[Tiger2/] Documentation

Laurent Romary, Amir Zeldes, Florian Zipser

Documentation

- DRAFT 25 May 2011 -

title:	<tiger2/> Documentation
doc version:	Draft version 25 May 2011
format version:	2.0.3
authors:	Laurent Romary, Amir Zeldes and Florian Zipser
e-mail:	tiger2@lists.hu-berlin.de
homepage:	http://korpling.german.hu-berlin.de/tiger2/

Preamble

<tiger2/> is an XML format serializing the ISO meta-model for syntactic annotations defined in the SynAF[1] framework. The format is based on the TigerXML format developed as part of the German TiGer Treebank annotation project[2]. The focus in <tiger2/> is on the ability to flexibly express all manner of syntactic annotation structures, including multiple, possibly conflicting, typed annotation nodes and edges with arbitrary annotation names expressing both constituent and dependency trees with modern, standard conformant mechanisms.

Graph structure

In <tiger2/> annotations are represented in a graph structure. The graph structure can be described as G=(V, E) with:

a set of nodes V and

a set of edges E with e=(v ∈ V, v ∈ V) ∈ E

A graph represents a bundle of interrelated nodes and edges. It is not specified which parts of a primary text are covered by a single graph: a graph can cover e.g. a sentence, a sub-sentence, a chapter or a whole text.

Nodes and Edges are more or less placeholders for linguistic annotations. An annotation is in general an attribute-value-pair. In <tiger2/> we used two kinds of nodes 1) terminal-nodes and 2) nonterminal-nodes.
A terminal-node represents the reference to the primary data. As we will see later on, this can be a direct reference to a text-span or an indirect one to other xml-elements in e.g. a MAF-file (Morphosyntactic Annotation Framework, see http://atoll.inria.fr/~clerger/MAF/html/index.html). Terminal nodes are represented by the <t>-element and are grouped together in a <terminals>-element.
A Nonterminal node is an inner node, referring directly or indirectly to a terminal node. It is represented by the <nt>-element and belongs to the surrounding element <nonterminals>.
An Edge always has to have a source and a target. Both of them can be either a terminal or a nonterminal-node. In <tiger2/> an edge is represented by the <edge>-element and its source is given by the surrounding element. This means that an <edge>-element always has to be a child-element of a <t> or an <nt> element. The target is given by the @target attribute and must refer to a node in the same XML-document.
Example 1 shows the <tiger2/> representation of the graph structure:

1 http://www.tc37sc4.org/document.php?p=tc37sc4_list_total.txt&search_text=SynAF&project_category=on
2 http://www.ims.uni-stuttgart.de/projekte/TIGER/

```
<graph xml:id=”s1_g1”>
      <terminals>
            <t xml:id="s1_t1"/>
            <t xml:id="s1_t2"/>
      </terminals>
      <nonterminals>
            <nt xml:id="s1_nt1">
                  <edge xml:id=”s1_e1” target="#s1_t1" />
                  <edge xml:id=”s1_e2”target="#s1_t2" />
            </nt>
      </nonterminals>
</graph>
```

Example 1: graph structure

Primary data and token representation

In <tiger2/> there are different possibilities to represent primary data and tokens. Tokens and primary data can either be included in the <tiger2/>-document (inline representation), or they can be referred to, being contained in another file (standoff representation). The inline representation makes it possible to have all data in just one <tiger2/>-document, whereas the standoff representation enables one to refer to other formats, for instance a MAF file containing tokens and wordForms.
Both options are possible and the decision is largely made based on the needs of the corpus.

Inline representation:

The benefit of the inline representation is that you can find all elements of a corpus in just one <tiger2/>-document, making it easier for humans to read it and for machines to process it. To go this way you can encode the primary data as part of the terminal-nodes. This can be done by using the @word attribute of the <t>-element. An example is given by example 2.

```
<t xml:id="s1_t1" word=”two”/>
<t xml:id="s1_t2" word=”words”/>
```

Example 2: graph structure

Example 2 shows two terminal-nodes, representing two tokens. The first token contains the word “two”, the second token contains the word “words”. The order of the tokens is given by the order of the xml-elements. In this case it means that the word “two” is followed by the word “words” in the primary data.

