TextGrid-Tools I

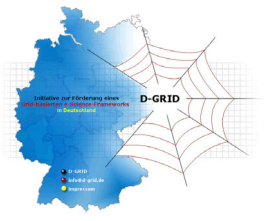
Version 5.2.2007

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

Amir Eskandari, SUB

Peter Gietz, DAASI

Martin Haase, DAASI

Fotis Jannidis, TU Darmstadt

Frank Knoll, DAASI

Wolfgang Pempe, Saphor

Christian Simon, IDS Mannheim

Markus Sosto, Saphor

Thorsten Vitt, TU Darmstadt

Andrea Zielinski, IDS Mannheim

Inhalt

Vorbemerkungen

Das vorliegende Dokument soll einen Überblick über die mit Arbeitspaket 2 (AP2) verbundenen Aktivitäten in den vergangenen 12 Monaten vermitteln. Hierzu werden im folgenden Abschnitt 1 die Stellung und die Abhängigkeiten von AP2 innerhalb von TextGrid skizziert – organisatorisch und technisch. In Abschnitt 2 werden die im Berichtzeitraum erfolgten Arbeiten beschrieben, sowohl unter organisatorischen (Arbeitsplan) als auch technischen (Prototyp) Aspekten. In Abschnitt 3 folgt die technische Dokumentation der im Berichtzeitraum begonnenen Module.

1 Überblick Arbeitspaket 2 (AP2)

1.1 AP2 – Funktion in TextGrid

Die Aufgabe von Arbeitspaket 2 (AP 2) ist die Entwicklung Community-spezifischer Werkzeuge (Tools) für die Erstellung, Bearbeitung, Annotation und Analyse von XML-codierten Textdaten. Eine Liste aller in AP 2 zu entwickelnden Tools findet sich im beiliegenden Arbeitsplan (Anhang 1).

Zwischen den Arbeitspaketen in TextGrid besteht eine Vielzahl von Abhängigkeiten. So müssen in AP2 einerseits die in AP1 erarbeiteten Evaluationsergebnisse berücksichtigt, andererseits die in Zusammenarbeit mit AP3 definierten Schnittstellen zur Middleware implementiert werden. Außerdem ist darauf zu achten, dass die im weiteren Projektverlauf hinzukommenden Arbeitspakete 4 (Muster-Applikation) und 5 (Semantic TextGrid) problemlos auf die bis dahin entwickelten Lösungen aufsetzen können, bzw. deren Vertreter sich bereits jetzt (d.h. im Berichtzeitraum) an der grundlegenden Diskussion beteiligen.

Um die für eine erfolgreiche Projektarbeit notwendige Arbeitspaket-übergreifende Koordination und Kooperation in allen technischen Fragen sicherzustellen, wurde im April 2006 die AG Architektur ins Leben gerufen. Zu ihren Mitgliedern gehören die technisch Verantwortlichen der Arbeitpakete 1 bis 5.

Als Ergebnisse der in der AG Architektur geführten Diskussionen wurden sowohl der Arbeitsplan für AP 2 umgestaltet (s. Abschnitt 2) als auch einige Tools gegenüber dem Antrag neu definiert (im Arbeitsplan = Anhang 1 grün hinterlegt): So wurden bspw. *Query Interface* und *Text Retrieval* zum *Recherchetool* zusammengefasst, der *Link-Editor* hingegen in mehrere Tools aufgespalten.

1.2 AP2 in der Gesamtarchitektur – die TextGrid-Workbench

Die Tools werden in der *TextGrid-Workbench* zusammengefasst. Horizontal fungiert sie als integrierendes Element für alle Tools, vertikal umfasst sie die obersten drei Schichten der TextGrid-Architektur (s. Report 3.2 „TextGrid Architektur“):

- Benutzerumgebung(en)
- Service Layer
- Middleware (Schnittstellen zum Datei- und Rechtemanagement)

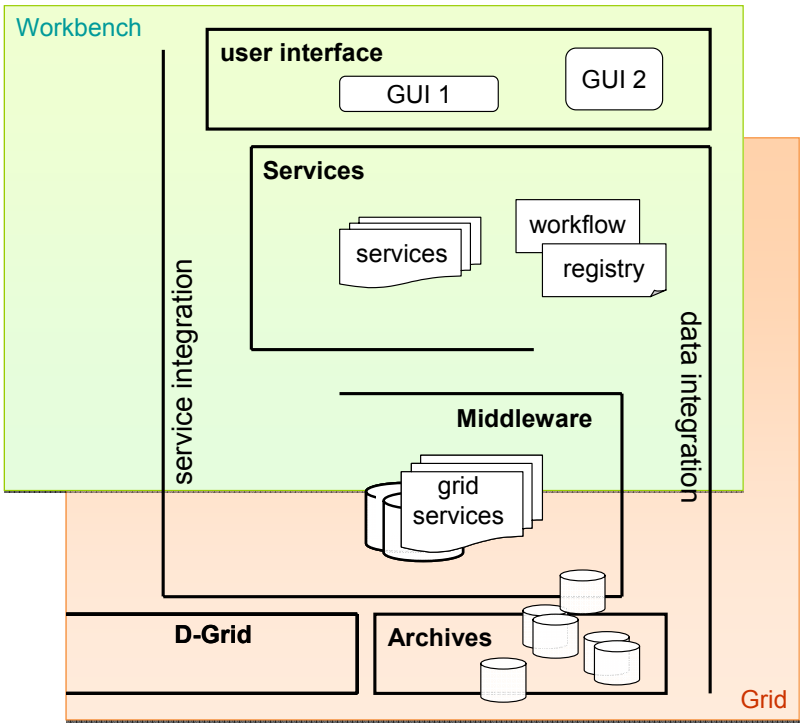
Entsprechend besteht jedes Tool aus maximal drei schichtspezifischen Komponenten:

1. Einer grafische Benutzerschnittstelle (GUI) zur Konfiguration und ggf. interaktiven Steuerung des Tools, die jeweils als Plugin in eine Eclipse-basierte *Rich Client Platform* (RCP) eingebettet ist (s.u., Abschnitt 3.2)
2. Der eigentlichen Anwendung (Service), die zumindest bei den Streaming Tools (s.u.) als WebService gekapselt ist

3. Einer Komponente, in der die Schnittstellen zur Middleware gekapselt sind (ebenfalls als WebServices implementiert)

Die in diesem Report behandelten Tools fallen in zwei Kategorien:

1. *Streaming Tools* besitzen eine Komponente, die als Service im Batchbetrieb läuft. Die Konfiguration erfolgt über eine GUI-Komponente in der RCP.
2. *Interaktive Tools* besitzen keine Batchkomponente, sondern werden in einem hoch interaktiven Prozess vom Benutzer bedient. Sie existieren nur in der RCP.



Die TextGrid-Workbench (grün) und ihre Beziehung zur Gesamtarchitektur
(Quelle: Report 3.2 „TextGrid Architektur“)

2 Arbeiten im Berichtzeitraum

2.1 Änderungen im Arbeitsplan

Die ursprüngliche Einteilung im Antrag sieht zunächst eine sequentielle Entwicklung der Tools, erst in M2.3 eine „Workbench“ und schließlich (M2.5 zum Projektende) die „Integration aller Tools in [die] Workbench“ vor.

Aufgrund der in der AG Architektur erzielten Ergebnisse wurde beschlossen, integrierende Arbeiten bereits am Anfang durchzuführen, gemeinsame Funktionalitäten der Tools gemeinsam zu entwickeln und einheitliche Standards – insbesondere Schnittstellen – für die Tools zu entwickeln, anstatt die Werkzeuge erst in einer späten Projektphase zusammenzuführen. Deshalb lag der Fokus auf der Entwicklung eines Prototypen, der die Schichten der Workbench-Architektur (s. Abschnitt 1.2) miteinander verbindet: Benutzerumgebung – Service Layer – Middleware. Nachdem der „Durchstich“ geglückt ist und anhand des Prototyps Musterlösungen für die Integration der schichtspezifischen Komponenten gefunden wurden, kann nun die Workbench horizontal um weitere Tools erweitert werden.

Entsprechend wurden die Milestones und Reports neu definiert, siehe hierzu den Arbeitsplan in Anhang 1. Für jedes Tool wurde eine verantwortliche Institution (Spalte E) bestimmt, die das Projektmanagement und die technische Dokumentation übernimmt. Die fachliche

Betreuung (Spalte N, nach Möglichkeit Repräsentanten der Zielgruppen) sorgt für die Formulierung der Anforderungen, stellt Testmaterialien zur Verfügung, definiert Testszenarien und übernimmt die Durchführung der Tests.

2.2 Prototyp

Wie oben erwähnt sollten anhand des Prototypen Musterimplementierungen für alle Workbench-Schichten geschaffen werden, an der sich die weitere, vor allem horizontale Entwicklung orientieren kann.

Benutzerumgebungen

- *Rich Client Platform* (RCP, Abschnitt 3.2): Eclipse-basierte Plugin-Struktur (Views) zur einheitlichen GUI-Modellierung für die einzelnen Tools. Bei (Streaming-)Tools, die als WebServices implementiert werden, dienen die GUI-Elemente der Steuerung halbautomatisch generierter WebService-Clients.
- Workflow-Editor (Abschnitt 3.5): Eigene GUI, wird demnächst in RCP integriert.

Service Layer

- Tokenizer (Abschnitt 3.3): Wurde ausgewählt, weil relativ schnell zu entwickeln, so dass genug Zeit für die Schnittstellenentwicklung (RCP und FileServices) bleibt.
- Lemmatisierer (Abschnitt 3.4): Ebenfalls schnell zu entwickeln, Applikationslogik liegt (z.Zt.) in Automaten, die sich sukzessive erweitern lassen, ohne am eigentlichen Programm etwas ändern zu müssen (in Zukunft ggf. Maßnahmen zur Performance-Verbessserung).
- Lexikon-Suche (AP5): Erster Schritt, die in Trier vorliegenden (Tcl-)Anwendungen als WebServices zu kapseln.
- Workflow-Enactor (Abschnitt 3.5): Zunächst Orchestrierung Tokenizer-Lemmatisierer.

Middleware

Dateimanagement, Zugriff auf Daten im Grid: FileServices (Anhang 2).

