Mapping TCF and LIF

Marc Verhagen

February 12, 2018

1 Introduction

This document contains mappings of data elements of the Text Corpus Format (TCF) used by WebLicht and the LAPPS Interchange Format (LIF) used by the LAPPS Grid. The descriptions here are based on the information in:

- The TCF Format explained.
  http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/The_TCF_Format
- The TCF schema
  https://github.com/weblicht/tcf-spec/blob/master/src/main/rnc-schema/textcorpus_0_4.rnc
- The specifications of the LAPPS Interchange Format: http://wiki.lappsgrid.org/interchange/
- The LAPPS vocabulary website: http://vocab.lappsgrid.org/

The mappings are provided in a series of tables and are structured around several categories (tokens, sentences, parsing, etcetera) and for each category examples of both TCF and LIF are given. But we start with some high-level observations to accompany the tables.

First, assuming that the LAPPS vocabulary contains the categories needed to express all information in a TCF layer, then a single view can be created for each layer. The one exception is the combination of the `tokens`, `POStags` and `lemmas` layers, which most naturally map to a single LIF view with `Token` annotation types. Second, since LIF has no strict rules on what annotation types co-occur in a view, we list the most common combinations. Third, one tricky issue that may not be solved is what to do with LIF representations that contain several layers with the same annotations. For example, it is totally okay if there are two Token views, it seems like TCF has no natural way of dealing with this.

The initial description of what's involved in the mapping is centered around two simple yet not trivial use cases. The first case is of a LAPPS user who has built a pipeline of several LAPPS services followed by one WebLicht service, this is partially depicted in Figure 1 where we se the LIF representation as created by the LAPPS services, how that LIF representation is mapped to TCF, and then how after WebLicht processing the TCF is mapped back to LIF.

For the WebLicht service to run the LIF representation first needs to be mapped to TCF (see the left two boxes of the figure. Here are some remarks on that mapping:

1. The text can relatively simply be mapped from LIF to TCF since the way the text is represented in LIF and TCF is essentially the same.

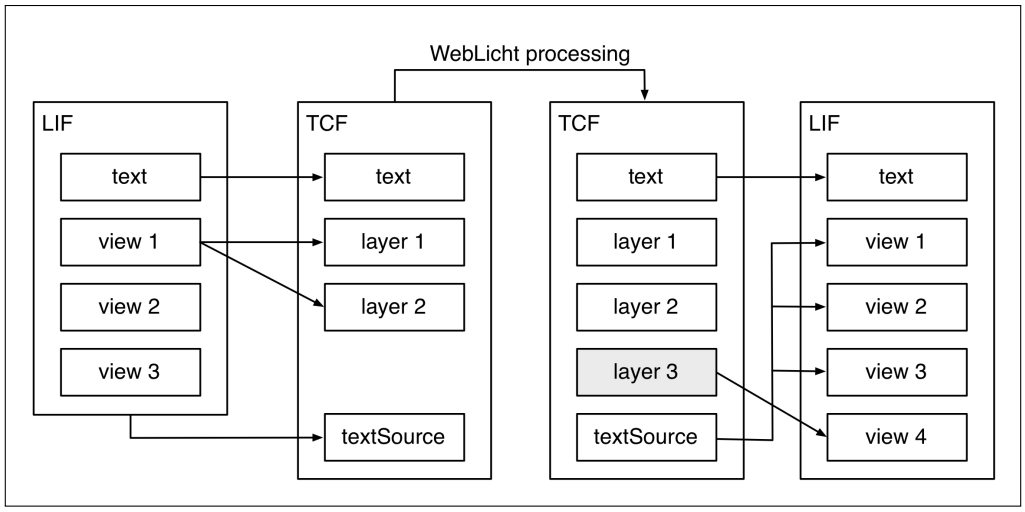

Figure 1: Accessing a WebLicht service from LAPPS

2. The example here has three LIF views where one of them maps to two TCF annotation layers. This could for example be a view with Token annotations where the Tokens each have a `pos` tag. In addition, there are two LIF views that have no clearly defined mapping to TCF layers or that are not needed for the processing performed by the WebLicht service.

