WebLicht Tool Integration

Erhard Hinrichs, Marie Hinrichs, Wei Qiu
University of Tübingen

Bundesministerium
für Bildung
und Forschung

Introduction

- WebLicht (Web-based Linguistic Chaining Tool)
  - Motivation
  - Overview of the architecture
  - TCF (text corpus format) data-exchange format
- Integration
  - Repository Sets
  - Comet (CMDI orchestration metadata tool)
  - Center Registry Update
- Tool Testing
  - Bombard (tool scalability testing)
  - Awesome (verify actual tool input/output against metadata)
- Usage Statistics
  - Piwik
- Links

Motivation

Many natural language processing tools must be invoked from the command line or from programming code.

- Sometimes difficult to install
- Not always available for all operating systems
- Confusing for those not accustomed to running command line tools or who don't like to program
- Not possible to create some annotations with one tool and other annotations with a different tool
  - Input/Output formats differ

WebLicht

- Makes NLP tools available online and enables users to mix-and-match tools to suit their needs.
- The WebLicht web interface guides the construction and execution of processing chains, and provides visualization of results.
- NLP tools are made available by CLARIN centers as web services.

Overview

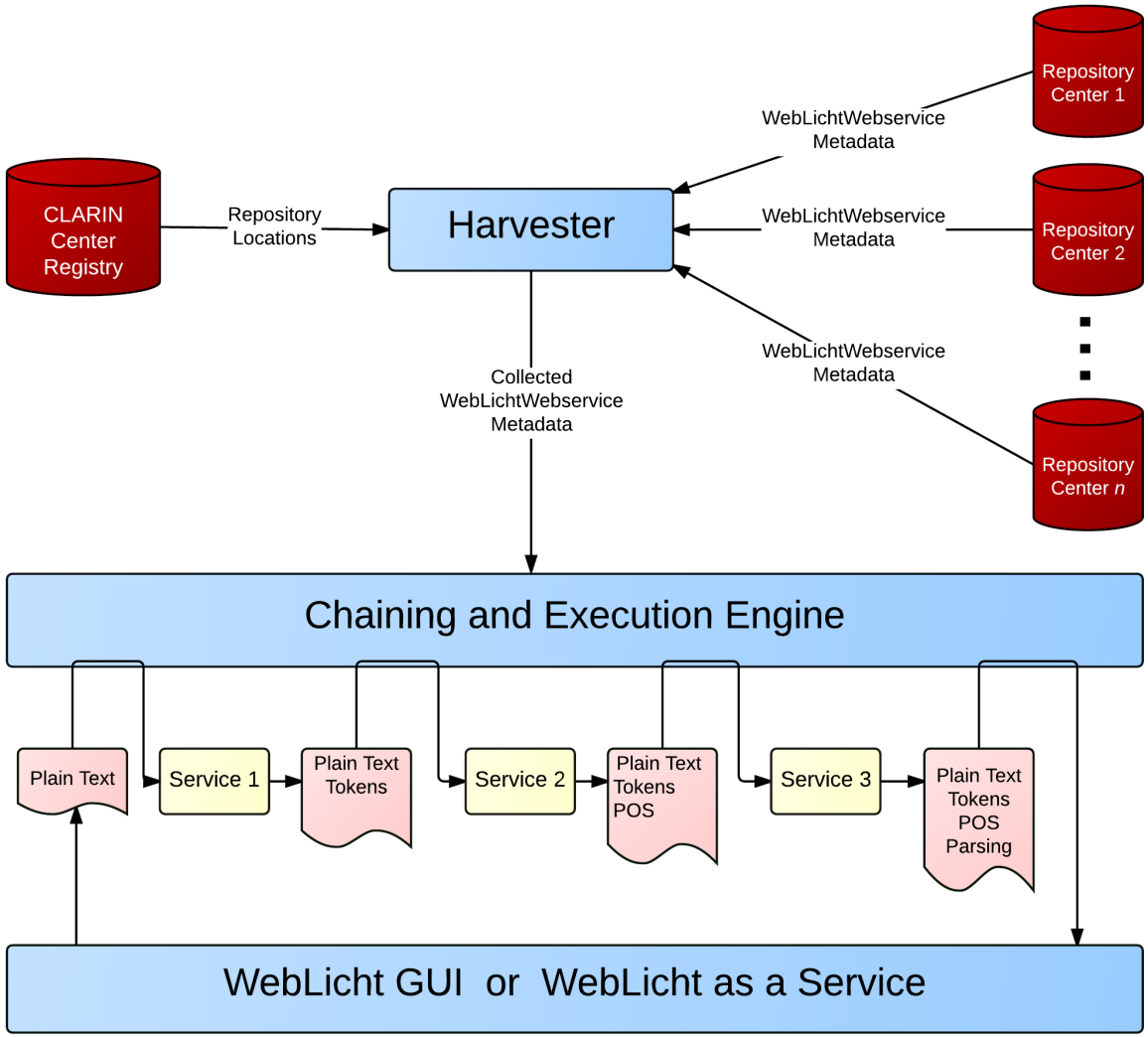
WebLicht is an execution environment for natural language processing pipelines:

- Uses a service-oriented architecture (SOA)
- Web services are combined to form a chain
- Chains are executed via sequential HTTP POST requests to services on the chain
- The output of service *n* is the input to service *n+1* in the chain
- Most services use Text Corpus Format (TCF) as their input and output
- Services add one or more annotation layer(s)

Main Components

- **Services** for data processing, distributed
- **Repositories** containing metadata about the services
- **Harvester**
  - Collects service metadata from repositories
- **Chaining engine**
  - Guides the construction of valid chains
  - Executes chains
- **Web interface** for building and executing chains
- **WaaS** (WebLicht as a Service)
  - RESTful web service to execute chains
  - Can be called from command-line, scripts, etc.

Component Interactions



TCF: Motivation

TCF (text corpus format) is WebLicht's data-exchange format

- Service chains are meant to be built flexibly
  - Mix-n-match, Lego style
  - Not "one service does it all"
- Output of service *n* is input of service *n+1*
- Services rely on the annotations in the input
- Services must be able to
  - Read annotations produced by previous services
  - Write annotations for use by later services

TCF: Details

- Each service adds one or more annotation layers
- Services are not permitted to change or remove existing annotations

TCF: Details

TCF is an XML format for storing linguistic annotations

- Text, tokens, sentences, lemmas, part-of-speech, morphology, parsing, etc
- Standoff format
  - Each type of annotation is stored in a separate XML element
  - token layer can be seen as the central, atomic element to which other annotation layers refer

TCF: Example

```
<?xml version="1.0" encoding="UTF-8"?>
<D-Spin xmlns="http://www.dspin.de/data" version="0.4">
  <MetaData xmlns="http://www.dspin.de/data/metadata">
        The execution engine records metadata
        about the services used to create the document here.
  </MetaData>
  <TextCorpus xmlns="http://www.dspin.de/data/textcorpus"lang="de">
    <text>Karin fliegt nach New York. Sie will dort Urlaub machen.</text>
```

TCF: Example

```
<tokens>
    <token ID="t_0">Karin</token>
    <token ID="t_1">fliegt</token>
    <token ID="t_2">nach</token>
    <token ID="t_3">New</token>
    <token ID="t_4">York</token>
    <token ID="t_5">.</token>
    <token ID="t_6">Sie</token>
    <token ID="t_7">will</token>
    <token ID="t_8">dort</token>
    <token ID="t_9">Urlaub</token>
    <token ID="t_10">machen</token>
    <token ID="t_11">.</token>
</tokens>
```

- Each token has a unique ID
- All other annotation layers (except the text layer) reference tokens directly or indirectly

TCF: Example

```
<sentences>
   <sentence ID="s_0" tokenIDs="t_0 t_1 t_2 t_3 t_4 t_5"></sentence>
   <sentence ID="s_1" tokenIDs="t_6 t_7 t_8 t_9 t_10 t_11"></sentence>
</sentences>
```

TCF: Example

```
    <POStags tagset="stts">
        <tag ID="pt_0" tokenIDs="t_0">NE</tag>
        <tag ID="pt_1" tokenIDs="t_1">VVFIN</tag>
        <tag ID="pt_2" tokenIDs="t_2">APPR</tag>
        <tag ID="pt_3" tokenIDs="t_3">NE</tag>
        <tag ID="pt_4" tokenIDs="t_4">NE</tag>
        <tag ID="pt_5" tokenIDs="t_5">$.</tag>
        <tag ID="pt_6" tokenIDs="t_6">PPER</tag>
        <tag ID="pt_7" tokenIDs="t_7">VMFIN</tag>
        <tag ID="pt_8" tokenIDs="t_8">ADV</tag>
        <tag ID="pt_9" tokenIDs="t_9">NN</tag>
        <tag ID="pt_10" tokenIDs="t_10">VVINF</tag>
        <tag ID="pt_11" tokenIDs="t_11">$.</tag>
    </POStags>
    ...
  </TextCorpus>
</D-Spin>
```