However it should be pointed out that this representation is not primary-data-preserving. This means, that we loose some information about the primary data when representing a primary text in this representation. On the one hand the inline representation does not care about whitespaces between tokens, and on the other hand untokenized parts of the primary data will be lost. Take a look at examples 3 and 4.

```
This is a sample.
```

Example 3: primary data

```
<t xml:id="s1_t1" word=”This”/>
<t xml:id="s1_t2" word=”is”/>
<t xml:id="s1_t4" word=”sample”/>
<t xml:id="s1_t5" word=”.”/>
```

Example 4: inline representation

In example 3 and 4 one can see that it is not possible to resolve whether there was a whitespace between the tokens or not. In the case of token “s1_t1” and “s1_t2” there was one, whereas in the case of punctuations like in tokens “s1_t3?” and “s1_t4” there was none. An unrepresented text span like the word “a” will be lost.

Because of the restrictions of XML-attributes, another problem occurs: reserved signs like an angle bracket (“<”) can not be represented inside an xml-attribute-value. Therefore one has to escape the angle bracket with “<”, see example 5.

```
<t xml:id="s1_t4" word=”&lt;”/>
```

Example 5: inline representation with escaping

When interpreting the primary-data in the case of example 5, it is not possible to resolve whether the origin text contains an angle bracket (“<”) or the literal escaping (“<”).

Standoff representation

The aforementioned problems can be avoided by using a standoff representation. In <tiger2/> you can refer to other files containing the tokens or even higher structures e.g. wordForms. You can do this using the @corresp attribute of the terminal-nodes. The @corresp attribute replaces the @word attribute and contains a reference in XPointer syntax to another resource, for example a MAF-file. You either can use the @word or the @corresp attribute. Example 6, 7 and 8 show the use of the standoff representation via the @corresp attribute.

```
<text>We can ...</text>
```

Example 6: primaryData.xml

```
<token      xml:id="t1"
            xlink:href=”primaryData.xml#xpointer(string-range(//text,'',0,2))”/>
<token      xml:id="t2"
            xlink:href=”primaryData.xml#xpointer(string-range(//text,'',3,3))”/>
...
<wordForm xml:id="wf_1" lemma="we" tokens="t1"/>
<wordForm xml:id="wf_2" lemma="can" tokens="t2"/>
```

Example 7: example.maf

```
<terminals>
      <t xml:id="s9_1" corresp="example.maf#wf_1" />
      <t xml:id="s9_2" corresp="example.maf#wf_2">
</terminals>
```

Example 8: example.tiger2

Example 6, 7 and 8 contains excerpts of three different files, first an excerpt of a file containing the primary text, second a MAF file containing the tokenisation and an aggregation of tokens to wordForms and third an excerpt of the <tiger2/> document refering the MAF-document.

Typing of nodes and edges

Nodes and edges can also be given types in <tiger2/>. A type is a special kind of annotation and is represented by the @type attribute contained in a <t>, <nt> or <edge> element. In example 9 we use the @type attribute to give a more special semantic meaning to the terminal nodes “s1_t1”, “s1_t2”, the nonterminal-node “s1_nt1” and the edges ”s1_e1”, ”s1_e2” and ”s1_e3”. The example shows the wordForm “house-builder”, in which the node house is seen as a direct object dependent of builder, a deverbal noun heading a synthetic compound. The non-terminal node for “house-builder” may be given a special type to analyze it as a morphological compound, as opposed to a syntactic phrase. Similarly, the dependency edge is given the type “dep” to distinguish it from the constituency edges in the graph.

```
...
<t xml:id="s1_t1" word="house-" type="morpheme" />
<t xml:id="s1_t2" word="builder" type="morpheme">
      <edge xml:id =”s1_e1” type= “dep” target=”#s1_t1” label=”obj”/>
</t>
...
<nt xml:id="s1_nt1" type="wordform">
      <edge xml:id =”s1_e2” type="comp" target="#s1_t1" />
      <edge xml:id =”s1_e3” type="comp" target="#s1_t2" />
</nt>
```