3. The entire LIF representation has a natural landing spot in the `textSource` tag in TCF (which is mostly used to store the TEI source of a document). Using this tag has the advantage that the mapping to TCF now becomes lossless. Some information in LIF views cannot naturally be expressed in TCF (character offsets for example) and having the entire LIF stored in a special field makes it possible to just transfer the information needed to use the WebLicht service.

WebLicht processing occurs in the middle of Figure 1 and for the example we simply assume that just one new layer is created. The deal now with mapping back to LIF is that the original LIF views can be restored from the TCF `textSource` layer. It would potentially be possible to restore view 1 from layers 1 and 2, but using `textSource` is more robust. The new layer is mapped to a new view by convention. Note that it is sometimes possible to fold the new TCF layer into an existing LIF view. This if for example the case when the TCF layer is a `lemmas` or `POStags` layer and the LIF representation already has a Tokens view. In that case having a new view may introduce redundancy, which is a common occurrence in LIF.

The second use case is that of a WebLicht user who has built a pipeline of several WebLicht services followed by one LAPPS service. This is depicted partially in Figure 2, in a similar way to how it was done for the reverse direction in Figure 1.

Here we assume that there are two TCF layers and they each happen to map to one view. We also assume that the LAPPS service used generates one new view. Technically, a LAPPS service could add annotations to an existing view, but it is recommended that services create new views and most services do that. So at this point we will not deal with those cases where existing views are updated.

The first thing to note is that LIF does not currently have the equivalent of the TCF `textSource` tag so restoring the TCF layers in the right of the picture require that the mapping from layers to

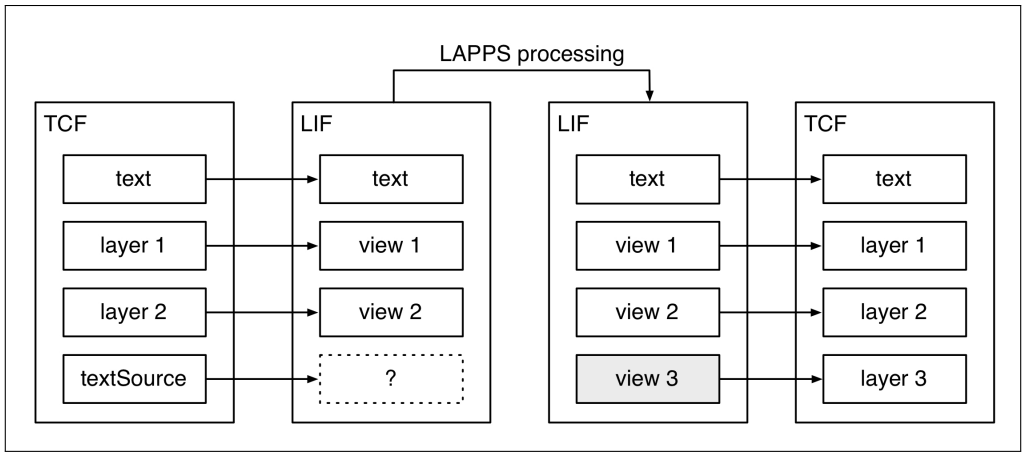

Figure 2: Accessing a LAPPS service from WebLicht

views is lossless. This is in most cases possible with just some changes in the LAPPS Vocabulary side of things. However, we will consider whether it is worthwhile to introduce a new structural element in LIF that can store an alternative representation.

◇◇◇

The next sections have more details on the mappings of individual layers and views. We concentrate on the annotation contents of views and layers and have little to say about meta data. We have structured the discussion into several topics.

For each topic we give the relevant fragments of the TCF schema and the LAPPS vocabulary as well as example TCF and LIF representations. For TCF, the concepts used are expressed in the schema, but not explicitly further semantically defined. There is the CLARIN Concept Registry at https://www.clarin.eu/ccr, but there are no explicit links from TCF to the registry. In LIF the concepts are referred to explicitly by using links to the LAPPS Vocabulary. In our discussion we focus on changes needed to the LAPPS vocabulary and the TCF schema and we again abstract away from the problem of having to generate character offsets when we map from TCF to LIF.

