

# TextGrid-Tools II

Version 05.3.2008

Arbeitspaket AP 2

verantwortliche Partner: TU Darmstadt, Saphor GmbH

## TextGrid

Modulare Plattform für verteilte und kooperative  
wissenschaftliche Textdatenverarbeitung –  
ein Community-Grid für die Geisteswissenschaften



Bundesministerium  
für Bildung  
und Forschung

Projekt: **TextGrid**

Teil des D-Grid Verbundes und der deutschen e-Science Initiative

BMBF Förderkennzeichen: 07TG01A-H

Laufzeit: Februar 2006 - Januar 2009

Dokumentstatus: final

Verfügbarkeit: öffentlich

Autoren:

Andreas Aschenbrenner, SUB

Stefan Büdenbender, Uni Trier

Fotis Jannidis, TU Darmstadt

Martin Haase, DAASI

Christoph Ludwig, FH Worms

Wolfgang Pempe, Saphor

Andrea Rapp, Uni Trier

Thorsten Vitt, TU Darmstadt

Werner Wegstein, Uni Würzburg

Andrea Zielinski, IDS Mannheim

# Inhalt

Inhalt.....	4
Vorbemerkungen.....	5
1 Arbeiten im Berichtszeitraum.....	5
1.1 Änderungen im Arbeitsplan.....	5
1.2 Roadmap.....	5
1.3 Web Service Integration.....	6
1.4. TextGrid Registry.....	7
2 Im Berichtszeitraum begonnene Module.....	8
2.1 Recherchetool.....	8
2.2 Streaming Editor I (XSLT-Komponente).....	15
2.3 Projektmanagement.....	17
2.4 Metadaten-Annotation.....	20
3 Im Berichtszeitraum weiterentwickelte Module.....	24
3.1 Eclipse-GUI (vormals: Rich-Client-Plattform).....	24
3.2 Tokenizer.....	26
3.3 Morphologische Analyse, Lemmatisierer.....	27
3.4 Workflow-Editor.....	28
3.5 XML-Editor.....	29
4 Planungsstand restliche Module.....	30
4.1 Grafischer Link-Editor.....	30
4.2 Link-Editor Text.....	32
4.3 Bibliographietool.....	33
4.4 Sortieren.....	37
4.5 Streaming Editor II.....	40
4.6 Kollationierung.....	42
4.7 Text Publisher Print.....	44
4.8 Text Publisher Web.....	45
4.9 OCR.....	45
5. Literatur.....	46

## Vorbemerkungen

Das vorliegende Dokument soll einen Überblick über die mit Arbeitspaket 2 (AP2) verbundenen Aktivitäten in den vergangenen 12 Monaten vermitteln, sowohl was die Entwicklung neuer Tools angeht (Abschnitt 2), als auch die Weiterentwicklung bestehender Anwendungen (Abschnitt 3). Der Planungsstand bezüglich der in den kommenden 12 Monaten zu implementierenden Tools ist in Abschnitt 4 dokumentiert.

## 1 Arbeiten im Berichtszeitraum

### 1.1 Änderungen im Arbeitsplan

Entgegen dem in Report 2.1 aufgestellten Arbeitsplan verzögert sich die Fertigstellung des Recherchertools (Entwicklung seit 2. Quartal 2007) aus verschiedenen Gründen (umfassendere Aufgaben, Umstellungen in der Middleware und im Metadatenformat, Probleme mit Frameworks) und wird voraussichtlich erst im 1. Quartal 2008 erfolgen. Die damit zusammenhängenden Tools Bibliographietool und Link-Editor Text, für die bereits Vorarbeiten existieren, können auch erst dann fertig gestellt werden. Dafür werden bereits jetzt Schritte zur Entwicklung des Text Publishers (Web) unternommen – zur Darstellung der Rechercheergebnisse im Web-Interface.

Für sämtliche Werkzeuge in TextGrid wird es mit TextGridLab eine integrierte, Plugin-basierte Benutzungsschnittstelle auf der Basis der *Eclipse Rich Client Platform* geben. In den vergangenen 12 Monaten wurden an der TU Darmstadt verstärkt Integrations-, Koordinations- und Architekturtätigkeiten im Zusammenhang mit dieser Nutzerschnittstelle durchgeführt sowie gemeinsame Komponenten (wie die Modellierung und der Zugriff auf die im Grid gespeicherten TextGrid-Objekte) entwickelt, die die Grundlage für die Eclipse-basierten Werkzeuge bzw. Frontends auch der anderen Projektpartner bilden. Durch diese Tätigkeiten wird die Entwicklung von Teilfunktionalität des XML- und Linkeditors ins nächste Jahr (für das im ursprünglichen Antrag Integrationsmaßnahmen vorgesehen waren) verschoben.

### 1.2 Roadmap

Um TextGrid bei den anvisierten Nutzergruppen bekannt zu machen und rechtzeitig Feedback einzuholen, wurde für den Rest der Projektdauer eine Roadmap erstellt:

#### **Nutzer (Besucher, Datenerzeuger)**

Im Jahr 2008 sind vor allem erste Nutzertests, die Veröffentlichung des Source Codes und eine grundlegende Dokumentation geplant. Organisatorische Strukturen zur Rückführung von Änderungen am Code und der Aufbau eines Community-Portals sind Pläne für TextGrid II.

#### **Entwickler**

Für die Projektphase "TextGrid I" gibt es zwei primäre Ziele:

- (A) externe Entwickler befähigen (Dokumentation, Samples, Tutorial-Workshop),
- (B) Kommunikation mit Interedition und möglichen anderen Partnern, um eine Konvergenz der technischen Ansätze und Konzepte zu erlangen. Der Aufbau eines Community-Portals, in

dem sich Entwickler untereinander austauschen können, Feedback wieder rückgeführt werden kann, etc. ist Aufgabe von TextGrid II.

## Institutionen

Identifizierung von Ansprechpartnern und Entscheidungsträgern, Sondierung bezüglich möglicher Kooperationsformen, Anbindung von Daten und Bereitstellung eigener Ressourcen.

## Termine

- 13.-16.2.2008: **Veröffentlichung der Roadmap** auf der MGH-Tagung und AG germanistische Edition
- Frühjahr 2008 - Test der Utilities, z.B. mit dMGH
- 25.-29. Juni: Vorstellung auf der Digital Humanities 2008
  - Veröffentlichung einer downloadbaren **Teaser Version** (Campe-Daten; eine Vorabversion des TextGridLab mit verminderter Funktionalität)
  - ein **Satellite Meeting** unter dem Banner von TextGrid / Interedition; Thema: "Service Networks in the Digital Humanities"; demonstriert werden Services von TextGrid: Streaming Editor, Sortieren, Workflow
- Frühsommer 2008 - erste Nutzertests mit ausgewählten Personen
- 1. September 2008: "**TextGrid Beta**"
  - Bereitstellung eines downloadbaren Produkts des TextGridLabs für Beta-Tester (öffentlicher Aufruf)
  - Veröffentlichung des kompletten Source-Codes
  - Dokumentation und Tutorials mit beispielhafter Implementierung von 'neuen' Diensten auf der TextGrid Website
- Oktober/November 2008 - **Entwickler-Workshop** zur Programmierung in TextGrid.
  - Andocken an TextGrid Utilities
  - TextGrid Service Netzwerk Konzepte;
  - eine Hands-On Session zur Programmierung eines Services, das Daten aus dem Grid liest und Funktionalität eines anderen Services wiederverwendet
- Oktober/November 2008 - **Informationsveranstaltung Institutionen**  
Vorstellung TextGrid, auch mit konkreten Ideen zu Organisationsmodellen und Nachhaltigkeit als Diskussionsgrundlage; eine Art Gründungskonvent für das TextGrid-Konsortium
- Januar 2009 - **Projektende** TextGrid I
- März 2009 - Veröffentlichung der **TextGrid v.1** auf einer CD zum Mit-nach-Hause-Nehmen am D-Grid All-Hands Meeting

### 1.3 Web Service Integration

Während des Berichtszeitraums wurden die Web Services mit verschiedenen Client-Applikationen und -Frameworks getestet (Python, Perl, PHP, SOAPUI, Axis2 u.a.m.). Siehe auch Report 3.4, Abschnitt „3.1.1 WS Interoperabilität“.

Bisher nur als *rpc style* publizierte WebServices wurden mit einer zusätzlichen *document style*-SOAP-Schnittstelle versehen (Tokenizer, Streaming-Editor, Lemmatizer)

Zusätzliche wurden für die einzelnen Tools Webservice-Instanzen, die die zu verarbeitenden Daten direkt aus der SOAP-Message empfangen und dort auch wieder hineinschreiben, aufgesetzt. Kein Zugriff auf Middleware, d.h. keine Authentifizierung, kein Logging und keine Schnittstelle zu *TG-crud*. Zugriff auf gleiche Programmbibliotheken. Diese Servicevarianten sind vor allem zur Nachnutzung durch externe Projekte (etwa im Rahmen von Interedition), die ihre Daten nicht in TextGrid ablegen, gedacht.

Derzeit (Feb. 2008) in Arbeit sind die Umstellung der FileService-Schnittstellen auf *TG-crud*, die Erweiterung der jeweiligen Tool-Schnittstellen um Parameter für *TG-auth\** und Logging, sowie Client-Plug-ins für die Eclipse-GUI.

## 1.4. TextGrid Registry

Aufgrund seiner Web Service basierten Architektur ist TextGrid sehr modular aufgebaut. Die einzelnen Werkzeuge können an verschiedenen Orten betrieben werden; der gleiche Dienst ggf. auch auf mehreren Servern. Es bedarf deshalb einer Registry, über die potentielle Clients die Endpoints der gesuchten Dienste in Erfahrung bringen können. Diese Dienste müssen auch nicht notwendig auf TextGrid beschränkt sein. Durch die Vernetzung mit anderen Initiativen wird TextGrid Teil eines *eHumanities Digital Ecosystems*<sup>1</sup> sein. Technologisch erfordert diese Vision eine Interoperabilitätsschicht auf technischer wie semantischer Ebene.

Tatsächlich war das Konzept der Registries von Beginn an Teil des Entwurfs Web Service basierter Architekturen. Dennoch haben sich Registries noch nicht im großen Rahmen durchgesetzt, wofür – wie von Küster, Moore und Ludwig<sup>2</sup> beschrieben – in erster Linie fundamentale Schwächen der einschlägigen Standards UDDI und ebXML verantwortlich sind. TextGrid wird deshalb eine Topic Maps (ISO/IEC 13250:2000) basierte Semantic Registry zum Einsatz bringen.

In einer solchen Registry kann den Daten eine beliebige Ontologie zu Grunde gelegt werden. Dadurch ist es möglich, die spezifischen Anforderungen des jeweiligen Anwendungsgebietes, hier also den eHumanities, zu berücksichtigen, so dass die Registry diejenigen Informationen vorhält, die benötigt werden, um einzelne Dienste zu für Geisteswissenschaftler hilfreichen Anwendungen zusammenzufügen.

Die Hauptaufgabe einer Registry bleibt, dass sie Dienste – oder allgemeiner: Ressourcen – auffindbar macht. Dafür braucht sie u. a. eine auch technisch wenig komplexe Schnittstelle, um nach Ressourcen zu suchen. Die TextGrid Registry wird hierfür sowohl eine SOAP-Schnittstelle als auch ein RESTful Interface<sup>3</sup> anbieten. Im einfachsten Fall kennt ein Client den Namen des Services, den er sucht, und erhält folglich auf eine Anfrage direkt eine Liste der Endpoints aller Instanzierungen dieses Dienstes. Ist dagegen nur klar, dass beispielsweise ein Lemmatisierungsdienst benötigt wird, so kann auch nach „Lemmatizer“ gesucht werden; die Registry wird dann eine Beschreibung aller Dienste liefern, die das abstrakte Konzept „Lemmatizer“ implementierung. Anhand dieser Beschreibung kann der Client dann den Dienst auswählen, der z. B. die eine Lemmatisierung für die passende Sprache bzw. Sprachstufe anbietet.

---

<sup>1</sup> Vgl. Küster, Ludwig, Aschenbrenner (2007) und Ludwig, Küster (2008).

<sup>2</sup> Küster, Moore, Ludwig (2007), 21–36.

<sup>3</sup> Richardson, Ruby (2007).

Wie schon festgestellt, ist angestrebt, dass sich TextGrid mit anderen eHumanities-Initiativen vernetzt, z. B. Interedition. Dann wird aber auch der Bedarf für eine TextGrid-übergreifende Suche nach Ressourcen bestehen. Auch die anderen Initiativen werden ggf. eigene Registries betreiben. Aus organisatorischen Gründen scheint es weder praktikabel noch wünschenswert, dass die einzelnen Initiativen die Kontrolle über ihre Registries zugunsten einer zentralisierten eHumanities-Registry aufgeben. Die Alternative ist eine Föderierung der Registries, die deshalb auch von der TextGrid Registry unterstützt werden muss. Wie in den bereits zitierten Artikeln beschrieben, hat sich hierfür ein Ansatz bewährt, in dem jede Registry ihre Änderungen über einen ATOM-Feed publiziert. Diese auf existierenden Standards aufsetzende Lösung ist robust, erfordert keine aufwendige Pflege der Listen abhängiger Registries, und ermöglicht sowohl Peer-to-Peer als auch hierarchische Registry-Netzwerke. Dieser Ansatz soll deshalb auch für die TextGrid Registry verfolgt werden.

## 2 Im Berichtszeitraum begonnene Module

### 2.1 Recherchetool

#### 2.1.1 Produktübersicht

Das Recherchetool bildet das Frontend zur Utility *TG-search*<sup>4</sup> und dient der Recherche und dem Retrieval von Struktur- und Metadaten.

Zum den technischen Rahmenbedingungen:

TextGrid-Objekte werden von *TG-crud* ins Grid eingecheckt. Dabei werden Metadaten und XML-Strukturdaten zusätzlich in eine XML-Datenbank geschrieben. XML-Daten werden dabei vom *Adaptor-Manager* in die TextGrid-Kerncodierung<sup>5</sup> überführt. *TG-crud* sorgt dafür, dass sowohl die Original- als auch die kerncodierten Daten in die Datenbank geschrieben werden. Teils aus den Metadaten, teils aus den XML-Daten werden Beziehungen (Relationen: *isVersionOf*, *isAlternativeFormatOf*, *hasAdaptor*, *hasSchema*, *isPartOf*, *refersTo*) extrahiert, die in eine RDF-Datenbank geschrieben werden. Sowohl die Metadaten als auch die Strukturdaten (Original und kerncodiert) erhalten als Datensatz-Namen den von *TG-crud* generierten URI. Das Recherchetool sucht ausschließlich in diesen Datenbanken. Die TextGrid-übergreifende Suche erfolgt auf den über die Kerncodierung vereinheitlichten Strukturdatenbestand.

#### 2.1.2 Zielbestimmung

##### 2.1.2.1 Musskriterien

Suche nach Element- und Attributwerten in Meta- und Strukturdaten. Auswertung von XQueries<sup>6</sup> unter Beachtung der Zugriffsrechte des jeweiligen Nutzers. Alles weitere unter 2.1.4, Produktfunktionen.

---

<sup>4</sup> Bislang in keinem Report ausführlich dokumentiert, da Konzeption noch relativ neu. Siehe vorerst Report 3.4, Abschnitt 3.5 „TG-search“ und Report 3.3 (Version 1), Abschnitt 3 „Schnittstellen zu den Metadaten und Strukturdaten“.

<sup>5</sup> [http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid\\_Kerncodierung\\_070615.pdf](http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid_Kerncodierung_070615.pdf)

<sup>6</sup> <http://www.w3.org/XML/Query/>

### 2.1.2.2 Wunschkriterien

#### *Automatische Summierung bei mehr als n Treffern*

Bei mehr als  $n$  Treffern für eine Suchanfrage stellt sich die Frage nach der Organisation der Suchergebnisse, um die Übersicht zu wahren. Eine Möglichkeit ist, dass Treffer innerhalb einer Objekthierarchie wie etwa *Person* → *Textsorte* → *Titel* aufsummiert werden. Man erhält als Suchergebnis also statt der eigentlichen Trefferliste eine Darstellung wie

```
Schiller, Friedrich, 1759-1805 (103)
+ Dramen (42)
  + Wilhelm Tell (23)
  + Die Räuber (19)
+ Lyrik (61)
  + Das Lied von der Glocke (30)
  + Der Taucher (17)
  + Die Bürgerschaft (14)
```

wobei jeder Werktitel ein Link auf die Trefferliste für das Werk ist.

#### *Exemplarische Suchergebnisse auf Objektebene bei mehr als n Treffern in einem Objekt*

Gibt es in einem Objekt mehrere Treffer, so wird nur die Gesamtzahl und ein, zwei Beispieltreffer dargestellt, mit einem Link um alle Treffer oder alle Treffer des Werks zu zeigen.