TCF: Processing

- More detailed information about TCF is available in the Developers Manual on the weblicht-wiki:
  - schema
  - online validator
  - Tutorial for reading/writing TCF using the Java library

Building a Tool

ToDo's for implementing a WebLicht tool:

- Gather resources needed for the tool
  - Models, lists, etc
- Implement the tool as a web service
- Deploy the web service on a public server

Integrating a Tool into WebLicht

Making a tool visible to WebLicht:

- Create a **set** in your center's repository for WebLicht web services
- Create CMDI metadata for the tool and place it in the repository set
- Assign a PID to the tool
- Update your center's data in the Center Registry to enable harvesting by WebLicht

Repository Set

- Each center offering WebLicht services must create a **set** in their repository for metadata about those web services
  - Metadata for WebLicht services should be part of this set
  - The WebLicht harvester only retrieves those sets
  - Avoids harvesting of unnecessary data
- Procedure for creating a **set** depends on the repository software (fedora, dspace,…)
- Set name can be anything, but keep it simple (no spaces or special characters)
  - e.g. WebLichtWebServices

Creating Tool Metadata with Comet

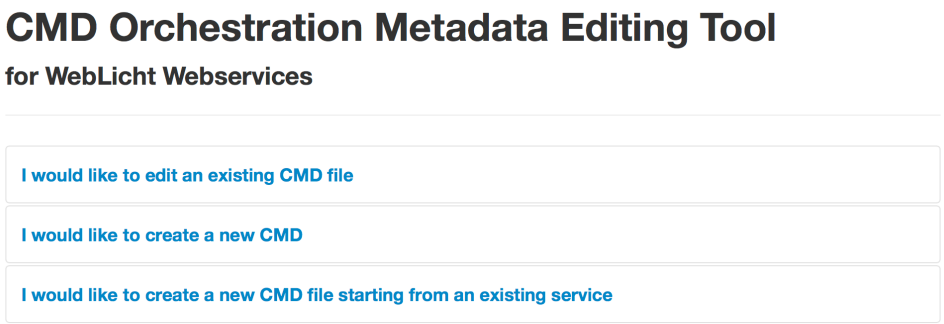
Comet (CMD Orchestration Metadata Editing Tool) can be used to create CMDI metadata using the **WebLichtWebService** profile

- Two main sections
  - General Information
    - Tool name, description, PID,…
  - "Orchestration" Information
    - Tool language, input/output specification
- Needed by the chaining and execution engine to
  - Guide users in building valid chains
  - Execute a processing chain