At the end we summarize results for each direction in a couple of tables.

2 Tokens, part-of-speech tags and lemmas

While these are separate annotation layers in TCF, we group these together because on the LIF side these typically end up in the same view. First we give a TCF example for tokens and the associated schema:

```
<tokens>
  <token ID="t_0">Karin</token>
  <token ID="t_1">fliegt</token>
</tokens>
```

```
tokens =
  element tokens {
    attribute charOffsets { "true" }?,
    element token {
      (attribute start { xsd:int },
       attribute end   { xsd:int })?,
      attribute ID    { xsd:ID }?,
      xsd:string
    }*
  }
```

This layer does have an optional `charOffsets` attribute which can be set to true for those cases when character offsets are available. This could be exploited by the TCF2LIF mapper, but in most cases there are no offsets alas. The `ID` attribute is listed as optional, but in fact it seems to be required.

For lemmas in TCF we have the following example and schema:

```
<lemmas>
  <lemma ID="le_0" tokenIDs="t_0">Karin</lemma>
  <lemma ID="le_1" tokenIDs="t_1">fliegen</lemma>
</lemmas>
```

```
lemmas =
  element lemmas {
    element lemma {
      attribute ID    { xsd:ID }?,
      attribute tokenIDs { xsd:IDREFS },
      xsd:string
    }*
  }
```

And finally, for part-of-speech tags we have the following.

```
<POStags tagset="stts">
  <tag ID="pt_0" tokenIDs="t_0">NE</tag>
  <tag ID="pt_1" tokenIDs="t_1">VVFIN</tag>
</POStags>
```

```
POStags =
  element POStags {
    attribute tagset { xsd:string },
    element tag {
      attribute ID    { xsd:ID }?,
      attribute tokenIDs { xsd:IDREFS },
      xsd:string
    }*
  }
```

◇ ◇ ◇

Mapping to LIF is simple, all that is needed is to create `Token` annotations in a view.

```
{ "id": "v1",
  "annotations": [
    { "@type": "http://vocab.lappsgrid.org/Token",
      "id":" t_0",
      "start": 0,
      "end": 5,
      "features": {
        "word": "Karin",
        "lemma": "Karin", "tcf_lemma_ID": "le_0",
        "pos": "NE", "tcf_POStag_ID": "pt_0" }
    },
    { "@type": "http://vocab.lappsgrid.org/Token",
      "id":" t_1",
      "start": 6,
      "end": 11,
      "features": {
        "word": "fliegt",
        "lemma": "fliegen", "tcf_lemma_ID": "le_1",
        "pos": "VVFIN", "tcf_POStag_ID": "pt_1" }
    }
  ]
}
```

In both TCF and LIF the tokens are the primary thing in that lemmas and pos tags both depend on the tokens to be present. In TCF this is expressed by the lemmas pos tags referring to a token object, in LIF this is done by embedding lemmas and pos tags as attributes on a features dictionary.

The main difference is that in TCF lemmas and pos tags are first-class citizens with their own object. In LIF, all information on the lemma or pos tag needs to be copied to the Token's feature dictionary, in the example above we use `tcf_lemma_ID` and `tcf_POStag_ID`. Note that this does not require any changes to the LAPPS Vocabulary since it is allowed to add arbitrary features. Another difference is that in TCF a lemma or pos tag can refer to more than one token. The only way to do that in LIF is to store the attributes involved on more than one Token.

We now try to abstract away from the syntax of TCF and LIF and express the difference between the two notations as concept mappings. A concept mapping can be seen as linking a path in the WebLicht TCF schema to an element of the LAPPS vocabulary. Note that elements in the LAPPS vocabulary are expressed at fixed positions in a LIF representation so on the LAPPS end we can use the intended semantic for the mapping. In TCF the semantics is expressed directly and implicitly in the schema syntax, there is no explicit semantics.