#### *»Greediness«*

In bestimmten Fällen gibt es zwei Möglichkeiten, wie mit Suchanfragen und nicht hinreichend codierten Informationen umzugehen ist:

##### Precision-orientiert

Der Benutzer will nur Suchergebnisse, auf die seine Beschränkungen vollständig und korrekt zutreffen, auch wenn er damit riskiert, Treffer in unzulänglich annotierten Dokumenten nicht zu erhalten

##### Recall-orientiert

Der Benutzer will möglichst alle Suchergebnisse, selbst solche, auf die seine Beschränkungen nur möglicherweise zutreffen.

#### *Beispielhafte Anwendungsfälle:*

Editionen / Sammelwerke mit fehlenden Metadaten für die Einzelobjekte

Die Edition *Der Junge Goethe* etwa enthält zahlreiche Texte von anderen Autoren (z.B. Briefe von Herder, oder die Bibel). Die Information, welcher Textteil von wem stammt, ist nicht annotiert, und der Aufwand, das nachzutragen, ist zu groß. *Der Junge Goethe* trägt deshalb als Metadatum die Information, dass da Texte von Fremdautoren drin sind. Eine Precision-orientierte Suche nach *Farbe* in Texten von Goethe fände überhaupt keine Treffer aus dem *jungen Goethe*, eine Recall-orientierte Suche hingegen möglicherweise auch Texte von Herder über Farbe.

Wörterbücher / sonstige spezifische Textsorten mit optionalen Codierungsbestandteilen

In den textsortenspezifischen Kerncodierungen sind bestimmte Codierungsmöglichkeiten optional. Bei einer Sucheinschränkung für eine solche optionale Codierung werden bei einer precision-orientierten Suche nur Objekte, die diese Codierung enthalten, berücksichtigt; bei einer recall-orientierten Suche werden die Objekte, die diese Codierung nicht enthalten, berücksichtigt und dort das beschränkende Kriterium aus der Suche nicht angewandt.



Zur Realisierung der Wahlmöglichkeit muss seitens der **Metadaten** für den ersten Falle ein Feld für diese »mixed content«-Eigenschaft vorhanden sein, im zweiten Fall muss in den Metadaten codiert werden, welche optionalen Elemente in der Kerncodierung verwandt werden.

In der UI wäre es wünschenswert, bei precision-orientierten Suchen einen Hinweis auf weitere potentielle Ergebnisse und bei recall-orientierten Suchen auf die Qualität der Ergebnisse unterzubringen (z.B. »Die Einschränkung ›nur in den Beispielen« wurde bei einigen durchsuchten Objekten nicht berücksichtigt«).

### 2.1.2.3 Abgrenzungskriterien

Aus Zeit- und Performancegründen werden folgende Punkte zunächst nicht umgesetzt:

- Indexierung außerhalb der von den Datenbank-Anwendungen generierten Indices (also mit einem separaten Tool wie Lucene).
- sprachspezifische Suche (Soundex Algorithmus und Derivate)<sup>7</sup>
- Einbindung von Normdaten-Katalogen.
- Eingabehilfe für nicht-lateinische Alphabete (evtl. können hier die Eingabehilfen des XML-Editors nachgenutzt werden).
- Ranking

## 2.1.3 Produkteinsatz

Streaming Tool, als Web Service gekapselt. Keine Grid-Features; diese werden von den in der Middleware bereitgestellten FileServices bzw. *TG-crud* zur Verfügung gestellt.

## 2.1.4 Produktfunktionen

Folgende Suchinterfaces stehen zur Verfügung

1. TextGrid-übergreifende Suche (Kerncodierung, Web / Eclipse GUI)  
Maskenbasierte Suche über Metadaten, Relationen, Volltext und Textsorten-spezifischen Strukturdaten (gemäß Kerncodierung). Beim Ergebnis handelt es sich je nach Typ der Anfrage um
  - die (ggf. gefilterten) Metadaten der gefundenen Objekte
  - eine Liste der gefundenen Objekte (mit Trefferhäufigkeiten)
  - eine Trefferliste im KWIC<sup>8</sup>-Modus, mit hervorgehobenem Suchbegriff

In jedem Fall bietet die Trefferliste Pfadangaben und Links, um tiefer in die Textdaten einzusteigen.

2. Projektspezifische Suche (Projektcodierung, Web / Eclipse GUI)  
Erstellung in Verantwortung des jew. Projekts, im Rahmen von AP4 1-2 vorgefertigte Beispiele in Anlehnung an TextGrid-übergreifende Suche.
3. Objektverwaltung (nur Metadaten, Eclipse GUI)  
s.u., Projektbrowser (3.1.2)
4. XQuery-Interface (Web / Eclipse GUI)  
Eingabe frei definierbarer XQueries, Ergebnisdarstellung in Quellcode-Ansicht.

---

<sup>7</sup> Siehe Report 1.3, Abschnitt 2.2.5.

<sup>8</sup> KWIC = **Key-Word In Context**.

Ein Nutzer darf nur Treffer aus/von Objekten angezeigt bekommen, für die er zumindest Leserecht besitzt, d.h. die Objekte sind entweder *publiziert*, d.h. der Allgemeinheit zugänglich, oder sind vom Projektleiter/-administrator für den jeweiligen Nutzer zum Lesen freigegeben worden. Das Recherchetool muss deshalb, bevor es Daten zurückliefert, anhand der URIs der gefundenen Objekte und den vom Client übergebenen Authentifizierungsinformationen (RBAC-Session-ID) über *TG-auth\** prüfen, zu welchen Objekten der Nutzer Informationen erhalten darf.

**TextGrid-übergreifende Suche**

Suchen nach ...

Textsuche starten   passende Objekte zeigen   Auswahl löschen    unsichere Ergebnisse mitliefern (Hilfe)

**Nur in Objekten mit ...**

Person (Autoren, Herausgeber, ...)  ...  
z. B.: Goethe, Johann Wolfgang von; Eibl, Karl

Titel (Werk)  ...  
z. B.: Faust oder Derjunge Goethe

Erscheinungsdatum nach  und vor:   
z. B.: nach 1800; nach 1945 und vor 2000-06-30

Sprache

Schlagwörter  ...  
z. B.: Roman; Frühwerk

Dokumenttyp  (egal)  Wörterbücher  Prosa  Drama  Briefe  Lyrik  Digitalisate

Sammlungstyp  (egal)  Edition  Linguistisches Korpus  Sonstiges

Projekt  ...  
z. B.: Derjunge Goethe online

**Textsortenspezifisches**

**Drama**

alles durchsuchen

Regieanweisungen

Sprechtext

Prolog, Epilog

Sprecher:  ...  
z. B.: Mephisto; Faust

**Edition**

alles durchsuchen

Apparat / Kommentar

Vorwort / Einleitung

Fertig   zotero

Screenshot zur TextGrid-übergreifenden Suche (Prototyp Web-Interface)

## 2.1.5 Produktdaten

### 2.1.5.1 Welche Daten sind aus Benutzersicht (langfristig) zu speichern

Komplexe Anfragen, deren Erstellung mitunter sehr viel Zeit kostet, sollten ebenso wie Suchergebnisse als Objekte abgelegt werden können.

Da Suchanfragen intern ohnehin als XQueries modelliert werden, ist es technisch gesehen kein Problem, diese zu exportieren, mit Metadaten zu versehen und als Objekt im Grid abzulegen.

Bei Suchergebnissen ist dies zunächst nur möglich, wenn es sich um valides XML handelt, das in den XML-Editor geladen (und weiterverarbeitet) werden kann, z.B. bei kompletten XML-Objekten, Fragmenten von Objekten, oder bei Ergebnissen, die im KWIC-Modus dargestellt werden, z.B.:

```
<?xml version="1.0" encoding="utf-8"?>
<tg:metaResponse xmlns:tg="http://textgrid.info/namespaces/metadata">
<tg:document>
<c:textgridmetadata xmlns="http://textgrid.info/namespaces/metadata/core">
  <!--...-->
</c:textgridmetadata>
<tg:result>
<tg:entry>
<tg:location>
<!--DER JUNGE GOETHE / Dramatische Schriften / Belsazar / [Zweiter Aufzug]
/ [Erster Auftritt]-->
<tg:locitem path="XPATH">DER JUNGE GOETHE</tg:locitem>
<tg:locitem path="XPATH">Dramatische Schriften</tg:locitem>
<tg:locitem path="XPATH">Belsazar</tg:locitem>
<tg:locitem path="XPATH">[Zweiter Aufzug]</tg:locitem>
<tg:locitem path="XPATH">[Erster Auftritt]</tg:locitem>
</tg:location>
<tg:context>Die <tg:match path="XPATH+QUERYSTRING">Liebe</tg:match>
hielte<note><hi>hielte</hi> Ältere süddeutsche Form.</note> mich in
sanftem Arm gebunden</tg:context>
</tg:entry>
<tg:entry>
<tg:location>
<!--DER JUNGE GOETHE / Dramatische Schriften / Der Lügner / Erster Aufzug.
/ Erster Auftritt / DORANT. CLITON.-->
<tg:locitem path="XPATH">DER JUNGE GOETHE</tg:locitem>
<tg:locitem path="XPATH">Dramatische Schriften</tg:locitem>
<tg:locitem path="XPATH">Der Lügner</tg:locitem>
<tg:locitem path="XPATH">Erster Aufzug.</tg:locitem>
<tg:locitem path="XPATH">Erster Auftritt</tg:locitem>
<tg:locitem path="XPATH">DORANT. CLITON.</tg:locitem>
</tg:location>
<tg:context>Zur <tg:match path="XPATH+QUERYSTRING">Liebe</tg:match>,
Gnädger Herr, ist das die größte Gabe.</tg:context>
</tg:entry>
<tg:entry>
<tg:location>
<!--DER JUNGE GOETHE / Dramatische Schriften / Der Lügner / Erster Aufzug.
/ Erster Auftritt / DORANT. CLITON.-->
<tg:locitem path="XPATH">DER JUNGE GOETHE</tg:locitem>
<tg:locitem path="XPATH">Dramatische Schriften</tg:locitem>
<tg:locitem path="XPATH">Der Lügner</tg:locitem>
<tg:locitem path="XPATH">Erster Aufzug.</tg:locitem>
<tg:locitem path="XPATH">Erster Auftritt</tg:locitem>
<tg:locitem path="XPATH">DORANT. CLITON.</tg:locitem>
</tg:location>
```

```
<tg:context>Der schenckt mit voller Hand, und wird doch nicht ge<tg:match
path="XPATH+QUERYSTRING">liebe</tg:match>t.</tg:context>
</tg:entry>
</tg:result>
</tg:document>
</tg:metaResponse>
```

#### 2.1.5.2 *Wo liegen die Daten, von wo aus greift die Applikation darauf zu*

Sowohl Meta- als auch Strukturdaten liegen derzeit in ein und derselben eXist-Datenbank<sup>9</sup>. in unterschiedlichen *collections* (s.u.) Mit steigender Datenlast werden bei Bedarf (Performance) separate Instanzen angelegt werden müssen für:

- Metadaten  
aktuell: /db/metadata
- Strukturdaten (Original)  
aktuell: /db/structure/original
- Strukturdaten (Kerncodierung)  
aktuell: /db/structure/baseline

Beziehungsdaten stehen in einer RDF-Datenbank (OpenRDF Sesame).

Für Abfragen stehen sowohl eine REST- als auch eine SOAP/WSDL-Schnittstelle zur Verfügung. Die REST-Schnittstelle wird aktuell auch intern von der Recherchetooll-Implementierung genutzt. Sämtliche Datenbank-Spezifika sind in jeweils einer Klasse gekapselt, so dass bei einem etwaigen Wechsel der Datenbank-Anwendung nur die betreffende Klasse ausgetauscht zu werden braucht.

#### 2.1.5.3 *Welche Datenmengen müssen verarbeitet werden*

Tatsächlich verarbeitet werden müssen nur Trefferlisten, die je nach Suchanfrage allerdings sehr umfangreich ausfallen können. Deshalb kann es – je nach Anfragentyp – sinnvoll sein, die Trefferanzeige in vom Nutzer definierbaren Portionen darzustellen. Eine diesbezügliche Entscheidung wird nach den Beta-Tests getroffen.

#### 2.1.5.4 *Daten-Format*

Bei maskenbasierten Suchanfragen wird stets valides XML zurückgegeben, Root-Element <tg:metaResponse> bzw. das Root-Element des jew. Dokuments. Bei XQueries lässt sich das Ergebnis-Format frei definieren (kein Binärformat).

### 2.1.6 **Produktleistungen**

Da die Menge der zu durchsuchenden Daten mit steigender Nutzerzahl stetig ansteigen wird, muss bereits jetzt darauf geachtet werden, dass sich die Antwortzeiten im Rahmen des Akzeptablen bewegen. Allerdings wird auf Nutzerseite ein gewisses Bewusstsein dafür vorausgesetzt, dass bei besonders komplexen und besonders allgemein gehaltenen Anfragen naturgemäß mit etwas längeren Antwortzeiten zu rechnen ist.

### 2.1.7 **Benutzeroberfläche**

- GUI-Plugin (Eclipse) für TextGrid-übergreifende Suche (Kerncodierung)
- Web-Interface für TextGrid-übergreifende Suche (Kerncodierung)

---

<sup>9</sup> <http://134.76.176.136:8080/exist>

- Web-Interface für projektspezifische Suche (Originalcodierung) – in Zuständigkeit des jeweiligen Projekts (1-2 exemplarische Formulare im Rahmen der Entwicklung des Text Publishers Web, s.u.)
- XQuery-Interface (Eclipse und Web)
- Spezialisierte (Eclipse-)Plugins und Schnittstellen für andere Tools
  - Projektbrowser/Navigationstool (s.u., 3.1.3)
  - Link-Editor Text (s.u., 4.2)
  - Graphischer Link-Editor (s.u., 4.1)
  - Bibliographietool (s.u., 4.3)
  - Text Publisher Web (s.u., 4.8)

## 2.1.8 Nichtfunktionale Anforderungen

### 2.1.8.1 Einzuhaltende Gesetze und Normen, Sicherheitsanforderungen, Plattformabhängigkeiten

Plattformunabhängig, da als Webservice implementiert.

### 2.1.8.2 Welche Nutzungsdaten sollen (oder dürfen nicht) von der Middleware erhoben werden

Logging, Durchreichen von Benutzerdaten, siehe Report 3.2 „TextGrid Architektur“.

## 2.1.9 Technische Produktumgebung

### 2.1.9.1 Software: für Server und Client, falls vorhanden

*Client:* Java ab 1.5, Eclipse 3.3, Apache Axis2 1.3<sup>10</sup>. Für das Web-Interface einen aktuellen Browser (Mozilla Firefox, Internet Explorer, Opera, Safari).

*Server:* Java 1.6, Apache Tomcat 6.0 (Servlet-Container)<sup>11</sup>, Apache Axis2 1.3 (WebService-Framework), eXist 1.1.2 (XML-Datenbank)<sup>12</sup>, OpenRDF Sesame 2.0 (RDF-Datenbank)<sup>13</sup>

### 2.1.9.2 Hardware: für Server und Client getrennt

Keine speziellen Anforderungen, Hardware sollte aus Performance-Gründen halbwegs aktuell sein, serverseitig sollten mind. 8 GB RAM zur Verfügung stehen.

### 2.1.9.3 Produkt-Schnittstellen

SOAP/WSDL: <http://134.76.176.136:8080/axis2/services/MetadataNew?wsdl>

REST: <http://134.76.176.136:8080/axis2/services/MetadataNew/meta>

(beides vorläufige URLs)

*Parameter:*

md      Metadaten, interne Syntax: METADATUM:WERT, mehrere Angaben möglich, durch “|“ voneinander getrennt. WERT = “\$“: Metadatum dient als Filter, standardmäßig werden komplette Metadatensätze zurückgegeben.  
Beispiel:

<sup>10</sup> <http://ws.apache.org/axis2/>

<sup>11</sup> <http://tomcat.apache.org>

<sup>12</sup> <http://www.exist-db.org>

<sup>13</sup> <http://www.openrdf.org>

<http://134.76.176.136:8080/axis2/services/MetadataNew/meta?md=person:goe>  
(liefert die Metadatenätze, bei denen die Zeichenfolge "goe" in <person> enthalten ist, z.B. <person role="author">Johann Wolfgang Goethe</person>)

- rel    get:    Kommaseparierte Liste aller gefundenen Relationen.  
filter:  Kommaseparierte Liste aller Relationen, die ausgefiltert werden sollen  
          (Werden nur die Metadaten vom Ursprungswerk, ohne seine Formate und  
          Versionen als Rückgabe gewünscht, dann filter:hasVersion,hasFormat)  
tree:    Integer – Tiefe der rekursiven get-Relationen-Suche.  
  
