Integration of Language Resources into Web service infrastructure

2011-03-14 Version: 1.0

Editors: Maciej Ogrodniczuk, Adam Przepiórkowski

The ultimate objective of CLARIN is to create a European federation of existing digital repositories that include language-based data, to provide uniform access to the data, wherever it is, and to provide existing language and speech technology tools as Web services to retrieve, manipulate, enhance, explore and exploit the data. The primary target audience is researchers in the humanities and social sciences and the aim is to cover all languages relevant for the user community. The objective of the current CLARIN Preparatory Phase Project (2008-2010) is to lay the technical, linguistic and organisational foundations, to provide and validate specifications for all aspects of the infrastructure (including standards, usage, IPR) and to secure sustainable support from the funding bodies in the (now 23) participating countries for the subsequent construction and exploitation phases beyond 2010.

Integration of Language Resources into Web service infrastructure

CLARIN-2010-D5R-3b

EC FP7 project no. 212230

Deliverable: D5R-3b – Deadline: 15.03.2011

Responsible: Adam Przepiórkowski

Contributing Partners: CNR-ILC, ICS PAS, ILSP, IMCS, IULA-UPF, RACAI, ULisbon, UTübingen, WROCUT
Contributing Members: BBAW, ULeipzig, UStuttgart

CLARIN references

• Language Resource and Technology Federation	CLARIN-2008-4	February 2009
• Metadata Infrastructure for Language Resource and Technology	CLARIN-2008-5	February 2009
• CLARIN Description of Work (Annex I to the Grant Agreement no 212230), amended version		March 2009
• Report on Web Services	CLARIN-2008-6	March 2009
• Web services and workflow creation	CLARIN-D2R-7a	June 2009
• Requirement Specification Web Services and Workflow systems – v1	CLARIN-D2R-6a	July 2009
• Requirement Specification Web Services and Workflow systems – v2	CLARIN-D2R-6b	January 2010
• Linguistic processing chains as Web Services: Initial linguistic considerations	CLARIN-D5R-3a	January 2010
• A first set of integrated Web services	CLARIN-M2R-7.1	February 2010
• Standards and Interoperability	CLARIN-D5C-3	December 2010
• Validation of technical standards and infrastructure prototype	CLARIN-D5R-4	December 2010

Frequently used acronyms

Abbreviation	Explanation
DCR	Data Category Registry
LRT	Language Resources and Technologies
MAF	Morphosyntactic Annotation Framework
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
TCF	Text Corpus Format
POS	Part of speech
NLP	Natural Language Processing
WSDL	Web Service Definition Language

Contents

Background and scope

This document is a result of work of CLARIN WG 5.6: LRT Integration, aiming at reviewing a number of Web services implementing linguistic processing chains and selecting appropriate standards for the resources and tools to be integrated. In contrast to other CLARIN work packages, it focuses on linguistic overview of issues related to adoption of standards to processing chains (or workflows).

The following document has been preceded with initial version of D5R-3 deliverable (Linguistic processing chains as Web Services: Initial linguistic considerations) containing less in-depth demonstrations of individual Web services forming the basis for the preliminary analysis of linguistic representation and annotation standards.

Technical aspects of Web services interoperability have been covered by WP2 and described in detail in Requirement Specification Web Services and Workflow systems (D2R-6b) document. Similarly, metadata- and architecture-related issues related to the persistent CLARIN infrastructure can be found in Web services and workflow creation (D2R-7a) document and are not repeated below.

This document will be discussed in the appropriate working groups and in the Executive Board. It will be subject of regular adaptations dependent on the progress in CLARIN.

Contributions

Section 1 has been contributed by Marie Hinrichs, Thomas Zastrow and Yana Panchenko from the Department of Linguistics at the University Tübingen and Helmut Schmid from the Institute for Natural Language Processing at the Stuttgart University.

Section 2 has been written by Maciej Ogrodniczuk, Michał Lenart, Agnieszka Patejuk and Adam Przepiórkowski from the Linguistic Engineering Group at the Institute of Computer Science, Polish Academy of Sciences.

Section 3 has been contributed by Normunds Grūzītis and Madars Virza from the Institute of Mathematics and Computer Science at the University of Latvia.

The annex provides a listing of language resources integrated into Web service infrastructure within CLARIN. Web service references contained there were included basing on responses of their respective authors and/or copyright holders.

Introduction

The need for text encoding standards for language resources (LRs) is widely acknowledged: within the International Standards Organization (ISO) Technical Committee 37 / Subcommittee 4 (TC 37 / SC 4), work in this area has been going on since the early 2000s, and working groups devoted to this issue have been set up in both CLARIN and FLaReNet (http://www.flarenet.eu). It is obvious that standards are necessary for the interoperability of tools and for the facilitation of data exchange between projects, but they are also needed within projects, especially where multiple partners and multiple levels of linguistic data are involved.

Given the existence of a number of proposed and de facto standards for various levels of linguistic annotation, the starting point of the design of a specific schema to be used in a particular project should be careful examination of those standards: in the interest of interoperability, creation of new schemata should be avoided. But, if a number of standards are applicable, which one should be chosen and on what grounds? And if constructing a schema for a particular linguistic level in a particular project should start with an overview and comparison of existing standards — a time consuming task which may be hard to justify within the limits of a project budget — would it not be easier and cheaper to construct one's own focused schema from scratch?

To answer these questions, three families of Web service-based processing chains are presented below in the form of extended showcases delivered by consortium members. These descriptions were selected to illustrate three variants of linguistic representation and were intended to provide exhaustive examples of applications built with popular frameworks. Importantly in this case, the text does not intentionally repeat findings from other CLARIN deliverables (such as technical recommendations for Web services or summary of representation standards), concentrating on cross-sectional approaches, metamodel concepts, power of expression, interoperability and notes on implementation processes rather than comparison of standards.

1 Standards in practice: WebLicht

WebLicht (Web Based Linguistic Chaining Tool, see http://www.d-spin.org for an overview) is a web-based service environment for the integration and use of language resources and tools. It includes NLP tools (realized as Web services), a repository for storing relevant information about each service, and a chaining mechanism that determines how the tools can be combined. These elements are brought together in a web application that enables users to create and run tool chains on their own texts.

The WebLicht infrastructure was designed and implemented in a collaborative effort by the following institutions:

- Seminar für Sprachwissenschaft, Universität Tübingen (Uni-Tübingen), Germany
- Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart (Uni-Stuttgart), Germany
- Abteilung Automatische Sprachverarbeitung, Universität Leipzig (Uni-Leipzig), Germany
- Berlin-Brandenburgische Akademie der Wissenschaften (BBAW), Berlin, Germany
- Das Institut für Deutsche Sprache (IDS), Mannheim, Germany

Other institutions and individuals have supported the project by providing tools in the form of Web services, including:

- Department of Modern Languages, University of Helsinki, Finland
- Romanian Academy Research Institute for Artificial Intelligence (RACAI), Bucharest, Romania
- Institute of Computer Science, Polish Academy of Sciences (PAN), Warsaw, Poland

WebLicht was developed to alleviate the problems associated with download-first tools and to provide a means for distributed tools to be combined into service chains. At present, digital language resources and tools (LRT) are available in different forms: some of them can be downloaded from the worldwide web, while others are shipped on CD-ROMs or DVDs. These tools are often restricted to a particular platform or environment, but some can be used independently of operating systems or data formats. In either case, the software must be installed or copied to a local computer or network. This can lead to problems that may be difficult to resolve. For example:

- The users' machines are often not able to handle the needed resources or tools. This includes incompatibilities between libraries and operating systems as well as the fact that even modern PCs are sometimes not powerful enough to fulfil the requirements of linguistic analysis.

- The lack of common data formats makes it impossible to create chains of tools or to compare different resources with each other easily.

In recent years, many attempts were made to find solutions to these persistent issues. Standardized text encodings like Unicode and markup languages like XML are addressing the lack of common data formats, while platform independent programming languages like Java and Python provide a promising way to make tools available across platforms. But still, the user must rely on the capabilities of his local machine.

To overcome this restriction, WebLicht makes the functionality of linguistic tools and the resources themselves available via the internet. To make use of the more than 120 tools and resources currently in WebLicht, the end user needs nothing more than a common web browser. The tools share a common data format, which means that any tools can make use of annotations that were produced earlier in the chain.

There are currently services available for German, English, Italian, French, Spanish, Romanian, Finnish, Czech, Hungarian, and Slovenian. However, the WebLicht infrastructure itself does not restrict the languages supported. A language is considered "supported" when one or more tools for processing it are available.

Currently approximately 120 webservices are available (at https://weblicht.sfs.uni-tuebingen.de, login via Shibboleth required), including:

- tokenizers,
- sentence border detection,
- part-of-speech taggers,
- named entity recognition,
- lemmatization,
- constituent parsing,
- coocurrence annotation,
- semantic annotator (GermanNet),
- data format converters (including MS Word/PDF/RTF to TCF, plain text to TCF, TCF <-> MAF, PAULA, Negra, TEI).

1.1 WebLicht architecture

In WebLicht, tools and resources are implemented as distributed services in a Service-Oriented Architecture (SOA). That means that they are not running on one central machine, but instead they are running on many different machines that are distributed across the web. A centralized database, the repository, stores technical and content-related metadata about each service. With the help of this repository, the chaining mechanism is implemented. The WebLicht user interface encapsulates this chaining mechanism in an AJAX-driven web application. It provides ways of uploading or generating a text corpus and guides the user in creating tool chains. Since web applications can be invoked from any browser, downloading and installation of individual tools on the user's local computer is avoided.

Service-Oriented Architecture

In most cases, the services are running at the institute where the tool was originally developed. This allows the authors to maintain control over their services.

Distributing the services in this way has several advantages. The computational workload is spread among various machines, resulting in better performance. It also has the advantage that when a tool is improved, the new version is immediately available. End users do not need to do anything to take advantage of the improved tool.

The services are created as *RESTful Web services*, which means that they make use of the HTTP protocol. Services do not interact directly with each other, but are invoked individually by sending the required document via HTTP.

Repository

To be available in WebLicht, a service must be registered in a centralized repository. The repository stores two different kinds of information about each service:

- Technical metadata: they provide details about what is required in the input document and what output the service generates. This information is needed to be able to compute valid tool chains. For example, a POS tagger normally needs a tokenizer to be applied to the text before it can be invoked.
- Descriptive metadata: concerning author, location of the service, legal information, description of the service etc.

The repository itself can be queried by invoking RESTful Web services, thus making it possible for both the chaining mechanism and the web GUI to retrieve information about the services available in WebLicht.

Chaining mechanism

The chaining mechanism determines which processing chains can be formed. This is done by matching the input requirements of the services in the repository with the data available in the document. The chaining process can be applied prior to actually invoking service tools, which makes it possible to build entire service chains quickly. This is achieved by creating a chain "profile" which is updated each time a service is added to the chain. The updated profile is used at each step to determine which services can possibly follow in the chain.

Interoperability

An important part of Service Oriented Architectures is ensuring interoperability between the underlying services. Interoperability of Web services, as they are implemented in WebLicht, refers to the seamless flow of data through the Web services in the processing chain. To be interoperable, these Web services must first agree on protocols defining the interaction between the services (WSDL/SOAP, REST, XML-RPC). They must also use a shared data exchange format, which is preferably based on widely accepted formats already in use (UTF-8, XML). WebLicht uses the REST-style API and a data format (Text Corpus Format, TCF) that is a valid XML format fully compliant with the *Linguistic Annotation Format* (LAF) and *Graph-based Format for Linguistic Annotations* (GrAF) developed in the ISO/TC37/SC4 technical committee (Ide & Suderman, 2007). The TCF format allows the combination of the various linguistic annotations produced by the tool chain. It supports incremental enrichment of linguistic annotations at different levels of analysis in a common XML-based format. Thus, the Web services within WebLicht are highly interoperable. Although TCF is used by most Web services, it is also possible to use other formats. Currently, converters to and from TCF and MAF, PAULA, TEI and Negra are available or in development.

Sample service chains

WebLicht's architecture allows a high degree of flexibility with respect to service processing chains. For example, Figure 1.1 shows a small subset of possible processing chains available for the German language. The circles represent individual webservices (labeled with the institute which created it and an identification number) and the rectangular boxes represent linguistic annotation states. An annotation state indicates which linguistic annotation layers are present in the data document. Often it is possible to reach an annotation state through different processing chains. This allows one to compare results from different tools and to pick-and-choose those tools which the user deems best suited to their needs.

Figure 1.2 shows the connectivity of all of the webservices currently available in WebLicht for the German language.

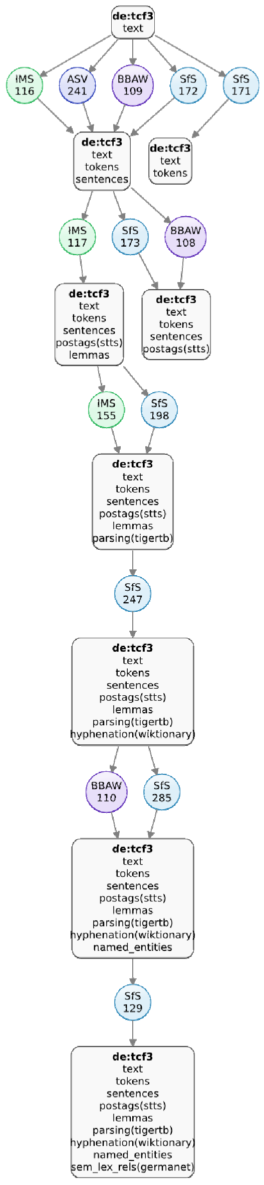
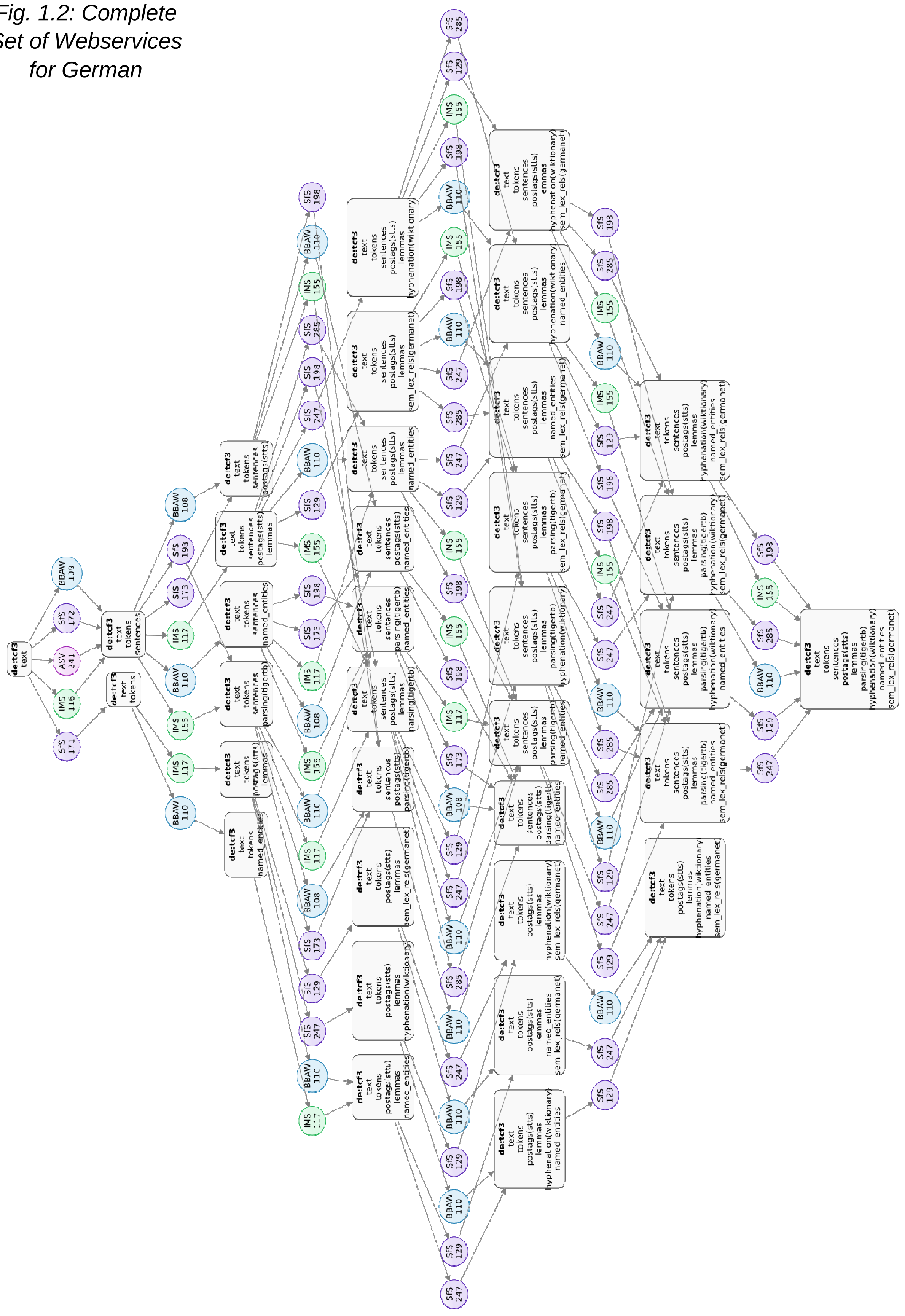


Fig. 1.1: Subset of Webservices available for German

Fig. 1.2: Complete Set of Webservices for German



1.2 Description of the TCF format

Introduction

The TCF data format is the primary data interchange format used by D-Spin webservices and was jointly developed by the D-Spin partners. TCF is an XML-based format which is encoded in Unicode. A typical TCF document contains a *text corpus* and a set of *linguistic annotations* such as tokens, part-of-speech (POS) tags, lemmas, or syntactic structures. TCF encodes each type of annotation in a separate *layer*.

A linguistic processing chain often starts with a raw corpus which only consists of a text. The corpus is sent through a pipeline of webservices. Each webservice adds one or more new annotation layers and the enriched corpus is sent to the next webservice of the processing chain for further annotation. Such processing chains can be composed with the WebLicht tool.

The following sections describe the TCF format in detail, and the appendix contains RelaxNG schemas for version 0.4 of the TCF format.

TextCorpus element

Text layer

Here is a very simple TCF document[1]:

```
<D-Spin version="0.4">
   <TextCorpus lang="de">
      <text>Peter aß eine Pizza. Sie schmeckte ihm.</text>
   </TextCorpus>
<D-Spin>
```

It contains a short text inside of a `<text>` element which is embedded in a `<TextCorpus>` element, which in turn is embedded in the top element `<D-Spin>`. The `<D-Spin>` element has an attribute "version" which specifies the version of the TCF format used for this document. The current version is 0.4 (as of February 2011). The `<D-Spin>` element has an optional attribute "lang" specifying the language of the corpus.

Token layer

Lingustic annotations are encoded by further XML elements which are added at the same level as the `<text>` element, i.e. below the `<TextCorpus>` element. The following TCF document contains a *tokens* layer:

```
<D-Spin>
  <TextCorpuslang="de">
    <text>Peter aß eine Salamipizza. Sie schmeckte ihm.</text>
    <tokens charOffsets="true">
      <token ID="t1" start="0" end="4">Peter</token>
      <token ID="t2" start="6" end="7">aß</token>
```

[1] This example is slightly simplified because it lacks name spaces and the obligatory <MetaData> element.

```
      <token ID="t3" start="9" end="12">eine</token>
      <token ID="t4" start="14" end="24">Salamipizza</token>
      <token ID="t5" start="25" end="25">.</token>
      <token ID="t6" start="27" end="29">Sie</token>
      <token ID="t7" start="31" end="39">schmeckte</token>
      <token ID="t8" start="41" end="43">ihm</token>
      <token ID="t9" start="44" end="44">.</token>
    </tokens>
  </TextCorpus>
</D-Spin>
```

Each token is represented by a `<token>` element which encloses the string of the token. The `<token>` element has an optional "ID" attribute which uniquely identifies it. It may also have "start" and "end" attributes to indicate the start and end position of the token in the character string of the `<text>` element. The `<token>` elements are embedded inside of a `<tokens>` element. The optional boolean "charOffsets" attribute of the `<tokens>` element indicates that start and end attributes are available at the `<token>` elements if its value is true.

Sentence layer

The segmentation of a text into sentences is represented with `<sentence>` elements. Each `<sentence>` element has a "tokenIDs" attribute which specifies the set of tokens that the sentence contains. The tokens are referred to by their IDs. Just like `<token>` elements, the `<sentence>` elements also have optional "ID" and "start" and "end" attributes. All `<sentence>` elements are enclosed in a `<sentences>` element. The "charOffset" attribute again indicates whether "start" and "end" attributes are present at the `<sentence>` elements.

The following partial TCF document shows how sentence boundary information is added to the previous document:

```
...
    </tokens>
    <sentences charOffsets="true">
      <sentence ID="s1" tokenIDs="t1 t2 t3 t4 t5" start="0" end="25"/>
      <sentence ID="s2" tokenIDs="t6 t7 t8 t9" start="27" end="44"/>
    </sentences>
  </TextCorpus>
</D-Spin>
```

POStag layer

Part-of-speech (POS) information is encoded in the *POStags* layer. The `<POStags>` element has an attribute "tagset" which specifies the tagset from which the inventory of POS tags is taken. The `<POStags>` element contains a sequence of `<tag>` elements which enclose the POS labels.

Each `<tag>` element has an attribute "tokenIDs" which specifies the token(s) to which the POS tag is assigned. It is possible to assign a single POS tag to a sequence of tokens. The token sequence may even be discontinuous which allows particle verbs such as "fährt ... ab" in "Der Zug fährt gleich ab" to be assigned a single POS tag.

It is also possible that two different `<tag>` elements refer to the same token. This could be used to represent the output of a POS tagger which leaves some ambiguities unresolved.

`<tag>` elements may also have an identifier attribute "ID".

```
...
    </sentences>
    <POStags tagset="STTS">
      <tag tokenIDs="t1">NE</tag>
      <tag tokenIDs="t2">VVFIN</tag>
      <tag tokenIDs="t3">ART</tag>
      <tag tokenIDs="t4">NN</tag>
      <tag tokenIDs="t5">$.</tag>
      <tag tokenIDs="t6">PPER</tag>
      <tag tokenIDs="t7">VVFIN</tag>
      <tag tokenIDs="t8">PPER</tag>
      <tag tokenIDs="t9">$.</tag>
    </POStags>
  </TextCorpus>
</D-Spin>
```

Lemma layer

The *lemmas* layer encodes lemma information for the tokens. A `<lemmas>` element contains a set of individual `<lemma>` elements which enclose the lemma string for the token (or tokens) referred to via the “tokenIDs” attribute.

```
...
    </POStags>
    <lemmas>
      <lemma tokenIDs="t1">Peter</lemma>
      <lemma tokenIDs="t2">essen</lemma>
      <lemma tokenIDs="t3">ein</lemma>
      <lemma tokenIDs="t4">Salamipizza</lemma>
      <lemma tokenIDs="t5">.</lemma>
      <lemma tokenIDs="t6">sie</lemma>
      <lemma tokenIDs="t7">schmecken</lemma>
      <lemma tokenIDs="t8">er</lemma>
      <lemma tokenIDs="t9">.</lemma>
    </lemmas>
  </TextCorpus>
</D-Spin>
```

Morphology layer

The *POStags* layer encodes the POS information as a flat string. The “morphology” layer provides an alternative representation for morphosyntactic information based on *feature structures*. It is also capable of representing the internal structure of morphologically complex words.