In the concept mappings we use the hash symbol between concepts and their attributes, where on the TCF end these refer to elements of the schema and on the LAPPS end these refer to elements in the vocabulary. For TCF, we use `cdata` as an attribute that stands for the CDATA in the XML tag. In all following tables, the first line is always the name of the TCF layer. There is no specific concept that this relates to on the LIF side, but in all cases the content of the layer is mapped to one view.

Here are the mappings for tokens and lemmas.

tokens	
tokens#charOffsets	No mapping, tokens always have character offsets
token	Token
token#cdata	Token#word
token#ID	Token#id
token#start	Token#start
token#end	Token#end

lemmas	
lemma	No direct mapping, attributes are embedded in Token
lemma#cdata	Token#lemma
lemma#ID	Token#tcf_lemma_ID In LIF lemma is an attribute that does not have its own identifier so we park it on the Token. Syntactically this attribute is expressed in the features dictionary.
lemma#tokenIDs	Token#id, but note this allows only one identifier

There is one things to note on the lemmas. Consider the use of `tcf_lemma_ID`. This is an attribute that has no natural place in LIF since lemmas have no identifiers in LIF. But we do not loose it so we park it in the features dictionary of the Token and give it a special name with the `tcf` prefix. We can use this for any possible feature on lemmas that do not have a LIF counter part on the Token. Note that if we have a special place in LIF to park the entire TCF representation then we would not need to do this.

Here are the concepts involved for pos tags:

POStags	
tag	No direct mapping, attributes are embedded in Token
tag#cdata	Token#pos
tag#ID	Token#tcf_POStag_ID In LIF pos is an attribute that does not have its own identifier so we park it on the Token. Syntactically this attribute is expressed in the features dictionary.
tag#tokenIDs	Token#id, but note this allows only one identifier

The same remarks on identifiers that we had for the lemmas layer hold here as well. One difference though is that in TCF lemmas can be referred to but POStags not, so we could get away with not keeping the identifier at all.

The mappings above include all tags and attributes defined in TCF for the three layers. But LIF and the LAPPS Vocabulary allow arbitrary extra attributes. Changing TCF to allow for all these is not trivial and instead we will rely on the `textSource` attribute (see Figure 1).

3 Sentences

TCF example and schema:

```
<sentences>
  <sentence ID="s_0" tokenIDs="t_0 t_1 t_2 t_3 t_4 t_5"/>
</sentences>
```

```
sentences =
  element sentences {
    attribute charOffsets { "true" }?,
    element sentence {
      attribute ID    { xsd:ID }?,
      (attribute start { xsd:int },
       attribute end   { xsd:int })?,
      attribute tokenIDs   { xsd:IDREFS }
    }*
  }
```

This is again a fairly simple mapping, but there are two ways to represent this in LIF. The first is what is used mostly by LIF services which is to give the start and end character offsets of the sentence:

```
{ "id": "v4",
  "annotations": [
    { "@type": "http://vocab.lappsgrid.org/Sentence",
      "id":" s_0",
      "start": 0,
      "end": 35,
      "features": { }
    }
  ]
}
```

The other way is to use the `targets` attribute:

```
{ "id": "v4",
  "annotations": [
    { "@type": "http://vocab.lappsgrid.org/Sentence",
      "id":" s_0",
      "targets": "t_0 t_1 t_2 t_3 t_4 t_5 ",
      "features": { }
    }
  ]
}
```

The Token identifiers used would then either need to be expressed in the same layer or they need to refer to another layer, so instead of `t_0` we would have `v1:t_0`.

We have the following mappings:

sentences	
sentences#charOffsets	No mapping, sentences can always be related to offsets
sentence	Sentence
sentence#ID	Sentence#id
sentence#tokenIDs	Sentence#targets
sentence#start	Sentence#start
sentence#end	Sentence#end

