1">
- <graf:as type="chunk">
  - <graf:a label="chunk_internal_info">
    - <ns1:fs>
        <ns1:f name="head" fVal="tok1"/>
        <ns1:f name="func" fVal="tok2"/>
      </ns1:fs>
    </graf:a>
  </graf:as>
```

c

```
<graf:node id="dep1">
- <graf:as type="dependency">
  - <graf:a label="dependency_info">
    - <ns1:fs>
        <ns1:f name="rel_type" fVal="D"/>
        <ns1:f name="score" fVal="1.246175"/>
      </ns1:fs>
    </graf:a>
  </graf:as>
</graf:node>
<graf:edge to="chunk2" id="e1000" from="dep1">
  <graf:as type="dep_head"/>
</graf:edge>
<graf:edge to="chunk1" id="edge1001" from="dep1">
  <graf:as type="dep_dependent"/>
</graf:edge>
```

Figure 9: LAF/GrAF compliant XML document for CaboCha (partial).

ated by the wrapper. Note here that the XML schema is an extension of the GrAF XML schema: any XML documents validated with the schemata are also GrAF-compliant. The benefit of such a wrapper in the Web service context is obvious, improving the re-usability and interoperability of the results of existing dependency parsers.

5. Discussions

There surely exist alternatives for representing dependency structures; these include the simple data format empolyed by CoNLL-2008 shared task for ”Joint Learning of Syntactic and Semantic Dependencies”[8] and the TEI P5 schema presented in (Przepiórkowski, 2008). The latter allows

[8]http://www.cnts.ua.ac.be/conll2008/

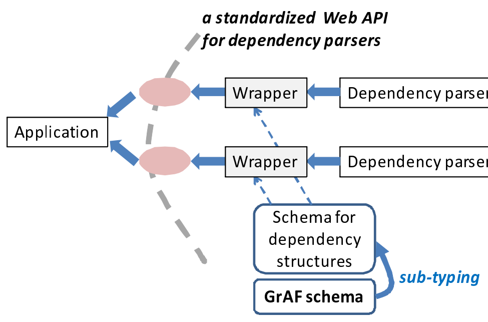


Figure 10: Configuration of the dependency parser wrapper.

chunk-based dependency structure as proposed in this paper. These schemata are especially dedicated to economical representation of dependency structures. The proposed sub-typing of LAF/GrAF for representing dependency structures, on the other hand, might be a kind of naïve adoption of the ISO proposal that is still under the discussion. However, as stated in the paper, the presented proposal demonstrates a possible way to bridge between SynAF and LAF/GrAF. Kountz et al. (2008), like our proposal, presented a schema for representing syntactic annotations based on LAF/GrAF. Their proposal includes an extension to LAF that allows flexible encoding of underspecified representation that is required for annotating syntactically ambiguous sentences.

As discussed in (Marneffe and Manning, 2008), fidelity is a key issue in the representation of linguistic structures. In the context of global interoperability of language resources in multiple languages, this might be reconsidered. We may need a rather coarse level representation, where language dependent idiosyncrasies are abstracted away. The Japanese B-chunk-based dependency structure could be a candidate representation level of the kind, and we think we should explore comparable chunking in other languages, say in English. If this direction is correct and feasible, proper chunking should be a part of the wrapper process. This direction also may facilitate the development of standardized inventory of dependency relation labels that could be more of a semantic type rather than purely grammatical.

6. Concluding Remarks

This paper proposed a sub-typing of LAF/GrAF framework for representing two types of dependency structures that should be essential in handling multiple languages. It also partially proved the value of an international standard like LAF/GrAF in the Web service context: an existing dependency parser can be, in a sense, standardized, once wrapped by a data format conversion process.

For future work, along with the issue of more semantic-oriented representation, a mechanism to link data syntax (XML schemata) with data semantics (ontological knowledge) (Hayashi et al., 2008) should be explored. This line of work would allow us to develop a method to (semi-)automatically generate a wrapper program given the ontological specification of a parser at hand.

7. Acknowledgments

The authors would like to thank both Ulrich Heid and Kerstin Eckart (IMS, Stuttgart) for the helpful discussion; Nancy Ide, and Keith Suderman for their useful suggestions on utilization of GrAF XML schema. The presented work has been supported by Strategic Information and Communications R&D Promotion Programme (SCOPE) of the Ministry of Internal Affairs and Communications of Japan, and by the project D-SPIN funded by the German Federal Ministry of Education and Research (BMBF).

8. References

Brants, S. Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER Treebank. *Proc. TLT'02*.

Declerck, T. (2008). A Framework for Standardized Syntactic Annotation. *Proc. LREC2008*.

Hayashi, Y., Declerck, T., Buitelaar, P., and Monachini, M. (2008). Ontologies for a Global Language Infrastructure. *Proc. ICGL2008*, pp. 105–112.

Ide, N., and Romary, L. (2006). Representing Linguistic Corpora and Their Annotations. *Proc. LREC2006*.

Ide, N., and Suderman, K. (2007). GrAF: A Graph-based Format for Linguistic Annotations. *Proc. Linguistic Annotation Workshop*, pp. 1–8.

Kountz, M., Heid, U., and Eckart, K. (2008). A LAF/GrAF based Encoding Scheme for Underspecified Representations of Syntactic Annotations. *Proc. LREC2008*.

Kübler, S., McDonald, R., Nivre, J. (2009). *Dependency Parsing*. Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers.

Kudo, T., and Matsumoto, Y. (2002). Japanese Dependency Analysis using Cascaded Chunking. *Proc. CONLL2002*, pp. 63–69.

Kurohashi, S., and Nagao, M. (2003). Building a Japanese Parsed Corpus while Improving the Parsing System. In: Anne Abeille (Ed). *Treebanks*, pp. 249–260, Kluwer Academic Publishers.

de Marneffe, M., and Manning, C.D. (2008). The Stanford Typed Dependencies Representation. *Proc. COLING2008 Workshop on Cross-framework and Cross-domain Parser Evaluation*.

Przepiórkowski, A. (2008). TEI P5 as an XML Standard for Treebank Encoding. *Proc. TLT8*