A `<morphology>` element has an optional boolean attribute “segmentation” to indicate whether word segmentation information is encoded or not and contains a set of `<analysis>` elements. Each `<analysis>` element has a “tokenIDs” attribute to refer to the annotated token(s) and contains a `<tag>` element and an optional `<segmentation>` element.

The `<tag>` element contains an `<fs>` element which encodes the morpho-syntactic information as a feature structure. The `<fs>` element contains a sequence of `<f>` elements which represent the features. Each `<f>` element has a *name* attribute which specifies the feature name and encloses either a character string which represents an atomic feature value or an `<fs>` element if the structure is recursive.

```
...
   <morphology segmentation="true">
     <analysis tokenIDs="t4">
       <tag>
         <fs>
           <f name="cat">noun</f>
           <f name="case">acc</f>
           <f name="gender">fem</f>
           <f name="number">singular</f>
         </fs>
       </tag>
       <segmentation>
         <segment cat="noun" start="0" end="10">
           <segment type="stem" cat="noun" func="comp" start="0" end="5">
             Salami
           </segment>
           <segment type="stem" cat="noun" func="base" start="6" end="10">
             Pizza
           </segment>
          </segment>
        </segmentation>
      </analysis>
    </morphology>
  </TextCorpus>
</D-Spin>
```

If the “segmentation” attribute of the `<morphology>` element is present and has the value “true”, then each `<analysis>` element also contains a `<segmentation>` element to encode the morphological structure of the word. The `<segmentation>` element contains a sequence of `<segment>` elements. Each `<segment>` element has the following optional attributes:

- “type” distinguishes between different types of morphemes such as “stem”, “prefix”, “suffix”.
- “cat” represents the syntactic category.
- “func” further specifies the function of the morpheme such as compounding, derivation or base stem, for instance.
- “start” and “end” encode the start and end position of the morphological segment in the character string of the word form.

The `<segment>` element either encloses the character string if the segment consists of a single morpheme, or a sequence of `<segment>` elements if it is morphologically complex.

Named Entity layer

The *namedEntities* layer encodes the output of a named entity recognizer. The `<namedEntities>` element has a “tagset” attribute which specifies the set of named entity categories used, such as MUC1990, TIMEX etc. The `<namedEntities>` element encloses a sequence of `<entity>` elements.

Each `<entity>` element has a "tokenIDs" attribute which specifies the set of tokens that the named entity consists of and a "class" attribute which specifies the class of the named entity.

```
...
   <namedEntities tagset="MUC1990">
      <entity class="PERSON" tokenIDs="t1"/>
    </namedEntities>
  </TextCorpus>
</D-Spin>
```

WordSplittings layer

The *WordSplittings* layer can be used to represent a syllabification or hyphenation or other kind of internal segmentation of words. The `<WordSplittings>` element has an attribute "type" which encodes the type of segmentation (e.g. syllabification or hyphenation) and encloses a set of `<split>` elements. Each `<split>` element encodes the segmentation of one token (which is referred to via the "tokID" attribute) by the enclosed sequence of integers which specify the positions at which the token is split.

```
...
    <WordSplittings type="hyphenation">
      <split tokID="t4">2 4 6 9</split>
    </WordSplittings>
  </TextCorpus>
</D-Spin>
```

Phonetic layer

The *Phonetics* layer represents the phonetic transcriptions of words. Its "transcription" attribute specifies the encoding of the phonetic symbols (e.g. IPA). The `<Phonetics>` element encloses a set of `<pron>` elements. Each `<pron>` element refers to a token via the "tokID" attribute and encloses a character string which represents the pronunciation of the token.

```
...
    <Phonetics transcription="IPA">
      <pron tokID="t7">....</pron>
    </Phonetics>
  </TextCorpus>
</D-Spin>
```

QueryResults layer

The *QueryResults* layer encodes the result of a corpus query. The "query" attribute of the `<QueryResults>` element has the query string itself as attribute. The `<QueryResults>` element encloses a set of `<match>` elements which represent the individual matches that the query processor has found. Each `<match>` element has a "tokenIDs" element which specifies the tokens belonging to this match result. The "tokenIDs" attribute of the optional `<key>` element indicates the subset of tokens which was actually matched by the query.

```
...
    <QueryResults query="[pos=VVFIN][]*[pos=NN] within s">
      <match tokenIDs="t1 t2 t3 t4">
        <key tokenIDs="t2 t4"/>
      </match>
    </QueryResults>
  </TextCorpus>
</D-Spin>
```

The `<token>` elements that the `<match>` elements refer to, must have been stored in a separate `<tokens>` layer. This has the advantage that those tokens can be POS tagged, lemmatized, and parsed just as any other corpus. If the query results are not complete sentences, the results of further processing steps might be suboptimal, however.

Parsing layer

Constituent parses are represented by the *parsing* layer. The "tagset" attribute specifies the inventory of non-terminal labels used in the parse trees. Each parse tree is stored in a separate `<parse>` element. A `<parse>` element has an optional "ID" attribute and contains a single `<constituent>` element. Each `<constituent>` element has a "cat" attribute which specifies the syntactic category and an optional "ID" attribute.

`<constituent>` elements which correspond to pre-terminal nodes of the parse tree (i.e. POS tags) have an additional "tokenIDs" attribute which specifies a sequence of tokens (a single word or a multi-word element) which this pre-terminal category expands to. Other `<constituent>` elements are recursive and enclose a sequence of `<constituent>` elements.

```
...
  <parsing tagset="Tiger">
   <parse>
    <constituent ID="c11" cat="TOP">
     <constituent ID="c12" cat="S-TOP">
      <constituent ID="c13" cat="NP-SB">
       <constituent cat="PPER-HD-Nom" ID="c14" tokenIDs="t6"/>
      </constituent>
      <constituent cat="VVFIN-HD" ID="c15" tokenIDs="t7"/>
      <constituent ID="c16" cat="NP-DA">
       <constituent cat="PPER-HD-Dat" ID="c17" tokenIDs="t8"/>
      </constituent>
     </constituent>
     <constituent cat="\$." ID="c18" tokenIDs="t9"/>
    </constituent>
   </parse>
  </parsing>
  </TextCorpus>
</D-Spin>
```

This representation is also able to encode trees with discontinuous constituents because the tokens at the bottom layer are referenced via their IDs and their order is separately specified in the tokens layer.

Dependency parsing layer

For dependency parses, there is a separate depparsing layer. The <parse> elements used here enclose a set of dependency relations. Each <dependency> element has a "depIDs" and a "govIDs" attribute to specfify the token(s) which the dependent or governor consists of, respectively. The optional “func” attribute represents the grammatical function of the dependent.

```
...
    <depparsing>
      <parse>
        <dependency func="SUBJ" depIDs="t1" govIDs="t2"/>
        <dependency func="SPEC" depIDs="t3" govIDs="t4"/>
        <dependency func="OBJ" depIDs="t4" govIDs="t2"/>
      </parse>
      <parse>
        <dependency func="SUBJ" depIDs="t6" govIDs="t7"/>
        <dependency func="OBJ" depIDs="t8" govIDs="t7"/>
      </parse>
    </depparsing>
  </TextCorpus>
</D-Spin>
```

Relations layer

Beyond syntactic dependencies, there are other kinds of linguistic relations between words or constituents which are encoded in corpora. A typical example are coreference chains.

The `<relations>` element is a general means for the encoding of such relations. Its “type” attribute indicates the type of relation such as *coreference*. It encloses a sequence of `<relation>` elements. Each `<relation>` element has an attribute “refIDs” which specifies the sequence of entities (tokens or parse constituents etc.) which enter this relation. The optional attribute “func” provides further information on the type of the relation.

```
...
    <relations type="coreference">
      <relation refIDs="c3 c17"/>
      <relation refIDs="c7 c13"/>
    </relations>
  </TextCorpus>
</D-Spin>
```

WordNet layer

The *sem_lex_rel* layer represents information about WordNet relations. The “src” attribute of the `<sem_lex_rel>` element encodes the WordNet version from which the information was taken. Each `<sem_rel>` element has a “type” attribute which specifies the type of the relation (such as synonymy, hyponymy, or antonymy) and encloses a set or `<orthform>` elements. Each `<orthform>` element refers to the word whose relations are displayed via a “tokenIDs” attribute, and encloses the sequence of words that the word is related to.

```
...
    <sem_lex_rels src="GermaNet 5.3">
      <sem_rel type="synonymy"/>
        <orthform tokenIDs="t25">Orange|Apfelsine</orthform>
      </sem_rel>
    </sem_lex_rels>
```

```
  </TextCorpus>
</D-Spin>
```

The MetaData element

Each TCF document must contain a `<MetaData>` element as a child of the `<D-Spin>` element. This element was not shown in the previous examples for the sake of simplicity. Currently, the only allowed element inside of `<MetaData>` is `<source>`. In the future, the `<MetaData>` element will be used to store meta data and provenance data for the corpus once this information becomes available. The XML schema for the `<MetaData>` element will then be further extended.

```
<D-Spin version="0.4">
  <MetaData>
    <source>IMS, Uni Stuttgart</source>
  </MetaData>
  <TextCorpus>
    <text>Peter aß eine Käsepizza. Sie schmeckte ihm.</text>
  </TextCorpus>
</D-Spin>
```

The Lexicon element

The TCF format cannot only be used to store corpus data, but also lexical information which is encoded with a `<Lexicon>` element rather than a `<TextCorpus>` element. Thus far, the `<Lexicon>` element was primarily used to encode the result of a collocation extraction webservice.

The <Lexicon> element has an optional attribute “lang” which specifies the language. Possible child elements (= annotation layers) of the `<Lexicon>` element are *lemmas*, *POStags*, *frequencies*, and *word-relations*.

The `<lemmas>` element specifies the basic entities of a Lexicon document. It contains a set of `<lemma>` elements. Each `<lemma>` element encloses the character string of a lemma and the optional attribute “ID” contains a unique identifier as value.

The `<POStags>` element encodes part-of-speech information for the lemmas. Its “tagset” attribute specifies the tag inventory from which the POS tags are taken. The `<POStags>` element contains a set of `<tag>` elements. Each `<tag>` element encloses a POS tag string for a lemma which is referred to via a “lemID” attribute.

In a similar way, the `<frequencies>` element contains a set of `<frequency>` child elements each of which specifies the frequency of a lemma. The lemma is again referred to via a “lemID” attribute and the frequency is represented as an integer value which is enclosed by the `<frequency>` element.

Finally the `<word-relations>` element is used to encode collocations and other word-word relations. `<word-relation>` encloses a set of word `<word-relation>` elements. Each `<word-relation>` element has optional attributes “type”, “func”, and “freq” to encode the type of the relation (e.g. “syntactic relation”), a further functional specification (e.g. “verb+direct-object”) and the frequency of occurrence, respectively. A `<word-relation>` element contains a sequence of at least 2 `<term>` elements which specify the terms that enter the word relation, and any number of `<sig>` elements. Each `<term>` element specifies a lemma either directly by enclosing its character string or by referencing it via a “lemID” attribute. The `<sig>` elements are used to encode the results of different word association measures. The “measure” attribute indicates the word association measure used such as “mutual information” or “log-likelihood ratio”.

The following example shows a Lexicon document complete with MetaData and name-space information:

```
<D-Spin xmlns="http://www.dspin.de/data" version="0.4">
  <MetaData xmlns="http://www.dspin.de/data/metadata">
    <source>IMS, Uni Stuttgart</source>
  </MetaData>
  <Lexicon xmlns="http://www.dspin.de/data/lexicon" lang="de">
    <lemmas>
      <lemma ID="l1">halten</lemma>
      <lemma ID="l2">Vortrag</lemma>
    </lemmas>
    <POStags tagset="basic">
      <tag lemID="l1">Verb</tag>
      <tag lemID="l2">Noun</tag>
    </POStags>
    <frequencies>
      <frequency lemID="l1">1257</frequency>
      <frequency lemID="l2">193</frequency>
    </frequencies>
    <word-relations>
      <word-relation type="syntactic relation" func="verb+direct-object" freq="13">
        <term lemID="l1"/><term lemID="l2"/>
        <sig measure="MI">13.58</sig>
      </word-relation>
    </word-relations>
  </Lexicon>
</D-Spin>
```

1.3 Description of visualization

Visualization of linguistic data is important for its analysis, for discovering of new information about the language, for getting an idea about language processing tools. Annotation Viewer is a visualization tool developed as a part of WebLicht and in accordance with SOA (Service Oriented Architecture) principles. It concerns user-friendly presentation of language resources and is used to view the results of the language tools processing.

WebLicht web application integrates Annotation Viewer by offering the user an option to view tool chain processing results in a user-friendly graphical interface (see Fig. 1.3).

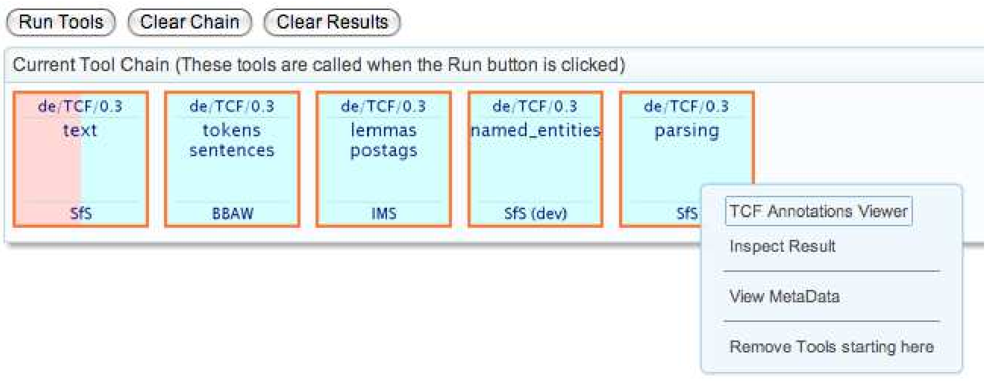
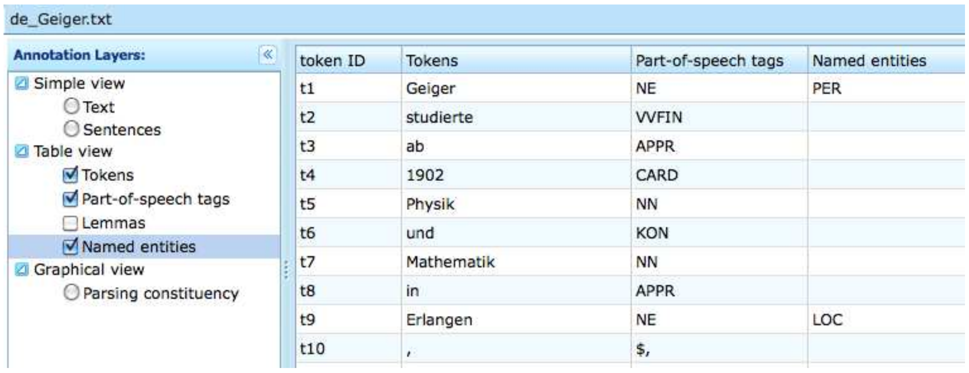

Run Tools
Clear Chain
Clear Results
Current Tool Chain (These tools are called when the Run button is clicked)
de/TCF/0.3 text SfS
de/TCF/0.3 tokens sentences BBAW
de/TCF/0.3 lemmas postags IMS
de/TCF/0.3 named_entities SfS (dev)
de/TCF/0.3 parsing SfS
TCF Annotations Viewer
Inspect Result
View MetaData
Remove Tools starting here

Fig 1.3. WebLicht user interface.

The visualization tool can also be used as a stand-alone web application available over the net. It can either be called from another application or used directly. In the latter case the users can upload data in TCF0.3 format and get a graphical view of their linguistic annotations.

Currently Annotation Viewer visualizes data present in lexicon and text corpus format TCF0.3, such as text, sentence annotations, tokens, lemmas, part-of-speech tags, named entity tags, lexical-semantic relations and parsing annotations. Inline annotations are visualized in a table and users can select to view the annotation type or types they are interested in (see Fig. 1.4).



token ID	Tokens	Part-of-speech tags	Named entities
t1	Geiger	NE	PER
t2	studierte	VVFIN	
t3	ab	APPR	
t4	1902	CARD	
t5	Physik	NN	
t6	und	KON	
t7	Mathematik	NN	
t8	in	APPR	
t9	Erlangen	NE	LOC
t10	,	$,	

Fig. 1.4. Annotation Viewer.

Parsing annotations are visualized as tree images (see Fig. 1.5).

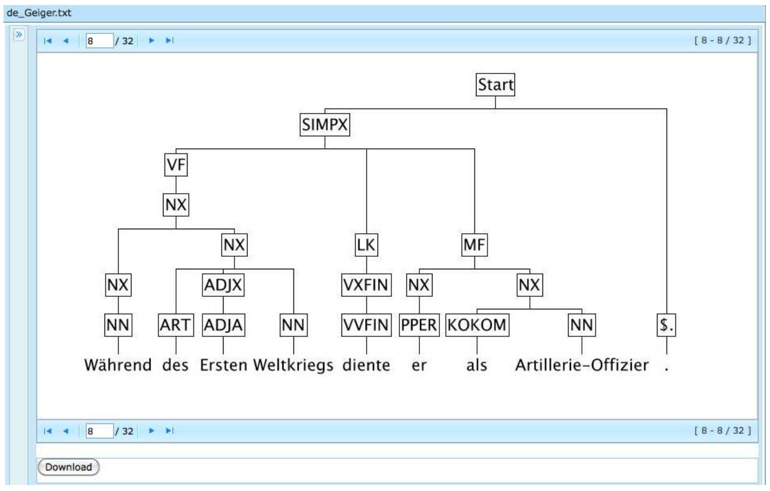


Fig. 1.5: Tree images.

The tree images are scrollable if their size exceeds the viewarea (see Fig. 1.6).

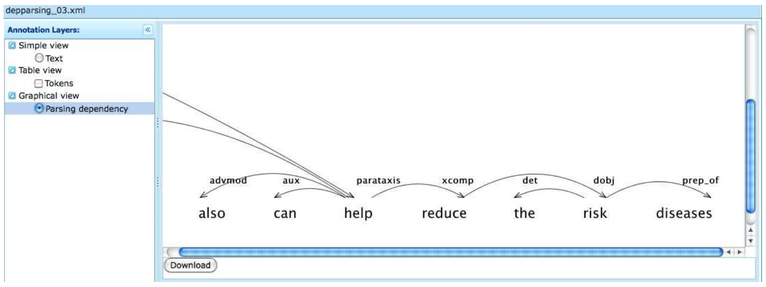


Fig. 1.6: Scrollable tree image.

The images can be downloaded in JPG, PDF and SVG formats.

Under the hood Annotation Viewer implementation uses a StAX-based library to read the data, and ZK Ajax framework to visualize the data. StAX parser allows for efficient use of CPU and RAM. The Java library, built on top of the StAX parser to bind the text corpus format allows for easy data access. The ZK toolkit provides a rich user experience. The benefits of the visualization tool for the end users is that they do not need to know anything about the internal implementation, they do not need to install anything and can run it on a computer with any operating system which has a modern browser available.

In future work, the Annotation Viewer will be made more interactive, to allow user not only to view, but also to edit linguistic annotations.

2 Standards in practice: ICS PAS service

The ICS PAS Web Service (later referred to as the *Multiservice*) intends to provide a common interface and methods of linguistic annotation of Polish texts basing on individual offline tools. Its format builds on results of the National Corpus of Polish, a recent project of a large (one billion – 10^9)-token size general corpus of Polish.

2.1 Background project and format[2]

Polish, the most represented Slavic language of the EU, had until recently been underrepresented by a national corpus, i.e., a large, balanced and publicly available corpus, which would be at least morphosyntactically annotated. Before 2008 there existed three contemporary[3] Polish corpora which were – to various extents – publicly available. The largest and the only one that fully morphosyntactically annotated was the IPI PAN Corpus (http://korpus.pl; Przepiórkowski 2004),

[2] This section is largely based on (Bański and Przepiórkowski 2009, Przepiórkowski 2009a, Przepiórkowski and Bański 2009b, and Przepiórkowski et al. 2010).

[3] Another – much smaller and dated, but historically very important – corpus is available in its entirety: http://www.mimuw.edu.pl/polszczyzna/pl196x, a 0.5-million word corpus created in the 1960s as the empirical basis of a frequency dictionary (Kurcz et al. 1990). An electronic version of the corpus has been made available in 2000s (Ogrodniczuk 2003).

containing over 250 million segments (over 200 million orthographic words), but – as a whole – rather badly balanced[4]. Another corpus, which is considered to be carefully balanced, the PWN Corpus of Polish (http://korpus.pwn.pl), contained over 100 million words, of which only 7,5 million sample were freely available for search. The third corpus, the PELCRA Corpus of Polish (http://korpus.ia.uni.lodz.pl), also contained about 100 million words, all of which were publicly searchable.

Between 2008 and 2010 the National Corpus of Polish (Pol. Narodowy Korpus Języka Polskiego; NKJP; http://nkjp.pl, see Przepiórkowski et al. 2008, 2009) was constructed using the funds of the Polish Ministry of Science and Higher Education (project number: R17 003 03).

The NKJP project involved 4 Polish institutions:

- Institute of Computer Science of the Polish Academy of Sciences (coordinator),
- Institute of Polish Language of the Polish Academy of Sciences,
- University of Łódź,
- Polish Scientific Publishers PWN.

Corpus texts gathered by project partners included books, newspapers, magazines, blogs, transcripts of spoken data, etc. The size of the result data set is one billion tokens whereas a one-million-token subcorpus had been prepared in a hand-verification process.

The resource has been designed to represent various kinds of metadata and structural information, namely:

- fine-grained word-level segmentation (including some segmentation ambiguities),
- morphosyntax (referring to disambiguated segmentation),
- coarse-grained syntactic words (e.g., for analytical tense forms consisting of multiple segments; referring to morphosyntax),
- named entities (referring to syntactic words),
- syntactic groups (the annotation of nominal and other phrases, without the requirement that each word in the sentence must be contained in some syntactic group, also referring to syntactic words),
- word senses (ignored in the further part of this document since they are not yet made available by the Multiservice).