4 Phrase structure parsing

TCF example (rather simplified) and schema:

```
<parsing>
  <constituent cat="NX" ID="c_9">
    <constituent cat="NE" ID="c_7" tokenIDs="t_3"/>
    <constituent cat="NE" ID="c_8" tokenIDs="t_4"/>
  </constituent>
</parsing>
```

```
parsing =
  element parsing {
    attribute tagset { xsd:string },
    element parse {
      attribute ID    { xsd:ID }?,
      constituent
    }*
  }

constituent =
  element constituent {
    attribute cat { xsd:string },
    attribute edge { xsd:string }?,
    attribute ID { xsd:ID }?,
    element cref {
        attribute constID {xsd:ID}, # target constituent ID
        attribute edge { xsd:string } # secondary edge label
    }*,
    (attribute tokenIDs { xsd:IDREFS } | constituent*)
  }
```

The most notable characteristic of the TCF parsing layer is that dominance relations are expressed by embedding of XML tags and not by features (with the exception of constituents that immediately dominate tokens). But in a LIF representation there is no embedding of annotation objects and dominance relations have to be expressed by attributes, as shown in the following example (which is also an example where one view refers to tokens in another view).

```
{
  "text": { "@value": "Sue sees herself" },
  "views": [
    { "id": "v1",
      "metadata": {
        "contains": {
          "Token": {
            "producer": "edu.brandeis.cs.lappsgrid.opennlp.Tokenizer:2.0.3",
            "type": "tokenizer:opennlp" }}},
      "annotations": [
         { "@type": "Token", "id": "tok0", "start": 0, "end": 3 },
         { "@type": "Token", "id": "tok1", "start": 4, "end": 8 },
         { "@type": "Token", "id": "tok2", "start": 9, "end": 16 }
      ]
    },
    { "id": "v2",
      "metadata": {
        "contains": {
          "PhraseStructure": {
            "producer": "edu.brandeis.cs.lappsgrid.SimpleParser:1.0.0",
            "categorySet": "ns/types/PTBcategories",
            "type": "PhraseStructure:SimpleParser" },
          "Constituent": {
            "producer": "edu.brandeis.cs.lappsgrid.SimpleParser:1.0.0",
            "categorySet": "ns/types/PTBcategories",
            "type": "PhraseStructure:SimpleParser" }}},
      "annotations": [
         { "@type": "PhraseStructure",
           "id": "phrase0",
           "start": 0,
           "end": 16,
           "features": {
              "constituents": [ "c0", "c1", "c2", "v1:tok0", "v1:tok1", "v1:tok2" ] }},
         { "@type": "Constituent", "label": "S", "id": "c0",
           "features": {
             "parent": null,
             "children": [ "c1", "c2"] } },
         { "@type": "Constituent", "label": "NP", "id": "c1",
           "features": {
             "parent": "c0",
             "children": [ "v1:tok0" ] }},
         { "@type": "Constituent", "label": "VP", "id": "c2",
           "features": {
             "parent": "c0",
             "children": [ "v1:tok1", "v1:tok2" ] }}]}]
}
```


This difference implies that not all information relevant for mapping from TCF to LIF can be expressed by the concept mappings.

Here are the mappings from TCF concepts to the LAPPS Vocabulary:

parsing	
parsing#tagset	PhraseStructure@categorySet
parse	PhraseStructure
parse#ID	PhraseStructure#id
constituent	Constituent
constituent#ID	Constituent#id
constituent#cat	Constituent#label
constituent#edge	No mapping
constituent#tokenIDs	Constituent#children
cref	No mapping
cref#constID	No mapping
cref#edge	No mapping

Note the use of the @ symbol in `PhraseStructure@categorySet`, this indicates that the property is a metadata property (these too are defined in the vocabulary). The LAPPS Vocabulary has `Constituent#parent` and `Constituent#children`. These have no counter part in TCF since there are expressed by embedding (except for the constituents that directly dominate tokens). The `cref` object allows a secondary edge to another constituent, which in effect turns the tree into a directed graph. These are not widely used in WebLicht services and we ignore them for now, but when needed we can most likely use one of the relation annotation types in the vocabulary.