Wird kein Tree-Parameter angegeben, wird der default-Wert zu tree:0 angenommen,  
d.h. es werden nur Objekte/Beziehungen zurückgegeben, die in direkter Beziehung  
zu den gesuchten Objekten stehen. Für alle "Related-Objects" werden auch die  
angeforderten Metadaten zurückgegeben.
- opt    Bislang nur für *autocomplete* (ac:1, bei REST) genutzt.
- q      Volltextsuche in den Strukturdaten, beliebiger String, Rückgabe der Metadaten zu  
den gefundenen Objekten und die Anzahl der Treffer je Objekt.
- xquery XQuery, als String
- sid    Authentifizierung (RBAC-Session-ID), wenn nicht angegeben, wird nur über den  
publizierten Datenbestand gesucht.
- log    Logging-Parameter (Logserver, Session-ID, Loglevel)

Parameter lassen sich miteinander kombinieren. Beispiele sind in Report 3.4 „TextGrid -  
Middleware-Tests“ dokumentiert.

## **2.2 Streaming Editor I (XSLT-Komponente)**

### **2.2.1 Produktübersicht**

Der Streaming Editor I ist ein Webservice-Wrapper um einen XSLT<sup>14</sup>-Prozessor.

### **2.2.2 Zielbestimmung**

#### *2.2.2.1 Musskriterien*

Regelbasierte Transformation eines XML-Objekts anhand eines XSLT-Stylesheets.

#### *2.2.2.2 Wunschkriterien*

XSLT 2.0 bietet die Möglichkeit, sowohl mehrere Dateien einzulesen, als auch in mehrere  
Ausgabedateien zu schreiben. Die SOAP/WSDL-Schnittstelle sollte dies über entsprechende  
Parameter unterstützen.

#### *2.2.2.3 Abgrenzungskriterien*

Die XSLT-Implementierung Saxon 9 ermöglicht es, beliebige Java-Klassen auszuführen<sup>15</sup>.  
Dieses Feature muss aus Sicherheitsgründen deaktiviert werden.

---

<sup>14</sup> <http://www.w3.org/Style/XSL/>

<sup>15</sup> <http://www.saxonica.com/documentation/extensibility/functions.html>

### **2.2.3 Produkteinsatz**

Streaming Tool, als Web Service gekapselt. Keine Grid-Features; diese werden von den in der Middleware bereitgestellten FileServices *bzw. TG-crud* zur Verfügung gestellt.

### **2.2.4 Produktfunktionen**

Transformation von XML-Daten anhand eines XSLT-Stylesheets.

Die zu transformierenden Daten können ebenso wie das für die Transformation zuständige XSLT-Stylesheet als URIs adressiert werden. In diesem Fall sorgt eine Schnittstelle zu *TG-crud* für das Einlesen der Daten aus dem Grid – und generiert ein Objekt, in das das Ergebnis der Transformation geschrieben wird. Wie bei allen Tools, die auf Daten im Grid zugreifen, müssen auch hier die Zugriffsrechte berücksichtigt werden.

Daneben existiert eine Schnittstelle, die die zu verarbeitenden Daten aus den Parametern der SOAP-Message ausliest und das Ergebnis über die SOAP-Response direkt zurückschreibt.

Eine weitere Schnittstelle liest und schreibt die Daten base64-codiert aus der und in die SOAP-Message.

### **2.2.5 Produktdaten**

#### *2.2.5.1 Welche Daten sind aus Benutzersicht (langfristig) zu speichern*

Quelle, Ziel, XSLT-Stylesheet.

#### *2.2.5.2 Wo liegen die Daten, von wo aus greift die Applikation darauf zu*

Die Daten liegen entweder im Grid oder können über den SOAP-Request mitgegeben werden.

#### *2.2.5.3 Welche Datenmengen müssen verarbeitet werden*

Beliebig große Datenmengen.

#### *2.2.5.4 Daten-Format*

Eingabedaten: XML, Ausgabedaten: beliebig, abhängig von XSLT-Stylesheet.

### **2.2.6 Produktleistungen**

XML-Daten sollten 1:1 weitergereicht werden. Da die in die SOAP-Messages eingebetteten XML-Daten seitens der client- und serverseitigen Frameworks hin- und her-escaped werden, besteht die Gefahr, dass bestimmte Sonderzeichen falsch interpretiert werden – deshalb auch die Webservice-Instanz, die base64-codierte Daten liest und schreibt.

### **2.2.7 Benutzeroberfläche**

GUI-Plugin (Eclipse).

### **2.2.8 Nichtfunktionale Anforderungen**

#### *2.2.8.1 Einzuhaltende Gesetze und Normen, Sicherheitsanforderungen, Plattformabhängigkeiten*

Plattformunabhängig, da als Webservice implementiert.

#### *2.2.8.2 Welche Nutzungsdaten sollen (oder dürfen nicht) von der Middleware erhoben werden*

Logging, Durchreichen von Benutzerdaten, siehe Report 3.2 „TextGrid Architektur“.

## 2.2.9 Technische Produktumgebung

### 2.2.9.1 Software: für Server und Client, falls vorhanden

*Client:* Java ab 1.5, Eclipse 3.3, Apache Axis2 1.3.

*Server (XSLT 2.0, Java-Umgebung):* Java 1.6, Apache Tomcat 6.0, Apache Axis2 1.3, Saxon-B 9.0 (XSLT-Prozessor, Open Sorce-Variante)<sup>16</sup>.

*Server (XSLT 1.0, Python-Umgebung):* Apache 2.2.4 unter mod\_python, Python 2.4.2, ZSI 2.1a<sup>17</sup>, 4Suite<sup>18</sup>, PyXML 0.8.4<sup>19</sup>.

### 2.2.9.2 Hardware: für Server und Client getrennt

Keine speziellen Anforderungen, Hardware sollte aus Performance-Gründen halbwegs aktuell sein, serverseitig sollten mind. 8 GB RAM zur Verfügung stehen.

### 2.2.9.3 Produkt-Schnittstellen

Python, XSLT 1.0:

<a href="http://ingrid.daasi.de/SExslt1_rpc.wsdl">http://ingrid.daasi.de/SExslt1_rpc.wsdl</a>	(rpc-style, Dateinamen/URIs als Param.)
<a href="http://ingrid.daasi.de/SExslt1.wsdl">http://ingrid.daasi.de/SExslt1.wsdl</a>	(docstyle, Dateinamen/URIs als Parameter)
<a href="http://ingrid.daasi.de/SExslt2_rpc.wsdl">http://ingrid.daasi.de/SExslt2_rpc.wsdl</a>	(rpc-style, zu verarbeitende Daten als Param.)
<a href="http://ingrid.daasi.de/SExslt2.wsdl">http://ingrid.daasi.de/SExslt2.wsdl</a>	(docstyle, zu verarbeitende Daten als Param.)
<a href="http://ingrid.daasi.de/SExslt3_rpc.wsdl">http://ingrid.daasi.de/SExslt3_rpc.wsdl</a>	(rpc-style, zu verarbeitende Daten als base64)
<a href="http://ingrid.daasi.de/SExslt3.wsdl">http://ingrid.daasi.de/SExslt3.wsdl</a>	(docstyle, zu verarbeitende Daten als base64)

Java, XSLT 2.0 (wird derzeit implementiert):

<http://134.76.176.136:8080/axis2/services/seditor?wsdl> (docstyle, Dateinamen/URIs als Param.)

<http://134.76.176.136:8080/axis2/services/seditor2?wsdl> (docstyle, zu verarb. Daten als Param.)

Parameter:

*infile* (Eingabedaten/URI), *outfile* (Ausgabedaten/URI), *xslfile* (XSLT-stylesheet/URI), *params* (optionale Key-Value-Paare für XSLT-Stylesheet), *sid* (RBAC-Session-ID), *log* (Logging-Info).

## 2.3 Projektmanagement

### 2.3.1 Produktübersicht

Die Projektverwaltung ermöglicht die Erstellung neuer Projekte durch den Projektleiter und die Zuordnung von weiteren Benutzern (in bestimmten Rollen) zu einem Projekt, sowie die Abfrage und das Setzen von Rechten an TextGrid-Objekten.

---

<sup>16</sup> <http://saxon.sourceforge.net>

<sup>17</sup> <http://pywebsvcs.sourceforge.net>

<sup>18</sup> <http://4suite.org>

<sup>19</sup> <http://pyxml.sourceforge.net>



### 2.3.2 Zielbestimmung

Ziel des Projektmanagements ist eine möglichst effiziente und intuitive Verwaltung der Zugriffsrechte von TextGrid-Ressourcen, den dazugehörigen Rollen sowie der Benutzer, die diese inne haben, bis hin zur Definition von Projekten.

#### *Musskriterien*

- Erstellung neuer Projekte - jeder kann ein neues Projekt erstellen und ist dann automatisch Projektleiter
- Anzeigen aller bzw. der Projekte, in denen ein Benutzer involviert ist
- Zuordnung von Personen in einer bestimmten (Meta-)Rolle (Beobachter, Bearbeiter, Projektleiter, technischer Administrator, ggf. weitere) zu einem Projekt (z.B. eine Edition); jede Rolle können mehrere Benutzer in einem Projekt innehaben, jeder Nutzer kann mit verschiedenen Rollen in ein Projekt eingeschrieben sein, jeder Nutzer kann an mehreren Projekten mitarbeiten
- Integration in andere Tools
- in der Navigationsleiste bzw. nach der Metadatensuche kann der Projektleiter/Administrator über einen Kontext-Menü-Eintrag bei einem Objekt die Rechte des Objekts spezifizieren
- Der TG-crud-Service legt Ressourcen in einem gewählten Projekt an und vergibt Default-Rechte (z.B. Owner:read,update,delete, Projektleiter:read,update,delete,delegate).
- TG-crud und TG-search (evtl. noch andere TextGrid-Komponenten) benötigen eine performant realisierte Funktion *CheckAccess*, die den Zugang zu Projekten/Ressourcen erlaubt oder verbietet
- Rechtevergabe mit den Rechten view, create, delegate (auf Projekt-Ressourcen) sowie read, update, delete, delegate (auf Datei-Ressourcen).
- Neben Owner (i.e. Ersteller) können diese Rechte pro Ressource beliebig auf die im Projekt definierten Rollen vergeben werden, z.B. Beobachter:read, Bearbeiter:read,update etc. So kann eine Ressource mehreren Rollen innerhalb eines Projekts „gehören“.
- Ressourcen können publiziert werden: lesender Zugriff TextGrid- (i.e. welt-) weit; unangemeldete Nutzer können diese öffentlichen Ressourcen einsehen. Eine Veröffentlichung darf nicht rückgängig gemacht werden.
- Deaktivierung/Wieder-Aktivierung von Projekten: Löschen eines Projekts ist verboten; Veröffentlichungen bleiben immer einem Projekt zugeordnet; d.h. bei einem deaktivierten Projekt sind nur noch die zuvor veröffentlichten Ressourcen sichtbar.
- Projektverwaltung: Konfiguration der Projekteinstellungen, z.B. Zuordnung von Workflows, Schemata, Adaptoren als besondere Ressourcen

#### *Wunschkriterien*

- Gestaltung einer (Web-)Seite (z.B. unterhalb von textgrid.de), auf der sich ein Projekt darstellen kann und ggf. Rohdaten und Publikationen publizieren kann (in Zusammenspiel mit dem Text Publisher Web)

#### *Abgrenzungskriterien*

- Die Objektverwaltung wird nicht mit der Projektverwaltung verquickt. Objekte (z.B. ein TEI-File, etc) werden aus den jeweiligen interaktiven Tools bzw. im Navigationstool verwaltet und können von dort aus mit Rechten versehen werden. D.h. die Verwaltung der

einzelnen Datei-Ressourcen hat keine eigene Projektverwaltungs-spezifische Benutzeroberfläche.

### **2.3.4 Produkteinsatz**

Interaktive Masken als Teil des *TextGridLab* und zugrundeliegende Web-Services. Keine Einbindung in Workflow; keine Grid-Fähigkeit.

### **2.3.5 Produktfunktionen**

Das Projektmanagement besteht zunächst aus einer LDAP-Datenbank, deren Inhalt mit einer Bibliothek von RBAC-Funktionen abgefragt und modifiziert werden kann. Diese Funktionen können bereits von TextGrid-Komponenten, z.B. TG-crud oder TG-search, verwendet werden.

Aus Benutzersicht wird dies einerseits in einem dedizierten graphischen Frontend zu diesen Funktionen deutlich, andererseits durch in andere Komponenten des TextGridLab eingebettete interaktive Schnittstellen zu diesen Funktionen.

### **2.3.6 Produktdaten**

*Welche Daten sind aus Benutzersicht (langfristig) zu speichern*

Daten über Projekte, Ressourcen, teilweise über Benutzer

*Wo liegen die Daten, von wo aus greift die Applikation darauf zu*

An zentraler Stelle (z.B. auf [textgrid-srv.gwdg.de](http://textgrid-srv.gwdg.de)) in einer LDAP-Datenbank

*Welche Datenmengen müssen verarbeitet werden*

Die meisten Funktionen verarbeiten nur wenige Daten, da der Benutzer normalerweise nur einzelne Änderungen am jeweiligen Projekt macht. Einige Funktionen müssen jedoch sehr performant implementiert sein, da sie entweder schnell reagieren müssen (*CheckAccess*) oder einen hohen Datendurchsatz haben (*filterBySid*, *getObjects*).

### **2.3.7 Benutzeroberfläche**

Siehe Abbildungen für eine beispielhafte Umsetzung.

**TEXT GRID**

Administration

- Projekte
- Rollen
- Benutzer
- Ressourcen

Bestehendes Projekt modifizieren

Projekt auswählen

auswählen

deaktivieren

aktivieren

Oder neues Projekt mit Standardrollen anlegen und \$ HTTP\_REMOTE\_USER als Projektleiter

Name Beschreibung

anlegen

**TEXT GRID**

Administration

- Projekte
- Rollen
- Benutzer
- Ressourcen

Projekt 4567: Junger Goethe

Fotis Jannidis	jannidis@linglit.tu-darmstadt.de	User löschen
	Projektleiter	Rolle löschen
	Rolle wählen	Rolle hinzufügen
Thorsten Vitt	vitt@linglit.tu-darmstadt.de	User löschen
	Administrator	Rolle löschen
	Bearbeiter	Rolle löschen
	Rolle wählen	Rolle hinzufügen
Martin Haase	martin.haase@daasi.de	User löschen
	Beobachter	Rolle löschen
	Rolle wählen	Rolle hinzufügen

Neuen Benutzer aufnehmen

Name E-Mail

Suchen

### 2.3.8 Technische Anforderungen

- Autorisierungsdatenbank (LDAP) muss auf zentralem Server liegen, kann ggf. als nur lesbare Kopien repliziert werden
- Funktionsbibliothek zur Abfrage muss auf allen LDAP-Replikaten vorhanden sein; Datenmodifikation nur auf dem Master;
- Software-Anforderungen:
  - Server: openLDAP, PHP mit SOAP-Erweiterung; Betriebssystem möglichst Linux/Unix

- Client: Rich Client Platform mit Web-Service-Clients und einem internen Browser zur Shibboleth-gestützten Authentifizierung

## 2.4 Metadaten-Annotation

### 2.4.1 Produktübersicht Metadaten-Annotation

Die "Metadaten-Annotation" ist ein generisches Werkzeug, das dazu dient, **strukturierte Daten** über eine definierbare Maske zu erfassen und an einer frei bestimmbar Position in einer Datei einzufügen / abzuspeichern.

Diese Aufgabe gliedert sich in folgende Tools:

- *Schemagenerator* - erlaubt es dem Projektleiter / Techniker eine Maske vorzudefinieren
- *Masken-Editor* - Darstellung der vordefinierten Maske, Überprüfung der Eingabe, Speichern der strukturierten Daten an einer bestimmten Position in einer Datei und im Grid

Zur Einbettung der Aufgabe in die Umgebung ist notwendig

- der *Masken-Editor* bekommt vom TextGridLab beim Aufruf mitübergeben, von welchem Hauptfenster aus welches Objekt (mit darin eingebetteten strukturierten Daten) geladen werden soll
- in der vom *Schemagenerator* vordefinierten Maske ist zudem spezifiziert, an welcher Stelle in einem Objekt die strukturierten Daten gelesen werden können, und ggf das Format (jeweils ein Adaptor zum Lesen und zum Schreiben der Daten)

Die Werkzeuge sollen so offen und flexibel gestaltet werden wie möglich. Dann können damit verschiedene Aufgaben erledigt werden, z.B.