2.3 Ergebnisse der Prototyp-Entwicklung

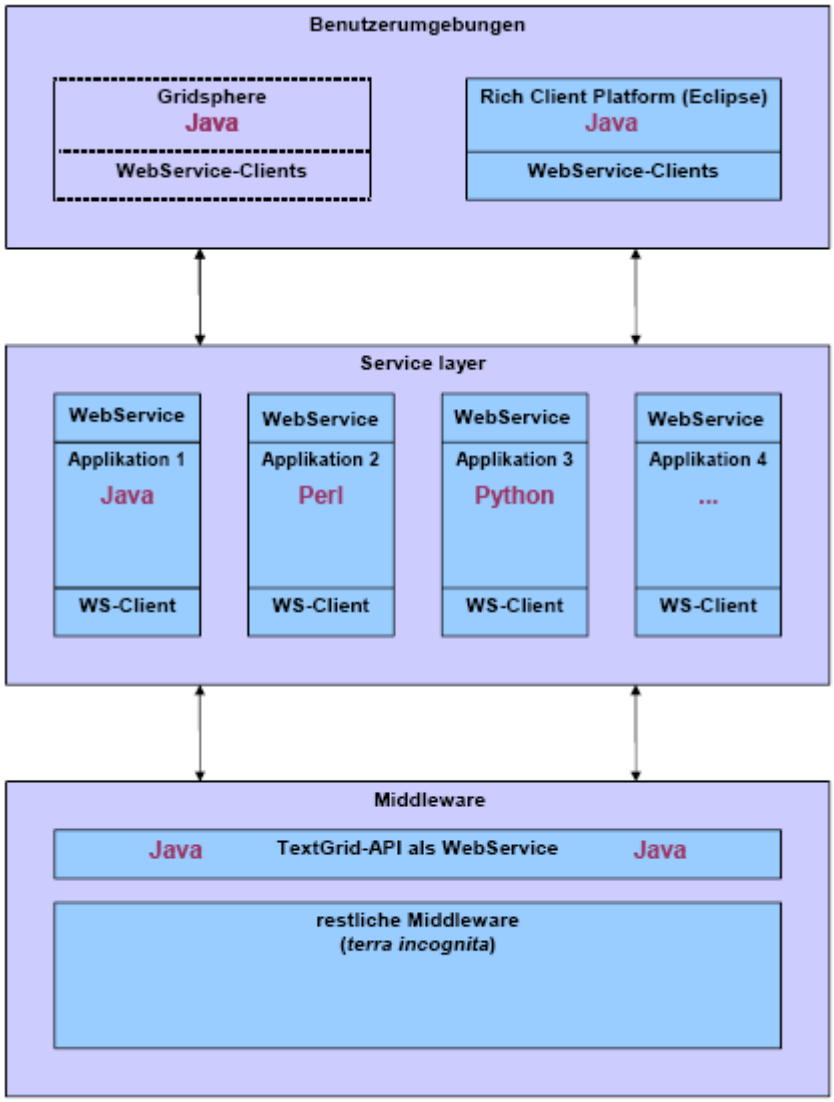
2.3.1 Horizontale Erweiterung der Workbench, Erkenntnisse und Empfehlungen

Der Service Layer ist sprach- und plattformunabhängig, da WebService-basiert. Insofern ist die Entwicklung von TextGrid-Tools nicht auf eine bestimmte Programmiersprache fixiert. Vielmehr besteht die Absicht, bei der Erweiterung der Workbench Tools neben Java und Python in weiteren Programmiersprachen zu implementieren, um mit den hierbei gesammelten und dokumentierten Erfahrungen potentiellen Kooperationspartnern die Integration eigener Anwendungen in TextGrid zu erleichtern. Mittelfristiges Ziel ist die Erstellung eines Intergrationsleitfadens, der die verwendeten

- Schnittstellen (GUI und Middleware)
- Programmiersprachen
- Bibliotheken
- Standards (z.B. WSDL)

dokumentiert – und ggf. Workarounds und Bugfixes bei in Entwicklung befindlichen Softwarepaketen (z.B. ZSI [Python]).

Im folgenden Abschnitt werden die Integrationsschritte kurz beschrieben.



TextGrid – sprachunabhängiger Service Layer

2.3.2 Schritte zur Integration einer Anwendung als WebService in den TextGrid Service Layer (Streaming Tools)

1. *Grundfunktionalität lokal testen*
   Nur bei Neuentwicklungen.

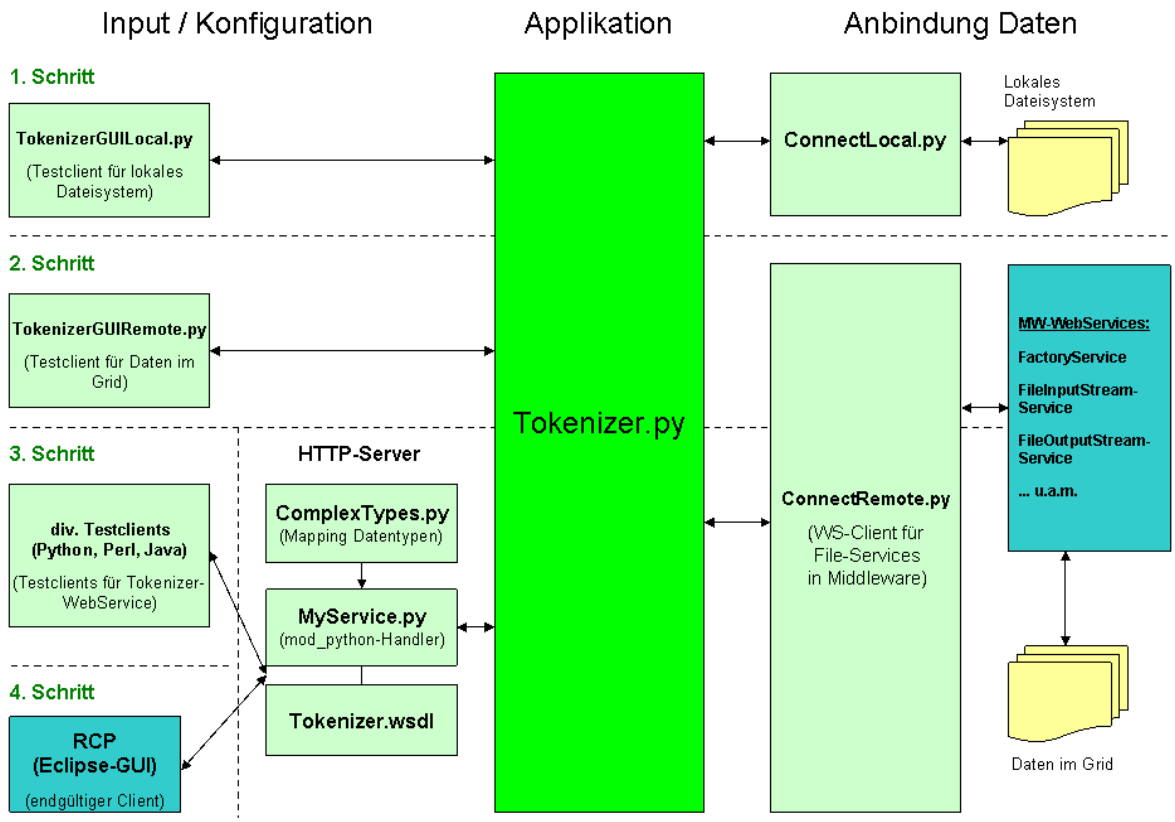
2. *Schnittstelle zum Dateisystem durch WebService-Client zu den FileServices ersetzen*
   Setzt eine modulare Architektur voraus. Hier liegt u.U. die einzige echte Hürde bei der Integration bereits bestehender Anwendungen. Für Python (2.4) existiert bereits eine entsprechende API-Implementierung (ConnectRemote.py), siehe Dokumentation Tokenizer (Abschnitt 3.3) und Lemmatisierer (Abschnitt 3.4) – desgl. für Java (Workflow-Enactor, Abschnitt 3.5). Im Zuge der horizontalen Erweiterung der Workbench werden höchstwahrscheinlich noch weitere Sprachen abgedeckt (Perl, PHP, Tcl [AP5-Tools]). Die entsprechenden Implementierungen können dann potentiellen Kooperationspartnern zur Verfügung gestellt werden.

3a. Anwendung *muss auf einem HTTP-Server laufen, ggf. Handler vorschalten*
   Lemmatisierer und Tokenizer laufen auf Apache 2.0.x unter mod_python (Python-Handler: MyService.py, siehe Abbildung unten), in Java implementierte WebServices

können alternativ unter Apache Tomcat laufen. In Perl oder PHP geschriebene Services machen keine Probleme, da beide Sprachen zur Web-Programmierung verwendet werden.

3b. *Publikation über WSDL-File*
Binding-Typ (XML-)RPC (`<soap:binding style="rpc">`), um sicherzugehen, dass alle verwendeten (SOAP-) Bibliotheken damit umgehen können. Beim Namespacing kann man sich an den bestehenden WSDL-Files (s. Dokumentationen in Abschnitt 3) orientieren – hier kann es bei der Client-Anbindung zu Problemen kommen.
Für die Erstellung von Test-Clients haben sich SOAP::Lite (Perl)[1] und SOAPpy (Python)[2] bewährt, mit beiden Bibliotheken lassen sich mit wenigen Zeilen Code entsprechende Anwendungen erstellen. Als Java-Client wurde Apache Axis[3] sowie für erste Tests der *Web Service Explorer* aus dem *Eclipse Web Tools Project* eingesetzt[4].

4. *Anbindung WebService an Eclipse-RCP oder andere GUI.*
Im Fall von Streaming Tools ist im Rahmen der RCP (Abschnitt 3.2) eine halbautomatische Generierung von Eingabemasken möglich.
Ansonsten sind prinzipiell auch andere GUIs denkbar, die auf einen oder mehrere WebService-Clients aufsetzen, z.B. GridSphere-Portlets.



Schrittweise Integration einer Anwendung als WebService am Beispiel des Tokenizers.

[1] Getestete Versionen 0.55 und 0.69. Bei der Parameterübergabe ist zu beachten, dass in Version 0.55 Spitzklammern (z.B. vom Tokenizer zu ergänzende Tags) als < bzw. > zu maskieren sind.
[2] SOAPpy 0.11.6 und 0.12 (http://sourceforge.net/project/showfiles.php?group_id=26590&package_id=18246).
[3] Apache Axis Version 1.4, http://ws.apache.org/axis
[4] http://www.eclipse.org/webtools/.

3 Im Berichtzeitraum begonnene Module

3.1 Überblick

Tool	verantwortlicher Projektpartner	Status
*Rich Client-Platform**	SUB Göttingen	beta
Tokenizer*	Saphor GmbH	beta
Lemmatisierung*	IDS Mannheim	beta
Workflow-Editor*	DAASI international GmbH	beta
XML-Editor	TU Darmstadt	alpha

* Prototyp

3.2 Rich-Client-Platform (RCP)

3.2.1 Produktübersicht

Die RCP dient als Tool-übergreifende Benutzerumgebung für die in TextGrid zur Verfügung stehenden Dienste vorwiegend für Editoren und Bearbeiter[5].

3.2.2 Zielbestimmung

Ziel der RCP ist es, eine integrierte Benutzungsumgebung anzubieten, in der Editoren und Bearbeiter die in TextGrid entwickelten Tools möglichst intuitiv bedienen und steuern können.

3.2.2.1 Musskriterien

- Intuitive Bedienung und Konfiguration
- Plattform-unabhängigkeit
- Online/Offline-Arbeiten möglich mit Synchronisierung der Daten (Import/Export in/aus dem Grid)
- Einhaltung von Konventionen und Schnittstellen der zugrundeliegenden Plattform (Eclipse), um eine Integration mit von dritter Seite für diese Plattform entwickelten Komponenten / Plugins zu erleichtern

3.2.2.2 Wunschkriterien

Fortschrittsanzeige bei der Verarbeitung längerer Jobs.

Automatische Generierung eines provisorischen maskenbasierten User-Interfaces für (externe) Web-Services, für die keine eigene GUI-Komponente vorliegt.

3.2.2.3 Abgrenzungskriterien

Während die GUI-Komponenten der einzelnen Tools (z.B. Tokenizer, XML-Editor oder Datei- und Rechtemanagement) im Rahmen dieser Tools entwickelt werden, entwirft das TextGrid-RCP-Projekt gemeinsame Komponenten sowie Leitlinien, um eine enge Integration und einheitliche Bedienbarkeit der GUI-Komponenten sicherzustellen.

[5] Zu den Rollen siehe *TextGrid Szenarien* (demnächst unter http://www.textgrid.de).

3.2.3 Produkteinsatz

Interaktiv, Frontend für Service- und Gridkomponenten, implementiert WebService-Clients.