5 Dependency parsing

TCF example and schema

```
 <depparsing tagset="tiger" emptytoks="false" multigovs="false">
  <parse ID="d_0">
    <dependency govIDs="t_1" depIDs="t_0" func="SB"/>
    <dependency depIDs="t_1" func="ROOT"/>
    <dependency govIDs="t_1" depIDs="t_2" func="MO"/>
    <dependency govIDs="t_4" depIDs="t_3" func="PNC"/>
    <dependency govIDs="t_2" depIDs="t_4" func="NK"/>
    <dependency govIDs="t_4" depIDs="t_5" func="--"/>
  </parse>
  <parse ID="d_1">
    <dependency govIDs="t_7" depIDs="t_6" func="SB"/>
    <dependency depIDs="t_7" func="ROOT"/>
    <dependency govIDs="t_10" depIDs="t_8" func="MO"/>
    <dependency govIDs="t_10" depIDs="t_9" func="OA"/>
    <dependency govIDs="t_7" depIDs="t_10" func="OC"/>
    <dependency govIDs="t_10" depIDs="t_11" func="--"/>
  </parse>
</depparsing>
```

```
depparsing =
  element depparsing {
    attribute tagset { xsd:string }?,
    attribute multigovs { xsd:boolean },
    attribute emptytoks { xsd:boolean },
    element parse {
      attribute ID { xsd:ID }?,
      element dependency {
        attribute func { xsd:string }?,
        attribute depIDs { xsd:IDREFS },
        attribute govIDs { xsd:IDREFS }?
       }*,
      element emptytoks {
        element emptytok {
            attribute ID { xsd:string },
             xsd:string?
          }+
      }?
    }*
  }
```

LIF example


```
{
  "text": { "@value": "Sue sees herself" },
  "views": [
    { "id": "v1",
      "metadata": {
        "contains": {
          "Token": {
            "producer": "edu.brandeis.cs.lappsgrid.opennlp.Tokenizer:n.n.n",
            "type": "tokenizer:opennlp" }}},
      "annotations": [
         { "@type": "Token", "id": "tok0", "start": 0, "end": 3 },
         { "@type": "Token", "id": "tok1", "start": 4, "end": 8 },
         { "@type": "Token", "id": "tok2", "start": 9, "end": 16 }
      ]
    },
    { "id": "v2",
      "metadata": {
        "contains": {
          "DependencyStructure": {
            "producer": "edu.brandeis.cs.lappsgrid.SimpleDependencyParser:1.0.0",
            "dependencySet": "ns/types/StanfordDependencies",
            "type": "DependencyStructure:SimpleDependencyParser" },
          "Dependency": {
            "producer": "edu.brandeis.cs.lappsgrid.SimpleDependencyParser:1.0.0",
            "type": "DependencyStructure:SimpleDependencyParser" }}},
      "annotations": [
         { "@type": "DependencyStructure", "id": "depstructure0", "start": 0, "end": 16,
           "features": {
              "type": "basic-dependencies",
              "dependencies": [ "dep0", "dep1", "dep2" ],
        { "@type": "Dependency", "label": "ROOT", "id": "dep0",
          "features":    {
            "governor": null,
            "dependent": "v1:tok1" }},
        { "@type": "Dependency", "label": "nsubj", "id": "dep1",
          "features":    {
            "governor": "v1:tok1",
            "dependent": "v1:tok0" }},
        { "@type": "Dependency", "label": "nobj", "id": "dep2",
          "features":    {
            "governor": "v1:tok1",
            "dependent": "v1:tok2" }}]}]}]
}
```

Here are the mappings:

depparsing	
depparsing#tagset	DependencyStructure@dependencySet
depparsing#multigovs	No mapping
depparsing#emptytoks	No mapping
parse	DependencyStructure
parse#ID	DependencyStructure#id
dependency	Dependency
dependency#ID	Dependency#id
dependency#func	Dependency#label
dependency#depIDs	Dependency#dependent
dependency#govIDs	Dependency#governor
emptytoks	No mapping
emptytok	No mapping
emptytok#ID	No mapping

Note that `depIDs` and `govIDs` are lists. These typically have one element, but if there is a second element than this would be expressed in LIF by using two `Dependency` elements. The LAPPS vocabulary defines two additional properties for `DependencyStructure`: `dependencyType` and `dependencies`. The first is not expressed in TCF and the second is expressed structurally by the list of `dependency` tags inside of the `depparsing` tag.