Example 9: using type for <t>, <nt> and <edge>-element

The domain of possible @type values is not defined by <tiger2/> and is therefore the responsibility of an interpreting tool.
For a more specific definition of a @type value you can define the domain of the values via an annotation with a <feature> element as we will show in the section Using the @type attribute. If you do not use the @type attribute inside a <t>, <nt> or <edge>-element, they will be implicitly set to a default value. In the case of an edge the type will be set to “prim” (a primary edge), in the case of a terminal-node it will be set to “t” and in the case of a nonterminal-node it will be set to “nt”. If you want to specify the domain for @type attributes, it is not possible to override the semantics of the three default values.
The @type attribute is also used for the declaration of the domain for annotation-names and the domain for annotation-values as we show in section Annotations.

Annotations

Annotations are realized as attribute-value-pairs and can be appended to nodes and edges. The name of an annotation is given by the name of an XML attribute and the annotation value is given by its XML attribute value. In general in <tiger2/> there are no restrictions for setting names or values for annotations depending on specific linguistic schools. Therefore you are free to append any attribute-value-pair to a <t>, <nt> or <edge> element. The only restriction is given by reserved attributes of <tiger2/> for example @type, @word, @corresp and @domain. These attributes have fixed semantics and cannot be overridden.

Example 10 shows the annotation mechanism for nodes and edges.

```
<terminals>
      <t id="s1_t1" word="I" lemma="I" pos="PP"/>
      ...
</terminals>
<nonterminals>
      <nt id="s1_nt1" cat="NP">
            <edge xml:id=”s1_e1” label="HD" target="s1_t1"/>
      </nt>
      ...
</nonterminals>
```

Example 10: annotation of a <t>, <nt> and <edge>-element

In example 10 we show a word “I”, annotated with the part-of-speech annotation “PP” and the lemma-annotation “I”. Neither “pos” nor “lemma” is reserved - they also can be named “part-of-speech” or “lem” or any other non-reserved name. The terminal “s1_t1” is headed by a nonterminal “s1_nt1” being annotated with the categorical annotation nominal phrase. The edge “s1_e1” corresponding “s1_nt1” and “s1_t1” also gets an annotation (“label='HD'”), which means the nonterminal acts as a head for the terminal.

Domain declaration

In <tiger2/> it is necessary to define a domain for annotation names, this must be done in the header section of <tiger2/>, namely in the <annotations> element inside the element <head>. This domain is referred to by the annotation name inside the node or edge. For example let us take a look again at example 10. The declaration of the domain for annotation-names can appear as shown in example 11.

```
<head>
      <annotations>
      <feature xml:id=”f1” name="pos" domain="t" />
      <feature xml:id=”f2” name="lemma" domain="t" />
      <feature xml:id=”f3” name="cat" domain="nt"/>
      <feature xml:id=”f4” name="label" domain="edge"/>
      </annotations>
</head>
```

Example 11: domain declaration for annotation-names and annotation-values

The mapping between the annotation name and the domain declaration is done by resolving the

name of the XML attribute of the annotation and the @name attribute of the element <feature>. The attribute @domain specifies that the declared feature is applicable for just the given kind of <tiger2/> elements. In the given example, this means, that the feature declarations for “pos” and “lemma” can only be used by <t> elements, “cat” only can be used by <nt> elements and “label” can only be used by <edge> elements. The attribute @domain is optional and need not be set. If it is not set it means that the feature declaration is available for all three kinds of elements.