1. Die Eingabe von Metadaten zu Inhaltsobjekten - z.B. eingebettet in den XML-Editor, den grafischen Link-Editor, etc.
2. Die Eingabe von Metadaten zu TextGrid spezifischen Objekten - z.B. Adaptoren im Streaming-Editor, Workflows im Workflow-Editor, etc.
3. allgemeine Eingabe strukturierter Daten wo auch immer in einer Datei (deshalb auch Grundlage des Bibliographietools)
4. allgemeine Annotation, jede Form strukturierter Daten wo auch immer in einer Datei (nicht in TextGrid-I).

### Konkrete Anwendung: Metadaten zu TextGrid-Objekten

Das oben beschriebene generische Konzept wird im ersten Anwendungsfall in TextGrid zur Beschreibung von TextGrid-Objekten verwendet. Am wichtigsten ist diesbzgl die Verarbeitung von TEI-Dateien, aber auch andere Inhaltsobjekte z.B. TEI-Dateien, Bilder (TIFF, PNG, etc); und TextGrid-spezifische Objekte Adaptoren, Workflows und andere.

Metadaten können von TextGrid-Nutzern mit den entsprechenden Rechten manuell eingeführt werden. Die in einer Datei vorhandenen Metadaten werden ständig mit der Darstellung im "Masken-Editor" synchronisiert, bereits vorhandene Metadaten können nach einem Konsistenzcheck so direkt übernommen werden. Zur weiteren Unterstützung der manuellen Eingabe können durch Interaktion mit dem Bibliographietool existierende Metadaten aus externen Quellen übernommen und angepasst werden (z.B. zvd, Kalliope).

Das durch den Schemagenerator erzeugte Metadaten-Schema orientiert sich in jedem Fall an den TextGrid Kernmetadaten - ein Schema, und das der gemeinsame Nenner für die Funktionalität von grundlegenden Tools in TextGrid ist. Für spezifische Typen von

Inhaltsobjekten wird das Schema weiter eingegrenzt - z.B. Inhaltsobjekte in TEI können als Metadaten (zusätzlich zu den Kernmetadaten) nur Felder aus dem TEI-Header enthalten.

## Use cases

### *Metadaten zu einem TG-Objekt*

Erlaubt die Eingabe, Darstellung und Bearbeitung von Metadaten zu einem beliebigen TG-Objekt, sei es eine XML- bzw. TEI-Datei. sei es eine Grafik, ein Skript usw.

### *Metadaten zu einer TEI-Datei = TEI-Header*

Erlaubt die Eingabe, Darstellung und Bearbeitung eines TEI-Headers.

### *Bibliographische Daten*

Erlaubt die Eingabe, Darstellung und Bearbeitung von bibliographischen Daten.

### *Strukturierte Daten*

Erlaubt die Eingabe von strukturierten Daten an einer beliebigen Stelle in einer XML-Datei.

Anmerkung: die Eingabe des TEI-Headers ist eigentlich nur Sonderfall von diesem hier, sollte aber auch die Darstellung und Bearbeitung beherrschen.

## 2.4.2 Schemagenerator

### *2.4.2.1 Produktübersicht Schemagenerator*

Der Schemagenerator ermöglicht die Auswahl bestimmter Elemente und Attribute aus einer vorgegebenen Liste und evtl. durch weitere Regeln deren möglichen Werte einzuschränken bzw. Standardwerte für bestimmte Felder vorzugeben. Außerdem ermöglicht das Werkzeug die Gestaltung einer Maske, in der diese Werte vom Philologen später einmal eingefügt / bearbeitet werden sollen.

Die folgende Zielbestimmung umreißt die Funktionalität des Tools.

### *2.4.2.2 Zielbestimmung*

Der Schemagenerator steht nur einem *Projektleiter* oder *technischen Administrator* zur Verfügung; definierte Schemata sind nur für diese beiden Rollen sichtbar. Sie können dadurch *Projektbearbeitern* gewisse Vorgaben machen. Die Eingabe der Metadaten entsprechend dem vorgegebenen Schema wird (gleichsam von allen Rollen) im Masken-Editor vorgenommen.

Neben der eigentlichen Funktionalität - dem "Zusammenklicken" einer Maske bzw eines Schemas aus vorhandenen Vorgaben, s.o. - muss der Schemagenerator auch für die Einbettung einer Maske ihre Umgebung sorgen. Der Masken-Editor sorgt für den Datenaustausch, aber der Schemagenerator muss die Interpretation der Daten übernehmen:

- ein reinkommendes, komplettes Schema wird interpretiert und in der Maske dargestellt (dazu ggf Adaptoren vom Streaming Editor nötig, der ein spezifisches XML-Schema interpretieren kann)
- nach Eingabe eines Wertes wird der Wert gleich direkt geschrieben (im Schemagenerator kann man angeben: wohin, und in welchem Format i.e. mit welchem Adaptor)

Im Fall der Metadaten-Annotation kann an 2 Orten gespeichert werden:

1. direkt im Inhaltsobjekt, also z.B. im TEI, TIFF, etc

2. im TextGrid-Schema, mit dem das Inhaltsobjekt ins Grid übertragen wird

Wenn der Masken-Editor ein Objekt einliest, wird angenommen, dass die Daten im Objekt und im Grid konsistent sind, und nur die Daten im Objekt ausgelesen.

#### 2.4.2.2.1 *Musskriterien*

Der Schemagenerator bietet ein GUI, mit folgenden Features:

- Liste möglicher Elemente je Objekttyp (z.B. für TEI Dateien die TEI-Header Elemente mit gemappten Kernmetadaten). Diese Liste möglicher Objekttypen und zugehöriger Elemente ist vom System (TextGrid) vorgegeben, aber durch TextGridler erweiterbar.
- Auswahlmöglichkeit von Elementen (i.e. "Einträge", "Felder") und evt.
  - Definition von Regeln (z.B. Wertebereich, Syntax), zur Einschränkung der Werte
  - Vorgabe von Standardwerten
  - ggf alternativ die Angabe einer dynamischen Quelle für Autocompletion (zB Autoren aus einer Autoredatenbank; der Nutzer kann direkt den Service-Aufruf eingeben als advanced feature)
- Zusammenstellung einer grafischen Maske zur Eingabe der ausgewählten Elemente; auch
  - wiederholbare Elemente; z.B. ein Werk kann von einem oder mehreren Autoren geschrieben worden sein
  - geblockte Elemente; z.B. eine *Person* und eine *Rolle* - z.B. J.W.Goethe, Autor; Jannidis, Editor; Flury, Übersetzer
- Speichern der Maske in einem Standard Format (empfohlen: XForms) im Grid
- Angabe, wohin Werte nach der Eingabe gespeichert werden (können auch mehrere Orte sein, in die parallel geschrieben wird)
  - welches Objekt
  - Position im Objekt
  - Adaptor (ein kleines Stylesheet, um dem Zielschema zu entsprechen; ebf. Auswahl eines Adaptors aus den im Grid vorhandenen, über den Streaming-Editor erzeugten "Metadatenanpassungsschemata")

#### 2.4.2.2.2 *Wunschkriterien*

- Import von existierenden Schemata: XML Schema, Relax NG
- Export eines Schemas: XML Schema, Relax NG
- automatische Ableitung eines Schemas aus einem vorhandenen TEI-Header

### 2.4.3 Masken-Editor

#### 2.4.3.1 *Produktübersicht*

Eine Maskendarstellung einschließlich einer Überprüfung der Eingabe in Bezug auf die jeweils gültigen Regeln bei der Eingabe der Werte, die in sinnvoller Weise in den XML-Editor integriert ist.

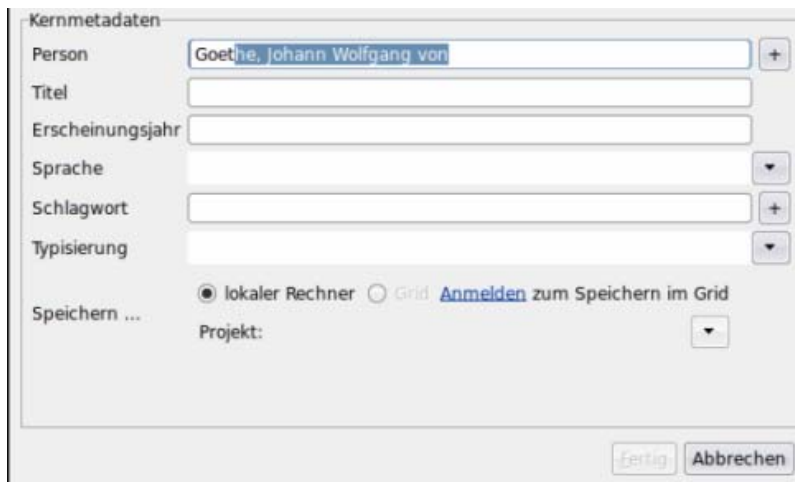
Die Maske wird durch den Projektleiter/-techniker im Schemagenerator vorgegeben.

(Die ursprüngliche Idee war, für die Beschreibung der Maske Xforms zu nehmen. Da auch hierbei Standards verwendet werden sollten, könnte man zur Beschreibung der Maske XForms verwenden).

Die folgende Zielbestimmung umreißt die Funktionalität des Tools.

#### 2.4.3.2 Zielbestimmung

Eine Eingabemaske nach den Vorgaben aus dem Schemagenerator. In der folgenden Grafik ist die Maske integriert mit dem "Speichern unter" Dialog im TextGridLab:



Man sieht Eingabefelder nach unterschiedlichen Typen. Wiederholbare Felder können durch das "+" am rechten Rand vervielfältigt werden.

##### 2.4.3.2.1 Musskriterien

- beim Öffnen einer Instanz des Masken-Editors wird aus der Umgebung übernommen: welches Objekt soll annotiert werden (die spezifische Maske ist mit dem Objekt verknüpft), und ggf. in welchem Eclipse-Fenster ist das Objekt gerade geöffnet
- Synchronisation mit dem Eclipse-Fenster: wird das Fenster geschlossen, wird der Masken-Editor auch geschlossen; wird im Masken-Editor etwas geändert, muss das im Fenster berücksichtigt werden; etc
- GUI zur Eingabe von strukturierten Daten, Maske nach Vorgabe des Schemagenerators (bzw. für die Metadaten: gibt es keine zugewiesene Vorgabe, dann entspricht die Maske dem Kernmetadaten-set)
- Feld, bei Eingabe: Überprüfung ob eingegebene Metadaten dem dahinterliegenden Schema entsprechen; bei einer Eingabe, die nicht dem Schema entspricht, bekommt der Nutzer einen aussagekräftigen Warnbildschirm
- Feld, nach Eingabe: Synchronisation der Metadaten-Maske mit einem (ggf. im TextGridLab geöffneten) Objekt
- Speichern der Metadaten an bestimmter Stelle in einem Objekt und im Grid (TG-crud)

##### 2.4.3.2.2 Wunschkriterien

- Einbettung von Bibliografiertool
- Drag-and-Drop Funktionalität
- Autocompletion bei Autoren, etc (siehe "Standardwerte" im Schemagenerator)

## 3 Im Berichtszeitraum weiterentwickelte Module

Updates zu den Pflichtenheften, sonstige Bemerkungen, weiterer Planungsstand.

### 3.1 Eclipse-GUI (vormals: Rich-Client-Platform)

Die grafische Benutzungsschnittstelle (GUI) zu den verschiedenen Tools in TextGrid basiert auf Eclipse und dessen *Rich Client Platform* genanntem Framework zur GUI-Entwicklung. Die Bestandteile (etwa spezifische GUIs für die einzelnen Tools) werden in einzelnen Plugins realisiert, die über gemeinsame Schnittstellen miteinander kommunizieren.

Dabei wird angestrebt, möglichst nahe an den Eclipse-eigenen Konzepten zu bleiben bzw. Eclipse-Mechanismen zu implementieren, um möglichst kompatibel zu unabhängig von TextGrid entwickelten Plugins zu bleiben. Damit kann

- eine eigenständige Applikation *TextGridLab* kompiliert werden, die auf die philologische Zielgruppe zugeschnitten ist,
- vorhandene, für die Zielgruppe sinnvolle Eclipse-Plugins in diese Applikation eingebunden werden,
- oder auch (obschon dies ein nachrangiges Ziel ist) eine vorhandene Eclipse-Installation mit TextGrid integriert werden.

Zu den im Integrationsprojekt zu leistenden Tätigkeiten gehört die Entwicklung des gemeinsamen Rahmenwerks mit Menü- und Perspektiveninfrastruktur, zusammen mit AP3 die Anbindung an die TextGrid-Utilities TG-auth und TG-search, die Entwicklung gemeinsamer Schnittstellen und Leitlinien sowie die Beratung der Entwickler der einzelnen Tools bzw. deren GUI-Bestandteile und zusammen mit AP4 die Koordination der Mittel zur Hilfestellung innerhalb von TextGridLab.

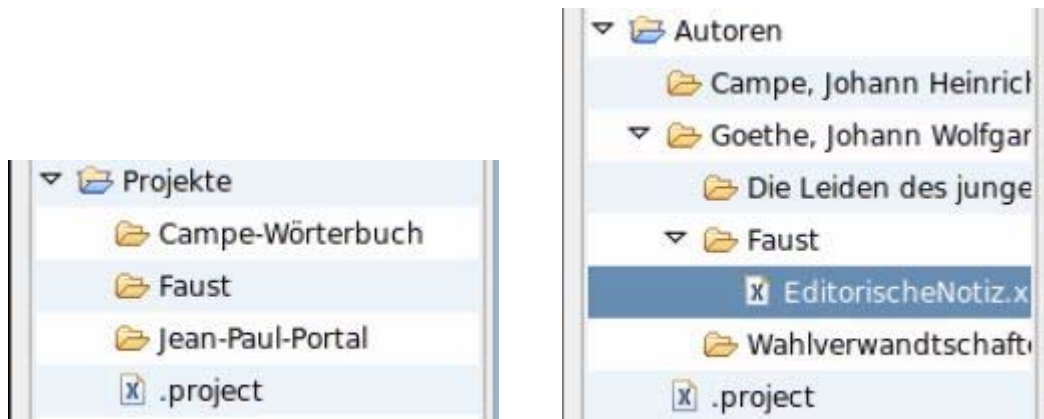
Zentral ist hier insbesondere die Modellierung der TextGrid-Objekte im Eclipse-Framework mit der Anbindung der TextGrid-spezifischen Infrastruktur zum Erzeugen, Lesen, Aktualisieren und Löschen von Objekten und ihren Metadaten (TG-crud, vgl. Report 3.3) an die Eclipse-spezifischen Implementierungen, sodass vorhandene Eclipse-Editoren zum Bearbeiten von TextGrid-Objekten benutzt werden können.

Ein detailliertes Pflichtenheft ist in Report 2.1 zu finden. Im Folgenden noch Beschreibungen zu spezifischen zentralen Komponenten des TextGridLab: Einem Projektbrowser zum Blättern durch die Projekte und Objekte des Benutzers sowie einem Willkommensbildschirm, der insbesondere neue Nutzer durch TextGridLab leiten wird.

#### 3.1.2 Projektbrowser

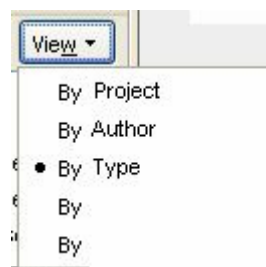
Das Projektbrowser ist ein ständiger Begleiter im TextGridLab. Es ermöglicht den einfachen Zugriff auf alle Projektmaterialien eines Benutzers. Der Navigator stellt damit einen möglichen Startpunkt für den Nutzer im TextGridLab dar.





### 3.1.2.1 Ziele

- Zugriff auf alle dem Benutzer zugänglichen Objekte
- primär eine hierarchische Darstellung, die auf verschiedenen Arten sortiert werden kann
  - nach Projekten → die darin enthaltenen Objekte nach einem zweiten Kriterium (Metadaten, siehe nachfolgend)
  - nach den TextGrid Kernmetadaten: Autor, Titel, Sprache, Objekttyp, etc
- Auswahl der Sortierung könnte ähnlich der Filter in der Firefox-History sein



- interaktive Tools können sich in das Context-Menü eintragen, sodass der Nutzer Objekte direkt z.B. mit dem XML-Editor, mit dem grafischen Link-Editor, etc öffnen kann
- die Visualisierung und Objektauswahl sollte in der Implementierung in einem öffentlich zur Verfügung stehenden Kontrollelement gekapselt werden, sodass andere GUI-Komponenten (ein Objekt-Auswahldialog etwa) diese Technik einfach nachnutzen können
- optional (an- und abschaltbar per Toolbar) sollte die Auswahl im Navigationswerkzeug mit dem aktuellen Editor gekoppelt werden, sodass in der Navigationsleiste das aktuell bearbeitete Objekt ausgewählt und ins Blickfeld gerückt wird (vgl. die diversen Eclipse-Navigationsleisten wie den Package Explorer).