6 Named entities

TCF example and schema:

```
<namedEntities type="CoNLL2002">
  <entity ID="ne_0" class="PER" tokenIDs="t_0"/>
  <entity ID="ne_1" class="LOC" tokenIDs="t_3 t_4"/>
</namedEntities>
```

```
namedEntities =
  element namedEntities {
    attribute type { xsd:string },
    attribute charOffsets { "true" }?,
    element entity {
      attribute class { xsd:string },
      attribute ID    { xsd:ID }?,
      (attribute start { xsd:int },
       attribute end   { xsd:int })?,
      attribute tokenIDs { xsd:IDREFS }
    }*
  }
```

LIF example:

```
{
  "@context": "http://vocab.lappsgrid.org/context-1.0.0.jsonld",
  "text": { "@value": "Jill sleeps." },
  "views": [
    {
      "id": "v1",
      "metadata": {

        "contains": {
          "http://vocab.lappsgrid.org/NamedEntity": {
            "producer":"edu.brandeis.cs.lappsgrid.stanford.corenlp.NER:2.0.3",
            "namedEntityCategorySet": "ner:stanford" }}}
      "annotations": [
         { "@type": "NamedEntity", "id": "c0", "start": 0, "end": 4,
          "features": { "category": "person", "gender": "female" } } ]
    }
  ]
}
```

Concept mappings:

namedEntities	
namedEntities#type	Not sure, probably NamedEntity@namedEntityCategorySet
namedEntities#charOffsets	No mapping, named entities are always related to offsets
entity	NamedEntity
entity#ID	NamedEntity#id
entity#class	NamedEntity#category
entity#start	NamedEntity#start
entity#end	NamedEntity#end
entity#tokenIDs	NamedEntity#targets

The WSEV `NamedEntity` annotation type also has `type` and`gender` attributes, there are no TCF equivalents for these.

7 Coreference

TCF example and schema:

```
<references typetagset="BART" reltagset="TuebaDZ">
  <entity>
    <reference ID="rc_0" tokenIDs="t_0" mintokIDs="t_0" type="nam"/>
    <reference ID="rc_1" tokenIDs="t_6" mintokIDs="t_6" type="pro.per3"
               rel="anaphoric" target="rc_0"/>
  </entity>
  <entity>
    <reference ID="rc_2" tokenIDs="t_3 t_4" mintokIDs="t_3 t_4" type="nam"/>
    <reference ID="rc_3" tokenIDs="t_8" mintokIDs="t_8" type="adv"
               rel="anaphoric" target="rc_2"/>
  </entity>
</references>
```

```
references =
  element references {
    attribute typetagset { xsd:string }?,
    attribute reltagset { xsd:string }?,
    attribute extrefs { xsd:string }?,
    entity*
  }

entity =
  element entity {
    attribute ID { xsd:ID }?,
    extref?,
    reference+
  }

extref =
  element extref {
   attribute refid { xsd:string }
  }

reference =
  element reference {
    attribute ID { xsd:ID }?,
    attribute rel { xsd:string }?,
    attribute target { xsd:IDREFS }?,
    attribute tokenIDs { xsd:IDREFS },
    attribute mintokIDs { xsd:IDREFS }?,
    attribute type { xsd:string }?
  }
```

As with phrase structure parsers, some of the information is encoded in the XML tree and not on the tags themselves.

LIF has several ways of dealing with this, see http://wiki.lappsgrid.org/interchange/coref.html, the closest to the TCF way is the following.

```
{
  "text": "Sue sees herself",
  "views": [
    { "id": "v1",
       "metadata": {
        "contains": {
          "Token": {
            "producer": "edu.brandeis.cs.lappsgrid.opennlp.Tokenizer:1.1.0",
            "type": "tokenizer:opennlp" }}},
      "annotations": [
         { "@type": "Token", "id": "tok0", "start": 0, "end": 3 },
         { "@type": "Token", "id": "tok1", "start": 4, "end": 8 },
         { "@type": "Token", "id": "tok2", "start": 9, "end": 16 } ]},
    { "id": "v2",
      "metadata": {
        "contains": {
          "Markable": {
            "producer": "edu.brandeis.cs.lappsgrid.opennlp.coref:1.1.0" },
          "Coreference": {
            "producer": "edu.brandeis.cs.lappsgrid.opennlp.coref:1.1.0" }}},
      "annotations": [
         { "@type": "Markable", "id": "m0", "targets": [ "v1:tok0" ] },
         { "@type": "Markable", "id": "m1", "targets": [ "v1:tok2" ] },
         { "@type": "Coreference",
           "id": "coref0",
           "features": {
             "mentions": [ "m0", "m1" ],
             "representative": "m0" }}]}]
}
```