3.2.4 Produktfunktionen

3.2.4.1 Benutzeroberflächen für TextGrid-Tools

Im Rahmen der RCP ist sicherzustellen, dass die im Rahmen der individuellen Tools entwickelten Benutzeroberflächen gut miteinander integriert sind. Dazu werden Leitlinien über die von den Tools zu implementierenden Schnittstellen entwickelt und die Anforderungen der individuellen Tools koordiniert.

Darüberhinaus wird hier GUI-Funktionalität entwickelt, die mehreren Tools gemeinsam ist, soweit sie nicht bereits von der Eclipse-Plattform angeboten wird.

Schließlich werden integrierende Mechanismen implementiert, um die GUI-Komponenten der diversen Tools zu einer Clientapplikation zu vereinen; dies umfasst die Generierung von installationsfertigen Binärpaketen der Gesamt-RCP für verschiedene Zielplattformen und die Verwaltung von Updates mit dem Eclipse-eigenen Mechanismus.

Besonderheiten beim Datenmanagement

Diverse Tools (unter anderem der XML-Editor) müssen auch ohne bestehende Anbindung ans Internet und somit ans Grid arbeiten können, etwa zur Arbeit in Archivräumlichkeiten. Die RCP sollte eine Schicht anbieten, um die Arbeit mit Online- bzw. Offline-Daten für die anderen Tools wie für den Benutzer weitestgehend transparent zu realisieren.

Dieser Teil der Funktionalität wird in enger Abstimmung mit dem Tool zum *Datei- und Rechtemanagement* implementiert; insbesondere Nutzerschnittstellen zum Verwalten der lokalen und entfernten Dateien werden in letzterem umgesetzt.

3.2.5 Produktdaten

Nutzer-/Authentifizierungsdaten, persönliche Anpassung der GUI seitens des Nutzers.

3.2.6 Produktleistungen

Im Gegensatz zu den Services kein Multi-User Betrieb.

3.2.7 Benutzeroberfläche

Siehe 3.2.4.1.

3.2.8 Nichtfunktionale Anforderungen

Nutzer-/Authentifizierungsdaten müssen an die Middleware weitergereicht werden.

3.2.9 Technische Produktumgebung

3.2.9.1 Software

Eclipse SDK 3.2.1 oder neuere Eclipse-Versionen

3.2.9.2 Hardware

Keine speziellen Anforderungen, Hardware sollte aus Gründen der Performance und der optimalen Darstellung (SWT) halbwegs aktuell sein.

3.2.9.3 Produkt-Schnittstellen

Für die Einbindung der WebService-Clients: SOAP, WSDL

3.2.10. Implementierungsentscheidung

Zur Umsetzung der oben genannten Anforderungen wurde die *Eclipse Rich Client Plattform* gewählt.

Eclipse[6] wurde ursprünglich als Plattform für integrierte Entwicklungsumgebungen (IDE) entworfen. Dabei ist es vollständig modular aufgebaut, die gesamte Funktionalität ist auf zahlreiche *Plugins* verteilt, die über wohldefinierte Schnittstellen miteinander kommunizieren können. Um die Plattform kann auch jenseits der Programmieranwendung zum Bau von eigenständigen Applikation und modular aufgebauten Arbeitsplatzumgebungen verwenden zu können, wurde die Eclipse Rich Client Plattform (RCP[7]) entworfen: Mit ihr können Entwickler einen Satz von plattformunabhängig entwickelten Plugins zu einer Anwendung integrieren und mit plattformspezifischen, vom Eclipse-Projekt zur Verfügung gestellten Startern und Anpassungen der Widgetbibliothek (üblichwerweise SWT[8]) zu einer integrierten Anwendung schnüren. Darüberhinaus können in die RCP eine Reihe von existierenden Eclipse-Plugins etwa zur Options- oder Updateverwaltung eingebunden werden.

3.3 Tokenizer

3.3.1 Produktübersicht Tokenizer

Zerlegt einen Text in eine Folge logischer Einheiten (Tokens), hier: Wörter. Diese werden durch Anfang- und Endemarkierungen gekennzeichnet.

3.3.2 Zielbestimmung

Die Tokenisierung der Texte dient der einfacheren Weiterverarbeitung mit Folgetools, z.B. Lemmatisierer.

3.3.2.1 Musskriterien

Bestimmung und Auszeichnung von Wörtern.

Die Verarbeitung 'komplexer' Tokens (Abkürzungen, Eigennamen, etc.) wird ebenfalls unterstützt.

3.3.2.2 Wunschkriterien

Eine weitere evtl. zu implementierende Funktion wäre das Taggen von Satzzeichen, z.B. mit TEI-Element `<c>`, Kategorisierung anhand STTS (Stuttgart-Tübingen Tagset: http://www.sfs.uni-tuebingen.de/Elwis/stts/stts.html) oder – differenzierter – gemäß den Zürcher Anpassungen (http://www.ifi.unizh.ch/CL/tagger/UIS-STTS-Diffs.html).

Erweitertes Preprocessing (s.u. 3.3.4.2), das über reines Patternmatching hinausgeht.

Einzelne Unicode-Bereiche (z.B. über Drop-Down-Liste) auswählbar.

Preview bzw. Anzeige der tokenisierten Daten in einem separaten Fenster (GUI).

3.3.2.3 Abgrenzungskriterien

Zu diskutieren bleibt, ob der Tokenizer auch zur Bestimmung und Auszeichnung größerer Texteinheiten, z.B. Sätzen, eingesetzt werden soll. Auch hierfür existiert eine Empfehlung des Unicode Konsortiums (s.u.), allerdings dürfte der Anteil an erforderlicher manueller

[6] Eclipse Project: http://www.eclipse.org.
[7] http://www.eclipse.org/home/categories/rcp.php
[8] *Standard Widget Toolkit* (http://www.eclipse.org/swt/).

Nachbesserung ungleich höher liegen als auf Wortebene, zumal dieser Anwendungsfall eher selten auftreten dürfte. Eher Anwendungsfall für die OCR-Nachbereitung.

3.3.3 Produkteinsatz

Streaming Tool, als WebService gekapselt. Keine Grid-Features; diese werden von den in der Middleware bereitgestellten FileServices zur Verfügung gestellt.

3.3.4 Produktfunktionen

3.3.4.1 Auszeichnung von Wörtern innerhalb von Textdaten

Dient der Auszeichnung von Wörtern innerhalb neu erfasster (manuell oder OCR – Plain Text) oder noch nicht bis auf Wortebene hinab ausgezeichneter (XML) Texte. Auf das so generierte Markup können weitere Tools wie der Lemmatisierer (Abschnitt 3.4) aufsetzen.

Markiert Wörter zunächst anhand der vom Unicode Konsortium vorgeschlagenen Regeln:
http://www.unicode.org/reports/tr29/tr29-9.html
Diese Regeln basieren auf der Zuordnung einzelner Unicode-Zeichen zu Zeichengruppen:
http://www.unicode.org/Public/UNIDATA/auxiliary/WordBreakProperty.txt
Gewünschte Anfang- und Endmarkierung der Wörter können in der Konfiguration angegeben werden. Voreinstellung ist <w> … </w>, vgl. http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-w.html

3.3.4.2 Optionales Preprocessing (in Arbeit)

Preprocessing, bei dem "Wörter", Namen, Patterns, in einer Konfigurationsdatei (XML) zu definieren sind, die in einem ersten Arbeitsschritt eingelesen und abgearbeitet wird – noch bevor die eigentliche Applikationslogik zum Einsatz kommt. Dieses Verfahren bietet sich z.B. für die Auszeichnung von (häufig textspezifischen) Zeichengruppen an, die nicht von o.g. Algorithmus als Wörter erfasst werden können, wie Eigennamen, Mehrwortlexeme (bedingt), Abkürzungen, Datumsangaben (als Pattern), Bindestrichwörter. Über die Konfigurationsdatei ist auch eine Zuordnung von Token-/Pattern-Listen zu Typen möglich, bspw. nach dem STTS (s.o., Abschnitt 3.3.2.2) möglich, z.B. `<w type="stts:NE">`Rufus T. Firefly`</w>`, oder mit einer noch detaillierteren Auszeichnung der Eigennamen gemäß ihres Typs (Person, Ort, Organisation).

NB: Das Preprocessing erfolgt ausschließlich über Patternmatching und kann deshalb kein "echtes" POS-Tagging[9] durchführen. Darum können Mehrwortlexeme, Klitika (Clitics) und Kardinal-/Ordinalzahlen nur ansatzweise als solche erfasst werden.

In der Konfigurationsdatei können außerdem im Quelltext vorkommende Elemente oder Bereiche angegeben werden, deren Inhalt nicht vom Tokenizer verarbeitet werden soll.

3.3.5 Produktdaten

3.3.5.1 Welche Daten sind aus Benutzersicht (langfristig) zu speichern

Quelldatei, Zieldatei, Konfigurationsdatei, Log-Datei.

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

Die Daten liegen im Grid, wo sie über die von der Middleware bereitgestellten FileServices (Anhang 2) angesprochen werden können. Lokale Daten werden über ein RCP-Plugin ins Grid eingecheckt.

---

[9] POS = *part-of-speech*.

3.3.5.3 Welche Datenmengen müssen verarbeitet werden

Beliebig große Datenmengen, die auch in Teilen aus der Middleware geladen werden können.

3.3.5.4 Daten-Format

Verarbeitet Plain Text oder bereits mit XML-Markup versehene Textdaten. Zeichensatz: UTF-8 ohne BOM (Byte Order Mark).

3.3.6 Produktleistungen

Die Daten dürfen abgesehen vom ausgewählten Markup nicht verändert werden (Zeichenformat, Zeilenfall, Linebreaks, etc.).

3.3.7 Benutzeroberfläche

Maske in der RCP für Parameter-Übergabe. Für XML-Dateien Preview über XML-Editor (Abschnitt 3.6) – noch zu realisieren.

Zu einem späteren Zeitpunkt evtl. auch GridSphere-Portlet.

3.3.8 Nichtfunktionale Anforderungen

3.3.8.1 einzuhaltende Gesetze und Normen, Sicherheitsanforderungen, Plattformabhängigkeiten

Plattformunabhängig, da als WebService implementiert.

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

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

3.3.9 Technische Produktumgebung

3.3.9.1 Software: für Server und Client, falls vorhanden

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)
  - fpconst 0.7.2 (http://cheeseshop.python.org/pypi/fpconst/0.7.2)

Client:

- Eingabemaske für Parameter (s.a. Abschnitt 3.2 zur RCP)

3.3.9.2 Hardware: für Server und Client getrennt

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

3.3.9.3 Produkt-Schnittstellen

Service ist publiziert unter http://ingrid.daasi.de/Tokenizer.wsdl

Parameter:

`infile`	Quelldatei
`outfile`	Zieldatei
`tzfile`	Konfigurationsdatei für Preprocessing
[`strvor`	vor einem Token (Wort) zu ergänzender String (Voreinstellung: `<w>`)]
[`strnach`	nach einem Token (Wort) zu ergänzender String (Voreinstellung: `</w>`)]
tokenelement	XML-Element, in das ein als Token identifizierter String überführt wird (neu, ersetzt `strvor` und `strnach`)
`mod`	Modus (txt/xml)
`cp`	Codepage (veraltet, aus der Middleware kommt nur UTF-8)

Konfigurationsfile für Preprocessing; FileServices Middleware (s. Anhang 2).

3.4 Morphologische Analyse, Lemmatisierer

3.4.1 Produktübersicht Lemmatisierer und Wortbildungsanalyse

Flexionsanalyse:

Analysiert einzelne Wortformen (Token) und liefert als Ausgabe

a) das zugehörige Lemma, d.h. das Token wird auf seine grammatische Grundform zurückgeführt,

b) die zugehörige Wortart (PartOfSpeech) und

c) weitere morphologische Merkmale (Numerus, Genus etc.).