### 3.1.2.2 Wunschziele

- bei einem Doppelklick auf ein Objekt wird ein passender Editor geöffnet. Hierzu gibt es eine Zuordnung Objekttyp Editor (customisable), sodass zB bei Bildern immer der grafische Link-Editor, bei xml-Dokumenten immer der XML-Editor, etc geöffnet wird.

### 3.1.2.3 Abgrenzung

- das Projektbrowser kümmert sich nur um Objekte aus Projekten, an denen ein Nutzer (in einer beliebigen Rolle: Projektleiter, Mitarbeiter, Beobachter) direkt teilnimmt. Veröffentlichte Objekte aus anderen Projekten - die zwar auch zugänglich sind, aber nicht zu einem Nutzer "gehören" - werden im Projektbrowser nicht abgebildet.

- Navigation in ein Objekt hinein (z.B. über ein Outline) könnte ggf. von einem eigenen Outline-Reiter ermöglicht werden. Dies ist nicht Bestandteil des Projektbrowsers.

### 3.1.3 Willkommensbildschirm

Nutzer (zumindest neue) sollten, wenn sie TextGridLab starten, durch einen Bildschirm begrüßt werden, der ihnen einen einfachen Einstieg in das Lab bietet.

Der Bildschirm sollte aus einem das gesamte Lab-Fenster einnehmenden View bestehen, der die wichtigsten Einstiegspunkte orientiert am Erfahrungsstand des Benutzers grob von links oben nach rechts unten präsentiert. Der Bildschirm sollte aufgeräumt wirken und eine enge Integration mit den Hilfetexten (AP4, Eclipse-Hilfe) und den vorhandenen Tools bieten.

#### 3.1.3.1 Musskriterien

##### 3.1.3.1.1 Einstieg zu Tools

Links mit Symbol und erläuterndem Text (ein Halbsatz, etwa ›Inhalte in TextGrid recherchieren‹, ›XML-Dateien bearbeiten‹) zu den wichtigsten Tools bzw. Ansichten. Die Links sollten idealerweise in eine Ansicht führen, in denen ein Tutorial/Cheatsheet oder die dynamische Hilfe eingeblendet wird.

##### 3.1.3.1.2 Einstieg zu TextGrid

Weiter unten sollten Links zum Anmelden bzw. Registrieren dargeboten werden und erläutert, welche Vorteile es bringt, sich als Nutzer anzumelden. Außerdem sollte ein Link auf die TextGrid-Projektseite bzw. das Community-Portal vorhanden sein. Ggf. kann auch der Anmeldebildschirm hier unten direkt integriert werden.

##### 3.1.3.1.3 Ggf. Kurzhilfe

... à la »Klicken sie hier für ....«, TODO Was genau anbieten? Das hängt wohl vom letztendlichen Aussehen des Labs ab

#### 3.1.3.2 Wunschkriterien

##### 3.1.3.2.1 Anpassung für angemeldete Benutzer

Login- bzw. Registrierschirm durch Logout ersetzen

##### 3.1.3.2.2 Anpassung an häufigere Benutzer

z.B. Liste zuletzt / häufig benutzter Objekte

## 3.2 Tokenizer

- Umstellung auf SAX-Parser, Modul aus 4Suite<sup>20</sup>.
- Die Regeln zur Identifizierung von Wörtern wurden als reguläre Ausdrücke formuliert, was zusammen mit der Parser-Umstellung einen erheblichen Performance-Gewinn bringt.
- Das in Report 2.1 unter „Wunschkriterien“ genannte Taggen von Satzzeichen wurde ebenso umgesetzt wie das über eine Konfigurationsdatei steuerbare Preprocessing.
- Die oben unter 1.3 genannten Umstellungen wurden auch für den Tokenizer vorgenommen.

---

<sup>20</sup> <http://4suite.org>  
Report\_2.2.doc

## 3.3 Morphologische Analyse, Lemmatisierer

### 3.3.1 Morphologische Analyse mit MorphHisto:

Die morphologische Analyse in TextGrid basiert auf der Finite State Technologie (siehe Report 2.1). Die derzeitige Implementation von MorphHisto verzichtet jedoch auf die AT&T FSM Tools bzw. Lextools, die lizenzrechtlich geschützt sind (eingeschränkte GPL).

Stattdessen werden im neuen Release die frei-verfügbaren SFST Tools der Universität Stuttgart verwendet (Schmid 2005), die bereits erfolgreich mit der Morphologiekomponente SMOR (Schmid, Fitschen, Heid 2004) eingesetzt wurden. Eine minimale Basisgrammatik bestehend aus den wesentlichen deutschen morphotaktischen und orthographisch / phonologische Regeln steht damit ebenfalls frei zur Verfügung. Da allerdings das lexikalische Wissen in SMOR nur auf wenige lexikalische Einträge (<500) beschränkt ist, wurden ca. 30.000 weitere lexikalische Einträge (mit Informationen zum Lemma, Wortklasse, Paradigma) ergänzt<sup>21</sup>.

Die Lemmaauswahl orientiert sich an der Liste der 30.000 häufigsten Wörter bezogen auf das Cosmas Korpus (DeReWo Liste; siehe <http://www.ids-mannheim.de/kl/derewo/>). Für diese hochfrequenten (oftmals lexikalisierten) Wörter der deutschen Sprache liefert der Transducer eine korrekte Analyse, was durch eine manuelle Verifizierung sichergestellt wurde. Die eigentliche Abdeckungsrate des Morphologietools ist dank der Wortbildungsanalyse jedoch wesentlich höher.

Mit Hilfe der SFST Tools (und Teilen der SMOR Grammatik) wurde zudem ein separater Transducer kompiliert, der die freien Wortklassen des Adellung-Wörterbuches (1793, <http://www.zeno.org/Adellung-1793/>) und somit einen Wortschatz von ca. 60.000 Einträgen abdeckt. Hierzu wurde das Adellung Wörterbuch in ein auf SMOR-basierendes Lexikonformat konvertiert.

### 3.3.2 Technische Produktumgebung

#### 3.3.2.1 Software: für Server und Client, falls vorhanden

Je nach Programmiersprache entspr. Installation auf Server bzw. Arbeitsplatzrechner (lokaler Einsatz)

Server:

- Apache 2.0.x mod\_python,
- Python 2.4.x mit zusätzlichen Paketen:
  - ZSI-2.0\_rc3 (<http://pywebsvcs.sourceforge.net>)
  - PyXML 0.8.4 (<http://pyxml.sourceforge.net>)
  - SOAPpy 0.11.6 ([http://sourceforge.net/project/showfiles.php?group\\_id=26590&package\\_id=18246](http://sourceforge.net/project/showfiles.php?group_id=26590&package_id=18246))
  - fpconst 0.7.2 (<http://cheeseshop.python.org/pypi/fpconst/0.7.2>)
  - SFST Tools (<http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/SFST.html>)
  - pysfst (<http://home.gna.org/pysfst/>)

---

<sup>21</sup> Das bereits vorhandene lexikalische Material (siehe Report 2.1) konnte in kurzer Zeit konvertiert und in SMOR integriert werden  
Report\_2.2.doc

Client:

- Java ab 1.5, Eclipse 3.3, Apache Axis2 1.3.

### 3.3.2.2 Hardware: für Server und Client getrennt

Entsprechend den allgemeinen Projektvorgaben.

### 3.3.2.3 Produkt-Schnittstellen

Derzeit implementiert:

*Front-End für den Wörterbuchzugriff bzw. Retrieval (einfache Normalisierung)*<sup>22</sup>

Service für das Nhd. ist publiziert unter <http://ingrid.daasi.de/lemmatizer.wsdl>

Service für das frühe Nhd. (Adelung) unter <http://ingrid.daasi.de/cgi-bin/morphisto3.cgi>

Parameter:

inword	Eingabewort
returnlemma	Analysestring (Lemma und morpho-syntaktisch Angaben)

*Front-End für den Batchbetrieb (Annotierung eines tokenisierten Files)*<sup>23</sup>

Service für das Nhd. wird publiziert unter [http://ingrid.daasi.de/lemmatizer\\_batch.wsdl](http://ingrid.daasi.de/lemmatizer_batch.wsdl)

Service für das frühe Nhd. unter <http://ingrid.daasi.de/cgi-bin/morphisto3.cgi>

Parameter:

infile	Eingabedatei
returnfile	Analysedatei (Lemma und morpho-syntaktisch Angaben)

## 3.4 Workflow-Editor

Der aktuelle Planung des Workflow-Editors entspricht weitgehend noch der im Report 2.1 (TextGrid-Tools I) niedergelegten Spezifikation. Seitdem wurden hauptsächlich evaluierende und integrative Arbeiten an verschiedenen Workflow-Editor und -Enactor-Lösungen durchgeführt. Ergebnisse:

- Evaluation verschiedener Workflow-Editoren, -Engines und integrierten Umgebungen. Weitere Kriterien: Open Source, Web Service-Fähigkeit, Unterstützung von UDDI o.ä. als WSDL-Repository, Format der WF-Beschreibungssprache.

Evaluierbare Systeme: Triana, Taverna, Kepler, Karajan, GridAnt, Intalio, OMII-BPEL, EclipseBPEL Designer+ActiveBPEL, P-Grade, g-Eclipse Workflow Editor, Virtual Data System.

Ergebnis:

---

<sup>22</sup> Für die Einwort-Lemmatisierung kann der Webservice nun auf einen speicherresistenten lokal laufenden Lemmatizierungs-Service ("Lemmatizer-Daemon") zurückgreifen, der das wiederholte Laden eines Lemmatizierungs-Automaten überflüssig macht

<sup>23</sup> Für die Batch-Lemmatisierung auf tokenisierten Daten wird die infl2-Funktion der SFST-Tools genutzt.

- WS-BPEL-basierte Lösungen sind sehr mächtig, aber kaum nutzerfreundlich und recht komplex. Trotz Standardisierung setzen verschiedene Engines verschiedene Zusatzinformationen voraus, die den Einsatz unabhängiger Editoren, z.B. Eclipse BPEL Designer erschweren; die Integration fehlt
- Als „kleine Lösung“ kann Taverna mit dem DAG-basierten XSCUFL-Format implementiert werden. Die Erstellung eines WF ist nicht zu komplex, das Format ist mächtig genug für TextGrid-Belange. Problem ist hier die fehlende Integration in Eclipse.
- Nähere Evaluation und Integration von ActiveBPEL Designer, Eclipse BPEL Designer und ActiveBPEL Engine. ActiveBPEL Designer (closed-source) und ActiveBPEL Engine (open source) arbeiten zusammen, aber nicht der Eclipse BPEL Designer (open source) mit der ActiveBPEL Engine; Integration wurde nicht erreicht
- Nähere Evaluation der „kleinen Lösung“: Programmierung eines eigenen Editors als Frontend zur Taverna-Engine auf Basis von Eclipse GEF. Versuch, das SCUFL-Objektmodell aus Taverna für GEF wiederzuverwenden. Das Taverna-Objektmodell ließ sich nur bedingt übertragen, Abhängigkeiten fast vom gesamten Taverna-Code wurden entdeckt; Integration ließ sich nicht erreichen.
- Weitere Evaluation des BPEL-Ansatzes, u.a. weitere Untersuchung des Eclipse BPEL Designers mit der ActiveBPEL Engine und des Intalio|BPMS Systems. Ergebnis der Arbeit steht noch aus. Erste erreichte Ziele: Integration von Eclipse BPEL Designers und ActiveBPEL Engine erreicht und an TextGrid-Workflows erfolgreich getestet; aktuell laufen Tests mit Intalio.
- Aktuelle Linie: Ausbau des Codes des Eclipse-basierten Workflow-Editors von g-Eclipse für den Export in das Taverna's Xscufl-Format und Verwendung der Engine von Taverna. Erste Evaluationen liefen und Kontakt zu beiden Entwicklergruppen ist hergestellt. Die g-Eclipse-Entwickler werden Anfang 2008 im Rahmen der Entwicklungen für die GRIA-Middleware auch Support für Taverna evaluieren.

Wie aus dem Vorangegangenen deutlich wird, geht es in dieser Phase hauptsächlich um die Ablösung des ursprünglichen, nicht integrierten Protoyps *Taverna* durch eine vollkommen auf Textgrid abgestimmte und in das *TextGrid Lab* eingebundene Implementierung.

### **3.5 XML-Editor**

Im Rahmen des XML-Editors fanden im Berichtszeitraum u.A. Weiterentwicklungen an Visualisierungen, eine neue Render-Infrastruktur zur flüssigeren Bearbeitung größerer Dateien, eine Anpassung an TEI, eine komfortable Eingabemöglichkeit für auch nicht auf üblichen Tastaturbelegungen vorhandene Unicode-Zeichen, die Entwicklung einer Einstellungsinfrastruktur sowie Vorbereitungen zur Integration von grafischem und Quellcodeeditor sowie einem gemeinsamen Datenmodell statt, darüber hinaus natürlich zahlreiche Bugfixes.

Die Entwicklung wird insbesondere in der ersten Hälfte des kommenden Jahres fortgeführt, sodass mit der Teaser-Version (vgl. die Roadmap in Abschnitt 1.2) eine stabile Version vorliegt.

## 4 Planungsstand restliche Module

### 4.1 Grafischer Link-Editor

#### 4.1.1 Überblick

Der grafische Linkeditor (man könnte ihn auch Topographie-Editor nennen) hat die Aufgabe, den XML-Editor bei der Alignierung von Text und Bildelementen zu unterstützen. Ziel ist die Erstellung einer Ausgabedatei, die die Textelemente und die topographische Beschreibung enthält (frei konfigurierbar: Standoff oder im Text; Format der Notation). Dabei sollen wie üblich Standards zur Anwendung kommen.

#### 4.1.2 Diskussion

Drei heikle Punkte:

##### 4.1.2.1 Headerinformation

Man wird dem Benutzer für die Transkription nicht erneut die Eingabe aller Metadaten abverlangen, die er bereits für die Grafik eingegeben hat. Also sollten diese Angabe soweit wie möglich weiterverwendet werden. 2 Szenarien:

- a) Bilder sind in TextGrid, also aus den TextGrid-Metadaten kopieren.
- b) Bilder sind lokal vorhanden, Metadaten werden über ein definiertes Format eingespielt.

##### 4.1.2.2 Erstellung einer vollständigen TEI-Datei

Die kleinste TEI-Datei besteht aus einem Header (s.o.) und einem Body, in dem die einzelnen Elemente auch noch einmal in div-Elemente eingepackt sind. Vorschlag: Man erlaubt dem User die Definition von Wrapperangaben, z.B. der User definiert jeweils den Ort von Wörtern (<w>) in einer Handschrift und er gibt dann als 1. Wrapper die Zeile (<l>) und als 2. Wrapper ein div ein. Die Gültigkeit des TEI sollte hier nicht geprüft werden.

##### 4.1.2.3 Konfiguration des Ausgabeformats

Das ist nicht ganz unproblematisch, da im Idealfall ein ganz freies Format gewählt werden können sollte (dann könnte man das Tool z.B. auch für HTML-Imagemaps verwenden). Das betrifft nicht nur die Schreibung der Koordinaten, sondern auch die Festlegung des Koordinatenursprungs. Außerdem wird man gerade bei hochwertigen Scans evtl. mit verkleinerten Abbildern arbeiten, dann kann man nicht mit den Pixelangaben im Bild arbeiten, sondern muss ein virtuelles Grid über das Bild legen und das als Koordinatensystem verwenden. Hier der Vorschlag, dies durch eine Konfigurationsdatei zu definieren.

### 4.1.3 Eingabe

#### 4.1.3.1 Eingabe der Bildmarkierung

Die Eingabe geschieht interaktiv, indem der Editor auf einem Bild der Handschrift die Zeichen, Worte, Zeilen, Absätze oder andere Einheiten grafisch markiert und dann den Text manuell eingibt oder aus einer vorhandenen Transkription zuordnet. Bei der Eingabe wählt der Editor zwischen den Möglichkeiten ein Rechteck aufzuziehen oder ein beliebiges Polygon zu zeichnen (Optionsmenü + Shortcuts wg. häufigem Wechsel). Die bereits markierten Ausschnitte in der Grafik bleiben erhalten, werden aber grau markiert im Gegensatz zur aktiven Markierung. Solche alten Markierungen können wieder aktiviert werden und der zugehörige Text wird dann ebenfalls angezeigt - um Fehler zu korrigieren.