NKJP and standards

From the very beginning NKJP was committed to following current standards and best practices, but it turned out that the choice of text encoding for multiple layers of linguistic annotation is far from clear. The conclusion of the overview paper, (Przepiórkowski and Bański 2009a), was that the guidelines of the Text Encoding Initiative (Burnard and Bauman 2008; http://www.tei-c.org) should be followed, as it is a mature and carefully maintained de facto standard with a rich user base. At the time of project start various proposed ISO TC 37 / SC 4 standards, including Morphosyntactic Annotation Framework (MAF), Syntactic Annotation Framework (SynAF) and Linguistic

[4] There exists a 30-million segment subcorpus of the IPI PAN Corpus which is relatively balanced (Przepiórkowski *et al.* 2008).

Annotation Framework (LAF), were still under development and, especially SynAF and LAF, had the form of general models rather than specific off-the-shelf solutions.

The Text Encoding Initiative (TEI Consortium, 2007) has been at the forefront of text annotation and resource interchange for many years. It has influenced corpus linguistic practices in at least three related ways. Firstly, the formalism itself, in the mature form, has been used to mark up linguistic corpora, e.g. the British National Corpus. An early application of the TEI, the Corpus Encoding Standard (CES; Ide and Priest-Forman 1995, Ide 1998; see http://www.cs.vassar.edu/CES/), together with its XML version, XCES (Ide et al. 2000; see http://www.xces.org/), have served as de facto standards for corpus encoding in numerous projects. Finally, the experience gained in creating and using XCES (together with e.g. the feature-structure markup of the TEI) has served as a foundation for the Linguistic Annotation Format (LAF, Ide and Romary, 2007), within ISO TC37 SC4. LAF promises to provide a standard interchange format for linguistic resources of many diverse kinds and origins.

The relationship between the TEI (especially in its stand-off version) and the LAF is straightforward. Both are implemented in XML, which makes transduction between a rigorous TEI format and the LAF "dump" (pivot) format mostly a matter of fleshing out some data structures. For lexical structures, TEI can be precisely mapped onto LMF model.

The current TEI P5 version is a de facto, constantly maintained XML standard for encoding and documenting textual data, with an active community, detailed guidelines and supporting tools. TEI recommendations for the encoding of linguistic information are limited, but it includes the ISO FSR standard for representing feature structures, which can be used to encode various kinds of information (cf., e.g., Witt et al. 2009).

TEI P5 was selected as the base text encoding standard for NKJP with certain additional assumptions, described below[5]. Nevertheless, when selecting from the rich toolbox provided by TEI, an attempt has been made to follow recommendations of these proposed ISO standards, as well as other common XML formats, including TIGER-XML (Mengel and Lezius 2000) and PAULA (Dipper 2005). This work, described in more detail in (Przepiórkowski and Bański 2009b) as well as in (Przepiórkowski 2009a), resulted in TEI P5 XML schemata encoding data models largely isomorphic with or at least mappable to these formats.

Morphosyntax

Following the common standard practice of stand-off approach to annotation, each linguistic level is represented in its own file, referring to lower layers, down to the primary text. More precisely, each corpus text is represented as a collection of files containing various annotation layers of the text, and each layer has the following minimal general structure (where `corpus_header.xml` contains the unique corpus header, and `header.xml` is the header of the particular text).

Depending on the type of text (written or spoken), `<body>` contains a list of `<p>`aragraphs (or possibly paragraph-length anonymous blocks, `<ab>`) or `<u>`tterance turns, further split into `<s>`entences. This structure, down to the sentence level, is preserved and made parallel at all annotation layers. The correspondence between different layers is expressed with the TEI attribute `@corresp`.

[5] Far more detailed information on the TEI encoding of the National Corpus of Polish can be found in section 4.2 of the CLARIN-D5C-3 document on Interoperability and Standards.

Whatever other annotation layers are present in the corpus, the existence of a morphosyntactic layer (`ann_morphosyntax.xml`) is mandatory. Each sentence at this layer is a sequence of `<seg>` elements implicitly (via a specification in the schema) marked as `@type="token"`, and each `<seg>` contains a feature structure specification of various morphosyntactic information about the segment.

Constituency

At the syntactic level (`ann_syntax.xml`), each `<s>`entence is a sequence of `<seg>` elements implicitly marked as `@type="group"`. More generally, for reasons of uniformity, <seg> elements with different values of `@type` are used for different kinds of linguistic units.

Just like word segments, syntactic groups also contain feature structure descriptions: in the simplest case, such a description may consist of a single attribute-value pair naming the node, but it could also be an LFG f-structure or a fully-fledged HPSG feature structure. Apart from a feature structure specification of the node, such a syntactic group element may contain any number of `<ptr>` elements pointing at the immediate constituents of the group.

Immediate constituents may be words specified at a different layer or other syntactic groups of the same layer. Any `<ptr>` element may specify the type of the constituency relation, e.g., head or nonhead, and each has an `@xml:id`, so that the relation may be referred to in the feature structure description of the node.

This schema allows for discontinuous constituents, as `<ptr>` elements within one `<seg>` do not have to point at neighbouring constituents. This freedom, combined with the representations for dependencies outlined in the following subsection, makes it possible to encode various linguistic analyses of possible conflicts between phrase structure and dependency, e.g., involving extraposed material or crossing dependencies.

Dependency

Dependency relations in TEI P5 are represented in equally straightforward way with the `<link>` element relating two syntactic nodes (words or groups). According to TEI Guidelines, `<link>` may be used to represent symmetrical (bidirectional) or asymmetrical (unidirectional) relations; in NKJP, by convention, `<link>` represents asymmetrical edges in the dependency graph, whose end vertices are specified in the value of the attribute `@targets`.

The value of `@type` specifies the kind of dependency, e.g. the dependency of type subject holds between a word (defined in `ann_morphosyntax`) and a syntactic group (defined elsewhere in the same file). In a “pure” dependency treebank, where dependencies are strictly between words, `<seg>` elements may be completely absent in this layer.

2.2 ICS PAS Multiservice format

To achieve the goal of providing a common interface and methods of linguistic annotation of Polish texts basing on heterogeneous offline tools, adopting a common representation and interchange format was necessary to bind the tools together and present results to the user. As stated above, such format has been developed for the National Corpus of Polish (NKJP) as a stand-off, TEI P5-based annotation which stores different levels of description in separate, interlinked files (Przepiórkowski and Bański, 2009b, 2010). For interchange, a certain method of packaging them had to be applied.

Among several possible methods (e.g. with additional minimalistic linkbase file referencing the annotation files and bundled together – similarly to OpenDocument representation) following the TEI path seemed the most reasonable choice since it allows maintenance of compatibility with the current standard while keeping encoding overhead not substantially different from competitive interchange formats (not taking into account the extensive metadata description by TEI).

The model presented below is a small, but useful extension of the representation prepared for NKJP. The idea of merging separate annotation levels into a single file facilitates usage of one the key features of stand-off annotation: interdependence of layers.

In its generality, the format is ready to store output of different variants of the same annotation type produced by competitive tools which fosters linguistic comparisons. Similarly, it is open to enlargement of scope of description (for Polish it is soon planned to be adopted by higher-level annotation tools such as coreference resolvers, named entities recognizers or word-sense disambiguators) and adoption to other languages.

The format has been tested in practice both in the process of representation of linguistic information (by NKJP) and its interchange – by the linguistic Web service for Polish.

The structure of the package

Four important decisions in the process of selecting the annotation model for representation of linguistic information are related to:

- basic representation of text encoding and format – made more or less indisputable by the advantages offered by XML and UTF-8, confirmed by widespread use of these standards,
- concrete encoding method within a consistent framework – in our case facilitated by earlier decision of using TEI P5 in the process of encoding NKJP,
- packaging method – initially commented in the introductory section and resulting in the easiest possible option of creating a corpus out of separate annotation layers,
- representation of linguistic information – here: NKJP-based, but potentially still open to improvements with respect to the richness of TEI and generality of feature structures (ISO:24610-1, 2005).

Of the above, the third and fourth decisions seem to need certain explanation. A competitive method of encoding a collection of annotations could rely on `<group>` elements grouping together a sequence of distinct texts (or groups of such texts) which are regarded as a unit for some purpose, as indicated in the TEI Guidelines[6]. However, this idea should be turned down due to numerous reasons: different practices of usage of this element (for *collected works of an author, a sequence of prose essays, etc.*)[1], extensive overhead in representation of annotation descriptions (such as necessity of using `<front>` for composite `<text>`s, impossibility of encoding different headers for annotation components (which is useful for separate processing of annotation layers) and, last but not least, incompatibility with NKJP encoding (inability to store `<teiCorpus>` elements inside any other element than `<teiCorpus>` itself).

As for representation of linguistic information and linking it to the base text, TEI offers numerous, often much simpler methods for this purpose (e.g. by means of `<text>`s with `<interp>`retations of a certain `type` linked to the base text with `inst` attribute). After serious consideration the NKJP format was selected for practical reasons: no significant benefit could be offered by any new format with respect to the number of existing tools and practices already applied in the currently mature NKJP project.

[6] See description of `<group>` element in (Burnard and Bauman 2008), Appendix C: Elements (http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-group.html).

Summarizing, the adopted annotation model consists of:

- a single corpus (`<teiCorpus>`) as a means of representation of the collection of source text and annotations,
- several subcorpora (second-level `<teiCorpus>` elements), each representing a different level of annotation (sentence-level segmentation, tokenization etc.),
- individual texts in every subcorpus (`<TEI>`), each representing a variant of annotation on a given level (e.g. results of tagging coming from different taggers).

The structure of the output package is therefore:

```
<teiCorpus>
  <teiHeader>
    <!- header information for the package ->
  </teiHeader>
  <teiCorpus>
    <!- single annotation level ->
    <teiHeader>
      <!- header information for the annotation level ->
    </teiHeader>
    <TEI>
      <!- single annotation variant ->
      <teiHeader>
        <!- header information for the variant annotation ->
      </teiHeader>
      <text>
        <body>
          <!- annotation details ->
        </body>
      </text>
    </TEI>
    <!- possibly more annotation variants of the current level
         in the form of <TEI> elements ->
  </teiCorpus>
  <!- possibly more annotation levels in <teiCorpus> elements ->
</teiCorpus>
```

As required by the TEI document model, the corpus package must always contain at least one annotation level with at least one annotation variant. This assumption is satisfied in very straightforward manner since every annotation model should contain at least basic segmentation in the form of the TEI-encoded source (see *Segmentation* below).

TEI headers

All three types of headers in the package (the package header, headers for subcorpora representing annotation variants and headers for individual variants) follow the TEI recommendation for a minimal header[7], supplying only the minimal level of encoding required.

The package header

The `title` element, one of the two required inside a file description (`<fileDesc>`), provides rough information on the type of returned content (e.g. "treebank representation", "morphological analysis" etc.). The responsibility statement (`<respStmt>`) refers to the multiservice only (without listing its components used in the annotation process; their names and types[8] are provided in nested headers) assigning it the role of text encoder (of the data analysed by constituent tools, referred to in the corpus substructure).

The publication statement (`<publicationStmt>`) contains information about the process of electronic delivery of the data. `date` and `time` fields are used to represent the moment of issuing the output "corpus" to the end user (in a normalized format, `YYYY-MM-DD` and `HH:MM:SS` respectively) and duration of the packaging (in a formalized W3C format[9]). The multiservice is listed as the `distributor` of the data. As `address` is intended to be used for postal addresses only, it was omitted (although storing information on Multiservice Internet address could be helpful).

The source description (`<sourceDesc>`) provides basic reference to the source of processed input – text retrieved from the Web form, supplied file or external URL, pointed to with a `<ptr>` in the latter case.

Following the above assumptions, the header may look like:

```
<teiHeader>
  <fileDesc>
    <titleStmt>
      <title>Treebank representation of the text</title>
      <respStmt>
        <resp>provided by</resp>
        <name type="webService">ICS PAS Multiservice</name>
      </respStmt>
    </titleStmt>
    <publicationStmt>
      <distributor>ICS PAS Multiservice</distributor>
      <date when="2010-11-17">
        <time when="15:30:00" dur="PT0.02S"/>
      </date>
    </publicationStmt>
```

[7] See (Burnard and Bauman 2008): 2.6 – Minimal and Recommended Headers, http://www.tei-c.org/ release/doc/tei-p5-doc/en/html/HD.html#HD7.

[8] Note that type must be a legal XML name (see http://www.w3.org/TR/REC-xml/#dt-name) which results in using e.g. camelCase instead of whitespace in attribute values.

[9] See http://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-data.duration.w3c.html.

```
    <sourceDesc>
      <p>Text retrieved from <ptr target="http://example.edu/
                                          test.txt"/>.</p>
    </sourceDesc>
  </fileDesc>
</teiHeader>
```

Annotation-level headers and variant annotation headers

At the annotation level the `<title>` stores information about the name and/or purpose of the layer (e.g. „morphological analysis"); information `<distributor>` is still the annotation generator, also responsible (by means of `<respStmt>`) for TEI P5-encoding of data. `<date>` and `<time>` fields refer to the execution of the processing at the given annotation level.

```
<teiHeader>
  <fileDesc>
    <titleStmt>
      <title>Morphological analysis</title>
      <respStmt>
        <resp>TEI P5-encoded by</resp>
        <name type="webService">ICS PAS Multiservice</name>
      </respStmt>
    </titleStmt>
    <publicationStmt>
      <distributor>ICS PAS Multiservice</distributor>
      <date when="2010-10-01">
        <time when="15:00:45" dur="PT0.02S"/>
      </date>
    </publicationStmt>
    <sourceDesc>
      <p>Text retrieved from <ptr target="http://example.edu/test.txt"/>.</p>
    </sourceDesc>
  </fileDesc>
</teiHeader>
```

TEI headers stored inside `<TEI>` elements representing variant annotation for a given level (within the same `<teiCorpus>`) provide information on the tool responsible for the concrete annotation. `<respStmt>` and `<distributor>` fields contain the tool name while `<date>`/`<time>` fields contain the timestamp of the end of the analysis process and the information about its duration.

```
<teiHeader>
  <fileDesc>
    <titleStmt>
      <title>Morphological analysis</title>
      <respStmt>
        <resp>provided by</resp>
        <name type="morphologicalAnalyser">Morfeusz</name>
      </respStmt>
    </titleStmt>
    <publicationStmt>
      <distributor>Morfeusz</distributor>
      <date when="2010-10-01">
        <time when="15:00:45" dur="PT0.02S"/>
      </date>
    </publicationStmt>
    <sourceDesc>
      <p>Text retrieved from <ptr target="http://example.edu/test.txt"/>.</p>
    </sourceDesc>
  </fileDesc>
</teiHeader>
```

Annotation components

Annotations are represented as TEI `<text>`s. Formally, the `<TEI>` element of a particular annotation variant contains a header information followed by a single `<text>`/`<body>` with `<p>`aragraphs grouping generic `<seg>`ments corresponding to annotation elements. Following NKJP concept, this structure is preserved and made parallel at all annotation levels (except for segmentation where `<s>`entences can be marked additionally, see *Segmentation* below).

To enhance human-readability of the text, segments are preceded with additional XML comments storing fragments of source related to the annotated unit. Linguistic features are preserved using TEI-embedded feature structure formalism.

Identifiers and links

Each corpus segment intended to be further referenced is given a unique identifier. To maintain consistency, identifiers take a standardized form of the following:

1. one-letter level symbol:
   - `s` – segmentation,
   - `l` – lemmatization,
   - `m` – morphological analysis,
   - `t` – tagging,
   - `w` – syntactic words,
   - `g` – syntactic groups,
   - `p` – deep parsing,
2. the number of the variant of particular layer (since there may be more than one e.g. tagging layer present, produced by rival taggers),
3. a minus sign,
4. a compound tag index (without intention to be further processed) composed of dot-separated paragraph number, token number, lemma number and MSD number (when respective segments are present).

For overlapping segments (e.g. resulting from different tokenization variants produced by a single annotation tool) additional substructure of the given part of the identifier is expressed with a minus-sign, so that e.g. `t1-seg2.3.4-5-6.7.8` identifier marks the segment representing eighth morphosyntactic interpretation attached to the seventh lemma corresponding to the sixth segment of a fifth variant of a fourth token coming from the third sentence in a second paragraph from the first segmentation-level annotation variant. When only one tokenization variant is present, the substructure is collapsed to a single token number. Similarly, when a token variant consist of a single segment, the third part of the token identifier string is omitted and it takes form of a token number-token variant compound.

References to corresponding elements are stored in `corresp` attributes of `<seg>`ments and take form of XML element identifiers (in contrast to NKJP, no string ranges are used to simplify usage of the

Segmentation

The segmentation model combines both the text structure and paragraph-, sentence- and token-level descriptions into a single layer. This differs from the NKJP approach, but seems to follow the assumption[10] that the data being annotated is never acquired as plain text so mimicking its "original" form is of little value. The segmentation layer can have varied, application-dependent complexity (with just text as paragraph or sentence content when no further annotation is needed – or with the full paragraph-sentence-token set). Storing original text fragments directly in segmentation layer offers major benefits for performance of further processing, based entirely on XML identifiers and never on string ranges. The only drawback is possible duplication of the source text when different variants of segmentation[11] are needed. This seems unusual for paragraph- and sentence-level segmentation and rare for tokenization, but the case of comparing different tokenization variants is difficult even with string-range references to any running text. In most cases the segmentation level would be a single reference layer for further annotation levels.

Paragraphs are adopted from the source text or an artificial single paragraph for shorter or unstructured texts is introduced. To maintain consistency and simplicity, paragraphs are used instead of anonymous blocks which could probably be more appropriate for this purpose, especially in cases when the complete analyzed text corresponds to a unit lower than sentence[12]. When paragraph-level annotation is sufficient, `<p>` tags can contain plain text.

Sentences are represented with an `<s>` tag nested directly in paragraphs. Similarly to paragraph-level segmentation, sentences can contain token-level annotation or plain text when it is requested to split text into sentences without performing any further analysis. When the text being analyzed does not constitute a complete sentence, but tokenization is needed, sentence tag is omitted.

Individual tokens are represented by `<seg>`ments of `token` type. Variant tokens resulting in overlapping text are stored in `<choice>` elements with variant `<seg>`ment subelements[13]. A token which was attached to the previous token (by means of not being separated with whitespace) is carrying a `subtype` attribute with `attached` value. In the following example it is used in two roles: for punctuation and subsequent (here: last) segment of a compound token.

---

[10] See discussion in Chapter 4 – *Text Structure* of (Przepiórkowski, Bański 2009b).

[11] This refers to variants produced by different tools and stored in separate variant layers of a segmentation level and not to variants of segmentation suggested by a single tool, which can be easily represented with `<choice>` tags; see tokenization description further in this section.

[12] The definition of `<ab>` element from the Elements chapter of the TEI Guidelines comes with the following explanation: an anonymous block contains any arbitrary component-level unit of text, acting as an anonymous container for phrase or inter level elements analogous to, but without the semantic baggage of, a paragraph.

[13] The semantically neutral bracket `<seg>`ment carrying `s1-seg1.1.1-2` identifier (represented in NKJP as `<nkjp:paren>`) became here a fully-fledged content unit (a *meta-token*) which can be referenced in further annotation layers.

```
<TEI>
  <!- header information for the layer ->
  <text>
    <body>
      <p xml:id="s1-p1">
        <!- Ważny list? ->
        <s xml:id-"s1-s1.1">
          <seg xml:id="s1-seg1.1.1" type="token">Ważny</seg>
          <seg xml:id="s1-seg1.1.2" type="token">list</seg>
          <seg xml:id="s1-seg1.1.3" type="token" subtype="attached">?</seg>
        </s>

        <!- Tu go miałem. ->
        <s xml:id-"s1-s1.2">
          <seg xml:id="s1-seg1.2.4" type="token">Tu</seg>
          <seg xml:id="s1-seg1.2.5" type="token">go</seg>

          <choice>
            <seg xml:id="s1-seg1.2.6-1" type="token">miałem</seg>
            <seg xml:id="s1-seg1.2.6-2">
              <seg xml:id="s1-seg1.2.6-2-1" type="token">miał</seg>
              <seg xml:id="s1-seg1.2.6-2-2" type="token" subtype="attached">em</seg>
            </seg>
          </choice>
          <seg xml:id="s1-seg1.2.7" type="token" subtype="attached">.</seg>
        </s>
      </p>
    </body>
  </text>
</TEI>
```

Lemmatization

Segments in lemmatization layer refer to segmentation layer and contain definitions of feature structures which in turn store interpretations as individual features. Value of each feature is a string preserving lemma for a given token; alternative values are grouped in value alternation elements (<vAlt>).

```
<TEI>
  <!- header information for the layer ->
  <text>
    <body>
      <p xml:id="l1-p1" corresp="p1-p1">
        <!- Ważny ->
        <seg xml:id="l1-seg1" corresp="t1-seg1">
          <fs type="lex">
            <f name="base">
```

```
              <string xml:id="l1-string1">ważny</string>
            </f>
          </fs>
        </seg>
        <!- list ->
        <seg xml:id="l1-seg2" corresp="t1-seg2">
          <fs type="lex">
            <f name="base">
              <vAlt>
                <string xml:id="l1-string2">list</string>
                <string xml:id="l1-string3">lista</string>
              </vAlt>
            </f>
          </fs>
        </seg>
      </p>
    </body>
  </text>
</TEI>
```

Presence of lemmatization-level annotation implies existence of tokenization layer.

Morphological analysis

Segments in morphological layer also refer to segmentation layer with individual features indicating respective lemmata (feature values storing them). Multiple analyses associated with a single lemma are linked by repeating its identifier in the reference attribute.

`morph` type feature structure serves as a wrapper to lexical interpretations; each of them contains reference to the base form existing in lemmatization layer (using `fval` pointer), POS tag information and morphosyntactic tag(s). Feature values are represented as symbols rather than strings since they come from a definite set. Such notation can be used for tools with arbitrary tagsets.

```
<TEI>
  <!- header information for the layer ->
  <text>
    <body>
      <p xml:id="l1-p1" corresp="p1-p1">
        <!- Ważny ->
        <seg xml:id="m1-seg1" corresp="t1-seg1">
          <fs type="morph">
            <f name="interps">
              <fs type="lex">
                <f name="base" fval="l1-string1"/>
                <f name="ctag">
                  <symbol value="adj"/>
                </f>
                <f name="msd">
                  <vAlt>
          <symbol xml:id="m1-symbol1" value="sg:nom:m1:pos"/>
          <symbol xml:id="m1-symbol2" value="sg:nom:m2:pos"/>
          <symbol xml:id="m1-symbol3" value="sg:nom:m3:pos"/>
          <symbol xml:id="m1-symbol4" value="sg:acc:m3:pos"/>
         </vAlt>
        </f>
       </fs>
      <f>
     </fs>
    </seg>
    <!- list ->
    <seg xml:id="m1-seg2" corresp="t1-seg2">
     <fs type="morph">
      <f name="interps">
       <vAlt>
        <fs type="lex">
         <f name="base" fval="l1-string2"/>
         <f name="ctag">
          <symbol value="subst"/>
         </f>
         <f name="msd">
          <symbol xml:id="m1-symbol5" value="sg:nom:m3"/>
          <symbol xml:id="m1-symbol6" value="sg:acc:m3"/>
         </f>
        </fs>
        <fs type="lex">
         <f name="base" fval="l1-string3"/>
         <f name="ctag">
          <symbol value="subst"/>
          </f>
          <f name="msd">
           <symbol xml:id="m1-symbol7" value="pl:gen:f"/>
          </f>
         </fs>
        </vAlt>
       </f>
      </fs>
     </seg>
   </p>
  </body>
 </text>
</TEI>
```

Inclusion of morphological analysis-level annotation in the package requires presence of lemmatization and tokenization layers.