Die Wortform *Hauses* beispielsweise lässt sich dem Lexem *Haus* mit der Wortart Substantiv zuordnen, der Kasus ist Genitiv und der Numerus ist Singular.

Wortbildungsanalyse:

Komplexe Wörter werden segmentiert, d.h. die Wortform wird in Morpheme zerlegt. Für die Wortform *Alpenwanderung* kann beispielsweise die Segmentierung *Alpen-wander-ung* ausgegeben werden. Als Wortbildungsprozesse gelten Derivation und Komposition.

3.4.2 Zielbestimmung

Die **Flexions- und Wortbildungsanalyse** sind für folgende Funktionen relevant:

- Basisfunktionalität für den anschließenden Wörterbuchzugriff (Lexical Lookup) aus einem laufenden Text heraus. Automatische Anzeige ein oder mehrerer Lexikoneinträge bei Markierung des entsprechenden Wortes im Text.
- Analyse von Texten; Annotierung der flektierten Wortformen im Text mit Angabe des Lemmas, der Wortart, der Zeitstufe und der morpho-syntaktischen Angaben.
- Basisfunktion für die Retrievalengine: Da sowohl der Suchterm als auch die in den Texten vorkommenden Terme in unzähligen Varianten vorkommen können (flektierte Form, alte/neue Rechtschreibung, historische oder regionale Varietäten etc.), welche der Benutzer nicht explizit eingeben möchte, ist eine Normalisierung notwendig. Somit wird die Routine sowohl bei der Index-Erstellung als auch zur Runtime bei der Analyse der Queryterme angesprochen. Der Benutzer sucht nach „Häuser“ und findet auch Textstellen, die *Haus*, *Häusern* oder *Hauses* enthalten sowie z.B. *Haus* als Wortbestandteil in *Wohnhaus* oder *Haustür*.

3.4.2.1 Musskriterien

- Analyse geschriebenen Hochdeutsches und älterer Sprachstufen für die Zeitepochen althochdeutsch, mittelhochdeutsch, frühneuhochdeutsch und neuhochdeutsch einschließlich regionaler Varietäten
- Differenzierung nach alter und neuer Rechtschreibung
- Angabe von morpho-syntaktischen Merkmalen zu den Wortformen
- vollständige Flexionsmorphologie
- Schneller Lexikonzugriff für idiomatisierte oder lexikalisierte Formen (z.B. Augenschein, Brombeere und Bahnhof)
- Grundwortschatz des Neuhochdeutschen ist abgedeckt
- Für das Althochdeutsche, Mittelhochdeutsche und Frühneuhochdeutsche werden Lernverfahren zur Erkennung graphischer und morphologischer Varianten entwickelt
- umfassende Derivations- und Kompositionsmorphologie, so dass auch neu-gebildete, nicht-lexikalisierte Wortformen analysiert werden können
- Möglichkeit zur Einbindung eigener projektspezifischer Wörterbücher
- Auswahl einer eindeutigen Zitierform (normalisierte Form) zu unterschiedlichen Varianten

3.4.2.2 Wunschkriterien

- Erkennung von Schreibvarianten
  - Wortbindestrichergänzung: Unterschiedliche Schreibweisen (*Zwei-Drittel-Mehrheit* vs. *Zweidrittelmehrheit*)
  - orthografischen Varianten: g*raphisch* vs. *grafisch*
  - etymologische Varianten (Hyperlemma): (*frouwa* im Althoch-deutschen, *vrouwe* im Mittelhochdeutschen, *Frau* im Neuhochdeutschen
- Ausgabe der korrekten Klammerstruktur bei komplexen Wortbildungen. Sobald eine Wortbildung mehr als zwei Bestandteile aufweist, sind mehrere Baumdarstellungen möglich
- Tokenfrequenzen (bezogen auf bestimmte Korpora) für das Ranking im IR
- Generierung einer Stoppwortliste für das Ranking im IR (Suche nach Dokumenten)
- fremdsprachliche Ausdrücke; Erkennung der Herkunft des Lexems (nativ | Sanskrit | Griechisch | Französisch)
- Tools zur automatischen Sprachenerkennung (Language Identification) für Texte oder Textausschnitte, deren Sprache noch nicht ausgezeichnet ist

3.4.2.3 Abgrenzungskriterien

Nicht abgedeckte Funktionen:

- Tippfehler, Rechtschreibfehler
- Tokenisierungsfehler (z.B. bei Zeilenumbrüchen getrennte Wörter, die nicht wieder zusammengefügt wurden)
- Erkennung von Eigennamen; d.h. welche nicht explizit als solche im Lexikon kodiert sind (z.B. *Annaberg*)

- Erkennung des semantischen Typs oder Sachgebietes einer lexikalischen Einheit
- Erkennung von Homonymen: *Fliege* (Insekt) vs. *Fliege* (Bekleidung)
- Erkennung von semantischen Verwandtschaften, z.B. Ober- und Unterbegriffe (z.B. *Haustür* als Unterbegriff zu *Tür*)
- Abbildung eines Graphems auf seine lautliche Gestalt (phonetische Tanskription), z.B. Lack [lak], lag [la:k]
- Kontextsensitive Analyse; d.h. Disambiguierung der morpho-syntaktischen Klassifizierung durch Auswertung von Kontextinformationen
- Entwicklung einer Terminologieverwaltungskomponente, die das Ergänzen von benutzerdefinierten Lexikoneinträgen ermöglicht

3.4.3 Produkteinsatz

Streaming Tool, das als WebService gekapselt ist. Batchprozess, der nach erfolgter Konfiguration als Service gestartet wird. Lokaler Einsatz (Arbeitsplatz-Rechner) prinzipiell möglich, erfordert aber erhöhten Speicherplatzbedarf (Größenordnung von 10 MB RAM).

3.4.4 Produktfunktionen

3.4.4.1 Morphologische Textanalyse

Morpho-syntaktische Auszeichnung eines Textes angelehnt an Korpus-Annotationsstandards wie TEI P5 http://www.tei-c.org/Guidelines2/ und der Corpus Encoding Standard von EAGLES http://www.xml-ces.org/. Der Eingabetext wurde bereits auf der Wortebene segmentiert (siehe Tokenizer, Abschnitt 3.3.) und es liegt ein XML-kodierter Text mit einer Folge von Wort-Elementen vor, z.B. <w> Hund</w>. Die darauf aufsetzende weitergehende Auszeichnung basiert auf dem erweiterten deutschen morpho-syntaktischen Tag-Set STTS des IMS Stuttgart und der Uni Tübingen http://www.sfs.uni-tuebingen.de/Elwis/stts/, welches derzeit einen Standard für die Kodierung darstellt. Die Annotation kann in demselben Text (in-line) oder als stand-off Annotation vorgenommen werden.

Die Handhabung der morpho-syntaktischen Annotation lehnt sich an MAF (Morphological Annotation Framework) an, wobei eine (komplexe) Wortform (<wordForm>) auf ein oder mehrere Tokens verweist (bei Mehrwortlexemen) und selber wiederum aus simplen Wortformen, d.h. Morphemen, bestehen kann (bei Komposita, Derivaten etc.). Die morpho-syntaktischen Informationen werden als komplexe Feature-Structure Repräsentation ausgegeben (<fs>). Ambiguitäten können explizit kodiert werden (<wfAlt>) und es lassen sich auch Kodierungsvarianten (z.B. unterschiedliche TagSets) definieren.

Beispiel für die morpho-syntaktische Annotation von „Frauen“: (blau markierte Feature/Value Paare gehören zu den Wunschkriterien):

Word Form	***Frauen***
Lemma	***Frau***
citationForm	*Frau*
language	*nhd.*
orthographicalVariant	*alte/neue Rechtschreibung*
geographicalVariant	*Fruu (nd).*

etymologicalVariant	*frouwa (ahd.), vrouwe (mhd.)*
hyphenation	*Frau-en*
stem/headword	*Frau*
POS	*NN*
number-gender	*pl-fem-nom/gen/dat/akk*
lexicalizedForm	*ja*
synonym	*-*

Beispiel für die morpho-syntaktische Annotation von „Staubecken“.

Potentielle Wortzerlegungen, die aus der Wortbildungsanalyse hervorgegangen sind, werden mit einem Wahrscheinlichkeitsmaß gekennzeichnet.

Word Form	***Staubecken***
Lemma	***Staubecke***
citationForm	*Staubecke*
language	*nhd.*
orthographicalVariant	*alte/neue Rechtschreibung*
stem/headword	*Ecke*
POS	*NN*
number-gender	*pl-fem-*
complexForm	*yes*
type:	*compound*
form:	*Staub+Ecke+ n; N + N;*
probability:	*0.9*

Lemma	***Staubecken***
citationForm	*Staubecken*
language	*nhd.*
orthographicalVariant	*alte/neue Rechtschreibung*
stem/headword	*Becken*
POS	*NN*
number-gender	*sg-neut-nom/dat/akk; pl-neut-nom/gen/dat/akk*
derivedForm	*yes*
type:	*compound*
form:	*Stau+Becken; N + N;*
probability:	*0.8*

3.4.4.2 Unterstützung des Frontends für den Wörterbuchzugriff

Automatische Anzeige ein oder mehrerer Lemmaeinträge im Lexikon bei Markierung des entsprechenden Wortes im Text, auch wenn es in einer flektierten Form auftritt. Damit wird der semasiologische Zugang zum Wörterbuch unterstützt.

Die Eingabe erfolgt interaktiv durch den Benutzer. Das Frontend generiert zu einer Wortform eine Liste möglicher Lemmata inklusive der morpho-syntaktischen Informationen. Zu den Merkmalen, die für einen direkten Zugriff auf das Wörterbuch relevant sein können, gehören die Features *Language, orthographicalVariant,POS* und *lexicalizedForm*.

Tools zur automatischen Sprachenerkennung sollen die Auswahl eines geeigneten Wörterbuchs triggern. Alternativ kann der Benutzer zuvor ein oder mehrere Lexika auswählen, in denen das Wort gesucht werden soll.

Ebenso kann das Tool auch zum Auffinden aller Belegstellen im Korpus verwendet werden, falls zuvor ein Lemmaeintrag aus dem Wörterbuch gewählt wurde (siehe 3.4.4.3).

3.4.4.3 Unterstützung des Frontends für das Retrieval

Normalisierung der Queryterme und der Indexterme durch Lemmatisierung und/oder Wortbildungsanalyse sowie – im Fall der älteren Sprachstufen (althochdeutsch, mittelhochdeutsche und frühneuhochdeutsch) – die Wahl einer Zitierform aus einer Menge von Varianten. Diese Varianten sind das Resultat eines (halb)automatischen Verfahrens, welches separat entwickelt werden soll.

a) Normierung der Suchanfrage

   Die Eingabe der Suchanfrage erfolgt interaktiv durch den Benutzer. Für eine Liste von ein oder mehreren Querytermen generiert das Front-End eine entsprechende normalisierte Liste. Dabei kann die Normierung das Ergebnis der Lemmatisierung sein (schwache Normierung)

Suchanfrage	***Goethes gesammelte Werke***	*Goethes Sammelwerke*
Normalisierung	***Goethe sammeln Werk***	*Goethe Sammelwerk*

   oder das Ergebnis der Derivations- und Kompositaanalyse (starke Normierung).