#### *Weitere Features der Eingabe:*

- Die Grafik ist frei zoombar.
- Einblendbar: ein kleines Übersichtsfenster über die gesamte Grafik mit Markierung des großen Ausschnitts. Diese Markierung kann auf dem Überblicksfenster verschoben werden und verändert den dargestellten Ausschnitt im großen Fenster
- Einblendbar: ein Fenster mit Bildausschnitten. Rechtecke können mit geringem Aufwand (aufziehen, rechte Maustaste, Menüpunkt) dort abgelegt werden) Vorbild: Keller-Edition. Dient dem Handschriftenvergleich.

#### *4.1.3.2 Texteingabe*

Prinzipielle Wahl zwischen „Neuen Text eingeben“ und „Vorhandenen Text zuordnen“

#### *Neuen Text eingeben:*

Sobald die Markierung eines Rechtecks bzw. Polygons abgeschlossen ist, öffnet sich eine Eingabemaske für die Texteingabe (verfügt über konfigurierbare Menüleiste, um Tags per Knopfdruck einzugeben, z.B. zur Kennzeichnung von Streichung, Einfügung usw.).

#### *Vorhandenen Text eingeben*

Sobald die Grafikmarkierung abgeschlossen ist, öffnet sich ein Textfenster, das den gesamten Text enthält. Position des Cursor nach der letzten Markierung. Benutzer markiert im Text die Zeichen, die zur Grafiknotation passen und bestätigt.

(In beiden Fällen statt sich öffnender Fenster wahlweise auch Anzeige des eingegebenen Textes in einem Fenster neben der Handschrift.)

### **4.1.4 Konfiguration**

- Strukturinfo mitabspeichern (default: ja)
- Art der Zeilen-/Strukturinfo (entweder <l>, <w> oder <milestone> oder <p> oder freie Eingabe)

(diese Optionen erlauben es, ganze Zeilen, Worte usw. zu markieren und auch die strukturelle Information gleich mitabzuspeichern)

- Ganze Grafik als Überblick anzeigen
- Toggle: Neuen Text eingeben – Vorhandenen Text markieren (Dateinamen zuordnen)
- Textfenster neben Grafik anzeigen
- Toggle: Ausschnitt markieren: Block – Vieleck
- Eingabe: Innenwrapper: (z.B. <lg> für Verse oder <line> für einzelne Wörter)
- Eingabe: Außenwrapper: (z.B. <div>)
- Eingabe: Herkunft des Headers: a) Datei (hier muss ein Format definiert werden), b) URI (für die TextGrid-Metadaten)
- Eingabe:
- Toggle: In eine Datei – Standoff Markup (Für die Ausgabe)
- Eingabe: Konfigurationsdatei zuordnen (enthält Angaben über die Art der Ausgabe)

Wenn die Konfigurationsdatei fehlt, dann wird sie automatisch generiert: Standard TEI P5

- Die Konfigurationsdatei enthält:
  - Template mit Variablen für Koordinaten / enthaltener String

- Informationen zur Berechnung/Skalierung der Koordinaten (Koordinatensystem, Skalierung)

Beispiel:

```
<config>
  <coordinates>
    <origin y-axis="top" x-axis="left"/>
    <!-- yyaxis: top | bottom | center; xaxis: left | right | center --!>
    <maxx>1</maxx>
    <!-- or <imgwidth/> -->
    <!-- statt floating point auch große Zahlen möglich!>
    <maxy>1</maxy>
  </coordinates>
  <template>
    <<[CDATA[ <rect x0="$x0" y0="$y0" x1="$x1"
y1="$y1">$content</rect>]]>
  </template>
</config>
```

Der Nutzer kann also umstellen, wo der Ursprung des Koordinatensystems liegt und welche Einheit (entweder Pixel oder ein virtuelles Grid, dessen Dimensionen frei festgelegt werden können) verwendet werden soll.

#### 4.1.5 Ausgabe

Die Ausgabe geschieht in XML. Entweder wird der Text zusammen mit den topographischen Informationen abgespeichert oder diese als stand-off Markup in eine eigene Datei.

Unterstützt wird: Ausgabe als TEI P5 (nur Rechtecke) oder TEI P5 + SVG für die Polygone. Der Editor verwendet ein internes Speicherformat und kann die konfigurierbaren XML-Formate nur schreiben.

#### 4.1.6 Preprocessing

Das Programm sollte so gebaut sein, dass zu einem späteren Zeitpunkt die Grafiken vorbereitet werden können. Szenario: User benennt einen Text, der eine Transkription enthält. Ein Preprocessor versucht Zeilen im Bild zu erkennen und ordnet diese Zeilen dem Text zu.

### 4.2 Link-Editor Text

Der Link-Editor Text ist eine Eingabehilfe für Links in XML-Dateien und verbindet Elemente von Recherchetooll und XML-Editor. Benutzer haben die Möglichkeit, Links zu beliebigen Elementen in TextGrid-Dokumenten in die aktuelle TEI-Datei einzugeben, in dem über Recherchetooll, Projektbrowser, XML-Outline-Darstellung oder direkt im XML-Editor das Zieldokument bzw. -element gewählt und über Kontextmenü oder ggf. Drag & Drop eine entsprechende Funktion ausgewählt wird; es wird dann die passende URI samt Fragment zum Einfügen in das Quelldokument des Links generiert.

Darüber hinaus wird er zur Validierung bestehender Links eingesetzt.



## 4.3 Bibliographietool

### Produktübersicht

Das Bibliographie-Tool ist ein Werkzeug, das dazu dient, bibliographische Daten entweder aus bereits vorhandenen Datenbeständen zu importieren, insbes. aus Bibliothekskatalogen (z.B. BDSL, zvdd, Kalliope) und den TEI-Headern (Metadaten) bestehender (TextGrid-)Objekte, Bibliographien über eine definierbare Maske zu erfassen, zu bearbeiten, zu verwalten und an einer frei bestimmbaren Position in einer Datei einzufügen / abzuspeichern bzw. bibliographische Daten in best. Standard-Formaten zu exportieren (z.B. TEI, MODS). Das Tool unterstützt Formatierungskonventionen (Zitierformate) nach üblichen Standards (z.B. MLA Style). Es gibt zahlreiche Berührungspunkte / Überschneidungen mit anderen Tools (Metadaten-Annotation, XML-Editor, etc.).

Die Erstellung der bibliographischen Daten steht üblicherweise am Anfang eines Projektes, begleitet dies jedoch auch permanent (Daten werden in der Regel kontinuierlich ergänzt bzw. verbessert). Sie werden in vorhandenen Katalogen und Verzeichnissen recherchiert bzw. selbst (nach Autopsie) neu erstellt.

Genauso wie Daten importiert werden können, sollen Daten auch (durch verwendete Standards) in bestehende Kataloge exportiert werden können. Die Daten können als Ganzes oder in Teildatensätzen exportiert werden, es kann eine Auswahl der zu exportierenden Felder getroffen werden.

### Musskriterien

#### *Ausgangspunkt*

- jedes *Projekt* hat eine Bibliografie, in der die Referenzen "flach" verwaltet werden; ggf Verwaltung durch Tags
- Einträge können dupliziert werden (von anderem Projekt, mit anderen Rechten bzw. um Zitierung unterschiedlicher Kapitel zu ermöglichen)

#### *Rechte*

- eine Projektbibliografie kann sein:
  - a. öffentlich lesbar oder nicht lesbar
  - b. von Mitarbeitern bearbeitbar oder nicht bearbeitbar→ Default-Einstellung ist öffentlich lesbar und von Mitarbeitern bearbeitbar

#### *Bildschirm (Mockup)*

- zentral ist ein Schirm mit den Bibliografiedaten - jede Zeile ist ein Eintrag; ein Eintrag kann ein Textgrid Objekt sein, oder ein reiner Metadatensatz
- Doppelklick auf eine Zeile öffnet das Metadaten-Annotationstool - man kann die Metadaten bearbeiten, und/oder direkt eine Datei zu dem Metadatensatz hochladen (wird zu einem TextGrid Objekt)
- eindeutige Identifikation eines Eintrags wird analog dem TextGrid URI aufgebaut (soweit sinnvoll): textgrid:'Projekt':'Titel':'Datum (created)' bzw einfach ein unique ID vergeben - ist manuell bearbeitbar

Entry...	Author ▲	Title	Year ▼
Misc	{American Council of Learned Soci...	Social Sciences: Our Cultural Commonwealth	2006
Misc	Anderson	E-Science for the Arts and Humanities: A discussio...	
Misc	Berliner and Polk	{CVS}: Concurrent Versions System	
Article	Beynon et al.	Human Computing - Modelling with Meaning	2006
Incollecti...	Bradley	Text Tools	
Book	Busa	Index Thomisticus	1974
Misc	Caesar and MacCalla	e-Humanities in a Digital Society	
Article	Childs et al.	A Single-Computer Grid Gateway Using Virtual Mac...	2005
Article	Colazzo et al.	A Typed Text Retrieval Query Language for {XML} Do...	2002
Misc	CollabNet	Subversion	
Article	Corcho et al.	An overview of S-OGSA: A Reference Semantic Grid ...	2006
Incollecti...	Crane	Classics and the Computer	
Manual	Cunningham et al.	Developing Language Processing Components wit...	
Techrep...	on Digital Archive Attributes	Trusted Digital Repositories: Attributes and Respon...	2002
Book	{European Commission}	From Grids to Service-Oriented Knowledge Utilities	2006
Misc	Fish	Better {SCM} Initiative: Comparison	

Abbildung 1 - Jabref TextGrid.bib

### Daten

- TextGrid Kernmetadaten - alles andere wird (vorläufig) "weggeschnitten"
- optional weitere Felder aus dem TEI-Header können angezeigt und manuell nachgetragen werden
- Verwaltung von Bibliothekssignaturen/Standorten (mehrere)
- Verlinkung mit Bibliothekskatalogen (extern)
- Daten können mehrfach vorgehalten werden, und haben keine Abhängigkeiten untereinander (wenn ein Datensatz geändert wird, ändern die anderen nicht mit)
  - zwischen Projekten, um z.B. unterschiedliche Rechte zu erlauben
  - innerhalb eines Projektes, für z.B. unterschiedliche Details (Kapitel x, Seite y)
- ein Zusatzeintrag ist ein Kurztitel (bibref), der automatisch unique vergeben wird, bzw manuell angepasst werden kann

### Daten-Import

- aus dem Recherchetooll können per Drag+Drop TextGrid Objekte in das Bibliografietool gezogen werden - Referenz zu TextGrid Objekt wird gelegt und Metadaten werden dargestellt
- Kataloge: zvdd und kalliope - diese Metadaten werden kopiert, potenziell unvollständig
- manuelle Eingabe
- ausserdem einlesbare Formate: MODS, TEI-Header
  - evtl. farbliche Darstellung, woher die Daten sind, aber keine explizite Trennung der Quellen in der Darstellung
  - Suche findet jeweils getrennt statt: in Recherche-Tool, in Katalogen oder in anderen (öffentlichen) Bibliografiedaten

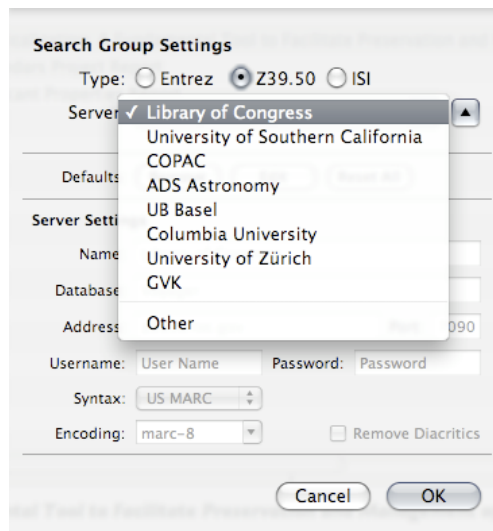


Abbildung 2 - Bibdesk: Katalogsuche

### *Daten-Export*

betrifft beides

1. Generierung lesbarer Referenzen für existierende TextGrid-Objekte (aus den Metadaten oder TEI-Headern der entsprechenden Objekte)
2. Export eines Metadatensatzes bzw mehrerer Metadatensätze bis hin zu einer kompletten Bibliografie in einem Standard Metadatenformat (z.B. MODS)

- einzelne Bibliografieeinträge sind über einen URI referenzierbar, über ein Fragment auch in einem bestimmten Format. z.B. `uri://textgrid:bibref#format=TEI`
- dadurch kann z.B. im XML-Editor eine Literaturliste in einem bestimmten Format erstellt werden (alle Werke sind einzeln mit Format-Fragment aufgelistet)
- mit Rechtsklick auf einen Bibliografieeintrag kann im Kontext-Menü über "Kopieren als ..." (z.B. TEI-Header) oder "Zitieren als ..." (vgl. MLA-Style) und der Wahl eines Formats eine formatierte Referenzierung kopiert werden
- Möglichkeit im Bibliografiertool mehrere Einträge zu markieren und in einem Format nach Wahl zu exportieren
- Formate sind in einem ersten Schritt: TEI-Header, MODS (über bibutils: TeX, Endnote)

### *Verknüpfung TextGrid*

- Erfassung Metadaten: bei Doppelklick auf einen Datensatz öffnet sich der *Metadateneditor*, und man kann direkt dort ein zugehöriges Objekt hochladen → wird zu TextGrid Objekt
- im *Schemagenerator* angegebene Einschränkungen für den Metadatensatz gelten auch im Bibliografiertool. In der Bearbeitung im *Metadateneditor* kann das direkt übernommen werden. Weiters müsste dieser Aspekt auch für die Darstellung in der Tabellenansicht übernommen werden.
- Literaturlisten: ein Eintrag im Bibliografiertool kann (in verschiedenen Zitierformaten) kopiert werden, und dann z.B. im *XML-Editor* eingefügt werden (ggf auch durch Drag+Drop)
- Tools wie z.B. der *Web-Publisher* greifen auf die REST-Schnittstelle des Bibliografiertools zu (siehe Abschnitt "Daten-Export"), um ggf. druckbare Bibliografien zu generieren