Note: in NKJP lemmatization and morphological layers are combined and lemma information is used directly in place of the base feature reference. Such description can be easily merged from the separated layers taking morphological layer as a start.

Tagging

The process of tagging results in selection of a single lemma and a single morphosyntactic description for each token. This produces a new annotation layer referring to morphological level (which, in consequence, results in selection of corresponding lemma and token variant, if alternations are present).

In case of multiple tokenization variants, only one is selected which is reflected in the presence of a single `<seg>`ment for a given fragment of the source text. Similarly, when multiple morphological analyses were present, only one is referred to from respective segment.

In contrast to NKJP where disambiguation information is a complex structure, listing dates and responsibilities, here it is stored in a single feature value since it is assumed that the disambiguation process is performed by a single tool referred to in the header.

```
<TEI>
  <!- header information for the layer ->
  <text>
    <body>
      <p xml:id="g1-p1" corresp="p1-p1">
        <!- Ważny ->
        <seg xml:id="g1-seg1" corresp="t1-seg1">
          <fs type="morph">
            <f name="disamb" fVal="m1-string1"/>
          </fs>
        </seg>
        <!- list ->
        <seg xml:id="g1-seg2" corresp="t1-seg2">
          <fs type="morph">
            <f name="disamb" fVal="m1-string2"/>
          </fs>
        </seg>
        <!- ? ->
        <seg xml:id="g1-seg3" corresp="t1-seg3">
          <fs type="morph">
            <f name="disamb" fVal="m1-string4"/>
          </fs>
        </seg>
      </p>
    </text>
  </body>
</TEI>
```

The result of tagging will always contain tokenization and morphological layers.

Syntactic words

Following Spejd approach (Buczyński 2007) each node of a parsing tree may be:

- a segment – a word with its morphosyntactic interpretation or simply a reference to an entity in morphological analysis layer,
- a syntactic word – a non-empty sequence of tokens and/or constituent syntactic words (which amounts to a non-empty sequence of tokens); such sequences of tokens, from the syntactic point of view, behave as single words and they may contain just a single simple word which stands for its own or many words that taken together may form complex named entities such as "United States of America",
- a token – a syntactic word or a segment,
- a syntactic group – non-empty sequence of tokens and/or syntactic groups.

Syntactic word layer is intended to help with syntactic parsing; since the process of morphological analysis (and, in turn, tagging) could result in identification of segments shorter than space-to-space words (such as "*biało*": en. *white* and "-" and "*czerwony*": en. *red* for a compound adjective "*biało-czerwony*": en. *white-and-red*), construction of parsing rules would be much more complex than with "traditional" understanding of segments.

```
<TEI>
  <!- header information for the layer ->
  <text xml:id="c-text1">
    <body>
      <!- a sequence of paragraphs, corresponding to the source layer ->
      <p xml:id="c1-p1" corresp="s1-p1.1">
        <!- biało-czerwony ->
        <seg xml:id="c1-seg1">
          <!- biało ->
          <ptr type="nonhead" target="t1-seg1.1.1" />
          <!- - ->
          <ptr type="nonhead" target="t1-seg1.1.2" />
          <!- czerwony ->
          <ptr type="head" target="t1-seg1.1.3" />
        </seg>
      </p>
    </body>
  </text>
</TEI>
```

Syntactic words can be referenced from the shallow parsing (syntactic group) layer.

Shallow parsing

Syntactic groups represented in the annotation model contain pointers (`<ptr>`s) to immediate constituents of the group – syntactic words specified in the preceding layer or other syntactic groups of the same layer, irrespective of their continuity. `<ptr>` elements may specify the type of the constituency relation (`head` or `nonhead`).

```
<TEI>
  <!- header information for the layer ->
  <text>
    <body>
      <p xml:id="c1-p1" corresp="p1-p1">
        <!- ważny list ->
        <seg xml:id="c1-seg1">
          <ptr type="head" target="t1-seg1"/>
          <ptr type="nonhead" target="t1-seg2"/>
        </seg>
      </p>
    </body>
  </text>
</TEI>
```

Deep parsing

Output format of the deep parsing tool is a shared parse forest, i.e. a collection of parse subtrees stored in a packed graph format, with each unique subtree stored only once. This proprietary representation has been adopted and included into the packaged format by means of standard `<seg>`ments carrying feature structure descriptions:

- correspondence between the forest root (output as segment) and the matching sentence is maintained with `corresp` attribute attached to the root `<seg>`ment or `forest` type,
- source text and all statistics are discarded,
- all `<node>`s are converted into generic `<seg>`ments of the `type` set to `nonterminal` or `terminal` respectively, basing on the name of the first subelement of the node,
- non-terminal categories become `subtypes` of `<node>`s,
- node identifiers (`nid` attributes) are incremented by 1 (to conform to the standard numbering model with first element number starting with 1 and not 0) and concatenated with the identifier of a forest segment to form unique identifiers even when multiple parsing trees are stored in one file; the compound identifiers are then stored in `xml:id`s of node `<seg>`ments,
- `from` and `to` attributes of element nodes are converted into `<f>`eatures of an anonymous feature structure (`<fs>`); the content of each `from`/`to` `<f>`eature is a `<symbol>` with text contents corresponding to the original value of `from`/`to` attribute,
- the number of subtrees (`subtrees` attribute of a `<node>`) and `chosen` attribute are discarded,
- the contents of the feature structure is supplemented with `<f>`eatures `named` after original types of source `<f>` elements and feature values stored in a `<symbol>` subelement with its text contents copied from original content of `<f>`,
- `<children>` elements are converted into `<seg>`ments with `children` type and `subtype` attribute filled with the value of `rule` attribute; `chosen` attribute is discarded,
- each `<child>` subelement of `<children>` node is converted into a `<ptr>` element (as in shallow parse layer) with the `type` attribute set to `head` string when `head` attribute was present and to `nonhead` when it was absent; `from` and `to` attributes are discarded,
- `nid` attribute of each `<child>` is converted into `target` attribute of the `<ptr>` element (since they both correspond to child node identifier),

- for terminal nodes `<orth>`, `<base>` and `<f type="tag">` elements are discarded along with `disamb` and `nps` attributes of the original `<terminal>` element; the value of `interp_id` attribute is stored in `fVal` attribute of a `<f name="msd-ref">` element in the terminal segment; `token_id` is discarded since the interpretation identifies the token explicitly.

```
<TEI>
  <!- header information for the layer ->
  <text>
    <body>
      <p>
        <seg type="forest" xml:id="d1-seg2" corresp="s1-seg2">
          <seg xml:id="d1-seg2.1" type="nonterminal" subtype="wypowiedzenie">
            <fs>
              <f name="from"><symbol>0</symbol></f>
              <f name="to">  <symbol>2</symbol></f>
            </fs>
            <seg type="children" subtype="w">
              <ptr type="head" target="d1-seg2.2"/>
              <ptr type="nonhead" target="d1-seg2.7"/>
            </seg>
          </seg>
          <seg xml:id="d1-seg2.7" type="nonterminal" subtype="znakkonca">
            <fs>
              <f name="from"><symbol>1</symbol></f>
              <f name="to">  <symbol>2</symbol></f>
              <f name="dest"><symbol value="neut"/></f>
            </fs>
            <seg type="children" subtype="int3">
              <ptr type="head" target="d1-seg2.8"/>
            </seg>
          </seg>
          <seg xml:id="d1-seg2.8" type="terminal">
            <fs>
              <f name="from"><symbol>1</symbol></f>
              <f name="to">  <symbol>2</symbol></f>
              <f name="msd-ref" fVal="m1-symbol5"/>
            </fs>
          </seg>
        </seg>
      </p>
    </body>
  </text>
</TEI>
```

2.3 Integrated tools

Offline versions of all integrated tools have been used by the linguistic community in Poland for several years and they proved their suitability and efficiency for linguistic engineering tasks. They constitute the basic building blocks of many local processing chains, but have never been made available online in a consistent manner[14] (in a common infrastructure and format). In this respect, bringing the tools online can go beyond just illustrating capabilities of the framework and the format and constitute a real value to the community and other interested parties. At the same time the framework can be the solid basis for integration of further tools, services, their new variants or levels of analysis. Until now all integrated tools are open source and all are actively maintained and developed.

Morfeusz

Morfeusz (Woliński 2006) is a morphological analyzer for Polish. It uses a positional tags starting with POS information followed by values of morphosyntactic categories corresponding to the given part of speech (Przepiórkowski and Woliński, 2003a). Current version of the tool, Morfeusz SGJP, is based on linguistic data coming from The Grammatical Dictionary of Polish (Saloni et al., 2007). The tool is available at http://sgjp.pl/morfeusz/index.html and is distributed under the terms of the GNU AGPL 3.

TaKIPI

TaKIPI (Piasecki and Wardyński, 2006) is a hybrid (multiclassifier) rule-based morphosyntactic tagger (disambiguator of morphological descriptions) of Polish. Plain text processing is based on Morfeusz's 2.1 results which assign a set of possible tags to each token. Then both hand-written rules encoded in JOSKIPI language and rules automatically acquired from a corpus are being used in disambiguation process. The achieved error rate is 6.56%.

TaKIPI has been successfully used in the process of automatic tagging of the current version of the IPI PAN corpus (Przepiórkowski, 2004) using IPI PAN tagset of Polish (Przepiórkowski, 2005; Przepiórkowski and Woliński 2003a) with around 1000 tags. TaKIPI is licensed under GNU GPL 3 and is also available as a separate Web service at http://plwordnet.pwr.wroc.pl/clarin/ws/takipi/takipi.wsdl.

Pantera

Pantera (Acedański and Gołuchowski 2009; Acedański 2010) is a recently developed morphosyntactic rule-based Brill tagger of Polish. It uses an optimized version of Brill's algorithm adapted for specifics of inflectional languages. The tagging is performed in two steps, with a smaller set of morphosyntactic categories disambiguated in the first run (part of speech, case, person) and the remaining ones in the second run. Due to free word order nature of Polish the original set of rule templates as proposed by Brill has been extended to cover larger contexts. The achieved error rate amounts to 10.8%, but the tagger is currently under active development.

[14] Demo online version of the older variant of the morphological analyser, Morfeusz SiAT, is still available at http://sgjp.pl/demo/morfeusz; TaKIPI is also wrapped as a separate Web service which can be tested at http://nlp.pwr.wroc.pl/clarin/ws/takipi/.

Spejd

Spejd (Przepiórkowski and Buczyński, 2007; Przepiórkowski, 2008) is an engine for shallow parsing using cascade grammars, able to co-operate with TaKIPI for tokenization, segmentation, lemmatization and morphologic analysis.

Parsing rules are defined using cascade regular grammars which match against orthographic forms or morphological interpretations of particular words. Spejd's specification language is used, which supports a variety of actions to perform on the matching fragments: accepting and rejecting morphological interpretations, agreement of entire tags or particular grammatical categories, grouping (syntactic and semantic head may be specified independently). Users may provide custom rules or may use one of the provided sample rule sets.

Spejd is also available as a separate online service at http://chopin.ipipan.waw.pl:8081/spejdws/services/SpejdService?wsdl.

Świgra

Świgra is a deep parser of Polish implementing Marek Świdzinski's formal metamorphosis grammar of Polish (Świdziński 1992), regarded as the largest and most precise formal description of general grammar of Polish. The parser has been implemented in Prolog by Marcin Wolinski within his doctoral dissertation (Woliński 2004). More information on Swigra can be found at http://nlp.ipipan.waw.pl/~wolinski/swigra/.

2.4 Architecture of the online service

To be able to process large amounts of text, requests sent to the Web service are handled in asynchronous manner. Invoking one of the available methods results in returning the request token (identifier) which can be used to check the request status and retrieve the result when processing completes. This design is directly inspired by TaKIPI Web service (Broda et al., 2010) prepared by ICS PAS and WROCUT.

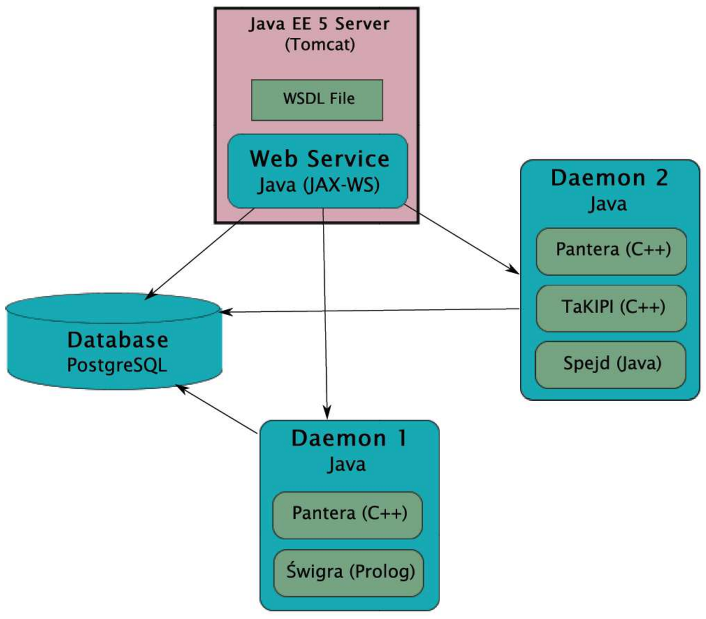


Fig. 2.1. Architecture of the Multiservice

Request manipulation

Following request manipulating methods are available:

- `analyzeChain(text, requestParts, inputFormat, outputFormat)` – enqueue a request to perform given list of operations (`requestParts`) on the given text. Each chain part consists of operation type (that is linguistic function such as tagging or shallow parsing), requested service (internal tool) name (e.g. "TaKIPI", "Pantera", "Spejd") and a map of properties specific to the provided tool. In this way one may request performing several operations at once (e.g. „tag text with Pantera, then perform shallow parsing with Spejd, possibly using tagging information provided by Pantera"). Returns a unique token of the request.
- `getStatus(token)` – returns a status of request with the given token (currently may be one of: `PENDING`, `IN_PROGRESS`, `DONE` or `FAILED`).
- `getResult(token)` – if the status of request is `DONE`, returns request result. If the request is `FAILED` returns an error message.

Removal of requests is currently not possible.

Error handling

Invalid parameters

When invoked with incorrect parameters, e.g.

- a non-existing operation type or service name is provided to `analyzeChain` method,
- `getStatus` or `getResult` is executed with invalid token,

each of the above-mentioned methods will return a SOAP message with fault info instead of regular output. When using SOAP tools like JAX-WS or Python-based Zolera Soap Infrastructure the message will be mapped to an exception.

Errors when performing request processing

Whenever (possibly correct) request fails at runtime, its status is set to `FAILED` and getting the result will return the failure details. Moreover some failures may indicate that a service is corrupted. In such cases not only the request status is set to `FAILED`, but also a daemon that was managing the failed tool is terminated.

The Web service

The Web service is the only part directly seen by the client. Whenever the `analyzeChain` function is invoked it stores request data in the database, queries for a daemon that is able to process the whole chain of operations (daemons that are not currently working are preferred) and sends it the request identifier. `getStatus` and `getResult` functions simply query the database for information about the request. The service communicates with SOAP messages and is implemented in Java using JAX-WS technology.

Daemons

Each daemon is a Java application listening on specified address and port retrieving request identifiers to process (sent by Web service) from the server socket. A daemon may integrate multiple internal tools and invoke them to process request chains. When finished, it saves the result in the database. Obviously, multiple daemons with different configurations may be run.

Database backend

The database is responsible for exchanging data between Web service and daemons.

The most important database tables are:

- `ServiceInfo` – contains information about internal tools (services) and operation names it provides,
- `DaemonInfo` – daemons with their network locations (for communication with Web service) and information about services they provide,
- `Request` – contains text, status, result and list of request parts,
- `RequestPart` – a linguistic function (e.g. shallow parsing) and service (tool) name (e.g. “Spejd”) to perform it.

Currently an underlying DBMS is PostgreSQL and it is accessed by daemons and the Multiservice with Java Persistence API 2.0 (using EclipseLink library).

2.5 Interface and usage

The Multiservice is intended to be used via a dedicated API. Additionally, a very simple interface has been prepared to let users test the service online. An example of method execution in Python has also been provided and initial notes on recent experimental integration of the service with WebLicht framework.

Input and output formats

Output of all functions of the Multiservice is always TEI P5. Input can be TEI P5 or plain text – which implies that for higher-order actions (such as deep parsing which requires precise morphosyntactic information) the segmentation tools, lemmatizer and morphological analyser will be run automatically. The annotation tools can accumulate output in the result file so that further steps of processing can build on the contents of the file. On the contrary, independent layers (e.g. shallow and deep parsing) can also be chained since no assumption is being made on the contents on the file after it has been processed by any of the service methods. As annotation tools can equally well add or remove data, this feature is planned to be used to filter the contents of e.g. lemmatization and morphosyntactic layers according to tagging results which could substantially simplify deep parsing.

Service execution example

The outline of service execution is the following:

- the user sends to the service a processing request with the linguistic function name and its parameters,
- the service generates a token for the request (further used to operate on the given request with the service), stores the request in the queue and returns the token to the user,
- the user keeps querying the service about the status of execution of a request identified with a given token until the status shows that the execution stopped because of error or ended successfully,
- if execution failed, the user retrieves an error message from the service and execution stops,
- if execution succeeded, the user retrieves the result from the service and execution stops.

The code excerpt below illustrates the execution process:

```
addr = ’http://chopin.ipipan.waw.pl:8083/WebService/ClarinWS?wsdl’
loc = ClarinWSServiceLocator()
port = loc.getClarinWS(addr)
req = analyzeChain()
req._text = ’Ala ma kota.’
req._parts = req.new_parts()
part1 = req._parts.new_part()
part1._operationType = ’Tagging’
part1._serviceName = ’pantera’
part2 = req._parts.new_part()
part2._operationType = ’Shallow parsing’
part2._serviceName = ’spejd’
req._parts._part = [part1, part2]
req._inputFormat = ’TEXT’
req._outputFormat = ’TEI’
token = port.analyzeChain(req)._token
status = None
while not status in [’DONE’, ’FAILED’]:
req = getStatus()
req._token = token
status = port.getStatus(req)._status
req = getResult()
req._token = token
res = port.getResult(req)._result
print status
print res
```

Building custom applications

Connecting a new linguistic tool to the framework requires:

- implementing a simple Java interface RequestExecutor with “execute” method invoking the tool for a given text (plain text format or XML TEI),
- adding the implementation to Daemons classpath (by simply copying the jar file to specified directory),
- adding configuration XML to let the Daemon recognize the new service.

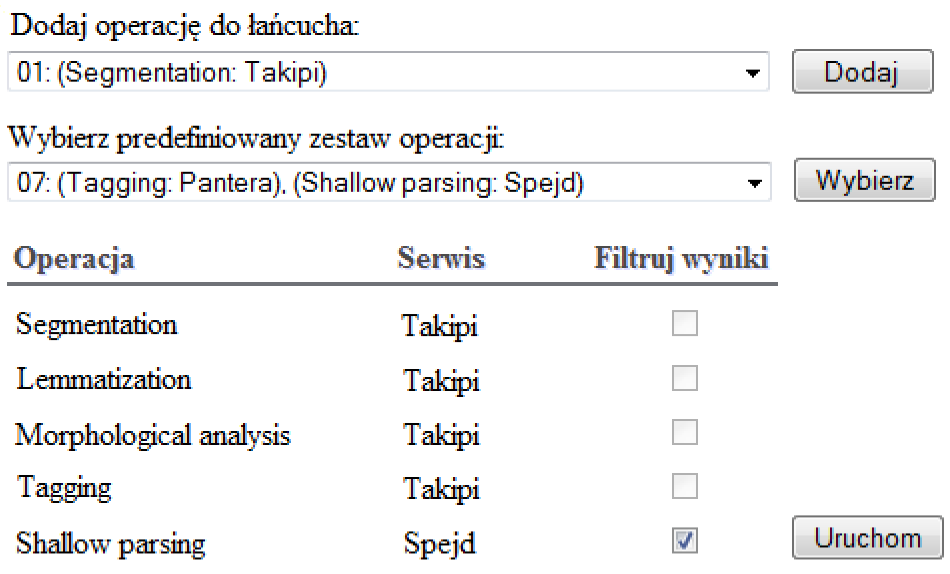
There is no need to extend or recompile any existing sources.

To facilitate integration several useful utilities are provided:

- `ExternalInvoker` – a simple, text-based program for testing invocation of external processes and their communication with the framework (via standard input and output) to help easily integrate tools written in languages other than Java into our infrastructure and run them as background daemon processes,
- parser of Packaged TEI P5-based documents – to convert TEI-encoded morphological analysis and tagging data into a tree of Java objects,
- XML utilities to easily add new layers and headers to existing TEI document (based on Saxon library).

Web interface prototype

A simple interface (see Fig. 2.2) has been prepared to test execution of the service from a Web page. Apart from the text-entry area it contains two chain-creating drop-downs – the first one (*Dodaj* *operację do łańcucha*) adds selected operation to the chain while the second one (*Wybierz predefiniowany zestaw operacji*) creates the complete chain using a predefined set of atomic operations. The checkboxes (*Filtruj wyniki*) allow for filtering results basing on disambiguation component which may significantly reduce input directed at further steps of the chain (e.g. deep parsing, otherwise resulting with surfeit of interpretations). After the text has been submitted (*Uruchom*) the results are displayed inline and a link to the result XML to facilitate its download is output.



Dodaj operację do łańcucha:

01: (Segmentation: Takipi) — Dodaj

Wybierz predefiniowany zestaw operacji:

07: (Tagging: Pantera), (Shallow parsing: Spejd) — Wybierz

Operacja	Serwis	Filtruj wyniki
Segmentation	Takipi	☐
Lemmatization	Takipi	☐
Morphological analysis	Takipi	☐
Tagging	Takipi	☐
Shallow parsing	Spejd	☑

Uruchom

Fig. 2.2. Multiservice prototype interface.

The asynchronous mode of chain execution is reflected in the interface: after starting the analysis, the Web application checks periodically (every 0.5 second) the status of the request. When it ended execution, the result is retrieved and displayed to the user. In case of a failure, appropriate error message is presented.