Feature values

Not only domains for annotation names can be declared in <tiger2/>, but also domains for annotation values. Declaring a domain for annotation values can be done by using the <value> element inside the <feature> element. The <value> element has a @name attribute, identifying the value name and also working as an anchor for annotation values. The element can contain a text, which acts as a description for the annotation-value. Example 12 shows the use of an annotation value domain declaration.

```
<head>
      <annotations>
      <feature xml:id=”f2” name="pos" domain="t">
            <value xml:id=”f2_1” name="PP">Personal pronoun</value>
      </feature>
      </annotations>
</head>
...
<terminals>
      <t id="s1_t1" word="I" pos="PP"/>
</terminals>
```

Example 12: referencing domains for annotation-name and annotation-value

In the case of example 12 the set of possible values for an annotation value with the annotation-name “pos” only contains a “PP” element, standing for a personal pronoun. In this case no other values are possible for this annotation for terminal nodes. To increase the set of possible annotation values just add further <value> elements to the <feature> element. The number of possible annotation values is not limited. If there are no <value> elements given inside a <feature> element, the set of annotation values is not defined and can be any string. In this case, a validation of annotation values will not take place. This is very useful in case of lemma annotations, for which it is not possible to define the domain of possible annotation values

DCR references

Often domain declarations are not only interesting for one corpus or just one project, they are interesting for a whole community. Therefore several data category registries have evolved. The most popular one is the ISOCat (see: http://www.isocat.org/). When using <tiger2/> you can benefit from the power and the community creating the registries, by making a reference from your domain declaration to the registries. To support this we provide the attribute @datcat, defined via the namespace “http://www.iso.org/ns/DCR”. This attribute can be used in <feature> and <value> elements. This can improve the usefulness of your corpus, thanks to a much better understanding of the annotations used. Example 13 shows the use of the @datcat attribute.

```
<corpus xml:id=”c1” tiger_version=”2.0.3” xmlns:dcr=”http://www.iso.org/ns/DCR”>
<head>
      <annotations>
      <feature    xml:id=”f2” name="pos" domain="t" dcrReference=”
                        http://www.isocat.org/datcat/DC-396 ”>
            <value      xml:id=”f2_1” name="PP"
                        dcr:datcat=”http://www.isocat.org/datcat/DC-1463 ”/>
      </feature>
      </annotations>
</head>
...
<terminals>
      <t id="s1_t1" word="I" pos="PP"/>
</terminals>
```

Example 13: using data category registry in annotation-declaration

In example 13 we use a terminal covering the word “I”, which is annotated by part-of-speech annotation having the value personal pronoun. The domain for the annotation name and annotation value is declared in the <annotations> element, as already seen in the examples above. But in this example we use the @datcat attribute to refer to the ISOCat system. The annotation name is not only declared by mentioning it in the <tiger2/> document, but also in the common registry. Therefore you do not necessarily need a description inside the <value> element, because it is given by the ISOCat entry, though <tiger2/> does not prohibit using both.

Using the @type attribute

If you need a finer granularity for the declaration of annotation domains, you can use the @type attribute inside the <t>, <nt> and <edge> element. As already mentioned in the section Typing of nodes and edges, this attribute on the one hand types the given element and on the other hand represents a correspondence to the declaration of the annotation-domain. The element <feature> also contains an optional @type attribute, with which you can constrict the set of nodes and edges using the annotation domain. Example 14 shows this mechanism.

```
<head>
      <annotations>
      <feature xml:id=”f2” type=”wordform” name="lemma" domain="t">
            <value xml:id=”f2_1” name="PP">Personal pronoun</value>
      </feature>
      </annotations>
</head>
...
<terminals>
      <t id="s1_t1" word="I" type=”wordform” lemma="I" pos="PP"/>
</terminals>
```

Example 14: using the @type attribute for annotation domain declaration

In example 14 the domain of the annotation name for terminal “s1_t1” is defined by the feature “f2”. The @type attribute guarantees that the feature only declares an annotation name domain for terminals of the type “wordform”. A terminal without this type cannot be corresponded to in that

domain. If no @type attribute is given in the <feature> element, the domain works for all elements respecting the domain declaration (<t>, <nt> or <edge> elements).