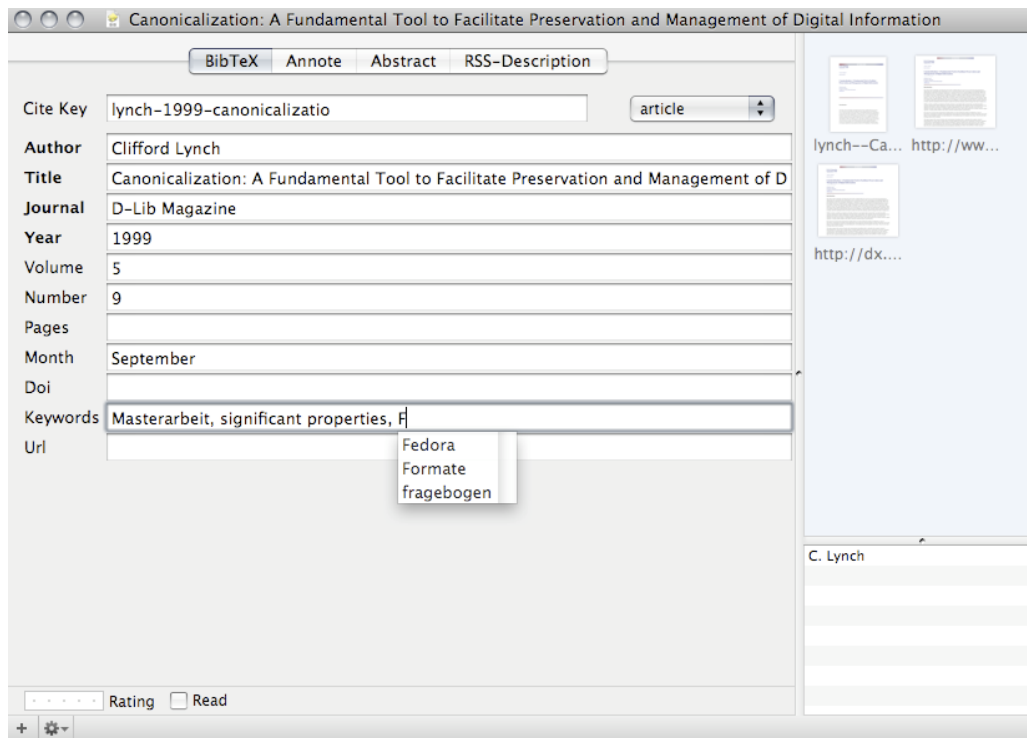


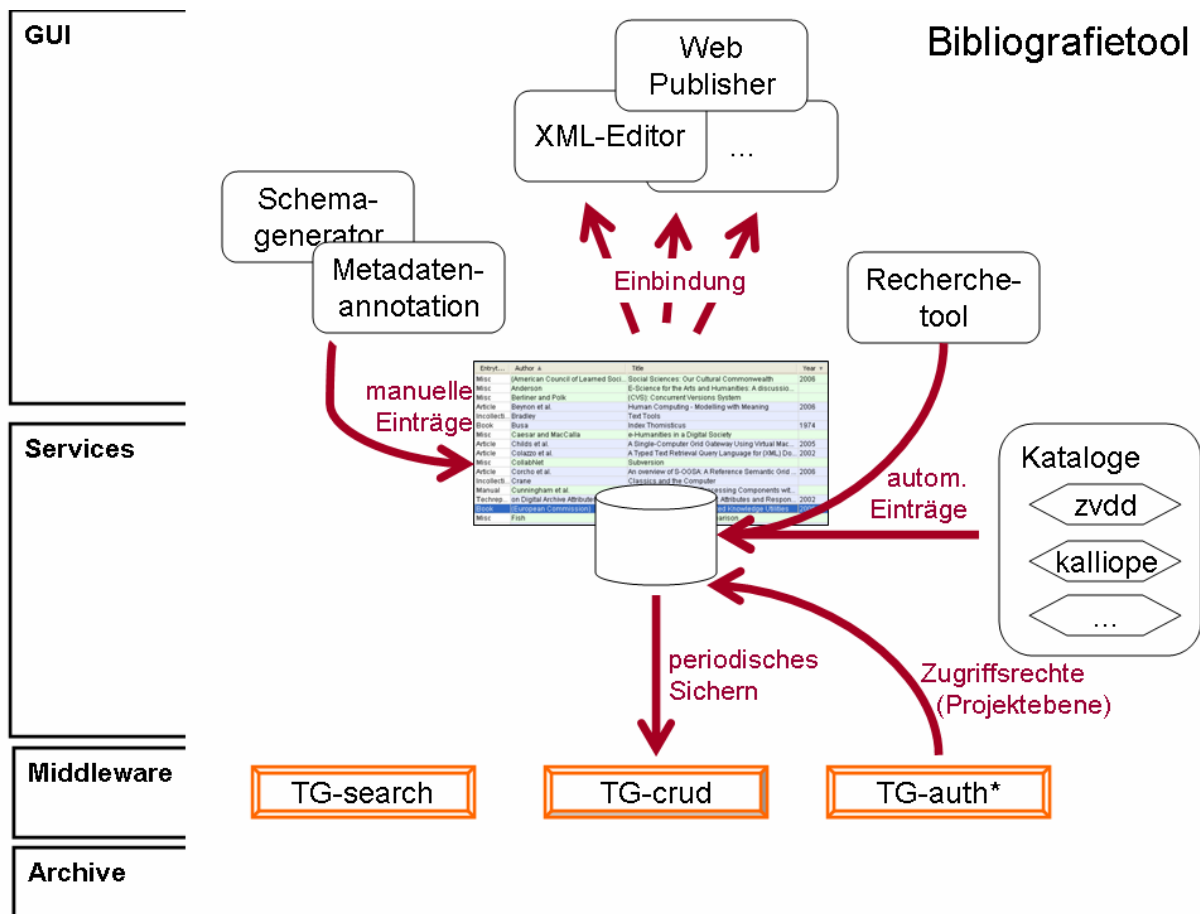
Abbildung 3 - Bibdesk: Metadateneingabemaske mit Upload, Tags (Autocompletion), etc

## Wunschkriterien

- Auto-Ergänzungsfunktion (z.B. über ISBN)
- Verschlagwortung/sachliche Gliederung (mehrere) → Einbindung von Normdateien
- Verlinkung zum Volltext bzw. zu Exzerpten (extern)
- Verwaltung individueller bzw. kollaborativer Bemerkungen/Anmerkungen
- Verknüpfung zu zotero?
- Import aus <http://www.bdsl-online.de/> (Dublin Core orientiertes Format)
- Export in MARC-XML, DocBook
- regelmässiger Export in das TextGrid als MODS Datei - zur Datensicherung (längerfristig)
- Verknüpfung mit Normdaten (Authority Files, PND, etc)
- Anbindung an die EDL (Katalog)

## Abgrenzungskriterien

Die Erstellung einer dynamischen Bibliografie a la Bibtext `\cite{}` in Dokumenten muss von dem Editor verwaltet werden. Der TextGrid XML-Editor, OpenOffice, oder andere Editoren könnten dies über das TextGrid Bibliografietool ermöglichen, dies ist aber nicht die Aufgabe des Bibliografietools.



## 4.4 Sortieren

### 4.4.1 Produktübersicht Sortieren

Mit Hilfe des Sortiertools kann ein Anwender gegebene Folgen von Zeichenketten gemäß kulturellen und fachlichen Erwartungen anordnen. Die dabei zu berücksichtigenden Sortierschlüssel und Sortierkonventionen kann der Nutzer frei vorgeben. Zur Vereinfachung der Handhabung erlaubt das Sortiertool aber auch, sich direkt auf einschlägige nationale und europäische Standards, etwa DIN 5007, NFZ44-001 (AFNOR), TK 34.1 (SIS) und ENV 13710 (CEN) zu beziehen.

### 4.4.2 Zielbestimmung

Wenn ein Textdokument - beispielsweise ein Index oder eine Sammlung von Wörterbuchartikeln - sortiert werden soll, so muss das entsprechende Tool zunächst die Möglichkeit bieten, Sortierfelder und -schlüssel zu definieren, Stoppwörter anzugeben, die bei der Sortierung ignoriert werden, und nicht zuletzt die gewünschte Sortierreihenfolge festzulegen.

Die in vielen Editionsprojekten bewährten Module #SORTIERE bzw. #SVORBEREITE aus TUSTEP verlangen, dass das Sortieralphabet für jeden Schlüssel explizit angegeben wird. Dies ist angesichts der Fülle der in Unicode zur Verfügung stehenden Zeichen nicht mehr zeitgemäß; von einem modernen Sortierer kann man erwarten, dass die Auswahl des Sortieralphabetes durch die Angabe eines Kulturraumes möglich ist; genauer gesagt durch die Benennung der Lokale, welche die Sortiergepflogenheiten des entsprechenden Kulturraumes wie in den einschlägigen Standards festgelegt beschreibt [Küs2001]. Weil es immer noch Sonderfälle oder in den Standards nicht erfasste Sonderzeichen geben kann,

muss es ferner möglich sein, die durch die Lokale gegebenen Sortierregeln zu ergänzen oder modifizieren.

Grundsätzlich erscheint die Aufteilung in drei Tools SVORBEREITE - SORTIERE - SNACHBEREITE jedoch sinnvoll, um komplexen Szenarien durch eine Zwischenschaltung des Streaming Editors gerecht zu werden. Standardsituationen können dann durch vordefinierte Workflows oder einen "Fassadenservice", der hinter den Kulissen die drei Dienste aufruft, abgehandelt werden.

#### 4.4.2.1 Musskriterien

##### *Markierung von Datensätzen/Feldern:*

- Über die Konfiguration muss der Anwender Beginn und Ende der Datensätze (z.B. eine Zeile in einer CSV-Datei oder ein Datensatz in einer XML-Datei) markieren können. Minimal genügt es, wenn diese Markierungen durch beliebige XPath-Ausdrücke definiert werden kann, denn reine Textdateien können ggf. mit dem Streamingeditor in ein einfaches XML-Schema konvertiert werden. Dann kann auf dem so entstandenen XML-Dokument mit XPath gearbeitet werden, da XPath auch reguläre Ausdrücke unterstützt.
- Die Software muss mehrere Marker für Datensätze unterstützen.
- Für die Markierung von Sortierfeldern gilt sinngemäß das gleiche wie für Datensätze.

##### *Modifikation von Einträgen:*

- Textblöcke, die für die Sortierung (nicht) einschlägig sind, müssen aus den Sortierschlüsseln gelöscht bzw. darin eingefügt werden können. Wörter, die regelmäßig ignoriert werden sollen, müssen über Stoppwortlisten konfiguriert werden können.
- Datensätze müssen für die Sortierung umgebaut werden können, so dass z.B. ein Eintrag „John von Neumann“ als „von Neumann, John“, „von Neumann, Johann“ oder auch als „Neumann, Johann von“ behandelt wird:
  - Konfiguration von Regeln zur (a) Erkennung und (b) Behandlung von Titeln u.ä. (vgl. hierzu auch Regeln für die Alphabetische Katalogisierung RAK)
  - Umformung von Sortierschlüsseln durch Zugriff auf existierende Normdatenbanken
- Auf Wunsch muss das Sortiertool Duplikate aus der Ergebnisliste streichen können, vergleichbar dem auf POSIX-Systemen verfügbaren Programm *uniq*.

##### *Sortierreihenfolgen:*

Die Anordnung von gegebenen Zeichenketten muss kulturellen und fachlichen Erwartungen der jeweiligen Nutzer entsprechen, weshalb Anwender die Konventionen explizit vorgegeben können müssen. Die in den einschlägigen nationalen und europäischen Standards definierten Kollationierungssequenzen müssen zur Vereinfachung der Bedienung direkt unterstützt werden. Dies bedeutet im Detail:

- Jedes Sortierfeld muss eigene Sortierreihenfolge unterstützen.
- Die in DIN 5007, NFZ44-001 (AFNOR), TK 34.1 (SIS) und ENV 13710 (CEN) beschriebenen Sortierreihenfolgen sind bereits vordefiniert.
- Regeln für die Transliteration anderer Schriften (z.B. griechische Ausdrücke in lateinischen Listen in Altertumslexika) können vom Anwender definiert werden.
- Die Definition eigener bzw. Modifikation existierender Sortierreihenfolgen erfolgt nach ISO/IEC 14651 oder alternativ UCA.

- Folgende Datentypen müssen als Sortierschlüssel nutzbar sein:
  - Unicodestrings
  - Zahlen (Ganzzahlen und Gleitkommazahlen in dezimaler & wiss. Notation, jeweils eingeschränkt auf den Wertebereich von 32 Bit signed Integern bzw. double precision gemäß IEEE 754)

#### 4.4.2.2 Wunschkriterien

- Eine Unterstützung von zusätzlichen Sortierschemata (sowohl kanonische, z.B. für Bibelstellen, als auch benutzerdefinierte) ist wünschenswert. Das gleiche gilt für die Nutzung von Datumsangaben als Sortierschlüsseln.

#### 4.4.2.3 Abgrenzungskriterien

Das Sortiertool ist in TextGrid als Streaming-Komponente vorgesehen. Eine spezialisierte graphische Nutzeroberfläche, etwa zur Definition von Sortierreihenfolgen etc., ist deshalb nicht vorgesehen.

### 4.4.3 Produkteinsatz

Das Sortiertool ist als nicht interaktiver Service konzipiert.

Spezielle Grid-Funktionalitäten werden nicht benötigt. Bei kleinen Datenmengen kann das Sortiertool als Stand-alone-Service genutzt werden (d. h. Ein-/Ausgabe vollständig in SOAP-Nachrichten), im Regelfall wird das Sortiertool aber seine Daten aus der Middleware beziehen und wieder dorthin zurück schreiben (TG-Crud).

Das Sortiertool ist prädestiniert, um in Workflows eingebunden zu werden. In der Regel wird das Ergebnis der Aufbereitung (in der die Datensätze und darin die Sortierfelder ausgezeichnet werden) in die "Ordering"-Einheit eingespeist, die für das Umordnen der Datensätze zuständig ist. In einem abschließenden Nachbereitungsschritt werden die bei der Aufbereitung eingefügten Auszeichnungen wieder entfernt. In komplexen Fällen kann es aber notwendig sein, dass zwischen diese Schritte weitere Transformationen geschaltet werden, z.B. um Sortierschlüssel mit Normdatenbanken abzugleichen. Wenn der oben beschriebene Regelfall als vordefinierter Workflow realisiert ist, dann kann dieser im Workflow-Manager ohne weiteres für die konkrete Anwendung geeignet adaptiert werden.

### 4.4.5 Vom Sortiertool genutzte Daten

Um das Ergebnis einer Sortierung reproduzierbar zu machen und Sortierreihenfolgen nachnutzen zu können, müssen folgende Daten persistent gespeichert werden:

- Workflows, über die Einzelschritte eines Sortierlaufs (Aufbereitung, ggf. Transformationen, Sortierung, ggf. Rücktransformation, Nachbereitung) zusammengefasst werden und die auch in anderen Workflows als fertige Bausteine eingesetzt werden können.
- Konfigurationen (Sortierreihenfolgen, Beginn-/Ende-Marker für Datensätze und Sortierfelder)
- Alle Daten, die nicht Teil der SOAP-Nachricht sind, müssen über TG-Crud aus der Middleware bezogen werden. Das Volumen dieser Daten ist potentiell sehr groß.

Das Sortiertool verarbeitet ausschließlich XML-Daten. Tokenisierter Text (nicht notwendigerweise XML) und Plain Text müssen zuvor mit dem Streaming Editor in ein XML-Format überführt (und ggf. nach der Sortierung zurücktransformiert) werden. Um eine unnötige Komplexität der Schnittstelle des Sortiertools zu vermeiden, muss eine evtl.

notwendige Konvertierung vom Anwender durchgeführt werden, z. B. mit Hilfe des Streaming Editors.

## **4.5 Streaming Editor II**

### **4.5.1 Produktübersicht**

Transformationen von Dateien aufgrund von Regeln, z.B. automatisierte Anreicherung potentiell unstrukturierter Texte mit XML-Strukturen. Die Eingabe muss nicht XML sein, sondern kann ein beliebiger Datenstream sein (etwa OCR-Rohdaten, reiner Text), Ausgabe wird üblicherweise, muss aber nicht XML sein. Das System wird allerdings auf die Ausgabe von XML-Daten optimiert sein. Module werden Streams aus ausgewählten Anwendungsformaten unterstützen.

### **4.5.2 Zielbestimmung**

#### *4.5.2.1 Musskriterien*

Datentransformation anhand eines Perl-Skripts. D.h. Webservice-Wrapper für Perl-Interpreter.

#### *4.5.2.2 Wunschkriterien*

- Einlesen mehrerer Quelldateien , Generierung mehrerer Zieldateien.
- Applikationsserver, der als Webservice-Wrapper für beliebige Skriptsprachen-Interpreter dient (z.B. Python, PHP, Ruby).
- Skriptverarbeitung erfolgt in einer Sandbox (ähnlich Java-Applet), siehe auch Abgrenzungskriterien.

#### *4.5.2.3 Abgrenzungskriterien*

Aus Sicherheitsgründen müssen die Skripte, die zur Ausführung kommen sollen, auf potentiell schädlichen Code überprüft werden. Nicht zugelassen werden:

- Zugriff auf das Dateisystem, d.h. Unterdrückung sämtlicher I/O-Operationen (Eingabedaten erhält das Skript über den Webservice-Wrapper, an diesen wird auch das Ergebnis zurückgeliefert)
- Verbindungsaufbau, Versenden von Mail, d.h. Unterdrückung sämtlicher Protokolle wie http(s), ftp, smtp etc.
- Aktionen, die zu einem Verlassen der Sandbox führen – etwa das Ausführen lokaler Programme
- Um Ressourcenverschwendung durch nicht terminierenden Code zu vermeiden, werden Skripte, die nach einer gewissen Laufzeit nicht beendet worden sind, abgebrochen.

### **4.5.3 Produkteinsatz**

Streaming Tool, als Webservice gekapselt. Keine Grid-Features; diese werden von den in der Middleware bereitgestellten FileServices bzw. *TG-crud* zur Verfügung gestellt.

### **4.5.4 Produktfunktionen**

Regelbasierte Transformation von Textdaten. Regeldefinition über Perl-Skripts. Die Anwendungsgebiete sind vielfältig, hier nur ein paar Beispiele:



- Konvertierung von Nicht-XML-Daten (z.B. OCR-Daten, Altdaten aus diversen Textverarbeitungsprogrammen) nach XML
- Konvertierung von XML-Daten in ein anderes Schema
- Konvertierung von XML-Daten in andere Formate (z.B. HTML, Office-Formate)
- Manipulation/Umstellung von Texteinheiten, Strings, Zeichen; Einfügen von Daten
- Vergabe von IDs, laufenden Nummern u.a.m.
- Verweisgenerierung und Prüfung in Fällen, in denen ein validierender Parser nicht eingesetzt werden kann, weil die Daten entweder nicht valide/validierbar sind oder weil das Verweisschema keinem Standard entspricht.
- Kann über den Workflow-Manager in Kombination mit dem Sortier-Tool (4.4) zur Generierung komplexer Register verwendet werden.
- evtl. als simple Lösung um Daten von Punkt A nach Punkt B zu kopieren.

#### **4.5.5 Produktdaten**

*4.5.5.1 Welche Daten sind aus Benutzersicht (langfristig) zu speichern*

Quelle(n), Ziel(e), Konfiguration.

*4.5.5.2 Wo liegen die Daten, von wo aus greift die Applikation darauf zu.*

Die Daten liegen entweder im Grid oder können über den SOAP-Request mitgegeben werden. Hierfür werden zwei separate Webservice-Instanzen zur Verfügung stehen, wie beim Streaming Editor I.