Plugging the multiservice into the WebLicht framework

WebLicht (Hinrichs et al., 2010) – see Sec. 1 – is a framework for managing execution of linguistic Web services in a chained manner. It features scalable architecture and consists of the registry of services, an interface for running the chains and several additional modules for e.g. finding a chaining path for a given input and output or visualization of annotation. At the moment there are around 120 external services available for more than 10 languages. Natively WebLicht-registered services use its proprietary TCF (Heid et al., 2008) format (standing for Text Corpus Format), but using other formats had also been described as feasible.

The integration task intended to test whether usage of a different format was possible and whether the tools using such foreign format could be used by the generic WebLicht interface. Following the integration guide Pantera, Spejd and Świgra methods of ICS PAS Multiservice have been successfully registered into the WebLicht infrastructure with each method serving as a "virtual service”. This way the Multiservice became a basis of 3 virtual services available for chaining[15]. Testing the WebLicht interface showed that it also seamlessly integrates with the foreign format and allows for execution of individual methods and chains easily.

Integration method

Invoking the Multiservice requires an additional step since WebLicht requires that subservices provide synchronous online access while the Multiservice is called asynchronously (user does not hang waiting for a request to complete, but gets a “ticket” with unique request token and then asks the service if it is completed). In order to make both execution methods compatible, a special RESTful Web service, written in Java, using JAX-RS is running alongside the Multiservice. When the user (possibly WebLicht) invokes it, the REST service places the request in database (like JAX-WS version of Multiservice would), checks cyclically for the request to complete, writes the result in HTTP response and finally closes the connection (just like any other WebLicht subservice).

TCF-aware services

Obviously, not all visualization tools (such as parsing tree viewer) could be used directly with the TEI format since they are highly TCF-dependent. In order to seamlessly plug the Multiservice into the WebLicht interface, TCF was adopted as a variant input/output format of the component services. Their sample execution method of is as follows:

```
wget --post-data="text" --header="Content-Type: text/plain; charset=encoding"
http://chopin.ipipan.waw.pl:8083/WebService/resources/Clarin/
Pantera?outputFormat=TCF --output-document=tcf.xml

wget --post-file=tcf.xml --header="Content-Type: text/xml"
http://chopin.ipipan.waw.pl:8083/WebService/resources/Clarin/
Spejd?filterTagging=true&inputFormat=TCF&outputFormat=TEI"
--output-document=tcf2spejd.xml
```

Integration notes

Difference in approaches of the application of TEI for Polish and current version of TCF resulted in the set of decisions while creating the converter. The following properties are not preserved:

- multiple interpretations of particular layer (e.g. results of two taggers over the same text),
- ambiguities in tokenization (see example from Sec. 2.2 with `<choice>` elements),
- variants of morphological analyses referring to different lemmas,
- group categories (“nominal group”, “adjectival group” etc.), syntactic words base forms and modified morphological analyses provided by a parser (resulting e.g. from using a different tagset: e.g. Spejd’s “subst” is equivalent to “Noun” in NKJP).

Distinction between ICS PAS TEI syntactic words and groups in TCF is described by @cat node attribute:

```
<constituent cat="S" ID="c_16">
  <constituent cat="Word" ID="c_3">
    <constituent cat="Token" ID="c_2">
```

[15] As shallow parsing (Spejd) and deep parsing (Świgra) processes are independent, following chains can currently be build: Pantera → Spejd, Pantera → Świgra, Pantera → Spejd → Świgra and Pantera → Świgra → Spejd.

```
          <tokenRef tokID="t_2"></tokenRef>
        </constituent>
      </constituent>
      <constituent cat="Group" ID="c_8">
        <constituent cat="Word" ID="c_5">
          <constituent cat="Token" ID="c_4">
            <tokenRef tokID="t_3"></tokenRef>
          </constituent>
        </constituent>
        <constituent cat="Word" ID="c_7">
          <constituent cat="Token" ID="c_6">
            <tokenRef tokID="t_4"></tokenRef>
          </constituent>
        </constituent>
      </constituent>
    </constituent>
```

The TCF format is constantly being reviewed and improved. Some of the omissions mentioned above are currently under review and may be available in a future version of TCF.

2.6 Notes on performance and scalability

Processing huge volumes of data proves to cause little trouble for the described infrastructure. Naturally, the Web service wrapper, XML post-processing and database operations add to the inevitable execution overhead, but the figures still leave room for improvement.

Comments on initialization, parsing and storage of intermediate results

Service plugin implementation class does not require the tool being integrated to run as a background daemon (in comparison to starting it for each request separately). However, all existing tools are initialized only once – at the daemon startup. New services should follow this good practice for obvious performance reasons.

Intermediate results are currently parsed using Saxon library. It appears to be too time-consuming for large data so switching to StAX-based parser is currently being considered.

The major challenge for linguistic services is also storing potentially huge amounts of data resulting from processing of rather small input. It is not uncommon e.g. for deep parsing systems which could generate plenty of different result trees even though their subtrees are identical among subsequent results. In case of the Multiservice the problem was diminished by using a shared parse forest representation with each unique subtree stored no more than in one instance.

Performance test results

A simple performance test has been carried out for one of the component services. It shows overhead caused by using the Web service when performing a simple request of tagging n-word text with Pantera tagger (against the execution of the offline tool alone). Results in Table 2.1 show durations of request execution through Web service on the local host for the most time-consuming operations.

Number of words	Total request time	Pantera execution time	XML post-processing	Database operations
1 000	3.2	1.0	1.2	0.1
2 000	4.5	1.4	2.1	0.6
5 000	9.2	3.7	3.3	1.4
10 000	14.9	7.0	4.8	2.7
20 000	32.6	14.7	10.3	5.7
30 000	48.3	22.4	15.7	8.1
40 000	98.5	29.7	45.3	22.1

Table 2.1. Results for performing one tagging request using Pantera.

It clearly shows that total request execution is 2-3 times longer than invoking Pantera locally. Even though for shorter texts such as a typical newspaper article (of approx. 1 000 tokens) the difference can still seem acceptable, various optimizations should be considered. Improvements in XML postprocessing (inclusion of headers, pretty-printing of document generated by C++ application with Pantera) should give the most significant performance boost.

It should also be noted that processing large documents is quite memory-consuming and using swap influences the overall tagging time. Currently the whole XML document tree (decomposed from e.g. 87 MB of TEI XML for a 40 000-word plain text source) is kept in memory by Saxon XSLT library to perform postprocessing. This common problem can be resolved by switching to StAX which could improve performance considerably when dealing with large data – at the cost of the coding time and possibly more error-prone implementation.

2.7 Towards semantic interoperability[16]

To review semantic interoperability issues related to Polish morphosyntax, the National Corpus of Polish (NKJP) tagset was defined in the ISOcat Data Category Registry. This sections attempts to comment on the experience of using this system and suggesting ways in which it could be improved.

NKJP tagset

The NKJP tagset (Przepiórkowski 2009b, see http://nkjp.pl/poliqarp/help/ense2.html for a concise version) is a slightly modified version of the IPI PAN tagset (Przepiórkowski and Woliński 2003b), a *de facto* standard tagset for Polish. There are 36 grammatical classes approximately corresponding to parts of speech, 13 grammatical categories and their possible values (36 in total). Each grammatical class has an associated list of appropriate grammatical categories, which may be specified as obligatory or optional for the particular class. Furthermore, there are a number of constraints on the possible values of categories appropriate for some classes.

[16] This section is based on (Patejuk and Przepiórkowski 2010).

Most grammatical classes have a list of categories for which they inflect or whose value is specified lexically. Gerunds (`ger`) inflect for number, case and negation while their aspect and gender (always neuter) are lexical. The complete morphosyntactic tag for 'piciem', `ger:sg:inst:n:imperf:aff`, provides the following information: it is a gerund whose values of the categories of number, case, gender, aspect and negation are singular, instrumental, neuter, imperfective and affirmative respectively. Certain categories may be optional – while some prepositions, like 'do' which is always `prep:gen`, have only one form, many others take different forms depending on the context: 'pod' `prep:inst:nwok` as opposed to 'pode' `prep:inst:wok` which have different values of vocalicity. Some classes such as conjunctions (`conj`) or predicatives (`pred`) are non-inflecting and, having no associated categories, their complete tags consist only of grammatical class tags: 'i' (`conj`), 'to' (`pred`). Finally, there are classes such as abbreviation (`brev`), bound word (`burk`) and unknown form (`ign`) which, rather than being traditionally understood parts of speech, serve technical purposes.

Alongside widely known traditional grammatical categories such as case (7 values), number, person, gender (5 values), degree, aspect, there are categories such as negation and accentability as well as some classes rather specific to Polish. These include accommodability which determines the syntactic behaviour of numerals, postprepositionality describing the behaviour of certain pronouns in relation to prepositions, agglutination optionally applicable to one class of verbs and vocalicity which regulates the distribution of agglutinates. The remaining category, fullstoppedness, is a technical category taking one of two values depending on whether the abbreviation segment has to be followed by a full stop.

ISOcat and its architecture

The ISOcat project is an implementation of the ISO 12620 standard which is described as follows in the abstract available on the ISO website (http://www.iso.org):

> ISO 12620:2009 provides guidelines concerning constraints related to the implementation of a Data Category Registry (DCR) applicable to all types of language resources, for example, terminological, lexicographical, corpus-based, machine translation, etc. It specifies mechanisms for creating, selecting and maintaining data categories, as well as an interchange format for representing them.

The architecture of ISOcat is therefore determined by the requirements set by the ISO 12620 standard: the DCR model consists of administrative information, descriptive information and linguistic information. These components of the DCR are reflected in the organisation of the data contained in individual Data Categories (DCs), all of which have an Administration Information Section, Description Section and potentially Conceptual Domains whose values are specified independently of the content of each other for different languages.

The Administration Information Section contains the Administration Record which provides information about the mnemonic identifier (Identifier, as opposed to PID, the unique Persistent IDentifier), version, registration status, justification together with its origin as well as the dates of creation and the last change made to the DC.

The Description Section (DS) contains the Language Section which organises information about a given DC according to language. It provides details such as the definition, its source and, optionally, some notes in the Definition Section. There is also the Name Section which specifies the DC names together with their status in the given language and the Example Section where relevant examples together with their sources can be provided. The DS also contains the Data Element Section which is intended as the place for storing language-independent names of the DC.

The Conceptual Domain (CD), an inherent feature of *complex* DCs, contains the information about its possible values in a given language, which in turn depend on the type of the particular DC. There are three types of complex DCs: *closed*, *open* and *constrained*. The CDs of a *complex/closed* DC are represented as finite sets of *simple* DCs which, being the only non-complex DC type, do not have any associated CDs and therefore have no associated values themselves. The two remaining types of *complex* DCs, *complex*/*open* and *complex*/*constrained*, are characterised by the fact that the sets of their values cannot be enumerated exhaustively: the *open* DC is a 'complex data category whose conceptual domain is not restricted to an enumerated set of values' while the *constrained* DC is a 'complex data category whose conceptual domain is non-enumerated, but is restricted to a constraint specified in a schema-specific language or languages' (ISOcat Glossary 2010).

More information about ISOcat can be found on the website of the project (http://www.isocat.org).

Defining the NKJP Tagset in ISOcat

The original idea was to enter into the ISOcat DCR the grammatical classes, grammatical categories and their corresponding values. After having completed this stage, the values would be attached to appropriate grammatical categories and these, subsequently, would be related to appropriate grammatical classes as their attributes. Ideally, the relations between a given grammatical class or category and its possible values or attributes would be expressed in its CD for the particular language. Adopting such a strategy would be preferable not only because of being the most economic one in terms of the amount of time necessary to define the NKJP Tagset as ISOcat DCs but also because it would closely reflect the design and structure of the tagset.

Elegant but impossible

Regrettably, such a solution, even though it would certainly be the most elegant one, could not be implemented because of the architecture of the ISOcat DC types. The DC type which matches best the requirements set by this task is, in most cases, the *complex* one as every complete morphosyntactic NKJP tag consists of a tag signalling the grammatical class followed by tags corresponding to values of appropriate grammatical categories, if there are any. Using the range of DC types offered by ISOcat, the values of grammatical categories were classified as *simple* DCs since, being simple atoms, they have no values themselves. They were subsequently related to corresponding grammatical categories whose DC type was set to *complex*/*closed* because they have well-defined repertoires of enumerable values. Ideally, it would be possible to list in an analogous way all the corresponding grammatical categories in the CD of a given grammatical class as its attributes. This way, both grammatical classes and categories would be classified as *complex*/*closed* DCs while values of grammatical categories would be *simple* DCs – it is here that a serious problem is encountered. While it is possible to provide *simple* DCs as values in the CD of a *complex*/*closed* DC, it is not possible to specify such a *complex*/*closed* DC as one of the attributes of another *complex*/*closed* DC, which would be the case here. The reasons are manifold: non-simple DC types cannot be linked to the CD of a *complex* DC, only *complex*/*closed* DCs can have CDs with enumerated content and, more importantly, there is no support for representing any relations other than that of *Value* in the CD at the moment.

Clever but impossible

Another approach at defining the tagset in ISOcat was based on the idea of entering complete NKJP tags directly into the DCR as *complex*/*open* DC types. A complete morphosyntactic NKJP tag, for instance `subst:sg:nom:m2`, the template for which would be `class:number:case:gender` has the following structure: the first element represents the grammatical class, followed by values of appropriate grammatical categories, if there are any. If the complete tag consists of more elements than the obligatory grammatical class, every segment is separated from the following one with a colon. With 36 grammatical classes, 13 grammatical categories and 36 values in total, there are more than 1500 possible complete tags, which makes the task of creating and entering them manually unfeasible. Due to this fact, the complete tags need to be either generated or extracted from the corpus. The latter solution is given preference because it avoids problems encountered in the case of baseline tag generation which include accounting for restrictions on the values of grammatical categories as well as the optionality of some categories. On the other hand, choosing to extract the complete tags from the corpus, there is the risk of some not being represented due to their absence from the data. Having obtained the complete tags by either method, the original tag separators must be replaced by some other character due to the fact that the use of colons in the DC Identifier is restricted and the system will not accept such DCs. Subsequently, automatic descriptions of DCs would be generated (`subst:sg:nom:m2` = noun, singular, nominative, animate masculine) and, together with corresponding complete morphosyntactic tags as identifiers, fitted into the frame provided by the Data Category Interchange Format (DCIF), the XML export format for DCs grouped into Data Category Selections (DCSs). Finally, the modified DCIF file would be fed into the DCR.

Unfortunately, this solution could not be implemented as DC import is not supported at all at the moment. According to the obtained information, although the DCIF DC import is given priority, there is no set date of its introduction and, more importantly, it is either not going to be publicly available or it is going to be subject to certain restrictions on the allowed data import limit.

Successful but time-consuming

Due to the fact that the implementation of the previous solution was not possible because of technical limitations, another strategy had to be adopted. 36 DCs defining grammatical classes which roughly correspond to parts of speech were created manually. Due to the uniqueness of many solutions adopted in the tagset which include, for instance, a separate class for depreciative forms (`depr`) and two distinct classes of pronouns, it was not possible to use already existing DCs and new ones tailored to the needs of NKJP were created. Definitions of NKJP DCs were written, with minor modifications, on the basis of extracts from publications about the IPI PAN Corpus (mainly, Przepiórkowski 2004) and NKJP with appropriate bibliographic source provided, following ISOcat guidelines.

Since the appropriate grammatical categories could not be represented in the CD of grammatical classes as their attributes, the definition is followed by a line containing detailed information about the grammatical categories associated with the particular grammatical class. In order to make it easier to trace associated grammatical categories, the list is accompanied by corresponding PIDs in plain text since the use of hyperlinks in definitions is not supported. Furthermore, if a category happens to be optional for some class, information about its optionality is also provided in brackets after the corresponding PID. Since grammatical classes are sets of lexemes which are not defined in ISOcat themselves, the DC type of defined grammatical classes was set to *complex*/*open*.

Another alternative

There is an alternative solution which, although it has not been implemented, is worth mentioning as it could improve the results achieved through the application of the previous one. This would, however, come at a considerable cost – grammatical classes would need to be reclassified as *complex*/*closed*, which would distort the ontology modelled in this implementation where grammatical classes are consistently *complex*/*open* DCs whose values are appropriate lexemes. The values of grammatical categories could be related directly, bypassing the level of categories, to appropriate grammatical classes in their CDs. Though the intermediate level of grammatical categories would not be represented in the CD of the given grammatical class, this information would be still available in its justification as well as definition. In this way, the information provided in the description of the Data Category (DC) would complement the specification of grammatical category values in its CD. Furthermore, such a solution would make it possible to account in a straightforward way for most constraints on the values of categories appropriate for classes, with the exception of more complex ones as in the case of imperative (`impt`) where the range of appropriate values of the category of person is restricted by the value of the category of number.

However, there are some serious drawbacks which have to be taken into consideration. These include a great deal of manual work due to the complete lack of support for templates or multiple changes to the DC or even the entire DCS. Moreover, since values in the CD are listed in an alphabetical order, the values of corresponding grammatical categories would not be grouped. Finally, there is no means to account for the optionality of values of certain categories directly in the CD of the given grammatical class – such information could only be retrieved in the definition and justification of the DC.

Technical issues

As it is openly acknowledged on the project website, ISOcat is constantly under development – it is emphasised that the Web Interface (WI) available at the moment is a beta version. As a result of this implementation which required a considerable amount of time spent using ISOcat WI, a few bugs were reported and many more features were requested. Most of the identified bugs were fixed while only some of the suggested functionalities have been introduced. This section recaps the main points concerning the technical side of the defining the NKJP Tagset in ISOcat and brings into focus some issues which require particular attention.

Current status

So far, only two of the requested features have been implemented, the first one being the update of the ISOcat DC search following earlier requests from other users. It is now possible to refine the search results using a variety of parameters such as the matching method, the language of keywords as well as fields, profiles and scopes to be considered. On the one hand the update introduces many more features than requested, but on the other it does not include an important functionality that was suggested – the possibility to use DC search when specifying the values in the CD of a DC. The second update brought the possibility to delete a DCS but not a DC which, being persistent, which is a part of ISOcat policy, cannot be removed once created. Since there is no way to delete a DC, the only solution at the moment is to recycle it – the only element of the DC which cannot be changed is its PID. In the future, however, an alternative in the form of having the possibility to deprecate a DC is going to be made available while at the moment it is only possible to set the name status of the DC to deprecated.

Further plans

There are many vital functionalities which could make the work with ISOcat significantly easier and more efficient but, unfortunately, are not supported at the moment. These include the introduction of basic tools such as DC templates which could be created from scratch or on the basis of a particular DC chosen by the user. A multiple change tool, possibly with regular expression support, making it possible to apply multiple changes at the same time instead of doing it manually would certainly be in place. It could easily be applied to editing the definitions (to change some key term shared by a number of DCs), their source, but also to changing features such as DC name status, DC type or even CD values (if some of them are shared). Multiple change tool would also be particularly useful when changing the scope of chosen DCs or even the entire DCS as currently changing the scope of a DCS does not result in an automatic change of the scope of DCs it contains. At the moment all of the above must be done manually, which is extremely time consuming when handling a greater number of DCs. The next feature request, whose importance is supported by ample evidence presented earlier, is the implementation of the DCIF import which would not only enable automating the management of the DCS to a large extent but it would also provide the first basic alternative to managing DCs via the WI which is currently the only means of accessing ISOcat.

Finally, there are some minor issues which could still have a considerable positive impact on the experience of using ISOcat. The introduction of password change would certainly be appreciated by many, not only for security reasons. It might be a good idea to resign from the obligatory comment required by the WI when saving a new DC, which would make working with ISOcat even more smooth and reduce the number of whitespace comments. Last but not least, the ISOcat WI provides a brilliant platform for work supported by state-of-the-art technology but a faster, more lightweight alternative would certainly be welcome.

Results

In spite of a number of problems encountered during this implementation, the main objectives were achieved – the NKJP Tagset has been successfully defined in the ISOcat DCR and it is now available as a public DCS, `nkjp`, owned by the `NKJP` group which is the first and used to be the only group with a public DCS in the ISOcat DCR. The `nkjp` DCS contains 36 *complex*/*open* DCs corresponding to grammatical classes, 13 *complex*/*closed* DCs defining grammatical categories and 36 *simple* DCs specifying values of these categories. In total, 85 DCs were created, all of which have an associated definition describing its function in the tagset, accompanied by relevant examples from Polish. Due to the technical limitations discussed at length, it was not possible to reproduce the original design of the tagset and alternative solutions had to be adopted.

At a glance, the NKJP Tagset was modelled in the ISOcat DCR in the following way: grammatical classes are *complex*/*open* DCs with appropriate lexemes as their values, grammatical categories are *complex*/*closed* DCs whose CDs are populated with their possible values modelled as *simple* DCs. Due to the fact that, using currently implemented ISOcat solutions, it was not possible to express formally the relation of *Attribute* between grammatical classes and categories, definitions of grammatical classes provide additional information about appropriate categories together with their PIDs, details about their optionality and, if applicable, constraints on their values.

The idea of standardising linguistic concepts is undeniably appealing and ISOcat provides a convenient platform to assist this process. It offers functionalities such as DC checking which support the creation of DCs in accordance with ISO standards and gives the unique possibility to submit DCs for standardisation. Though the experience of defining the NKJP Tagset in the ISOcat DCR suggests that there is still some room for improvement, the current implementation of ISOcat was flexible enough to allow a successful realisation of this task – the first public ISOcat definition of any complete tagset, for any language.

3 Standards in practice: IMCS services

Several NLP tools for Latvian, a a highly inflective Baltic language, have been made available via experimental webservices (http://valoda.ailab.lv/ws/)[17]. Some of them recently have been standardized according to the ISO family of standards, namely, a tokenizer and sentence splitter, a POS tagger, and a morphological analyzer. The first three are intended for use in a webservice chain; the latter one can be called from other services/applications, but it is not appropriate for use in a chain (see Fig. 3.1).

[17] By the Institute of Mathematics and Computer Science (IMCS) at the University of Latvia.

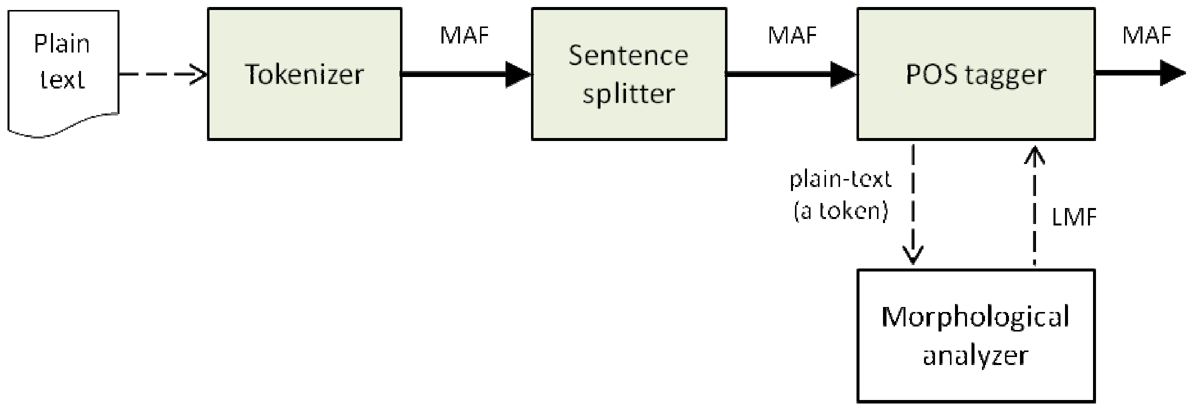


Fig. 3.1. IMCS services — a potential chain.