Suchanfrage	*Goethes gesammelte Werke*	***Goethes Sammelwerke***
Normalisierung	*Goethe sammeln Werk*	***Goethe sammeln Werk***

b) Normierung der Indexterme

   Bei der Generierung eines (Volltext)Indexes im Retrieval werden in der Regel dieselben Normierungsfunktionen verwendet wie bei der Analyse der Suchanfrage. Für alle im Index enthaltenen Indexterme muss die normalisierte Form hinzugefügt werden. Somit wird der gesamte Index um eine Spalte ergänzt. Die Normierung hat einen wesentlichen Einfluss auf das Ranking (z.B. bei TF/IFD Gewichtung), da die Termfrequenzen der normalisierten vs. unnormalisierten Indexterme sich unterscheiden.

3.4.5 Produktdaten

3.4.5.1 Welche Daten sind aus Benutzersicht (langfristig) zu speichern

Für die *Morphologische Textanalyse*: Quelldatei, Zieldatei, Konfigurationsdatei, Log-Datei.
Für das *Front-End für den Wörterbuchzugriff/Retrieval*: Konfigurationsdatei, Log-Datei

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

Die zu bearbeitenden Daten können im Grid verteilt liegen. Zugriff erfolgt über die Rechner, auf denen der Prozess läuft. Konfigurationsdaten können lokal oder im Grid (Middleware) abgelegt werden.

3.4.5.3 Welche Datenmengen müssen verarbeitet werden

Hängt von der Anwendung ab: Für die Analyse des Query-Strings und den Wörterbuchzugriff werden nur wenige Token analysiert. Bei der Annotation von großen Korpora mit mehreren Gigabytes dagegen muss die Analyse mit minimaler Rechenzeit geschehen.

3.4.5.4 Daten-Format

Plain Text, XML (TEI), jew. Unicode-Zeichensatz (UTF-8).

3.4.6 Produktleistungen

Textanalyse:

Da die morphologische Analyse kontextfrei ist, ziehen lokale Änderungen am Textkorpus nicht eine komplette Neuannotierung nach sich. Es ist lediglich erforderlich, die Segmente (Sätze, Paragraphen), die sich geändert haben, neu zu annotieren.

Unterstützung des Front-Ends für das Retrieval:

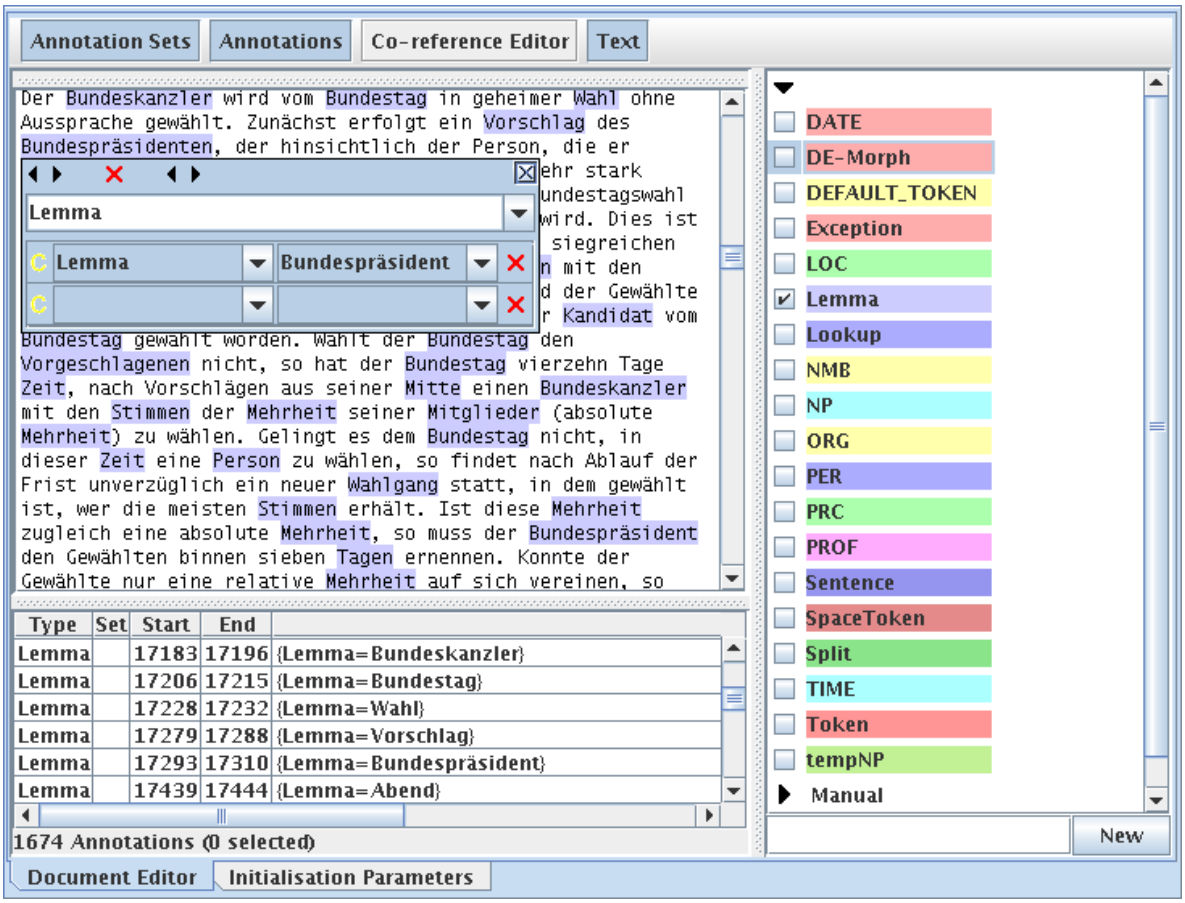
Wird das Textcorpus verändert, muss eine neue Indexierung durchgeführt werden.

3.4.7 Benutzeroberfläche

RCP: Konfigurierbares Benutzerinterface mit erweiterbaren Plug-ins.

a) Anzeige separater Fenster für Quelldatei und Zieldatei
b) Anzeige von Text und Annotation in einem Fenster. Die Annotierung kann farblich markiert werden.
c) Anzeige von Text und Annotation in getrennten Fenstern (stand-off).
d) Eingabemaske für Parameterübergabe im Batchbetrieb.

Der Benutzer kann spezifische vordefinierte Queries verwenden und unterschiedliche Perspektiven wählen.



Nur ein Beispiel – Durm Lemmatizer http://www.ipd.uka.de/~durm/tm/lemma/

3.4.8 Nichtfunktionale Anforderungen

3.4.8.1 einzuhaltende Gesetze und Normen, Sicherheitsanforderungen, Plattformabhängigkeiten

Plattformunabhängig, da als WebService implementiert

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

Logging der Suchanfragen und Benutzerdaten

3.4.9 Technische Produktumgebung

3.4.9.1 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)
  - fpconst 0.7.2 (http://cheeseshop.python.org/pypi/fpconst/0.7.2)
  - AT&T FSM Tools (http://www.research.att.com/sw/tools/fsm/)

– AT&T LexTools (http://www.research.att.com/sw/tools/lextools/)

Client:

- Siehe Abschnitt 3.2 zur RCP

3.4.9.2 Hardware: für Server und Client getrennt

Entsprechend den allgemeinen Projektvorgaben.

3.4.9.3 Produkt-Schnittstellen

Derzeit implementiert:

- Front-End für den Wörterbuchzugriff bzw. Retrieval (einfache Normalisierung)
- Verarbeitung tokenisierter Texte

Service ist publiziert unter http://ingrid.daasi.de/lemmatizer.wsdl

Parameter:

`modus`	Modus (`word/text`) Einzelwortanalyse/tokenisierter Text
Einzelwortanalyse (einfache Normalisierung)	
`inword`	Eingabewort
`returnlemma`	Analysestring (Lemma und morpho-syntaktisch Angaben)
Verarbeitung tokenisierter Texte	
`infile`	Quelldatei
`outfile`	Zieldatei
`tokenelement`	Element, mit dem zu analysierende Wörter ausgezeichnet sind
`lemmaattr`	Attribut zu `wordelement`, dessen Wert auf `returnlemma` gesetzt wird.

3.5 Workflow-Editor

3.5.1 Produktübersicht

Die Workflow-Komponenten sind im Wesentlichen auf den zwei oberen Schichten der TextGrid-Architektur angesiedelt: der Workflow-Editor in der Benutzerumgebung und der Workflow-Enactor im Service Layer. Hier soll es hauptsächlich um den Editor gehen. Als Enactor soll Software von externen Projekten zum Einsatz kommen, ohne bzw. mit möglichst geringen Modifikationen.

Der Workflow-Editor ist ein interaktiver Bestandteil der Benutzerumgebung und erlaubt es, die Automatisierung von Arbeitsabläufen (Workflows) zu definieren (orchestrieren). Neben den Servicekomponenten der Streaming-Tools können dabei auch interaktive Prozesse in einen Workflow eingebunden werden: In diesem Falle wird der Bearbeiter etwa per E-Mail oder IRC benachrichtigt, dass nun ein interaktiver Prozess erreicht ist, und es wird eine einfache Möglichkeit angeboten, die Benutzerschnittstelle entsprechend dieser aktuellen Aufgabe zu konfigurieren.

Der *Workflow-Editor* erlaubt eine weitgehend intuitive Spezifikation von Abläufen. Zu jedem einzelnen Streaming-Tool ist eine Maske verfügbar, über die es konfiguriert werden kann (Eingabe- und Ausgabedaten, falls nicht temporär, sowie weitere Konfigurationsoptionen des Tools). Hauptaufgabe des Workflow-Editors ist die Erstellung einer Prozessdefinitionsdatei, die alle zur Ausführung eines Workflows notwendigen Angaben enthält und an den Workflow-Enactor übergeben wird. Eine zusätzliche, aber optionale Aufgabe des Editors ist

das Anstoßen und die Überwachung der Ausführung des Workflows, welche im Enactor erfolgt.

Der *Workflow-Enactor* ist für die Ausführung des im Editor definierten Workflows zuständig. Die Zweiteilung Editor-Enactor hat sich bewährt und ermöglicht die Unabhängigkeit des Enactors von der konkreten Anwendung; hier kann deshalb auf vorhandene Software zurückgegriffen werden. Der Enactor empfängt die Prozessdefinition vom Editor und führt die darin spezifizierten Tasks in der festgelegten Abfolge aus. Im Falle von TextGrid muss der Enactor zumindest Web Services als Tasks ausführen können. Das Interface zum Enactor kann wiederum als Web Service realisiert sein; Funktionen zum Starten, Anhalten und Verwalten von verschiedenen Workflows sind hier verfügbar.

Als Prozessdefinition kommen prinzipell unterschiedliche XML-Formate in Frage. Nachteil der meisten ist, dass sie, wenn auch offen, so doch zumindest proprietär sind. Gerade im Bereich von Web Services – die hauptsächlich in TextGrid verwendet werden – hat sich als Workflow der Standard WS-BPEL (Business Process Execution Language) etabliert. Hierfür stehen eine Reihe kommerzieller und Open-Source-Implementierungen von Workflow-Enactoren zur Verfügung, weshalb es mittelfristig anzustreben ist, diesem Standard zu implementieren.

Der Workflow-Editor ist eine zentrale Komponente der Benutzerumgebung und muss intuitiv und ergonomisch bedienbar sein. Er ist sozusagen ein Meta-Tool, das andere Tools, wie den Tokenizer, Lemmatisierer oder XML-Editor, koordiniert.

In der jetzigen Aufbaustufe des Prototypen wird ein Workflow-Editor verwendet, der noch nicht in die RCP integriert ist. Es ist in der nächsten Phase zu prüfen, ob ein BPEL-konformer Workflow-Editor unter Nutzung bereits vorhandener entsprechenden Eclipse-Plugins, die den Anforderungen entsprechend angepasst werden, implementierbar ist. Mit der jetzigen prototypischen Implementierung mittels Taverna steht schon ein Workflow-Editor für die ersten Integrationsbemühungen zur Verfügung.