Creating Tool Metadata With Comet

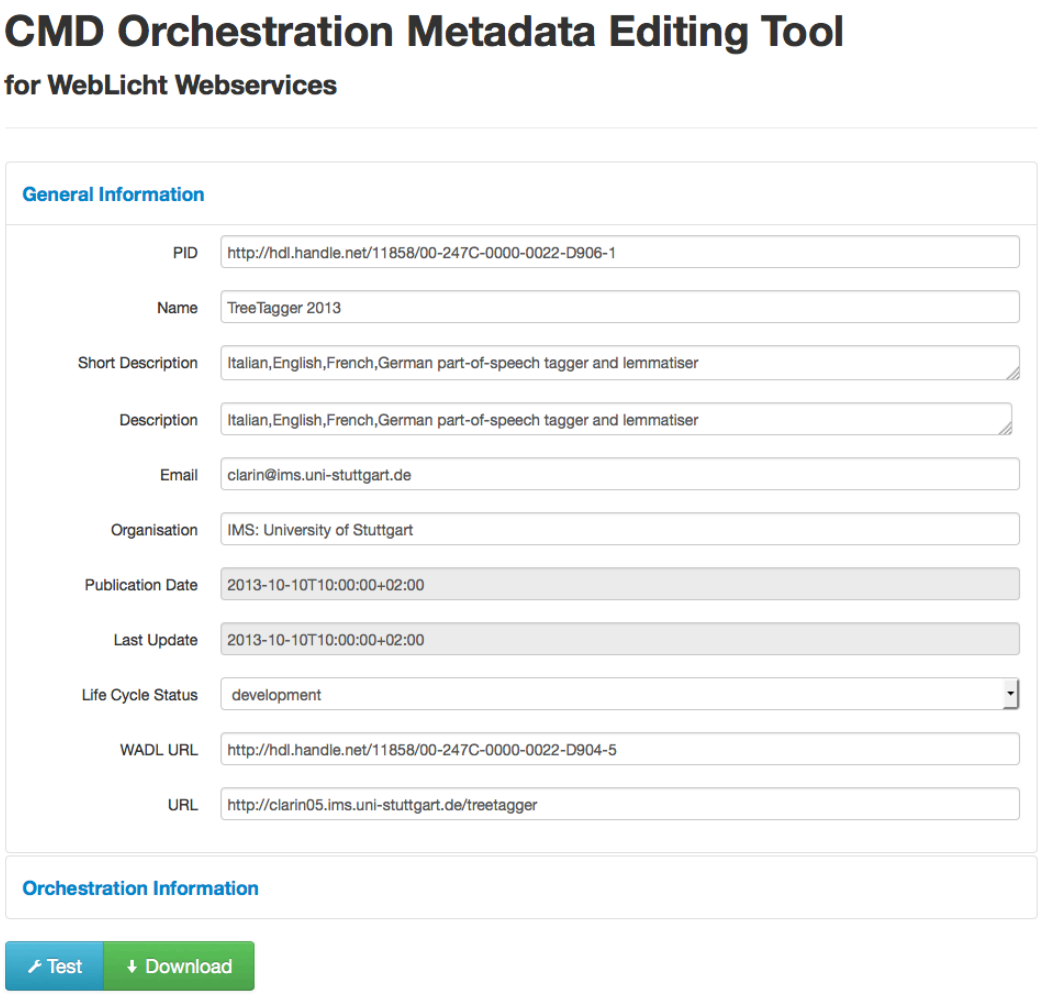
Comet can be used to create metadata for a WebLicht webservice:

- create from scratch
- upload metadata for editing
- use existing metadata as a template