Initially, the POS tagger service provided also the tokenization and sentence splitting functionality, however, currently it is separated out. This enables not only a modular use of components, but also allows reusing third-party services[18] for the pre-processing tasks, which are largely language-independent and would require only some configuration parameters.

The IMCS services are implemented in the RESTful manner. It is planned to integrate the tokenizer, sentence splitter and POS tagger services in the WebLicht framework (see Sec. 1) as they are. However, to use the full potential of WebLicht infrastructure and to allow for potential exploitation of these services in other/further chains, a converter from MAF to WebLicht's TCF (Text Corpus Format) is needed.

The main annotation formats that we use are *Morpho-syntactic annotation framework* (MAF; ISO/DIS 24611) and *Lexical markup framework* (LMF; ISO/IS 24613:2008). It should be noted that MAF is not an official standard yet, however, since 2008-12-07 it has the Draft International Standard status, which means that most likely there will be no significant changes to the proposal. A consequence is that MAF and other ISO proposals that have just emerged as standards are poorly documented and have no extensive guidelines available as it is in the case of the well established de facto standards (like TEI).

The stages of development of an ISO standard comprise:

1) initial proposal of a new work item,
2) preparation of a Working Draft (WD),
3) production and acceptance of the Committee Draft (CD),
4) production and acceptance of the Draft International Standard (DIS), to be distributed to ISO member bodies for commenting and voting,
5) approval of the Final Draft International Standard (FDIS), which has to pass the final vote,
6) the publication of the International Standard (IS).

Other related ISO standards are Feature structure representation (FSR; ISO/IS 24610-1:2006), Word segmentation of written texts (WordSeg1; ISO/IS 24614-1:2010) and Linguistic annotation framework (LAF; ISO/DIS 24612).

[18] Provided that the data formats are compliant or that there is a converter available between the formats.

For specifying morphological categories we use ISOcat — an implementation of the Data Category Registry standard (DCR; ISO/IS 12620:2009; for more details see Sec. 2.7). Note that ISOcat is under active development as well (currently, beta version).

The intention of the ISO TC37 SC4 standards is to provide universal interchange (pivot) formats and to facilitate reusability, merging and comparison of linguistic information independent of the language used.

Apart from other, a characteristic feature of MAF (and other ISO *xAF* specifications) is its recommendation to use the standoff notation that provides independence from the original document by providing a way to reference intervals (character offsets) in documents (ISO:24611, 2008). Alternatively, the embedding notation may be used, but it is not recommended for two reasons: the morpho-syntactic annotations may conflict with other annotations, and the content of the textual material separating the textual content embedded within `<token>` elements is not precisely defined (white-spaces etc.). In the case of a typical text corpus, the second argument is usually irrelevant, and the first one can at least be postponed while there are no services available that could introduce conflicting (ambiguous) annotations. Thus, due to the ease of implementation we are using the embedding notation at the token level and the standoff notation at all the higher annotation levels.

3.1 Tokenizer and sentence splitter

Leveraging the existing codebase, the tokenizer and sentence splitter functionality is currently combined in a single webservice. In the next version of IMCS services, these functions should be uncoupled into separate services or at least into separate methods[19], fulfilling the scenario that is depicted in Figure 3.1. There are two reasons for this. First, the tokenizer could be easily replaced by some alternative and more advanced service (our simplified implementation rely on white-spaces only). The sentence splitter potentially could be replaced as well, but it is more language-dependent. Second, annotation of sentence level segments is not supported by MAF. This issue has been discussed by ISO TC37 SC4 working group in an early stage of the proposal (Clergerie, de la and Clément, 2004), but it turns out that finally the support has been rejected. The problem is that it is difficult to capture the notion of sentence — should it be addressed at the segmentation or syntax level? Also, it should not be used as an embedding XML element because of possible ambiguities. However, it was suggested that there should be a sentence marker within MAF annotation flow, similar to `wordForm` (see Sec. 3.2).

Sentence level annotation is provided by Syntactic annotation framework (SynAF; ISO/IS 24615:2010), however, we are not dealing with syntactic annotations yet — at this point we treat a sentence as a segmentation unit not a syntactic unit. Information about sentence boundaries is required by the POS tagger (see Sec. 3.2) and also by some other potential services (e.g., a parser); such information might be useful also in end-user applications. Therefore we have slightly extended the MAF schema, following suggestions by (Clergerie and Clément, 2004) and introducing the `sentence` element. In this way, the additional annotations that are produced by the sentence splitter can be just ignored by services and tools that are MAF compliant and are not aware about this extension.

The MAF format (incl. our extension) is self-explanatory. For instance, for the input text *“Es zinu ceļu. Un tu?”* the output will be:

[19] Each method can be registered in WebLicht as separate virtual service — similarly as in the case of ICS PAS Multiservice (see Section 2.5).

```
<maf>
  <token id="t0">Es</token>
  <token id="t1">zinu</token>
  <token id="t2">ceļu</token>
  <token id="t3">.</token>
  <token id="t4">Un</token>
  <token id="t5">tu</token>
  <token id="t6">?</token>
  <sentence tokens="t0 t1 t2 t3"/>
  <sentence tokens="t4 t5 t6"/>
</maf>
```

The input for this webservice is a stream of plain-text. The webservice extracts all tokens from the input stream and annotates them according to the MAF embedding notation. It also enumerates all tokens belonging to a sentence in a `sentence` element.
The proposed sentence annotation format of our extended version of MAF is very similar to the TCF format (see Sec. 1.2). The main difference is that the attribute `id` is not used in our approach to identify sentences (in TCF it is actually an optional attribute). This is due to the temporal characteristics of the sentence boundary annotations and also is in line with the standard MAF approach (consider, for instance, the `wordForm` annotation flow). Another difference is that character offsets are not indicated — due to the embedding annotation approach at the token level. The last minor difference of MAF is that elements of the same type (the same annotation layer) are not grouped by embedding elements (like `tokens` and `sentences` in the case of TCF), but this is not a functional deficiency.

3.2 POS tagger

The input for the part-of-speech tagger is the output of the tokenizer/sentence splitter service: a list of tokens along with additional information of which tokens belong to the same sentence. The tagger is context-insensitive in the sense that sentences are processed independently from each other — this is a common characteristic of HMM-based taggers.

The output consists of the repetition of input, i.e., the list of tokens and sentence markers, in order to make the response self-contained, and a set of `wordForm` elements, each containing the most probable feature structure that describes the corresponding word form. The feature and value identifiers are received from the morphological analyzer and correspond to those of ISOcat. It should be mentioned that token attachment in Latvian is one-to-one. At this level of analysis we are not considering syntactic units (parts of sentence) that might be multi-word units (e.g., analytical forms of a verb and prepositional constructions) — this is a task of a syntactic parser — at the syntactic annotation layer.

Currently the POS tagger calls the morphological analyzer locally (via the standard API), however, it could be decoupled (as depicted in Figure 3.1), most likely incurring a performance hit, but allowing for greater flexibility in future.

Below is an example of a complete response for a request that passes the data produced in Sec. 3.1:

```
<maf>
  <token id="t0">Es</token>
  <token id="t1">zinu</token>
  <token id="t2">ceļu</token>
  <token id="t3">.</token>
  <token id="t4">Un</token>
  <token id="t5">tu</token>
  <token id="t6">?</token>
  <sentence tokens="t0 t1 t2 t3"/>
  <sentence tokens="t4 t5 t6"/>
  <wordForm entry="url:lexicon:lv:es" lemma="es" tokens="t0">
    <fs>
      <f name="partOfSpeech"><symbol value="pronoun"/></f>
      <f name="grammaticalNumber"><symbol value="singular"/></f>
      <f name="case"><symbol value="nominativeCase"/></f>
      <f name="person"><symbol value="firstPerson"/></f>
    </fs>
  </wordForm>
  <wordForm entry="url:lexicon:lv:zināt" lemma="zināt" tokens="t1">
    <fs>
      <f name="partOfSpeech"><symbol value="verb"/></f>
      <f name="reflexivity"><symbol value="nonreflexive"/></f>
      <f name="verbFormMood"><symbol value="indicative"/></f>
      <f name="grammaticalNumber"><symbol value="singular"/></f>
      <f name="grammaticalTense"><symbol value="present"/></f>
      <f name="person"><symbol value="firstPerson"/></f>
    </fs>
  </wordForm>
  <wordForm entry="url:lexicon:lv:ceļš" lemma="ceļš" tokens="t2">
    <fs>
      <f name="partOfSpeech"><symbol value="noun"/></f>
      <f name="grammaticalGender"><symbol value="masculine"/></f>
      <f name="grammaticalNumber"><symbol value="singular"/></f>
      <f name="case"><symbol value="accusativeCase"/></f>
    </fs>
  </wordForm>
  <wordForm entry="url:lexicon:lv:." lemma="." tokens="t3">
    <fs>
      <f name="partOfSpeech"><symbol value="punctuation"/></f>
    </fs>
  </wordForm>
  <wordForm entry="url:lexicon:lv:un" lemma="un" tokens="t4">
    <fs>
      <f name="partOfSpeech"><symbol value="conjunction"/></f>
    </fs>
  </wordForm>
  <wordForm entry="url:lexicon:lv:tu" lemma="tu" tokens="t5">
    <fs>
      <f name="partOfSpeech"><symbol value="pronoun"/></f>
      <f name="grammaticalNumber"><symbol value="singular"/></f>
      <f name="case"><symbol value="nominativeCase"/></f>
      <f name="person"><symbol value="secondPerson"/></f>
    </fs>
  </wordForm>
  <wordForm entry="url:lexicon:lv:?" lemma="?" tokens="t6">
    <fs>
      <f name="partOfSpeech"><symbol value="punctuation"/></f>
    </fs>
  </wordForm>
</maf>
```

Before moving to MAF compliance, the tagger webservice employed an ad-hoc output format where each input word was followed by a positional Multext-East style tag. In such a fashion the preceding example would have been encoded in a much more compact form, like

```
Es/p-sn1 zinu/vnisp1 ceļu/nmsa ./t
```

where `v` indicates "verb", `1` — "first person" etc., and dashes stand for positions that the tagger is unable to determine or that are inappropriate for the particular word form (e.g., the gender for a `firstPerson` pronoun).

This clearly had the benefit of interface being very simple and was successfully used in applications like machine translation that did not require intricate knowledge of the tag contents. However for most of the other uses this approach required every client to know how to parse the positional tags. Initially we used to distribute a parser library for such tags, however, any changes in the tagset required each client to update this library and turned out to be a maintenance liability.

The feature structure is clearly superior in this regard — it is a much more flexible approach, and the tagger responses have the tags already "parsed", completely eliminating the version discrepancy problem of the parser library. Furthermore, the use of ISOcat data categories will facilitate development of cross-lingual applications. The only downside is the verboseness of this format, which is suboptimal for storing a large amount of pre-tagged sentences, such as factored corpora for statistical machine translation.

3.3 Morphological analyzer

The morphological analyzer (its lexicon) is based on the Dictionary of the Standard Latvian Language[20]. Currently it recognizes more than 1.3 million word forms (see Table 3.1). The lexicon is stored in a database table where all word forms are listed along with their morphological features, thus the analysis is reduced to a simple and efficient search task. This allows exploiting the analyzer also as a synthesizer — in the same simple manner.

	Lemmas	**Word forms**	**Features**
Nouns	32 386	355 488	710 976
Verbs	12 002	347 729	1 174 964
Adjectives	6 086	681 632	3 408 160
Other	6 958	7 379	944
	57 432	**1 392 228**	**5 295 044**

Table 3.1. Statistics of the morphological lexicon for Latvian.

The analyzer service has two respective methods: `analyze` and `synthesize`. The first one expects a single parameter, a word form; the second one — two parameters: a lemma and a POS code. All the input parameters are plain-text values, but the result returned by both methods is in the LMF

[20] This dictionary is available via a Web service as well. It contains ca. 64 000 structurally rich XML-encoded entries. Currently a proprietary schema is used; LMF compliance is planned in the future.

format[21]. For instance, if the word form “ceļu” is passed for analysis, the following result is returned[22]:

```
<LexicalResource dtdVersion="16">
  <GlobalInformation>
    <feat att="languageCoding" val="ISO 639-3"/>
  </GlobalInformation>
  <Lexicon>
    <feat att="language" val="lav"/>
    <LexicalEntry>
      <feat att="partOfSpeech" val="verb"/>
      <feat att="reflexivity" val="nonreflexive"/>
      <Lemma>
        <feat att="writtenForm" val="celt"/>
      </Lemma>
      <WordForm>
        <feat att="writtenForm" val="ceļu"/>
        <feat att="verbFormMood" val="indicative"/>
        <feat att="grammaticalTense" val="present"/>
        <feat att="grammaticalNumber" val="singular"/>
        <feat att="person" val="firstPerson"/>
      </WordForm>
    </LexicalEntry>
    <LexicalEntry>
      <feat att="partOfSpeech" val="noun"/>
      <feat att="grammaticalGender" val="masculine"/>
      <Lemma>
        <feat att="writtenForm" val="ceļš"/>
      </Lemma>
      <WordForm>
        <feat att="writtenForm" val="ceļu"/>
        <feat att="grammaticalNumber" val="singular"/>
        <feat att="case" val="accusativeCase"/>
      </WordForm>
      <WordForm>
        <feat att="writtenForm" val="ceļu"/>
        <feat att="grammaticalNumber" val="plural"/>
        <feat att="case" val="genitiveCase"/>
      </WordForm>
```

[21] http://www.lexicalmarkupframework.org/

[22] There are some more analysis variants (lexical entries) possible for this word form; to save the space, we haven’t included them in the example.

```
    </LexicalEntry>
  </Lexicon>
</LexicalResource>
```

As we can see, the word form “ceļu” is morphologically ambiguous: it can be a verb or a noun, and, in the case of a noun, two alternative sets of attribute-value pairs can be assigned. Note that different POS categories usually are described by different sets of features (categories) — the generic feature structure representation (FSR) approach that is used in LMF allows to capture them in a simple uniform way. Moreover, different feature (sub)sets often are necessary even in the scope of a single POS, for instance, in the case of verbs. Below is a result returned by the `synthesize` method that received parameters `writtenForm=celt` and `partOfSpeech=verb` in the input:

```
<LexicalResource dtdVersion="16">
  <GlobalInformation>
    <feat att="languageCoding" val="ISO 639-3"/>
  </GlobalInformation>
  <Lexicon>
    <feat att="language" val="lav"/>
    <LexicalEntry>
      <feat att="partOfSpeech" val="verb"/>
      <feat att="reflexivity" val="nonreflexive"/>
      <Lemma>
        <feat att="writtenForm" val="celt"/>
      </Lemma>
      <WordForm>
        <feat att="writtenForm" val="celt"/>
        <feat att="verbFormMood" val="infinitive"/>
      </WordForm>
      <WordForm>
        <feat att="writtenForm" val="ceļu"/>
        <feat att="verbFormMood" val="indicative"/>
        <feat att="grammaticalTense" val="present"/>
        <feat att="grammaticalNumber" val="singular"/>
        <feat att="person" val="firstPerson"/>
      </WordForm>
      <WordForm>
        <feat att="writtenForm" val="celšot"/>
        <feat att="verbFormMood" val="relative"/>
        <feat att="grammaticalTense" val="future"/>
      </WordForm>
      <WordForm>
        <feat att="writtenForm" val="jāceļ"/>
        <feat att="verbFormMood" val="debitive"/>
      </WordForm>
      <WordForm>
        <feat att="writtenForm" val="celiet"/>
        <feat att="verbFormMood" val="imperative"/>
        <feat att="grammaticalNumber" val="plural"/>
        <feat att="person" val="secondPerson"/>
      </WordForm>
      ...
    </LexicalEntry>
  </Lexicon>
</LexicalResource>
```

If compared to positional tags, or schemas where the morpho-syntactic categories are encoded in (embedding) element names, providing a compact representation, the generic FSR approach is much more verbose, however, it allows for more robust processing and flexible updates.

The morphological analyzer/synthesizer uses ISOcat data categories for representing the morpho-syntactic features and their values. In the case of verbs, however, few additional data categories are used that currently are not included in the ISOcat registry: values `relative` and `debitive` are used for the category `verbFormMood`, and a new category `reflexivity` has been introduced. We have suggested to include the missing data categories in the Morphosyntax profile of ISOcat, and the first response from a member of the respective thematic domain group (TDG) of ISOcat is positive (the proposed lexical identifiers, of course, might change). While relative and debitive moods of a verb are quite specific for Baltic languages, reflexivity is a common category in many languages.

Notice that in the ISOcat data model lexical identifiers are not unique parameters. But PIDs do uniquely identify a data category (version). For example, for `grammaticalGender` there are two data categories: http://www.isocat.org/datcat/DC-1297 and http://www.isocat.org/datcat/DC-245. This is because various thematic domains (in this case, Morphosyntax and Terminology), or even specific applications, might have different semantics.

The DCR (ISO 12620) standard provides a small XML vocabulary to embed data category persistent identifiers (PID) in XML documents[23]. For the sake of simplicity we haven't provided PID references in the previous examples, but a general example would be:

```
<fs xmlns:dcr="http://www.isocat.org/ns/dcr">
  <f name="grammaticalGender" dcr:datcat="http://www.isocat.org/datcat/DC-1297"
     value="neuter" dcr:valueDatcat="http://www.isocat.org/datcat/DC-1884"/>
</fs>
```

Note that there are features (apart from `partOfSpeech`) that belong to an entry as a whole, i.e., that are true for all of its word forms (like gender in the case of nouns, and reflexivity in the case of verbs), while most features are assigned to each particular word form.

The complete set of data categories (except POS categories) that are used by the Latvian morphological webservice is given in Table 3.2. Although several possible categories are missing (the analyzer does not support participles and numerals yet, and some more specific categories might be included), most of them are common to several parts of speech, thus, from the FSR perspective (in contrast to the positional tag approach), the "tagset" is rather small even for such an inflective

[23] The schema for this reference vocabulary can be found at: http://www.isocat.org/files/12620.html. We would like to thank Menzo Windhouwer for providing information on this issue.

language like Latvian. On average, 3.8 features are used to describe a word form (entry level features like the gender in the case of nouns are not taken into account).

Data category	
Lexical identifier	**Persistent Identifier**
grammaticalGender	http://www.isocat.org/datcat/DC-1297
feminine	http://www.isocat.org/datcat/DC-1880
masculine	http://www.isocat.org/datcat/DC-1883
grammaticalNumber	http://www.isocat.org/datcat/DC-1298
singular	http://www.isocat.org/datcat/DC-1387
plural	http://www.isocat.org/datcat/DC-1354
case	http://www.isocat.org/datcat/DC-1840
nominativeCase	http://www.isocat.org/datcat/DC-1331
genitiveCase	http://www.isocat.org/datcat/DC-1293
dativeCase	http://www.isocat.org/datcat/DC-1265
accusativeCase	http://www.isocat.org/datcat/DC-1226
locativeCase	http://www.isocat.org/datcat/DC-1326
degree	http://www.isocat.org/datcat/DC-1419
positive	http://www.isocat.org/datcat/DC-1420
comparative	http://www.isocat.org/datcat/DC-1421
superlative	http://www.isocat.org/datcat/DC-1422
definiteness	http://www.isocat.org/datcat/DC-1926
definite	http://www.isocat.org/datcat/DC-2004
indefinite	http://www.isocat.org/datcat/DC-2005
reflexivity	
reflexive	
nonreflexive	
verbFormMood	http://www.isocat.org/datcat/DC-1427
infinitive	http://www.isocat.org/datcat/DC-1312
indicative	http://www.isocat.org/datcat/DC-1885
relative	
debitive	
conditional	http://www.isocat.org/datcat/DC-1258
imperative	http://www.isocat.org/datcat/DC-1844

Data category	
Lexical identifier	**Persistent Identifier**
grammaticalTense	http://www.isocat.org/datcat/DC-1286
present	http://www.isocat.org/datcat/DC-1367
past	http://www.isocat.org/datcat/DC-1347
future	http://www.isocat.org/datcat/DC-1291
person	http://www.isocat.org/datcat/DC-1328
firstPerson	http://www.isocat.org/datcat/DC-1288
secondPerson	http://www.isocat.org/datcat/DC-1384
thirdPerson	http://www.isocat.org/datcat/DC-1402

Table 3.2. Morpho-syntactic categories used by the IMCS webservice.

One has to be careful while choosing the right identifier and PID for an ISOcat data category: the same or similar term might be used not only in different thematic domains as mentioned above, but also in “third-party” profiles that are registered in ISOcat (like the Polish tagset; see Sec. 2.7).

In the case of the few missing categories for Latvian, we could register them in a separate profile as well, however, this would not make much sense, and this is not the mission of ISOcat — only the approved “common sense” terms are “meaningful”, especially if we are considering automatic services and tools that might not be aware of the “proprietary” profiles of ISOcat.

4 Web services and representation standards

4.1 General issues

Representation vs. interchange

The liaison between representation standards and interchange standards is tight – what is represented, can be exchanged between systems and the easiest way to achieve a common exchange format is to reuse the representation one. This is why most remarks related to representation of linguistic features gathered e.g. in the process of creation of corpora, can be referred to while evaluating interchange standards.

It also needs to be clearly stated that the Web Service aspect seems orthogonal to the representation aspect when linguistic properties are taken into consideration. As Web services offer just new manifestations of LR standards, D5C-3 document should be referred to for the purpose of “selecting the best standard to represent linguistic data in an application wrapped as a Web service”. The question of usefulness of standards in processing pipelines composed of WSs is in fact the question about the quality of these standards.

Stand-off annotation and Web services[24]

The requirement of stand-off character of a representation/interchange linguistic standard (or at least making stand-off annotation possible as one of its variants) seems indisputable since many levels of annotation can involve conflicting hierarchies. Another requirement is keeping the object of description (the text) maximally divorced from its possible theoretical views (annotations), in as neutral form as possible. This way it remains open to future analyses and to the creation of new views, i.e., new annotation layers (which improves extensibility). Moreover, the annotations can serve as the basis for comparison of tools and theories. Security (immutability) of the text itself is also guaranteed by non-destructiveness of the process with respect to the resource that gets annotated[25]. Interoperability values the ease of transduction, both of the source text and its annotations. Sometimes, the ease of mapping a single layer of annotation to another resource (e.g. a translated document) is also important.