3.5.2 Zielbestimmung

Mit dem Workflow-Editor können TextGrid-Tools orchestriert werden.

3.5.2.1 Musskriterien

- Es müssen mittels WSDL definierte Web Services in XML-RPC und Document Style über SOAP/HTTP und URLs als synchrone Prozesse parallel und sequentiell orchestriert werden können. Unterstützung von Streaming-Tools.
- Komplette Integration in die TextGrid-RCP und zu den Konfigurationsschnittstellen der orchestrierten Tools. Letztere soll über in den SOAP-Messages übergebene Parameter (entweder die einzelnen Konfigurationsparameter oder der Name einer Konfigurationsdatei) erfolgen.
- Workflows mit Metadaten können im Grid gespeichert und anderen Benutzern zugänglich gemacht werden.
- Unterstützung von UDDI zum Auffinden von TextGrid-Services, oder Modellierung über den GT4 Index Service
- Verwaltung asynchroner Prozesse, z.B. für interaktive Bearbeitung von Dokumenten – hier auch Unterstützung von Nicht-Streaming-Tools sowie von E-Mail- oder sonstiger Kommunikation mit Benutzern

3.5.2.2 Wunschkriterien

(Die Reihenfolge spiegelt in etwa die Gewichtung wieder, mit wichtigeren Kriterien zuerst.)

- Konformität zum BPEL-Standard (möglichst Version 2.0), u.A. für einfache Austauschbarkeit des Enactors
- Mächtigkeit der Workflow-Prozessdefinition über gerichtete azyklische Graphen (DAG) hinaus, also erlaubte Zyklen, z.B. für iterative Workflows
- Ansprechen der Prozesse über WS-Addressing, besser noch sollten WSRF- (also Grid-) Services orchestriert werden können.
- Orchestrierung von GT2 GRAM oder WS-GRAM-Jobs
- Weitere Protokolle neben SOAP/HTTP (z.B. SMTP)

3.5.2.3 Abgrenzungskriterien

Der Workflow-Editor wird kein Daten-Scheduling betreiben. Es werden logische Dateinamen verwaltet (in den Konfigurationsmasken der Tools) und diese Dateinamen werden in die Prozessdefinition aufgenommen. Für ein evtl. notwendiges Scheduling müsste der Workflow-Enactor modifiziert werden, was zumindest zusätzlichen Programmieraufwand bedeutet.

3.5.3 Produkteinsatz

Definition von Workflows hauptsächlich für Streaming-Tools, Einbettung der Konfigurationsschnittstellen der Tools

3.5.4 Produktfunktionen

Aufruf des Workflow-Editors von der TextGrid-Benutzerumgebung, möglichst eingebettet und im gleichen Look-and-Feel. Auswahl von zur Verfügung stehenden Tasks (also TextGrid-Tools bzw. Web Services) aus einem Verzeichnis, z.B. über die TextGrid-Middleware und/oder über UDDI. Für jeden Task können interaktiv Konfigurationen erzeugt werden (Pop-Up-Fenster o.ä. mit Submasken, z.B. Dateibrowser); diese Funktionalität wird entweder über die in der RCP vorhandenen Eingabemasken der TextGrid-Tools oder über eine generische, im Workflow-Editor integrierte Konfigurationsmaske, die für jedes Tool einzeln im Workflow-Editor konfiguriert wird, zur Verfügung gestellt. Alternativ können aus der WSDL-Beschreibung eines Web Services dynamisch Masken für die notwendigen Eingabewerte generiert werden, vgl. den *Eclipse WTP Web Service Explorer*[10]. Die Tasks werden auf einer Arbeitsfläche platziert. Die ausgewählten Tasks können nun mit gerichteten Links verbunden werden, die entweder Kontrollcharakter (also Sequenz) oder Datenflusscharakter (also Verbindungen von einem Ausgabewert des einen Tasks zu einem Eingabewert des anderen Tasks) haben. Bei Datenlinks wählt der Workflow-Editor automatisch jeweils eine temporäre Datei zum Zwischenspeichern der Ergebnisse. Parallelität lässt sich implizit formulieren, indem von einem Task mehrere Kontroll- oder Daten-Links abzweigen.

Aus der grafischen Anordnung von Tasks und Links wird eine textuelle/XML-basierte Repräsentation generiert, die *Prozessdefinition*. Diese kann nun dem Workflow-Enactor übergeben sowie gespeichert und wiederverwendet werden.

Soll ein Workflow ausgeführt werden, müssen die Eingabedaten für den gesamten Workflow festgelegt werden. Der Workflow-Editor erlaubt deren Definition und das Abspeichern von Eingabekonfigurationen für Workflows. Sobald die Eingabedaten feststehen, kann der Workflow ausgeführt werden. Dies geschieht zweckmäßigerweise im Look-and-Feel der Benutzerumgebung; Details über Dateitransfers zum Enactor o.ä. bleiben vor dem Nutzer verborgen. Abhängig von der Granularität der Rückmeldungen des Enactors (z.B. „*running/*

[10] http://www.eclipse.org/webtools

suspended/stopped/completed" bis hin zu einer genauen Angabe, welcher Task gerade ausgeführt wird) wird der aktuelle Status eines übergebenen Workflows dem Benutzer angezeigt.

Ausgabewerte in einfachen Formaten (String, Zahl, Datum) können mit internen Betrachtern angezeigt werden, als Option können für komplexere Formate bzw. ganze Dateien separate Tools wie z.B. der XML-Editor (vgl. Abschnitt 3.6 XML-Editor) aufgerufen werden.

Der Workflow-Editor kann auch ein toolübergreifendes Logging verwalten und übergibt den Tools automatisch den Namen der Logdatei und den Loglevel.

Die Benutzerauthentifizierung kann ebenfalls toolübergreifend durch den Workflow-Editor übernommen werden, indem dieser mit den entsprechenden Web Services der TextGrid-Middleware kommuniziert.

3.5.5 Produkdaten

3.5.5.1 Welche Daten sind aus Benutzersicht (langfristig) zu speichern

Workflow-Prozessdefinition, Eingabekonfiguration (also die Gesamtheit aller Eingabewerte, evtl. mehrere Dateien), Ausgabekonfiguration (auch evtl. mehrere Dateien), evtl. Logdateien des Enactors

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

Die Daten können sowohl lokal als auch im Grid abgelegt werden.

3.5.5.3 Welche Datenmengen müssen verarbeitet werden

Keine Begrenzung in der Größe des Workflows (=der Prozessdefinition); Ein- und Ausgabekonfigurationsdaten haben ebenfalls keine Größenbegrenzung. Datenmengen der Tools sind von diesen selbst bzw. der Grid-Middleware abhängig.

3.5.5.4 Datenformat

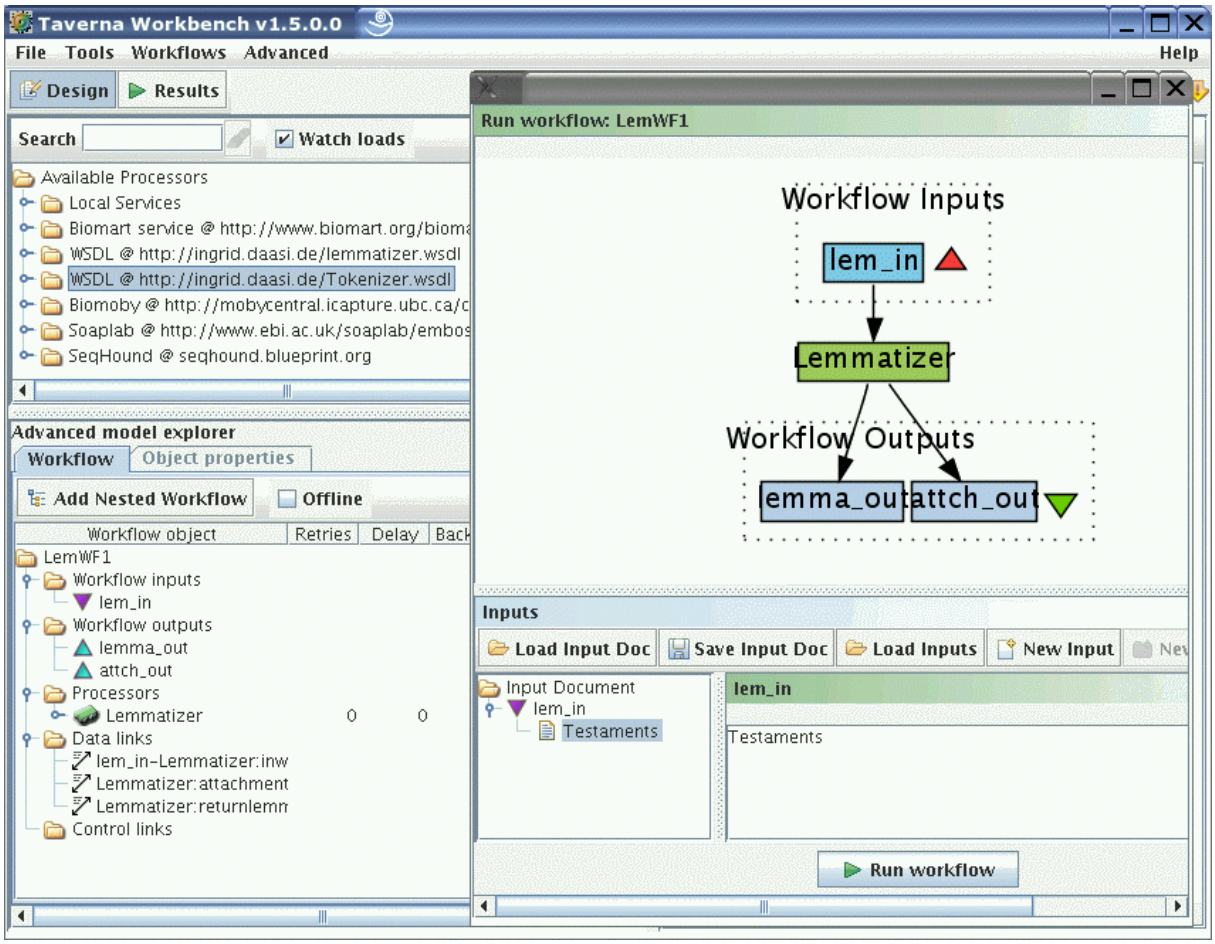
Strings sind in UTF-8 kodiert, sonst Format nicht spezifiziert.

3.5.6 Produktleistungen

Interaktiver Editor, mittels dessen man Workflows grafisch definieren und als XML-Prozessdefinitionsdatei abspeichern kann.

3.5.7 Benutzeroberfläche

Der jetzige Prototyp ist nicht in die RCP eingebunden. Das hierbei verwendete und im Folgenden beschriebene Produkt *Taverna* besitzt ein eigene Benutzeroberfläche (siehe Abbildung).



Screenshot Taverna

3.5.8 Nichtfunktionale Anforderungen

3.5.8.1 Einzuhaltende Gesetze und Normen, Sicherheitsanforderungen, Plattformabhängigkeiten

Plattform: Java / (möglichst Eclipse). Bei einer Portletimplementierung JSR 168.

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

Logging, Durchreichen von Benutzerdaten, siehe Bericht Architektur (Report R.3.2)

3.5.9 Technische Produktumgebung

Die folgenden Spezifikationen basieren auf den Anforderungen für das externe Tool „Taverna“ (http://taverna.sourceforge.net), das in einer ersten Implementierungsphase ohne Einbindung in die RCP verwendet wird.