Creating Tool Metadata With Comet

General Information



Creating Tool Metadata With Comet

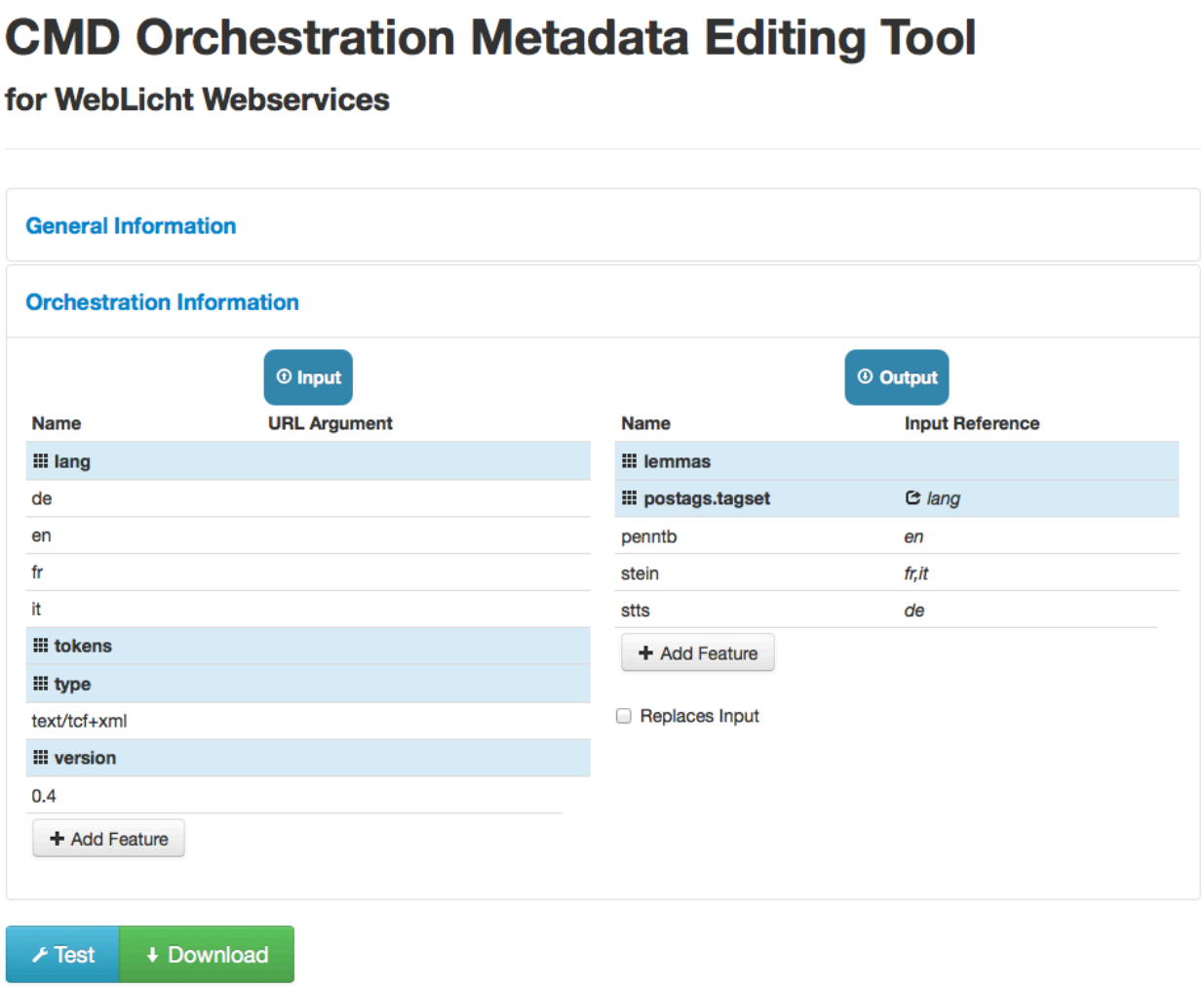
Orchestration Information

Input (TCF):

- tokens
- lang (de|en|fr|it)

Output (adds to input):

- lemmas
- part-of-speech tags
  - tagset depends on input language



Update Center Registry Data

Ask your Center Registry contact person to add these elements to your center‘s data:

- **WebServicesSet**
  - The name of the set in your repository containing CMDI metadata for WebLicht web services
  - No spaces or special characters allowed
  - e.g. WebLichtWebServices
- **WebServiceType:** WebLicht
- **MetadataScheme**: CMDI
- **OaiAccessPoint**
  - URL for harvesting
  - e.g. http://repository.my.org.countrycode/oaiprovider

Tool Testing

Some useful testing software for tool developers:

- Bombard for scalability testing
- Awesome for testing metadata correctness

Tool Testing: Bombard

Bombard for scalability testing

- Simulates users invoking tool chains
- Flexible test-case configuration
- Multiple test cases can be run simultaneously during one “bombardment”
- Reports statistics for each tool:
  - successes/failures
  - average processing times

Improving Scalability

One way to improve scalability of web services:

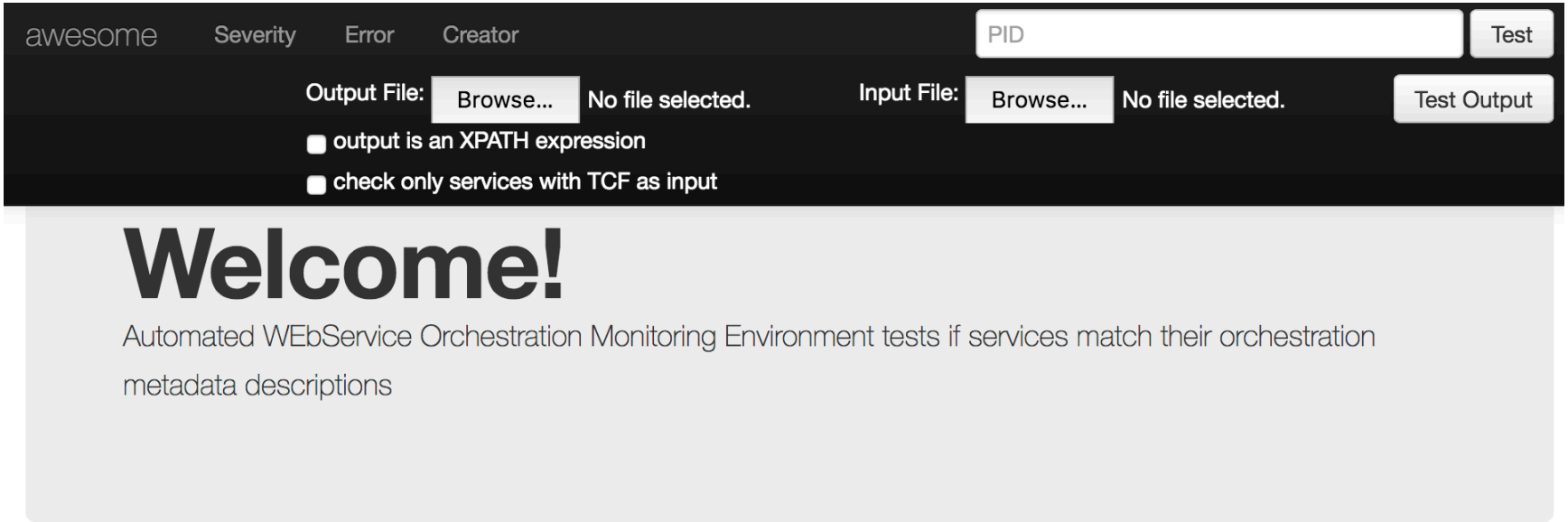
- Jesque - a distributed task queue framework
- Exploit the parallel processing capabilities of modern servers and computing clusters
  - parallel processing within requests
  - concurrent processing of requests
  - guarantees with respect to usage of resources
  - fairness (small requests should not have to wait for large ones)