On the other hand, (Rehm *et al.* 2010) point out that stand-off approaches are not optimal from the point of view of sustainability because they require dedicated tools in order to merge annotations with the source text. This argument is true, but can be in most cases put down to the evolution of XML technology as long as the stand-off annotation layers can be handled by generic XML tools. The solution which (Rehm *et al.* 2010) suggest is *multiply-annotated text* which also uses layers of annotation but each of these layers contains an exact copy of the source text, and thus achieves sustainability through redundancy. It is regarded by the authors of this notion to have more advantages than stand-off approaches that keeping a single copy of the source text. Additionally, in principle, both approaches can be mixed in e.g. crowd-sourced corpora where annotation layers are contributed by external parties.

Independently on the representation approach, Web service environment requires dedicated machinery for making the stand-off annotated resources available, but this issue is more or less technical (however, it was partially addressed by introducing the “packaged” format described in Sec. 2.2).

Empirical validation principle

The necessity of using standards for interchange and development of new resources and tools is indisputable. However, the selection of a concrete representation seems difficult in the presence of multitude of competitive representations. As presented in previous chapters, all three candidate recommendations satisfy the standardization needs and it is only scientific community who will select which one will be used predominantly (or possibly, that all three will be equally represented).

Taking this point into account, the WebLicht approach seems very interesting since it shows how a project-internal processing format (TCF) can go beyond its initial purpose with multitude of services built around it and integrated converters available. Obviously, directly comparison of TCF and TEI makes no sense since both frameworks were designed with different intentions in mind: TEI as an “all-around” format for maintaining compatibility and exchangeability of linguistic data and TCF as

[24] This section is based on (Bański 2010).

[25] On sustainability of LRs in general, see e.g. (Bird and Simons 2003) and Simons & Bird, 2008. On sustainability in the context of the TEI, see Witt et al., 2009b.

For more on interoperability of LRs, see e.g. Ide & Romary, 2007 or Witt et al., 2009a. We use the stand-off terminology in accordance with Goecke et al., 2010.

a compact format with main emphasis on transporting data between Web services and being machine readable in an effective way. Nevertheless, the growing number of tools processing TCF makes it an interesting alternative to other interchange format even outside WebLicht.

4.2 Sustainability and interoperability of language resources

Technical interoperability

Apart from the technical Web service standards (such as REST/SOAP protocols, WSDL descriptions etc. – see CLARIN D2R-6b deliverable, Requirement Specification Web Services and Workflow systems) the technical interoperability of LRs is maintained by wide adoption of general standards such as:

- Unicode – a character encoding standard in data processing and interchange,
- XML – a text format for information encoding and interchange,
- ISO 639 – codes for the representation of names of languages,
- IMDI, TEI or OLAC – for component metadata,

all recommended by CLARIN. Assuming that XML interchange formats follow official standards, technical interoperability at resource-structure level can be achieved easily.

Syntactic vs. semantic interoperability

Current standards, especially the official ones, give a good way of establishing formal or syntactic interoperability, which is a fundamental prerequisite for interoperability at large. Assuming XML interchange formats following official representation standards, interoperability at resource-structure level can be achieved.

However, the problem of semantic interoperability still remains open: even ensuring isomorphic structures, the meaning of the descriptors may still be unknown. With this respect, the idea of formally mapping (proprietary) descriptors to (some) standard concepts, as those in ISOCat, appears to be, at present, a practical and tangible solution.

Internal interoperability

The notion of technical, formal and semantic interoperability seems to have an independent aspect which may be named *internal interoperability*. This relates to the fact of extended generality of a standard which hinders uniformity of representations still created with full conformance to the standard.

In the case of TEI, using its model requires designing a particular text encoding schema by choosing the most appropriate mechanisms from the TEI toolbox and/or, less frequently, by introducing new XML elements or attributes. This inbuilt flexibility leads to multiple possibilities to encode the same phenomenon, which basically makes it necessary to choose between several options and thereby constraining or simplifying the TEI specifications in order to ensure interoperability. Since the specifications aim to capture all aspects of textual resources, they contain not only markup for the logical structuring of textual data but also for typographic information needed for purposes of publication.

(Helbig 2001) connects the notion of interoperability with homogeneity (usage of the same formalism for representing different levels of linguistic description) and communicability (good documentation and possibility of sharing resources). To build on these notions, maintenance of internal interoperability of a standard would require ensuring adequate level of communicability by means of presentation of detailed guidelines and examples. TEI Guidelines tend to serve this purpose to a great extent, yet they still offer numerous diverse possibilities of encoding the same set of linguistic properties.

5 Closing notes

CLARIN does not seem to be giving precedence to concrete formats and/or data categories, rather exploring LRT applications and promoting standards in general. Standardization is essential, but overstandardization (the use of a single dominant representation format at every level of use) would be a move in the wrong direction and as such was never promoted by any major standardization initiatives. The golden mean for document encoding is reaching some standardized representation of data and program output for sharing and re-using data, without controlling its every aspect by a specific model. Therefore, the adoption of general, broad-ranging metamodels which can accommodate many different representations should be recommended.

Growing complexity of encodings of language resources (by means of presence of large, multilingual or language-independent toolsets and frameworks covering practically all aspects of linguistic annotation) must be (and is) reflected in the construction of relevant standards. It drives existing standards into large families of metastandards and metamodels which in turn creates unavoidable problems of balancing their flexibility and precision: being abstract and permissive they cannot add too much to the original representation – and being too specific, they may overtly constrain representation of resource properties to the limitations of the standard.

The straightforward solution to this problem is adoption of a design pattern concept with three specific rules: implementing reusable solutions at the general design level (segments, feature structures, links etc.) while being specific at the instantiation level (e.g. a particular tagset) and retaining possibility to enhance model if necessary. This principle seems to be implicitly followed by most of currently implemented CLARIN Web Services, but, more importantly, is being offered by large families of general standards such as ISO or TEI. When applying them, maintenance of internal interoperability of the newly created framework-based representation seems important, so even with the most general encoding infrastructures the first step would be adopting existing structures and basing on earlier experiences and use cases rather than exploiting the general features to create a new semi-standard, still carrying, for instance, a TEI P5 label, but in fact being far from its spirit. This would allow for keeping the highest level of uniformity between subsequent representations of similar linguistic ideas which supports interoperability. In this respect, there is one more demand which calls for adequate attention: careful documentation of the adopted model. Independently of the selected encoding model, keeping reference to existing standards of linguistic data encoding (such as ISO DCR, even despite its immaturity) is essential.

At the same time, looking at the example of how Unicode lately changed the text representation world with its simplicity and universality, we could easily imagine a linguistic standard created in a similar manner and covering most of the needs of the linguistic community. The emerging families of standards are probably slowly heading towards this utopian direction, but the costs of reaching it make it a distant target. In contrast, LRT practitioners may opt to work with a greater number of well-defined, but local and/or specific standards such as TIGER-XML or the original version of XCES rather than with conceptually more attractive standard interchange formats. Both solutions have their dedicated followers and the battle over standards is definitely far from reaching the end – but this way or the other, the standards are going to win it.

List of CLARIN Web services

Following Web services have been registered as a technical result of CLARIN within deliverable D5R-3:

	Web service name	Institution(s) responsible	Languages	URL
1	ULei – Sentences	ASV	DE	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Sentences?language=de
2	ULei – Baseform	ASV	DE	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/BaseForm?language=de
3	ULei – Cooccurrences	ASV	DE	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Cooccurrences?language=de
4	ULei – Frequency	ASV	DE	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Frequencies?language=de
5	ULei – TextCorpus2Lexicon	ASV	DE	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/TextCorpus2Lexicon?language=de
6	ULei – Tokenizer	ASV	DE	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Tokenizer
7	ULei – Sentences	ASV	EN	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t5/Sentences
8	ULei – TextCorpus2Lexicon	ASV	EN	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/TextCorpus2Lexicon?language=en
9	ULei – Frequency	ASV	EN	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Frequencies?language=en
10	ULei – Cooccurrences	ASV	EN	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Cooccurrences?language=en
11	ULei – TextCorpus2Lexicon	ASV	ES	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/TextCorpus2Lexicon?language=es

	Web service name	Institution(s) responsible	Languages	URL
12	ULei – Cooccurrences	ASV	ES	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Cooccurrences?language=es
13	ULei – Frequency	ASV	ES	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Frequencies?language=es
14	ULei – Sentences	ASV	ES	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Sentences?language=es
15	ULei – Sentences	ASV	FI	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Sentences?language=fi
16	ULei – TextCorpus2Lexicon	ASV	FI	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/TextCorpus2Lexicon?language=fi
17	ULei – Frequency	ASV	FI	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Frequencies?language=fi
18	ULei – Cooccurrences	ASV	FI	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Cooccurrences?language=fi
19	ULei – Sentences	ASV	FR	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Sentences?language=fr
20	ULei – TextCorpus2Lexicon	ASV	FR	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/TextCorpus2Lexicon?language=fr
21	ULei – Frequency	ASV	FR	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Frequencies?language=fr
22	ULei – Cooccurrences	ASV	FR	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Cooccurrences?language=fr
23	ULei – TextCorpus2Lexicon	ASV	IT	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/TextCorpus2Lexicon?language=it
24	ULei – Frequency	ASV	IT	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Frequencies?language=it

	Web service name	Institution(s) responsible	Lan gua ges	URL
25	ULei – Cooccurrences	ASV	IT	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Cooccurrences?language=it
26	ULei – Sentences	ASV	IT	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Sentences?language=it
27	ULei – Sentences	ASV	RO	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Sentences?language=ro
28	ULei – TextCorpus2Lexicon	ASV	RO	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/TextCorpus2Lexicon?language=ro
29	ULei – Cooccurrences	ASV	RO	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Cooccurrences?language=ro
30	ULei – Frequency	ASV	RO	http://clarin1.informatik.uni-leipzig.de:5000/dspin_v03_t6/Frequencies?language=ro
31	BBAW Person Name Recognizer (TCF 0.4)	BBAW	DE	http://dspin.dwds.de/services/ne_v_0_4
32	BBAW Part-of-Speech Tagger (TCF 0.4)	BBAW	DE	http://dspin.dwds.de/services/tagger_v_0_4
33	BBAW Tokenizer and Sentence Splitter (TCF 0.4)	BBAW	DE	http://dspin.dwds.de/services/tokenizer_v_0_4
34	BBAW Plaintext Converter (TCF 0.4)	BBAW	DE	http://dspin.dwds.de/services/rohling_v_0_4
35	BBAW C4-corpus query (TCF 0.4)	BBAW	DE	http://dspin.dwds.de/services/ddc_v_0_4
36	English-Lithuanian Machine Translation Service	CCL VMU	LT	http://vertimas.vdu.lt/twsas/
37	PhonoMorpho	CNR-ILC	IT	http://www.clarin-it.it/Simple/services/PhonoMorphoSOAP?wsdl
38	Syntax	CNR-ILC	IT	http://www.clarin-it.it/Simple/services/SyntaxSOAP?wsdl
39	Semantic	CNR-ILC	IT	http://www.clarin-it.it/Simple/services/SemanticSOAP?wsdl

	Web service name	Institution(s) responsible	Languages	URL
40	ExportLMF	CNR-ILC	IT	http://www.clarin-it.it/Simple/services/ExportLMFSOAP?wsdl
41	Croatian Lemmatization Server	FFZG	CR	http://hml.ffzg.hr
42	Sanskrit Heritage Site	Gérard Huet	SA	http://sanskrit.inria.fr/
43	Tokenizer (TCF 0.3, Finnish)	GL: Uni Helsinki	FI	http://sysdb.cs.helsinki.fi/tomcat/avihavai/hfst/fi/tokenizer
44	Plaintext Converter (TCF 0.3, Finnish)	GL: Uni Helsinki	FI	http://sysdb.cs.helsinki.fi/tomcat/avihavai/hfst/fi/plaintext
45	Morph analyzer (TCF 0.3, Finnish)	GL: Uni Helsinki	FI	http://sysdb.cs.helsinki.fi/tomcat/avihavai/hfst/fi/morphanalyzer
46	Spejd	ICS PAS	PL	http://chopin.ipipan.waw.pl:8083/WebService/resources/Clarin/Spejd
47	Swigra	ICS PAS	PL	http://chopin.ipipan.waw.pl:8083/WebService/resources/Clarin/Swigra
48	Pantera	ICS PAS	PL	http://chopin.ipipan.waw.pl:8083/WebService/resources/Clarin/Pantera
49	Cosmas2 Goethe corpus query	IDS	DE	http://cosmas2.ids-mannheim.de:6382/services/WebLicht
50	Morphological analyzer/synthesizer for Latvian	IMCS	LV	http://valoda.ailab.lv/ws/morph.jsp
51	POS tagger for Latvian	IMCS	LV	http://eksperimenti.ailab.lv/tagger/
52	Text-to-speech synthesizer for Latvian	IMCS	LV	http://valoda.ailab.lv/ws/tts.jsp
53	Dictionary of the Standard Latvian Language	IMCS	LV	http://valoda.ailab.lv/ws/llvv.jsp
54	TreeTagger (TCF 0.4, French)	IMS	FR	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tree-tagger4.perl
55	TreeTagger (TCF 0.3, French)	IMS	FR	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tree-tagger3.perl

	Web service name	Institution(s) responsible	Languages	URL
56	TreeTagger (TCF 0.4, Italian)	IMS	IT	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tree-tagger4.perl
57	TreeTagger (TCF 0.3, Italian)	IMS	IT	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tree-tagger3.perl
58	RFTagger (TCF 0.4, Czech)	IMS	CZ	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/new_hp/rftagger.py
59	Tokenizer (TCF 0.4, Czech)	IMS	CZ	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser4.perl
60	RFTagger (TCF 0.4, German)	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/new_hp/rftagger.py
61	Converter TEI2TCF	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tei2tcf4.perl
62	Tokenizer (TCF 0.4, German)	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser4.perl
63	Constituent Parser (TCF 0.4, German)	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/bitpar4.perl
64	TreeTagger (TCF 0.4, German)	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tree-tagger4.perl
65	SMOR (TCF 0.4, German)	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/smor4.perl
66	Tokenizer (TCF 0.3, German)	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser3.perl
67	TreeTagger (TCF 0.3, German)	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tree-tagger3.perl
68	Converter MAF2TCF	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/maf2tcf.perl

	Web service name	Institution(s) responsible	Languages	URL
69	Constituent Parser (TCF 0.3, German)	IMS	DE	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/bitpar3.perl
70	Tokenizer (TCF 0.4, English)	IMS	EN	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser4.perl
71	Constituent Parser (IMS, TCF 0.4, English)	IMS	EN	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/bitpar4.perl
72	TreeTagger (TCF 0.4, English)	IMS	EN	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tree-tagger4.perl
73	Tokenizer (TCF 0.3, English)	IMS	EN	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser3.perl
74	TreeTagger (TCF 0.3, English)	IMS	EN	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tree-tagger3.perl
75	Constituent Parser (TCF 0.3, English)	IMS	EN	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/bitpar3.perl
76	Tokenizer (TCF 0.4, French)	IMS	FR	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser4.perl
77	Tokenizer (TCF 0.3, French)	IMS	FR	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser3.perl
78	Converter TEI2TCF	IMS	FR	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tei2tcf3.perl
79	RFTagger (TCF 0.4, Hungarian)	IMS	HU	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/new_hp/rftagger.py
80	Tokenizer (TCF 0.4, Hungarian)	IMS	HU	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser4.perl
81	Tokenizer (TCF 0.4, Italian)	IMS	IT	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser4.perl

	Web service name	Institution(s) responsible	Languages	URL
82	Tokenizer (TCF 0.3, Italian)	IMS	IT	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser3.perl
83	RFTagger (TCF 0.4, Slovenian)	IMS	SL	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/new_hp/rftagger.py
84	Tokenizer (TCF 0.4, Slovenian)	IMS	SL	http://gelbaugenpinguin.ims.uni-stuttgart.de/cgi-bin/dspin/tokeniser4.perl
85	Assigning lemmas and part-of-speech to wordform lists	ISJ ZRC SAZU	SL	http://bos.zrc-sazu.si/dol_lem.html
86	Pdf2txt	IULA-UPF	ALL	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/format_conversion.pdftotext?wsdl
87	Doc2txt	IULA-UPF	ALL	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/format_conversion.catdoc?wsdl
88	Html2txt	IULA-UPF	ALL	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/format_conversion.html2text?wsdl
89	XSLTtransformer	IULA-UPF	ALL	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/format_conversion.xsltproc?wsdl
90	Iconv	IULA-UPF	ALL	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/format_conversion.iconv?wsdl
91	Corpus Work Bench CWB (CQP)	IULA-UPF	ALL	http://gilmere.upf.edu:8080/iulaws/services/cqp?wsdl
92	Corcondancer (kwic)	IULA-UPF	ALL	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/text_mining.kwic?wsdl
93	Ngrams Statistics Package	IULA-UPF	ALL	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/statistics_analysis.ngrams?wsdl
94	Vocabulary Analysis	IULA-UPF	ALL	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/statistics_analysis.vocabulary_analysis?wsdl

	Web service name	Institution(s) responsible	Languages	URL
95	Catalan Press	IULA-UPF	ALL	http://gilmere.upf.edu/access_girona/queries
96	Catalan Annotated Corpora	IULA-UPF	ALL	http://gilmere.upf.edu/access_distribuit
97	IULA preprocess tool	IULA-UPF	EN ES CA	http://kurwenal.upf.edu:8080/soaplab2-axis/typed/services/chunking_segmentation.iula_preprocess?wsdl
98	Freeling tokenizer	IULA-UPF	EN ES CA	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/tokenization.freeling_tokenizer?wsdl
99	IULA tokenizer	IULA-UPF	EN ES CA	http://kurwenal.upf.edu:8080/soaplab2-axis/typed/services/tokenization.iula_tokenizer?wsdl
100	Freeling morpho(logical analysis)	IULA-UPF	EN ES CA	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/morphosintactic_tagging.freeling_morpho?wsdl
101	IULA morphological analyser	IULA-UPF	EN ES CA	http://kurwenal.upf.edu:8080/soaplab2-axis/typed/services/stemming_lemmatization.iula_lexicon_lookup?wsdl
102	Freeling	IULA-UPF	EN ES CA	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/morphosintactic_tagging.freeling?wsdl
103	Freeling pos-tagger	IULA-UPF	EN ES CA	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/morphosintactic_tagging.freeling_tagging?wsdl
104	IULA TreeTagger (IULA POS tagger based on TreeTagger)	IULA-UPF	EN ES CA	http://kurwenal.upf.edu:8080/soaplab2-axis/typed/services/morphosintactic_tagging.iula_tagger?wsdl

	Web service name	**Institution(s) responsible**	**Languages**	**URL**
105	Freeling parser	IULA-UPF	EN ES CA	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/syntactic_tagging.freeling_parsed?wsdl
106	Freeling dependency parser	IULA-UPF	EN ES CA	http://gilmere.upf.edu:8080/soaplab2-axis/typed/services/syntactic_tagging.freeling_dependency?wsdl
107	MIMORE Web services	Meertens Institute	NL	http://www.clarin.nl/node/70#MIMORE
108	DynaSAND	Meertens Institute	NL	http://www.meertens.knaw.nl/sand/soap/
109	MAND	Meertens Institute	NL	http://www.meertens.knaw.nl/mand/soap/
110	DiDDD	Meertens Institute	NL	http://www.meertens.knaw.nl/diddd/soap/
111	Dutch Database of Family Names	Meertens Institute	NL	http://www.meertens.knaw.nl/nfd/
112	Dutch Database of First Names	Meertens Institute	NL	http://www.meertens.knaw.nl/voornamen/VNB/
113	Language identification	RACAI	ALL	http://nlp.racai.ro/webservices/LangIdWebService.asmx?WSDL
114	Tokenizer / POS Tagger English	RACAI	EN	http://ws2.racai.ro/TTL-en-tokenizer-and-postagger
115	WordNet Browser	RACAI	EN RO	http://nlp.racai.ro/wnbrowser/
116	Tokenizer / POS Tagger French	RACAI	FR	http://ws2.racai.ro/TTL-fr-tokenizer-and-postagger
117	Tokenizer and Sentence Splitter	RACAI	EN	http://ws2.racai.ro/TTL-en-tokenizer
118	Lemmatization for English	RACAI	EN	http://ws2.racai.ro/TTL-en-lemmatizer
119	Chunker for English	RACAI	EN	http://ws2.racai.ro/TTL-en-chunker
120	PlainText to TCF	RACAI	EN	http://ws2.racai.ro/TTL-en-textconverter

	Web service name	Institution(s) responsible	Languages	URL
121	Tokenizer and Sentence Splitter	RACAI	FR	http://ws2.racai.ro/TTL-fr-tokenizer
122	Chunker for French	RACAI	FR	http://ws2.racai.ro/TTL-fr-chunker
123	Lemmatization for French	RACAI	FR	http://ws2.racai.ro/TTL-fr-lemmatizer
124	Plain Text to TCF Converter (French)	RACAI	FR	http://ws2.racai.ro/TTL-fr-textconverter
125	Tokenizer / POS Tagger Romanian	RACAI	RO	http://ws2.racai.ro/TTL-ro-tokenizer-and-postagger
126	Chunker for Romanian	RACAI	RO	http://ws2.racai.ro/TTL-ro-chunker
127	Lemmatization for Romanian	RACAI	RO	http://ws2.racai.ro/TTL-ro-lemmatizer
128	PlainText to TCF converter (Romanian)	RACAI	RO	http://ws2.racai.ro/TTL-ro-textconverter
129	Tokenizer and Sentence Splitter	RACAI	RO	http://ws2.racai.ro/TTL-ro-tokenizer
130	LexPar	RACAI	EN, RO	http://ws.racai.ro/lxpws.wsdl
131	TextProcessing	RACAI	EN, RO	http://nlp.racai.ro/WebServices/TextProcessing.asmx?WSDL
132	WordNetBrowser	RACAI	EN, RO	http://nlp.racai.ro/wnbrowser/Wordnet.asmx?wsdl
133	SearchRoWiki	RACAI	RO	http://nlp.racai.ro/WebServices/SearchRoWikiWebService.asmx?WSDL
134	Statistic for POS Tags	SfS – Thomas Zastrow	DE	http://www.thomas-zastrow.de:2001/xsltcf/transform?ss=posStatistics.xsl
135	GeoVisualization	SfS – Thomas Zastrow	DE	http://www.thomas-zastrow.de:2001/tcf2kml/gmap
136	GeoLocations	SfS – Thomas Zastrow	DE	http://www.thomas-zastrow.de:2001/tcf2kml/extract
137	Tcf2Negra	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/Tcf2NegraWS/Tcf2Negra
138	Hyphenation	SfS – Thomas Zastrow	DE	http://www.thomas-zastrow.de/wikDB/addToTCF?field=hyphenation