*4.5.5.3 Welche Datenmengen müssen verarbeitet werden.*

Beliebig große Datenmengen.

*4.5.5.4 Daten-Format*

Eingabe: maschinenlesbare Daten (keine Binärformate).

Ausgabe: beliebig (keine Binärformate).

#### **4.5.6 Produktleistungen**

Siehe sicherheitskritische Aspekte unter 4.5.2.3 Abgrenzungskriterien.

#### **4.5.7 Benutzeroberfläche**

GUI-Plugin (Eclipse).

#### **4.5.8 Nichtfunktionale Anforderungen**

*4.5.8.1 einzuhaltende Gesetze und Normen, Sicherheitsanforderungen, Plattformabhängigkeiten*

Plattformunabhängig, da als Webservice implementiert., Sicherheitsanforderungen siehe unter 4.5.2.3 Abgrenzungskriterien.

*4.5.8.2 Welche Nutzungsdaten sollen (oder dürfen nicht) von der Middleware erhoben werden*

Logging, Durchreichen von Benutzerdaten, siehe Report 3.2 „TextGrid Architektur“.

## 4.5.9 Technische Produktumgebung

### 4.5.9.1 Software: für Server und Client, falls vorhanden

Client: Java ab 1.5, Eclipse 3.3, Apache Axis2 1.3.

Server: Apache 2.2.4 unter mod\_python, Python ab 2.4. (zum Filtern potentiellen Schadcodes), ZSI 2.1a, Perl ab 5.6.

### 4.5.9.2 Hardware: für Server und Client getrennt

Keine speziellen Anforderungen, Hardware sollte aus Performance-Gründen halbwegs aktuell sein.

### 4.5.9.3 Produkt-Schnittstellen

Noch zu definieren, analog zu anderen Streaming Tools.

## 4.6 Kollationierung

### Anforderungen Kollationierer

#### Kurzbeschreibung

Zwei oder mehr TEI-kodierte Dokumente werden verglichen und ihre Unterschiede werden nach TEI kodiert und notiert.

#### Technische Eckdaten:

- Die maximale Anzahl der vergleichbaren Dokumente muss nach pragmatischen Erwägungen festgelegt werden, ebenso wie deren maximaler Umfang.
- Codierung: XML Daten (Behandlung des Auszeichnungsgerüsts siehe „Ignore-Listen“), UTF8 (evtl. auch andere)
- verschiedene Bearbeitungsmodi (Batch/interaktiv) und Ausgabeformate/Visualisierungen.
- Präferenzen und projektübergreifende Parameter können in Konfigurationsdateien abgespeichert werden
- Verlinkung zu evtl. vorhandenen Images

#### Modi:

- Batch: Vollständiger Durchlauf mit zuvor gewählten Parametern
- Interaktiv: Von Abweichung zu Abweichung, mit der Möglichkeit, die Parameter dynamisch zu ändern (z.B. Erstellen von Ignore-listen, setzen von Aufsetzpunkten, Ignore-Bereiche; transparentes Format in Parameterdatei)

Beide Modi sollen interoperabel sein, d.h. Parameter, die im interaktiven Prozess festgelegt werden, sollen in einem transparenten Format in einer eigenen Parameterdatei abgespeichert werden, die in späteren Batchprozessen genutzt und, wo es sinnvoll erscheint, auch für andere Projekte verwendet werden kann.

#### Ausgabe:

- Protokoll, konfigurierbare Ausgabeformate (Reihenfolgen, Trennzeichen etc.) Springen von protokollierter Fehlerstelle in Textzeugen.

Wünschenswert: Hilfestellung zur Herstellung eines kritischen Hypertexts (Auswahl einer Referenzhandschrift, Verlinkung mit den Varianten; eventuell durch eine Nachbereitung mit dem streaming-editor)

- Bildschirm: Möglichkeit zumindest eines paarweisen synoptischen Vergleichs, synoptisches scrollen, Histogrammfunktion (vgl. Juxta)

Neben einem paarweisen Vergleich wäre es auch denkbar, das Protokoll grafisch so aufzubereiten, dass Abweichungen zur Leithandschrift grafisch symbolisiert werden. (Es würden dann mehrere „Fieberkurven“ untereinanderliegen, was helfen würde, um sich einen schnellen Überblick zu verschaffen.)

Evtl. verschiedene Farbcodes/Markierungen für verschiedene Klassen von Abweichungen (konfigurierbar)

### **Steuerung des Kollatierungsprozesses:**

- Behandlung von Umstellungen:

Für einen Kollationierer ist es sehr aufwendig, völlig autonom zu erkennen, dass in einzelnen Quellen ganze Textblöcke ausgelassen oder gar verschoben wurden, speziell wenn die korrespondierenden Blöcke noch zusätzliche mehr oder weniger zahlreiche Varianten enthalten. Deshalb soll der für TextGrid zu entwickelnde Kollationierer sog. Aufsetzpunkte verarbeiten können, also Markierungen, von denen ausgehend der Vergleich aller Fassungen wieder synchronisiert wird. Weil diese Aufsetzpunkte nicht für alle Fassungen in der gleichen Reihenfolge definiert sein müssen, kann der Editor damit dem Kollationierer Verschiebungen größerer Textblöcke manuell mitteilen. Dieser Prozess kann vom Programm zu einem gewissen Grad durch Vorschläge unterstützt werden, die auf einer ggf. auch unscharfen Suche basieren.

- Ignore Listen (= Unterschiede, die ignoriert werden sollen):

Eine elektronische Edition ermöglicht es, grundsätzlich alle in den benutzten Quellen festgestellten Varianten darzustellen; der Platz ist nicht länger notwendig ein limitierender Faktor. Aber neben der reinen Dokumentation fällt einer Edition auch die Aufgabe zu, die Textgenese für den Leser (oder besser: Nutzer) nachvollziehbar zu machen. Wenn sämtliche Varianten gleichgewichtig dargestellt werden, so sinkt die Übersichtlichkeit rapide mit der Zahl der Quellen. Auch bei einer elektronischen Edition muss sich eine Editor also im Allgemeinen entscheiden, welche Variationen er als signifikant betrachtet. Der Editor muss deshalb einem Kollationierer in TextGrid Regeln übergeben können, welche Unterschiede als nachrangig behandelt werden sollen. (Etwa wenn der Editor der Schreibung eines Wortes mit dem Graphem „u“ oder dem Graphem „v“ keine besondere Bedeutung beimisst.) Als Teil dieser Regeln muss der Kollationierer auch Ausnahmelisten unterstützen.

Die Regeln können global definiert werden (Akzentbuchstaben sollen behandelt werden wie Grundbuchstaben; Groß/Kleinschreibung oder Satzzeichen sind irrelevant), für individuelle Zeichen (é soll behandelt werden wie e) oder für stellungsbedingte Varianten (z.B. auslautende Vokale; auslautendes h wie ch behandelt). Entsprechende Listen können während des interaktiven Kollationierungsprozesses erstellt und dynamisch erweitert werden. Auch die Behandlung des Auszeichnungsgerüsts wird so gesteuert: es kann entweder ganz oder in Teilen (z.B. Attributwerte, IDs) ignoriert werden oder selbst den Gegenstand der Kollationierung darstellen (d.h. es wird nach Unterschieden in der Auszeichnung gesucht, die Textdaten werden ggf. ignoriert).

Darüber hinaus soll die Möglichkeit bestehen, manuell Textpassagen zu kennzeichnen, die insgesamt vom Vergleichsprozess ausgenommen werden sollen.

## Gui-Elemente

- Dateiverwaltung:

Auswahl von Textzeugen, evtl. Vorschau auf verlinkte Images, Visualisierung, welche Handschriften gerade miteinander verglichen werden sollen

- Textansicht: beim interaktiven Kollationieren, beim Aufbereiten der Textgrundlage (Aufsatzpunkte), beim Überprüfen des Protokolls wird der Text aufgeschlagen, Sprung in evtl. assoziierte Images möglich.
- Darstellung der Ergebnisse

Bildschirmausgabe: Synoptische Darstellung der betroffenen Texte, Histogramm zeigt die Verteilung der Abweichungen

Protokoll kann ebenfalls eingesehen werden, Sprung in Textzeugen.

## Beziehung zu anderen TextGrid-Tools

- grafischer Linkeditor: beim Arbeiten etwa mit Handschriften kann es sinnvoll sein, an kritischen Stellen der Transkription das Original zu Rate zu ziehen. Dieses kann, so vorhanden, über den grafischen Linkeditor mit dem Text verknüpft werden.
- Tokenizer/Streaming Editor: die Aufbereitung des Grundtextes für den Kollationierungsvorgang (Codierung der Wortgrenzen, Normalisierung) geschieht durch den Tokenizer und den Streaming Editor.

## Noch offen/Ausbaustufen:

Ähnlich wie bei der Lemmatisierung sind in der Literatur verschiedene Vergleichs-Algorithmen beschrieben. In einer der neuesten Arbeiten schlagen Spencer und Howe [SH2004] eine Methode vor, die ohne Festlegung auf einen Basistext auskommt und auf Verfahren aus der Bioinformatik aufbaut. Auch hier lässt sich nicht aus dem Ergebnis der Literaturrecherche ableiten, dass TextGrid einem der vorgeschlagenen Algorithmen den Vorzug geben sollte, weshalb die Entscheidung, welche Kollationierungsmethode als erste implementiert wird, nach pragmatischen Gesichtspunkten getroffen werden kann. Der modulare Aufbau sollte die Integration verschiedener Algorithmen ermöglichen, auch verschiedene sich daraus ergebende Ergebnisdarstellungen (Abweichungen von der Basis vs. Gruppierung von Texten und Abweichungen nach Klassen von Ähnlichkeiten)

## 4.7 Text Publisher Print

Ziel des Print Publishers ist der Satz komplexer XML-Daten, der den Anforderungen an wissenschaftliche Publikationen (z. B. kritische Editionen mit synoptischem Satz und mehreren Apparaten) genügt. Anders als in existierenden Satzprogrammen mit grafischer Benutzerschnittstelle soll der Anwender direkt mit den semantisch annotierten Daten arbeiten, ohne diese erst in das programmspezifische Layoutsystem überführen zu müssen. Vielmehr soll das Programm den Anwender bei der Erstellung von Stylesheets zur regelgeleiteten Formatierung seiner Daten unterstützen und die Ausgabe nach PDF ermöglichen. Eine moderne grafische WYSIWIG-Benutzerschnittstelle wird benötigt, um den Anwender bei der Erstellung von Regeln für die Gestaltung des Textes zu helfen.

Der Print Publisher soll allen textproduzierenden Wissenschaftlern die einfache Gestaltung ihrer Daten und Ausgabe nach PDF ermöglichen, sei es für den Druck oder für die Langzeitarchivierung. Auch für die Akzeptanz digitaler Editionen ist dieses Tool von Bedeutung: So ist zwar die Präsentation einer Edition im Browser als HTML-Daten

akzeptabel, aber deren Ausdruck ist in der Regel satztechnisch nicht befriedigend; mit dem Print Publisher können dagegen Exzerpte in ansprechender Qualität bei Bedarf gesetzt und ausgedruckt werden.

Es hat sich allerdings gezeigt, dass die Anforderungen an ein derartiges Tool – dargestellt z. B. im TextGrid Publishing Report 1.4 den Umfang dessen, was im Rahmen von TextGrid geleistet werden kann, bei weitem sprengt. Die Projektpartner an der FH Worms und an der Universität Trier haben deshalb zusammen mit Saphor einen DFG-Antrag für ein separates Vorhaben vorbereitet, in dessen Rahmen eine derartige Print-Komponente entwickelt werden soll.

## **4.8 Text Publisher Web**

### **Überblick**

Der Webpublisher dient Projekten dazu, für Ihre Daten einen Internetauftritt zu gestalten.

### **Diskussion**

Problematisch ist das Einbinden des Web Publishers in die Architektur von TextGrid. Soll es einen zentralen Server geben, von dem aus die Daten im Internet zugänglich gemacht werden?

### **Funktionalität**

Der Projekttechniker kann den Webauftritt durch die Gestaltung von xslt- und css-Stylesheets bestimmen. Zusätzlich können weitere Präsentationsmodule (etwa zur Visualisierung komplexer Datenstrukturen) eingebunden werden.

Upload der Stylesheet geschieht durch das RCP-Interface.

Da der Webpublisher vor allem aus Stylesheets besteht, sollte eine exemplarische Anwendung – z.B. das Campewörterbuch und/oder die von-Arnim-Edition damit online publiziert werden; dies wird im Rahmen von AP4 geschehen.

Features dieser Publikation:

- a) Alle Daten werden über (definierbare) Navigationsseiten, z.B. Alphabetische Liste beim Wörterbuch, Liste der Handschriften usw. bei der Edition, aufgerufen. (Frage: statische oder dynamische Seiten?) (Browsen im Textbestand)
- b) Integration von Text-und-Bild (Grundlage ist die Ausgabe unseres Grafik-Link-Editors)
- c) Integration des Wörterbuch-Services (eingeschränkt auf ein für den Text sinnvolles Wb.)
- d) Volltextsuche + Strukturdaten + Metadaten
- e) Ausgabe

Es gibt die Möglichkeit, eine Suche über die Projektdaten in den Webauftritt zu integrieren.

## **4.9 OCR**

Tests im Hinblick auf eine prototypische Einbindung eines OCR-Moduls zur Erkennung von Frakturschriften rechtfertigen beim derzeitigen Stand der Technik nicht die Entwicklung eines OCR-Tools für Frakturschriften zumindest des 18. und frühen 19. Jahrhunderts.

Gründe:

1. Die in FineReader XIX eingebaute OCR-Erkennung ist frakturschriftabhängig und daher nicht auf beliebige Drucke übertragbar.
2. Die Nutzung der in FineReader XIX eingebauten Frakturschriftfonts führt bei den üblichen Mischungen von Antiqua- und Frakturschrift zu deutlich höheren Fehlerraten im Vergleich zu eigenständig für ein Projekt trainierten Fonts.
3. Typographische Merkmale werden nicht (z.B. im Fall von Sperrung) oder nur unzureichend (Schriftart- bzw. -schnittwechsel, Einrückungen) erkannt und sind - soweit sie überhaupt erkannt werden - nur auf extremen Umwegen in das Ausgabeformat zu transferieren.

## 5. Literatur

Ludwig, Küster (2008): C. Ludwig and M. W. Küster, „Digital Ecosystems of eHumanities Resources and Services.“ in *DEST 2008* (to appear)

Küster, Ludwig, Aschenbrenner (2007): M. W. Küster, C. Ludwig, and A. Aschenbrenner, “TextGrid as a digital ecosystem,” in *DEST 2007* (E. Chang, ed.), 2007.

Küster, Moore, Ludwig (2007): M. W. Küster, G. Moore, and C. Ludwig, “Semantic Registries,” in *XMLTage 2007 in Berlin* (R. Tolksdorf and J.-C. Freytag, eds.), (Berlin 2007)

Report 1.3, Text Retrieval,

[http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid\\_Report\\_1\\_3.pdf](http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid_Report_1_3.pdf)

Report 1.4, Publishing, [http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid-R1\\_4\\_Publishing.pdf](http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid-R1_4_Publishing.pdf)

Report 2.1, TextGrid-Tools I,

[http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid\\_Report\\_2\\_1.pdf](http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid_Report_2_1.pdf)

Report 3.2, TextGrid-Architektur,

[http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid\\_Report\\_3\\_2.pdf](http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid_Report_3_2.pdf)

Report 3.3, UserSpezifikation aller von der TextGrid-Middleware zu bedienenden Grid-Schnittstellen (Version 2, noch nicht erschienen)

Report 3.4, TextGrid - Middleware-Tests unter Berücksichtigung der Werkzeuge (AP2) und Musterapplikationen (AP4)

Richardson, Ruby (2007): L. Richardson and S. Ruby, *RESTful web services*. Farnham: O'Reilly, 2007

Schmid, Helmut (2005): A Programming Language for Finite State Transducers In: *Proceedings of the 5th International Workshop on Finite State Methods in Natural Language Processing (FSMNLP 2005)*, Helsinki, Finland.

Schmid, Helmut und Arne Fitschen und Ulrich Heid (2004): SMOR: A German Computational Morphology Covering Derivation, Composition, and Inflection In: *Proceedings of the IVth International Conference on Language Resources and Evaluation (LREC 2004)*, p. 1263-1266, Lisbon, Portugal.

Spencer, Matthew und Howe, Christopher (2004): Collating Texts Using Progressive Multiple Alignment. In: *Computers and the Humanities* 38(3), S. 253–270, Dordrecht: Springer. <http://dx.doi.org/10.1007/s10579-004-8682-1>