Concept mappings (very incomplete):

references	
references#typetagset	
references#reltagset	
references#extrefs	
entity	Coreference
entity#ID	Coreference#id
extref#class	
extref#refid	
reference	Markable
reference#ID	Markable#id
reference#rel	
reference#target	
reference#tokenIDs	Markable#targets
reference#mintokIDs	
reference#type	

8 Other layers

In TCF there are various other layers that we have left out of the discussion because in the current phase of the project we have no intention to integrate WebLicht tools that produce these layers. In many cases, these layers are not widely used within WebLicht. And, similarly, the LAPPS Vocabulary contains a few annotation types that have no equivalent in TCF. We will not go into detail on these layers and annotation types, but the summary tables in the next section do refer to them with some minor remarks.

9 Summary tables of mappings between TCF and LIF/WSEV

TCF layer	**Mapping to LIF/WSEV**
tokens	Simple mapping to Token.
sentences	Simple mapping to Sentence.
lemmas	With the tokens layer a simple mapping into Token, using the lemma attribute.
POStags	With the tokens layer a simple mapping into Token, using the pos attribute. Tag sets are different though and would need to be properly defined.
parsing	Relatively simple mapping from the `parse` tag to PhraseStructure and from `constituent` to Constituent, but the tree structure needs to be flattened out.
depparsing	DependencyStructure and Dependency. No clear LIF equivalent for `emptytoks` and `multigovs`
namedEntities	Maps to NamedEntity, Date, Location, Organization and Person. Soon, the WSEV will change and keep only the NamedEntity annotation type and use an attribute for the subtypes.
references	Maps to the Coreference type, which may need a few more features.
synonymy	No equivalent in LIF/WSEV.
matches	No equivalent in LIF/WSEV.
WordSplittings	No equivalent in LIF/WSEV.
geo	No real equivalent in LIF/WSEV, but NamedEntity can be used with a few changes.
Phonetics	No equivalent in LIF/WSEV.
textstructure	Maps to Paragraph and Sentence, but the TCF example also has a `page` tag, which has no equivalent in WSEV.
orthography	No equivalent in LIF/WSEV.
wsd	No equivalent in LIF/WSEV.

LIF view	**mapping to TCF**
Token	Token information can be distributed over the tokens, lemmas and POStags layers in a straightforward way.
Sentence	Simple mapping to sentences layer.
Paragraph	Probably maps to the textstructure layer, but note that the latter is more expressive.
NounChunk VerbChunk	No equivalent in TCF, but adding a chunks layer should be quite straightforward.
NamedEntity	Simple mapping to namedEntities layer (since the latter seems a bit more expressive).
Markable	No equivalent in TCF. The Markable annotation type is introduced in LIF/WSEV so we can simply refer to a span of text or a list of identifiers without typing the resulting annotation.
Coreference	Maps to the references layer. But note that a coreference view often includes Markable or Token annotations so the information in the view would also need to be distributed to an other layer and TCF would also need a place to put Markables.
PhraseStructure Constituent	Maps to the parsing layer. Often a view with phrase structure information contains Tokens and in that cases the view's contents would also be added to the tokens layer.
DependencyStructure Dependency	Maps to the depparsing layer. Again, Tokens are often included in the view so the tokens layer would also be involved.
GenericRelation	No equivalent in TCF, but is not widely used in LIF.
SemanticRole	No equivalent in TCF, but is not widely used in LIF.