	Web service name	Institution(s) responsible	Languages	URL
139	Plaintext Converter (TCF 0.4, German)	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/convert-0_4/de
140	German Named Entity Recognizer	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/service-ner-de/model-h
141	Semantic Annotator (TCF 0.3, German)	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/gnet/rest/gnAnnotatorTCF03
142	Microsoft Word Converter (TCF 0.3, German)	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/doc/to/textcorpus/de
143	Plaintext Converter (TCF 0.3, German)	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/txt/to/textcorpus/de
144	RTF Converter (TCF 0.3, German)	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/rtf/to/textcorpus/de
145	Berkeley Parser – Berkeley NLP	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/BerkeleyParser/resources/parser
146	Tokenizer – OpenNLP Project	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/tokenizer
147	Tokenizer/Sentences – OpenNLP Project	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/toksent
148	POS Tagger – OpenNLP Project	SfS	DE	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/tagger
149	Plaintext Converter (TCF 0.4, English)	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/convert-0_4/en
150	RTF Converter (TCF 0.3, English)	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/rtf/to/textcorpus/en
151	Plaintext Converter (TCF 0.3, English)	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/txt/to/textcorpus/en

	Web service name	Institution(s) responsible	Lan gua ges	URL
152	Microsoft Word Converter (TCF 0.3, English)	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/doc/to/textcorpus/en
153	Constituent Parser – OpenNLP Project	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/parser
154	OpenNLP Named Entity Recognizer	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPne/rest/ners
155	Tokenizer – OpenNLP Project	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/tokenizer
156	Tokenizer/Sentences – OpenNLP Project	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/toksent
157	POS Tagger – OpenNLP Project	SfS	EN	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/tagger
158	Plaintext Converter (TCF 0.4, Spanish)	SfS	ES	http://weblicht.sfs.uni-tuebingen.de:8080/convert-0_4/es
159	Plaintext Converter (TCF 0.3, Spanish)	SfS	ES	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/txt/to/textcorpus/es
160	POS Tagger – OpenNLP Project	SfS	ES	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/tagger
161	Tokenizer/Sentences – OpenNLP Project	SfS	ES	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/toksent
162	Tokenizer – OpenNLP Project	SfS	ES	http://weblicht.sfs.uni-tuebingen.de:8080/openNLPservice/TCF_0_3/tokenizer
163	Plaintext Converter (TCF 0.4, Finnish)	SfS	FI	http://weblicht.sfs.uni-tuebingen.de:8080/convert-0_4/fi
164	Plaintext Converter (TCF 0.4, French)	SfS	FR	http://weblicht.sfs.uni-tuebingen.de:8080/convert-0_4/fr

	Web service name	Institution(s) responsible	Languages	URL
165	Microsoft Word Converter (TCF 0.3, French)	SfS	FR	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/doc/to/textcorpus/fr
166	RTF Converter (TCF 0.3, French)	SfS	FR	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/rtf/to/textcorpus/fr
167	Plaintext Converter (TCF 0.3, French)	SfS	FR	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/txt/to/textcorpus/fr
168	Plaintext Converter (TCF 0.4, Italian)	SfS	IT	http://weblicht.sfs.uni-tuebingen.de:8080/convert-0_4/it
169	Microsoft Word Converter (TCF 0.3, Italian)	SfS	IT	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/doc/to/textcorpus/it
170	RTF Converter (TCF 0.3, Italian)	SfS	IT	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/rtf/to/textcorpus/it
171	Plaintext Converter (TCF 0.3, Italian)	SfS	IT	http://weblicht.sfs.uni-tuebingen.de:8080/convert/dspindoc03/from/txt/to/textcorpus/it
172	Plaintext Converter (TCF 0.4, Romanian)	SfS	RO	http://weblicht.sfs.uni-tuebingen.de:8080/convert-0_4/ro
173	SBLEX	Språkbanken	SV	http://spraakbanken.gu.se/eng/research/swefn/sblex
174	SALDO	Språkbanken	SV	http://spraakbanken.gu.se/eng/saldo/ws
175	Latvian language morphological analysis/generation reference system	Tilde	LV	http://www.letonika.lv/morphology/
176	Wörterbuchnetz	Trier Center for Digital Humanities	DE	http://www.woerterbuchnetz.de
177	Apertium morphological analyser for old Catalan	Universitat d'Alacant	CA	http://xixona.dlsi.ua.es/~fran/oldca/servei.php
178	LXService	University of Lisbon	PO	http://nlxserv.di.fc.ul.pt/axis/services/LXService?wsdl

	Web service name	**Institution(s) responsible**	**Languages**	**URL**
179	ANNIE with GATE XML output	University of Sheffield	EN	https://gate.ac.uk/projects/clarin/USFD-webservices-ANNIE.pdf
180	ANNIE for English with RDF-XML output	University of Sheffield	EN	http://services.gate.ac.uk/clarin/annie-rdf/
181	Morphosyntactic analysis for English with MAF XML output	University of Sheffield	EN	https://gate.ac.uk/projects/clarin/USFD-webservices-morphosyntactic-analysis-EN.pdf
182	Chunking for English with SynAF XML output	University of Sheffield	EN	https://gate.ac.uk/projects/clarin/USFD-webservices-Chunker-EN.pdf
183	NER for German with RDF-XML output	University of Sheffield	DE	https://gate.ac.uk/projects/clarin/USFD-webservices-GERMAN-NER.pdf
184	Corpus query for Estonian corpora	University of Tartu	ET	http://www.keeleveeb.ee/
185	TaKIPI WS	WROCUT	PL	http://plwordnet.pwr.wroc.pl/clarin/ws/takipi/takipi.wsdl
186	SuperMatrix WS	WROCUT	PL	http://plwordnet.pwr.wroc.pl/clarin/ws/supermatrix/ supermatrix.wsdl
187	plWordNet WS	WROCUT	PL	http://plwordnet.pwr.wroc.pl/clarin/ws/plwordnet/plwordnet.wsdl

List of institutional acronyms

The following table lists acronyms of institutions used throughout the document:

Acronym	**Full name / location**
ASV Universitaet Leipzig	ASV – Department of Computer Science, NLP Group, Leipzig
BBAW	Berlin-Brandenburgische Akademie der Wissenschaften
CCL VMU	Center of Computational Linguistics, Vytautas Magnus University
CNR-ILC	Consiglio Nazionale delle Ricerche, Istituto di Linguistica Computazionale, Pisa
FFZG	University of Zagreb, Faculty of Humanities and Social Sciences, Institute/Department of Linguistics
ICS PAS	Linguistic Engineering Group, Institute of Computer Science, Polish Academy of Sciences, Warsaw
IDS	IDS – Institute for the German Language, Mannheim

Acronym	Full name / location
ILSP	Institute for Language and Speech Processing, Athens
IMCS	Institute of Mathematics and Computer Science, University of Latvia, Riga
IMS	IMS – Institute for Natural Language Processing, University of Stuttgart
ISJ ZRC SAZU	Fran Ramovš Institute of Slovenian Language ZRC SAZU Corpus Laboratory
IULA-UPF	Institut Universitari de Lingüística Aplicada, Universitat Pompeu Fabra, Barcelona
RACAI	Romanian Academy Research Institute for Artificial Intelligence
SfS	Department of Linguistics, University Tübingen
Språkbanken	Språkbanken, Dept. of Swedish Language, Göteborg University
WROCUT	Wrocław University of Technology

Bibliography

Acedański, S. (2010). *A Morphosyntactic Brill Tagger for Inflectional Languages*. In: Loftsson, H., Rögnvaldsson, E., Helgadóttir, S. (eds.) *Advances in Natural Language Processing*. Lecture Notes in Computer Science, vol. 6233, pp. 3–14. Springer (2010), http://ripper.dasie.mimuw.edu.pl/~accek/homepage/wp-content/papercite-data/pdf/ace10.pdf.

Acedański, S. and K. Gołuchowski, K. (2009). *A Morphosyntactic Rule-Based Brill Tagger for Polish*. In: *Recent Advances in Intelligent Information Systems*, pp. 67–76. Academic Publishing House EXIT, Kraków, Poland (June 2009). http://ripper.dasie.mimuw.edu.pl/~accek/homepage/wp-content/papercite-data/pdf/acegol09.pdf.

Bański, P. (2010). *Why TEI stand-off annotation doesn't quite work and why you might want to use it nevertheless*. Presented at Balisage: The Markup Conference 2010, Montréal, Canada, August 3-6, 2010. In *Proceedings of Balisage: The Markup Conference 2010. Balisage Series on Markup Technologies*, vol. 5 (2010), pp. 1–29. http://www.balisage.net/Proceedings/vol5/html/Banski01/BalisageVol5-Banski01.html.

Bański, P. and A. Przepiórkowski. (2009). *Stand-off TEI Annotation: the Case of the National Corpus of Polish*. In: *The proceedings of the LAW-III workshop at ACL-IJCNLP 2009*, Singapore, pp. 64–67. http://aclweb.org/anthology-new/W/W09/W09-3011.pdf.

Biemann, C., S. Bordag, U. Quasthoff and C. Wolff. (2004). *Web Services for Language Resources and Language Technology Applications*. In: *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal.

Bird, S. and G. Simons. (2003). *Seven dimensions of portability for language documentation and description*. Language 79(3), pp. 557–582.

Broda B., M. Marcińczuk and M. Piasecki. (2010). *Building a node of the accessible language technology infrastructure*. In: *Proceedings of the 7th conference on International Language Resources and Evaluation (LREC 2010)*, Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odjik, Stelios Piperidis, Mike Rosner, Daniel Tapias (eds.), May 19-21. Valetta, Malta.

Buczyński A. (2007). *An Implementation of Combined Partial Parser and Morphosyntactic Disambiguator*. In: *Proceedings of the ACL 2007 Student Research Workshop*, Prague, June 2007. Association for Computational Linguistics, pp. 13–18, http://acl.ldc.upenn.edu/P/P07/P07-3003.pdf.

Burnard, L. and S. Bauman, editors. (2008). *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Oxford. http://www.tei-c.org/Guidelines/P5/.

Clergerie É., de la and Clément L. (2004). *MAF — Morphosyntactic Annotation Framework*. Slides from ISO TC37 SC4 meeting, Pisa, November 16-19. ftp://ftp.inria.fr/INRIA/Projects/Atoll/Eric.Clergerie/MAF-slides-PISA04.pdf.

Dipper, S. (2005). *Stand-off representation and exploitation of multi-level linguistic annotation*. In: *Proceedings of Berliner XML Tage 2005* (BXML 2005), pp. 39–50, Berlin.

Erjavec T. (2010). *MULTEXT-East and TEI: an Investigation of a Schema for Language Engineering and Corpus Linguistics*. In: D. Tufiş and C. Forăscu. *Multilinguality and Interoperability in Language Processing with Emphasis on Romanian*. Editura Academiei Române. Bucureşti, 2010, pp. 19–47.

Głowińska, K. (1999). *Taksonomia morfologiczna dla Słownika frekwencyjnego* [in Polish]. http://www.mimuw.edu.pl/polszczyzna/pl196x/doc/files/taksonomia.pdf.

Goecke, D., D. Metzing, H. Lüngen, M. Stührenberg and A. Witt. (2010). *Different views on markup, distinguishing levels and layers*. In: *Linguistic modeling of information and markup languages. Contributions to language technology*. Springer Netherlands, pp. 1–21.

Hajic, J. and B. Hladka. (1998). *Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset*. In: *Proceedings of ACL/Coling'98*, European Language Resources Association (ELRA), pp. 483–490, Montreal, Canada. http://shadow.ms.mff.cuni.cz/pdt/Morphology_and_Tagging/Tagging/Doc/References/col98.pdf

Ide, N. and L. Romary. (2007). *Towards International Standards for Language Resources*. In: Dybkjaer, L., Hemsen, H., Minker, W. (eds.), *Evaluation of Text and Speech Systems*, Springer, 263-84.

Heid, U., H. Schmid, K. Eckart and E. Hinrichs. (2008). *A Corpus Representation Format for Linguistic Web Services: The D-SPIN Text Corpus Format and its Relationship with ISO Standards*. In: *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*. ELRA, Marrakech.

Helbig, H. (2001). *Die semantische Struktur natürlicher Sprache: Wissensrepräsentation mit MultiNet*. Springer, Berlin.

Henrich V., E. Hinrichs, M. Hinrichs and T. Zastrow (2010). *Service-oriented architectures: from Desktop Tools to Web Services and Web Applications*. In: D. Tufiş and C. Forăscu. *Multilinguality and Interoperability in Language Processing with Emphasis on Romanian*. Editura Academiei Române. Bucureşti, 2010, pp. 69–92.

Hinrichs, M., T. Zastrow and E. Hinrichs. (2010). *Weblicht: Web-based LRT Services in a Distributed eScience Infrastructure*. In: *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*. ELRA, Valletta, Malta.

Ide, N. (1998). *Corpus Encoding Standard: SGML guidelines for encoding linguistic corpora*. In: *Proceedings of the 1st International Conference on Language Resources and Evaluation*, LREC 1998, pp. 463–470, Granada. ELRA.

Ide, N. and Priest-Dorman, G. (1995). *Corpus Encoding Standard*. http://www.cs.vassar.edu/CES/.

Ide, N., P. Bonhomme and L. Romary. (2000). *XCES: An XML-based standard for linguistic corpora*. In: *Proceedings of the 2nd International Conference on Language Resources and Evaluation*, LREC 2000, Athens. ELRA, pp. 825–830.

Ide, N. and L. Romary. (2007). *Towards International Standards for Language Resources*. In: Dybkjaer, L., Hemsen, H., Minker, W. (eds.), *Evaluation of Text and Speech Systems*, Springer, pp. 263–84.

ISO:24610-1. (2005). *Language resource management – feature structures – part 1: Feature structure representation*. ISO/DIS 24610-1, 2005-10-20.

ISO:24611. (2008). *Language resource management — Morpho-syntactic annotation framework*. ISO/DIS 24611, 2008-12-07.

ISOcat Glossary (2010). http://www.isocat.org/interface/JSXAPPS/ISOcat/help/ISOcatGlossary.html

Janus D. and A. Przepiórkowski. (2006). *POLIQARP 1.0: Some technical aspects of a linguistic search engine for large corpora*. In: Waliński J., K. Kredens and S. Goźdź-Roszkowski (eds.) *Proceedings of Practical Applications of Linguistic Corpora 2005*, Peter Lang, Frankfurt am Main, Germany.

König, E., W. Lezius and H. Voormann. (2003). *TIGERSearch 2.1: User's Manual*. Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart.

Kurcz, I., A. Lewicki, J. Sambor, K. Szafran and J. Woronczak. (1990). *Słownik frekwencyjny polszczyzny współczesnej* [in Polish]. Wydawnictwo Instytutu Języka Polskiego PAN, Cracow.

Lezius, W. (2002). *TIGERSearch — ein Suchwerkzeug für Baumbanken*. In: S. Busemann (ed.) *Proceedings der 6. Konferenz zur Verarbeitung natürlicher Sprache* (KONVENS 2002), Saarbrücken.

Mengel, A. and W. Lezius. (2000). *An XML-based encoding format for syntactically annotated corpora*. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation*, LREC 2000, Athens. ELRA, pp. 121–126.

Młodzki R. and A. Przepiórkowski. (2009). *The WSD Development Environment*. In *Proceedings of the 4th Language & Technology Conference*, pp. 185–189. Poznań, Poland.

Ogrodniczuk, M. (2003). *Nowa edycja wzbogaconego korpusu słownika frekwencyjnego* [in Polish]. In: Gajda S. (ed.) *Językoznawstwo w Polsce. Stan i perspektywy*, Polska Akademia Nauk – Komitet Językoznawstwa, Uniwersytet Opolski – Instytut Filologii Polskiej. Opole 2003, pp. 181–190, ISBN 83-86881-36-4. http://bc.klf.uw.edu.pl/104/.

Patejuk, A. and A. Przepiórkowski. (2010). *ISOcat Definition of the National Corpus of Polish Tagset*. In: *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*, Workshop on LRT Standards, Valletta, Malta.

Piasecki, M. and A. Wardyński. (2006). *Multiclassifier Approach to Tagging of Polish*. In: *Proceedings of 1st International Symposium Advances in Artificial Intelligence and Applications*. http://www.proceedings2006.imcsit.org/pliks/148.pdf.

Przepiórkowski A. (2004). *The IPI PAN Corpus: Preliminary version*. Institute of Computer Science, Polish Academy of Sciences. Warsaw, Poland. http://nlp.ipipan.waw.pl/~adamp/Papers/2004-corpus/book_en.pdf.

Przepiórkowski, A. (2005). *The IPI PAN Corpus in Numbers*. In: Vetulani, Z. (ed.) *Proceedings of the 2nd Language & Technology Conference*. Poznań. http://nlp.ipipan.waw.pl/~adamp/Papers/2005-ltc-numbers/ipipan-corpus.pdf.

Przepiórkowski, A. (2008). *Powierzchniowe przetwarzanie języka polskiego* [in Polish]. Academic Publishing House EXIT, Warszawa. B5, 322 pages.

Przepiórkowski, A. (2009a). *TEI P5 as an XML Standard for Treebank Encoding*. In: Passarotti, M., Przepiórkowski, A., Raynaud, S., Van Eynde, F. (eds.) *Proceedings of the 8th International Workshop on Treebanks and Linguistic Theories (TLT8)*, pp. 149–160. Milan, Italy.

Przepiórkowski, A. (2009b). *A comparison of two morphosyntactic tagsets of Polish*. In V. Koseska-Toszewa, L. Dimitrova, and R. Roszko (eds), *Representing Semantics in Digital Lexicography: Proceedings of MONDILEX Fourth Open Workshop*, pp. 138–144, Warsaw.

Przepiórkowski, A. and P. Bański. (2009a). *Which XML standards for multilevel corpus annotation?* In: *Proceedings of the 4th Language & Technology Conference*. Poznań, Poland.

Przepiórkowski, A. and P. Bański. (2009b). *XML Text Interchange Format in the National Corpus of Polish*. In: S. Goźdź-Roszkowski (ed.) *Proceedings of the Practical Applications in Language and Computers Conference (PALC 2009)*. Peter Lang, Frankfurt am Main, Germany.

Przepiórkowski, A. and P. Bański. (2010). *The TEI and the NCP: the model and its application*. In: *Proceedings of the 7th International Conference on Language Resources and Evaluation* (LREC 2010), Workshop on Language Resources: From Storyboard to Sustainability and LR Lifecycle Management, Valletta, Malta.

Przepiórkowski, A. and A. Buczyński. (2007). *Spejd: Shallow parsing and disambiguation engine*. In: *Proceedings of the 3rd Language & Technology Conference*. Poznań, http://nlp.ipipan.waw.pl/~adamp/Papers/2007-ltc-spade/Spade.pdf.

Przepiórkowski A. and G. Murzynowski. (2009). *Manual annotation of the National Corpus of Polish Anotatornia*. In: S. Goźdź-Roszkowski (ed.) *Proceedings of the Practical Applications in Language and Computers (PALC 2009)*. Peter Lang, Frankfurt am Main, Germany.

Przepiórkowski, A. and M. Woliński. (2003a). *A Flexemic Tagset for Polish*. In: *Proceedings of Morphological Processing of Slavic Languages (EACL 2003)*. http://nlp.ipipan.waw.pl/~adamp/Papers/2003-eacl-ws12/ws12.pdf.

Przepiórkowski, A. and M. Woliński. (2003b). *The unbearable lightness of tagging: A case study in morphosyntactic tagging of Polish*. In: *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, EACL 2003, pp. 109–116.

Przepiórkowski, A., R. L. Górski, B. Lewandowska-Tomaszczyk and M. Łaziński. (2008). *Towards the National Corpus of Polish*. In: *Proceedings of the 6th Language Resources and Evaluation Conference (LREC 2008)*. Marrakesh, Morocco: 827-830.

Przepiórkowski, A., R. L. Górski, M. Łaziński and P. Pęzik. (2010). *Recent Developments in the National Corpus of Polish*. In: *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*. Valletta, Malta, ELRA.

Przepiórkowski A., Z. Krynicki, Ł. Dębowski, M. Woliński, D. Janus and P. Bański. (2004). *A Search Tool for Corpora with Positional Tagsets and Ambiguities*. In: *Proceedings of the 4th*

International Conference on Language Resources and Evaluation (LREC 2004), pp. 1235–1238. Lisbon, Portugal.

Saloni, Z., W. Gruszczyński, M. Woliński and R. Wołosz. (2007). *Grammatical Dictionary of Polish – Presentation by the Authors*. Studies in Polish Linguistics 4, pp. 5–25. http://www.ijp-pan.krakow.pl/sipl/saloni.pdf, see also http://www.info.univ-tours.fr/~savary/Polonium/Papers/prezentacja-SGJP-Tours.pdf.

Saloni, Z., W. Gruszczyński, M. Woliński and R. Wołosz. (2007). *Słownik gramatyczny języka polskiego* [in Polish]. Wiedza Powszechna, 177 pp., CD.

Simons, G. F. and S. Bird. (2008). *Toward a global infrastructure for the sustainability of language resources*. In: *Proceedings of the 22nd Pacific Asia Conference on Language, Information and Computation: PACLIC 22*, pp. 87–100.

Świdziński, M. (1992). *Formal grammar of Polish* [in Polish]. Warsaw University Dissertations, Warsaw, Poland.

Váradi, T., S. Krauwer, P. Wittenburg, M. Wynne, and K. Koskenniemi. (2008). *CLARIN: Common language resources and technology infrastructure*. In: European Language Resources Association (ELRA) (eds.), *Proceedings of the 6th international language resources and evaluation (LREC 2008)*. Marrakech, Morocco.

Witt, A., U. Heid, F. Sasaki and G. Sérasset. (2009). *Multilingual language resources and interoperability*. In *Language Resources and Evaluation*, vol. 43:1, pp. 1–14. http://www.springerlink.com/content/pp7x18343662g5k2/fulltext.pdf.

Witt, A., G. Rehm., E. Hinrichs, T. Lehmberg and J. Stemann. (2009). *SusTEInability of linguistic resources through feature structures*. Literary and Linguistic Computing, 24(3), pp. 363–372.

Woliński, M. (2004). *Computer-aided verification of Świdziński's grammar* [in Polish]. PhD. diss., Institute of Computer Science, Polish Academy of Sciences. Warsaw, Poland. http://www.ipipan.waw.pl/ wolinski/publ/mw-phd.pdf.

Woliński, M. (2006). *Morfeusz – a practical tool for the morphological analysis of Polish*. In: Kłopotek, M. A., S. T. Wierzchoń and K. Trojanowski (eds.) *Proceedings of the International IIS: IIPWM'06 Conference*, pp. 511–520, Wisła, Poland.