3.5.9.1 Software: für Server und Client, falls vorhanden

Betrieb auf Linux, Windows oder Mac OS X unter Java JRE 1.5; Weitere Voraussetzung für Linux ist die separate Installation der „dot“-Anwendung (Paket Graphviz von AT&T Research) für die graphische Visualisierung von Workflows.

3.5.9.2 Hardware: für Server und Client getrennt

Mindestens 512 MB RAM; empfohlen 1 GB sowie Prozessor mit 1800+ Mhz. Wichtig: Netzwerkzugang, da beim ersten Start notwendige Bibliotheken installiert werden; außerdem werden dynamisch Informationen über vorhandene Web Services eingeholt.

3.5.9.3 Produkt-Schnittstellen

WebService, also SOAP, WSDL; mittelfristig BPEL.

3.6 XML-Editor

3.6.1 Produktübersicht XML-Editor

Der XML-Editor bildet eine der zentralen Komponenten des TextGrid-Clients: Er ist ein interaktives Tool, das zur Neuerfassung wie zur nachträglichen (manuellen) Annotation von Texten in XML-basierten Formaten dient. Darüberhinaus wird er mit anderen TextGrid-Tools in der *Rich Client Platform* integriert und kann so etwa für Streaming-Tools als Vorschau und zur Ergebnisdarstellung verwendet werden.

3.6.2 Zielbestimmung

Der Editor muss unterschiedlichen Benutzergruppen zur Erfüllung ihrer Aufgaben dienen: *Textwissenschaftlern* muss es möglich sein, bei Erfassung und Annotation den Überblick über den Text zu wahren und gerade Auszeichnungen wie Überschriften und Hervorhebungen mit den aus dem Druckbild gewohnten Mitteln visualisert zu sehen. *Bearbeitern* innerhalb eines Projekts soll über entsprechende Schemata möglichst präzise vorgegeben werden, welche Annotationen mit welchen Mitteln möglich sein sollen, um die Konsistenz in ihrem Projekt zu wahren – und die verfügbaren Annotationsmöglichkeiten müssen den Benutzern natürlich vermittelt werden. *Techniker* wollen etwa für Konvertierungsaufgaben auch einen Blick auf die präzise Serialisierung des XML-Dokuments haben.

Eine detaillierte Aufstellung der zu implementierenden Funktionen ist in Abschnitt 3.6.4 (Produktfunktionen) zu finden.

3.6.2.1 Musskriterien

Die wichtigste Funktion des XML-Editors ist es, Benutzer, die keine PC-Fachleute sind, in ihrer Arbeit mit XML-Texten möglichst weitgehend zu unterstützen. Ein wichtiges Element dieser Unterstützung ist die Möglichkeit, die visuelle Komplexität umfangreichen Markups zu reduzieren, indem in einem Dokument verschiedene *Darstellungsformen* unterstützt werden. Der Benutzer kann zwischen ihnen frei umschalten: Eine XML-Quellcodeansicht mit Syntaxhighlighting, eine Darstellung, in der die XML-Tags ausgeblendet und der Text mithilfe eines Stylesheets gestaltet wird (vgl. **Fehler! Es wurde kein Textmarkenname vergeben.**Abbildung 1 auf Seite 29**Fehler! Es wurde kein Textmarkenname vergeben.**) und eine hybride Darstellung mit symbolischer Darstellung der Elemente und Ausblendung der Attribute (vgl. Abbildung 2**Fehler! Es wurde kein Textmarkenname vergeben.**).

Der XML-Editor muss XML-Dokumente entsprechend einem in einer der gängigen XML-Schemasprachen (DTD, W3C XML Schema, Relax NG) formulierten *Schema* bearbeiten und validieren können. Der Benutzer soll bei seiner Arbeit möglichst weitgehend unterstützt werden, indem ihm angezeigt wird, welche XML-Elemente an einer bestimmten Position erlaubt sind und außerdem, welche Elemente er bereits häufiger verwendet hat. Hinzu kommt die Möglichkeit zum Auszeichnen vorhandenen Texts per Maus.

Darüber hinaus müssen dem Benutzer Möglichkeiten zur Orientierung und Navigation in seinem XML-Dokument gegeben werden.

Der Editor muss sich eng in das zu entwickelnde Eclipse-GUI-Framework mit der Rich Client Platform und den anderen zu entwickelnden Tools einfügen.

3.6.2.2 Wunschkriterien

Die Darbietung der möglichen Elemente bzw. des möglichen Inhaltsmodells an der Cursorposition sollte möglichst visuell und einfach verständlich geschehen. Zudem sollten die oft umfangreichen Wahlmöglichkeiten durch Beschränkung zugänglicher gestaltet werden, etwa in dem die Elementauswahl auf bereits verwendete Elemente oder eine vorkonfigurierte Auswahl reduziert werden kann.

Der Benutzer sollte die Oberfläche möglichst nach seinen Präferenzen konfigurieren können und sich so etwa häufig benötigte XML-Elemente oder Unicode-Zeichen auf Symbolleisten bzw. Tastenkombinationen legen können.

Die Bearbeitung sehr großer XML-Dokumente, ohne dass diese zuvor mit anderen TextGrid-Mitteln wie den Streaming-Tools zerlegt werden müssen, ist ebenfalls wünschenswert.

3.6.2.3 Abgrenzungskriterien

- Die Verwaltung von Dateien im Grid sollte vom Tool für Datei- und Rechtemanagement übernommen werden.
- Für die automatisierte Transformation von XML-Daten existieren die Streaming-Tools, insbesondere der Streaming-Editor.
- Die Erzeugung von XML-Schemata ist nicht Aufgabe des XML-Editors. Hierfür existieren spezielle Werkzeuge, für TEI-Anpassungen etwa das TEI-Tool *Roma*[11].
- Masken zur Eingabe von bereichsspezifischen, stark strukturierten Angaben wie etwa Metadaten, werden mit dem Metadateneditor definiert.

3.6.3 Produkteinsatz

Der XML-Editor wird vorwiegend zur Transkription existierender Quellen, zur Annotation vorhandener Texte, zum Verfassen neuer Texte und zur Vorschau automatisierter Bearbeitungen genutzt:

3.6.3.1 Transkription existierender Quellen

Existierende Quellen wie etwa Handschriften oder Drucke werden von Editoren oder Bearbeitern mithilfe des XML-Editors erfasst. Die Vorlagen können dabei sowohl bereits als Digitalisate als auch nur in originaler Form als physische Vorlage vorliegen. In letzterem Fall ist zu erwarten, dass die Transkription möglicherweise mit dem Laptop vor Ort in einem Archiv durchgeführt werden muss, aus dem das Original nicht entfernt werden darf.

3.6.3.2 Auszeichnung, Bearbeitung und Kommentierung vorhandener Texte

Bereits in XML- oder (digitaler) Textform vorliegende Dokumente werden vom Editor oder Bearbeiter mit zusätzlichen Annotationen versehen, um Kommentare ergänzt, bearbeitet und neu arrangiert. Vorhandene Annotationen werden bearbeitet.

3.6.3.3 Verfassung neuer Texte

Originäre Texte des Editors werden erfasst und ausgezeichnet – etwa editorische Notizen oder einleitende Kapitel.

3.6.3.4 Vorschau automatisierter Bearbeitungen

Editoren, Bearbeiter und technische Betreuer benutzen Streaming-Tools wie den Lemmatisierer oder den Streaming-Editor zur automatischen Bearbeitung von XML-

[11] http://tei.oucs.ox.ac.uk/Roma/

Dokumenten. Der XML-Editor wird als Komponente zur Darstellung und manuellen Überprüfung der Ergebnisse dieser Tools verwendet.

3.6.4 Produktfunktionen

3.6.4.1 Unterstützung von XML- und Schema-Standards

Gängige XML-Standards sollen unterstützt werden. Dies betrifft einerseits die Validierung von XML-Dokumenten (bzw. die Überprüfung auf Wohlgeformtheit), andererseits die in Abschnitt 3.6.4.5 (Unterstützung bei der Eingabe von Auszeichnungen) dargestellten schemaabhängigen Funktionen.

Die mindestens zu unterstützenden Standards umfassen:

- XML
- Die Schema-Sprachen
  - Relax NG
  - W3C XML Schema
  - DTDs

3.6.4.2 Einbindung in die Workbench

- Enge Integration in das sonstige TextGrid-Framework und insbesondere in die Eclipse-basierte Rich Client Platform
- Nutzung des XML-Editors zur Vorschau und Ergebnisanzeige für Input und Output der TextGrid-Webservices, z. B. Tokenizer, Lemmatisierer oder Streaming-Editor
- Aufruf der Benutzungsschnittstellen der Streaming-Tools aus dem XML-Editor heraus

3.6.4.3 Visualisierung des XML-Dokuments

Anzeigemodi

Der Benutzer kann zwischen verschiedenen Anzeigemodi umschalten:

- WYSIWYM[12]-Darstellung, in der Auszeichnungen durch aus dem Druckbild bekannte typografische Eigenschaften visualisiert werden (vgl. Abbildung 1). Die Konfiguration erfolgt mit *Cascading Style Sheets* (CSS), für gängige TEI-Konstrukte werden Stylesheets mitgeliefert. Liefert eine grundlegende Übersicht über den Text.

[12] What You Mean Is What You Get – eine Darstellung, die sich an einem möglichen Druckbild orientiert, aber (im Gegensatz zu WYSIWYG: What You *See* Is What You Get) statt auf eine 1:1-Umsetzung desselben eher auf eine sinnvolle Darstellung im Programmkontext zielt.

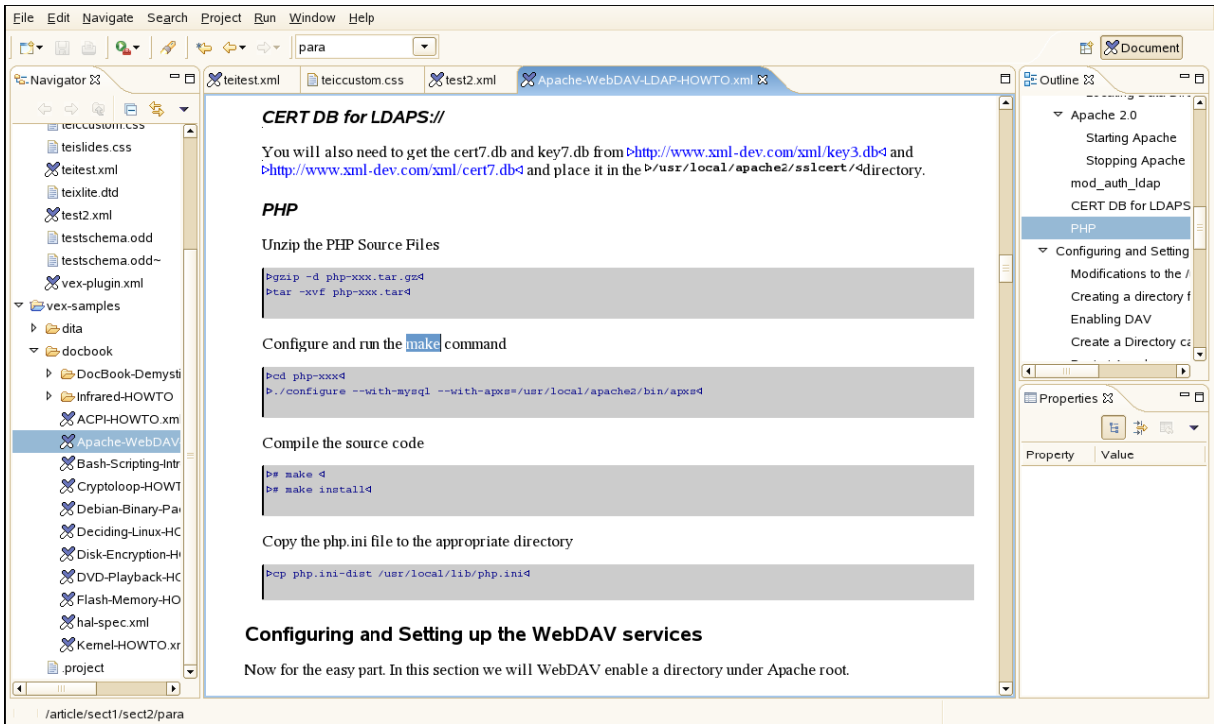


Abbildung 1: Beispiel für WYSIWYM-Darstellung (aus Vex[13])

- WYSIWYM-Darstellung mit einer zusätzlichen symbolischen Visualisierung aller Auszeichnungselemente, vgl. Abbildung 2. Zur Durchführung und Kontrolle der Auszeichnungen, ohne den Überblick über den Text im Spitzklammerwald zu verlieren.