As we already mentioned in the section Typing of nodes and edges, in <tiger2/> it is also possible to declare the domain for types. The mechanism to do so is the same as for declaring annotation values, namely via the <feature> and <value> element. Example 15 shows the declaration of the domain for types.

```
<feature xml:id="f3" name="type" domain="nt">
      <value xml:id="f2_1" name="compound">
            a unit of text, which acts as a kind of word
      </value>
      ...
</feature>
```

Example 15: declaration of domain for the @type attribute

In example 15 we declared the domain for types, by adding an entry for the type "compound". Note that if you declare such a domain, it is not possible to have non-declared values in the @type attribute of the corresponding element (<t>, <nt> or <edge>). This enables the validation of the @type attribute. The default types "t" for domain t, "nt" for domain nt and "prim" for domain edge, are implicitly given and do not have to be declared explicitly.

Grouping

Several graphs can be grouped by a segment, which is a semantic unit. For example, a segment can stand for a sentence, a paragraph and so on. A segment is represented in <tiger2/> via the <s> element. Such a segment must contain at least one graph, the upper limit of graphs is not bound. Example 16 shows the containment:

```
<s xml:id="s1">
      <graph xml:id="s1_g1"/>
      <graph xml:id="s1_g2"/>
      ...
</s>
<s xml:id="s2">
      <graph xml:id="s2_g1"/>
      <graph xml:id="s2_g2"/>
      ...
</s>
```

Example 16: grouping mechanism via <s> element

A corpus or sub-corpus (as described in the section Corpus-structure) can contain an unbounded number of segments.

Corpus-structure

In <tiger2/> you can represent a whole corpus structure containing several sub-corpora in just one <tiger2/> document. But in each <tiger2/> document you can represent just one root corpus represented by the <corpus> element. The corpus element must specify the format version of the data using the @tiger_version attribute. A corpus structure or more precisely a subcorpus can be

created by using the <subcorpus> element inside the <corpus> element. @tiger_version is inherited by subcorpora and should not be specified individually for each subcorpus. The <subcorpus> element is a recursive structure and enables one to create an unbounded depth of subcorpora. Example 17 shows the corpus-structure in <tiger2/>.

```
<corpus xml:id="c1" tiger_version="2.0.3" xmlns:dcr="http://www.iso.org/ns/DCR">
      <head/>
      <body/>
      <subcorpus xml:id="c2">
            <head/>
            <body/>
            <subcorpus xml:id="c3">
                  <head/>
                  <body/>
            </subcorpus>
      </subcorpus>
      <subcorpus xml:id="c4">
            <head/>
            <body/>
      </subcorpus>
</corpus>
```

Example 17: corpus structure with corpus and sub-corpora

Example 17 shows a corpus "c1" containing in total 3 subcorpora. Two of them ("c2" and "c4") are directly contained in "c1", and "c3" is directly contained in "c2" and therefore indirectly contained in "c1".

Meta-annotations

A corpus or subcorpus can be annotated by meta-annotations. Using meta-annotations differs from the way nodes and edges are annotated. In the case of meta-annotations in <tiger2/> we give a fixed set of possible meta-annotations. Meta-annotations are limited to the header of a corpus or subcorpus. They can be identified by the enclosing element <meta>. These meta-annotations comes form the TigerXML-format and are usable for the <corpus> and the <subcorpus> element. The possible meta-annotations are the following ones: name, author, date, description, format and history. A meta-annotation is represented by an XML element with the same name, containing the value as text inside the element. Example 18 shows the use of meta-annotations.

```
 <meta>
      <name>name of the corpus</name>
      <author>author of the corpus</author>
      <date>creation date of the corpus</date>
      <description>description of the corpus</description>
      <format>
            original format, if the corpus was not born as tiger-2-corpus
      </format>
      <history>version of the corpus</history>
</meta>
```

Example 18: metadata representation

All meta-annotation are optional and do not have to be set and can also contain empty texts.