Tool Testing: Awesome

Awesome (Automated WEbService Orchestration Monitoring Environment)

- Tests that services' orchestration metadata matches actual usage
- POSTs a small test file containing the required input annotations to the service
- Checks that the output returned contains the expected additional annotation layers
- Runs tests automatically at regular time intervals
  - Reports grouped by creator, error code, or severity
- Can test individual services
  - Specify PID, input file, expected output file

Tool Testing: Awesome



Statistics with Piwik

- WebLicht uses the CLARIN installation of the analytics platform Piwik to gather user statistics
- Each web service invocation from WebLicht or WaaS is recorded

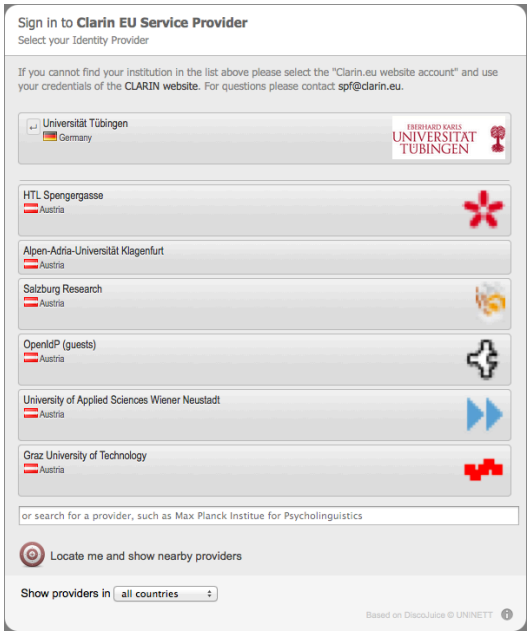
weblicht.sfs.uni-tuebingen.de	431	53691
clarin05.ims.uni-stuttgart.de	177	18285
/cgi-bin/dspin/tokeniser4.perl	61	3997
/RFTaggerMorph	48	14135
/treetagger2008	45	126
/cgi-bin/dspin/bitpar4.perl	18	20
/cgi-bin/dspin/tei2tcf3.perl	2	3
/rftagger	2	3
/cgi-bin/dspin/smor4.perl	1	1
dspin.dwds.de:8080	26	34
ws1-clarind.esc.rzg.mpg.de	12	13
kaskade.dwds.de	6	6
chopin.ipipan.waw.pl:8083	1	1

Current Tools

- Languages:
  - German, English most represented
  - Some tools for French, Italian, Spanish,…
  - Need support for more languages
- Tools:
  - Tokenization / Sentence splitters
  - POS taggers
  - Lemmatizers
  - Morphological analyzers
  - Parsers

WebLicht: Login

http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/



Links

- Developer Manual
  - Tutorial for creating WebLicht web services
  - TCF in detail
- Comet for creating tool metadata
- Bombard: *wlsupport [at] sfs.uni-tuebingen.de*
- Awesome tests web services against metadata
- Harvester list of harvested WebLicht services
- Jesque to improve scalability
- WaaS WebLicht as a Service

Thank You!