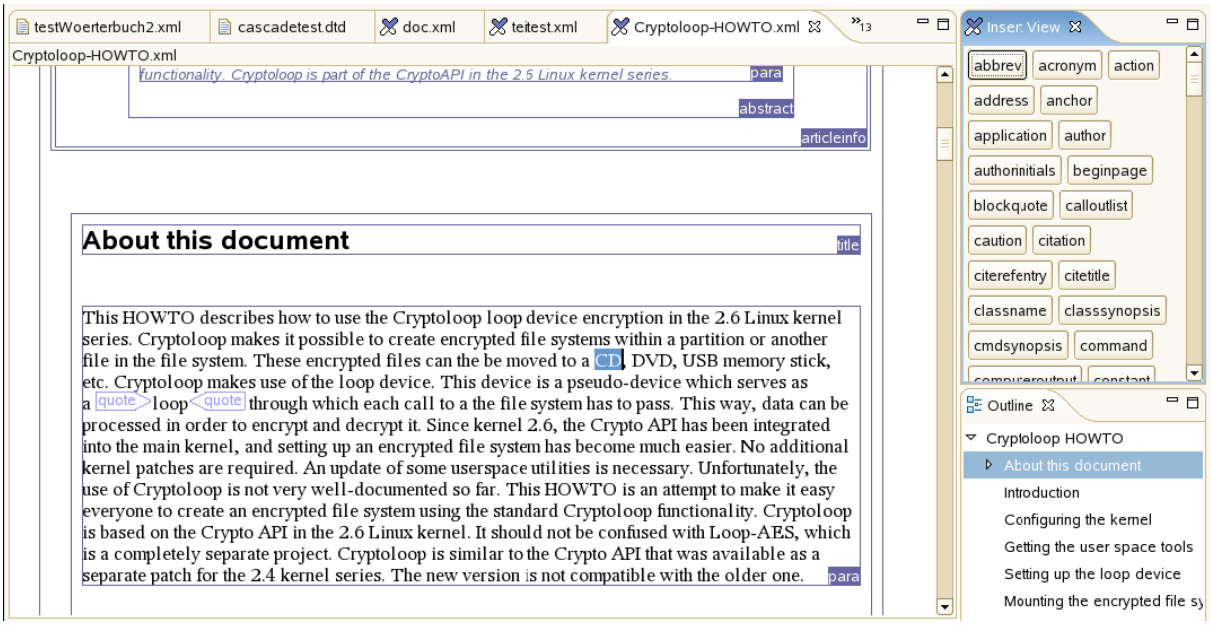


Abbildung 2: WYSIWYM-Darstellung mit symbolischer Visualisierung der Auszeichnung (aus der Entwicklerversion des XML-Editors)

- XML-Code mit Syntaxhighlighting. Zur präzisen Kontrolle über technische Details des XML-Dokuments.

[13] http://vex.sourceforge.net/

weitere Visualisierungsfunktionen

- Anzeige von Tabellen im WYSIWYM-Modus
- Folding: Der Benutzer sollte unerwünschte Teilstrukturen des Dokuments (etwa bestimmte Abschnitte) ausblenden können, um sich auf den aktuellen Teilbereich zu konzentrieren (Wunschkriterium)
- Elementausblendung: Der Benutzer sollte unerwünschte Elemente des Dokuments (etwa mit dem Text des Editors, nicht des Autors, in einer Edition) mitsamt ihrem Inhalt ausblenden lassen können (Wunschkriterium)
- Anzeige von Bildern im WYSIWYM-Modus. Die Darstellung wird über CSS geregelt.

3.6.4.4 Unterstützung bei der Orientierung im Dokument

- Gliederungsansicht: Der Benutzer sollte bestimmte XML-Elemente als Gliederungselemente festlegen können. Daraus wird dann eine Gliederungsansicht generiert, mit der der Benutzer durch das Dokument navigieren kann (siehe die Ansicht rechts unten in Abbildung 2).
- Elementhierarchie: Die aktuelle Position in der Elementhierarchie des Dokuments sollte anzeigbar sein (etwa als *TEI/text/body/div/p*)
- XPointer/XPath: Die aktuelle Position im Element sollte angezeigt werden, etwa in einer XPointer- bzw. XPath-artigen Struktur wie *TEI/text/body/div[4]/p[8]*.

3.6.4.5 Unterstützung bei der Eingabe von Auszeichnungen

- Die an der Cursorposition entsprechend dem Schema möglichen Elemente und sonstigen Inhalte sollten dem Benutzer dargeboten werden. Das umfasst:
  - Eine Anzeige der an der Cursorposition bzw. für den markierten XML-Abschnitt erlaubten Elemente mit der Möglichkeit, das Element an der aktuellen Position bzw. um die aktuelle Markierung herum einzufügen. Eine einfache prototypische Implementierung ist rechts oben in Abbildung 2 dargestellt.
  - Die Anzeige und Bearbeitung der vorhandenen und möglichen Attribute zum aktuellen Element.
  - Textauszeichnung durch Markieren des Texts und Anklicken des Auszeichnungselements.
- Die möglichen Elemente sollten auch mit der Tastatur in effizienter Art und Weise eingefügt werden können. Dies wird realisiert durch
  - Tag Completion in der Quellcodeansicht für alle erlaubten öffnenden Tags (mit Beschränkung auf diejenigen, deren Anfang bereits eingegeben wurde) bzw. die schließenden Tags aller offenen Elemente (mit automatischem Einfügen der »übersprungenen« schließenden Tags)
  - Eine vergleichbare einfache Elementauswahl an der Cursorposition in den anderen Ansichten, ebenfalls mit Unterstützung für das Eintippen eines Präfixes
- »Umwandlung« von Elementen, ohne dass der sonstige Text geändert wird. Realisierung zum Beispiel: Rechtsklick auf Tag → ›Tag ersetzen‹ im Kontextmenü wählen → Auswahlliste erlaubter Tags → Ersetzen von Anfangs- und Endtag.
- Kontextabhängige Einblendung der Dokumentation für XML-Elemente, wie sie beispielsweise aus den Dokumentationselementen der Schemata bzw. dem ODD-

Schema[14] abgeleitet werden können, in den Hilfen zur Elementauswahl bzw. in einem eigenen Programmbereich.

- Beschränkung der Liste der möglichen Elemente auf die bereits im Dokument verwendeten
- Automatische Einfügung der minimalen durch das Schema vorgegebenen Grundstruktur
- Möglichkeit zur Zusammenstellung ›beliebter‹ Elemente, ggf. mit Attributen vorkonfiguriert, etwa auf einer Symbolleiste; Zuordnung zu Tastenkombinationen

Wunschkriterien umfassen:

- Visualisierung des Modells an der Cursorposition auf einfach verständliche Weise, etwa als Railroad-Diagramm (vgl. Abbildung 3)

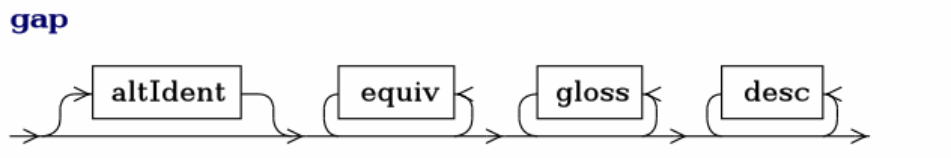


Abbildung 3: Das TEI-Element `gap` als Railroad-Diagramm.
(Quelle: Barry Cornelius, Sebastian Rahtz: *TV – A TEI Visualizer*)

3.6.4.6 Unterstützung für Sonderzeichen im Unicode

- Der Editor muss mit beliebigen Unicode-Zeichen umgehen können. Zeichen, die nicht dargestellt werden können, müssen dennoch erhalten bleiben.
- Für Sonderzeichen, die nicht in gängigen Tastaturbelegungen vorhanden sind, sollte es Eingabehilfen geben. Das umfasst
  - konfigurierbare Tastenkombination
  - konfigurierbare Symbolleisten
  - einen Auswahldialog bzw. eine Auswahlliste

  Dabei sind insbesondere auch die *Private Use Areas* von Unicode zu berücksichtigen.

3.6.4.7 Textverarbeitungsfunktionen

Folgende übliche Textverarbeitungsfunktionen werden in den Editor integriert:

- Ausschneiden, löschen, Kopieren, Einfügen
- Suchen, Ersetzen (Unterstützung regulärer Ausdrücke) (Optionen: Richtung oben/unten/gesamt; Einschränkung auf bestimmte Zeilen;
- Suchen mithilfe von XPath-Ausdrücken

3.6.5 Produktdaten

3.6.5.1 Welche Daten sind aus Benutzersicht (langfristig) zu speichern

Der XML-Editor soll dem Bearbeiten und natürlich auch dem Speichern beliebiger XML-Dokumente dienen, eine besondere Unterstützung für TEI-konforme Dokumente ist sinnvoll. Darüberhinaus ist ggf. die Zuordnung von Dokumenten oder Namespaces zu Schemata und Stylesheets (für den visuellen Modus des XML-Editors) zu speichern, ebenso Anpassungen

[14] http://www.tei-c.org.uk/wiki/index.php/ODD

der Arbeitsumgebung durch den Benutzer, soweit dies nicht durch die Rich Client Platform geschieht.

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

Die Daten liegen im Grid oder lokal vor. Im Grid können sie über die von der Middleware bereitgestellten FileServices angesprochen werden können. Für die Offline-Arbeit z. B. in Archiven ohne Internetzugang muss es auch möglich sein, Daten aus dem Grid lokal zwischenzuspeichern und später wieder ins Grid zu übertragen oder lokal erfasste Daten ins Grid einzuspielen.

Diese Funktionalität sollte für alle interaktiven Tools gemeinsam im Werkzeug zum Datei- und Rechtemanagement bzw. in der Rich Client Platform implementiert werden.

3.6.5.3 Welche Datenmengen müssen verarbeitet werden

Grundsätzlich ist die Bearbeitung sehr großer XML-Dokumente wünschenswert. Als Muss-Kriterium ist jedoch zunächst nur die Arbeit mit Daten im Umfang von mehreren Megabyte umzusetzen, größere Dokumente können ggf. vor der Bearbeitung mithilfe der Streaming-Werkzeuge zerlegt werden.

3.6.6 Produktleistungen

Beim Bearbeiten bestehender XML-Dateien sollten möglichst die Eigenschaften gemäß den XML-Datenmodellen *XML Information Set* und *XQuery 1.0 and XPath 2.0 Data Model* erhalten bleiben. Zudem sind (auch wenn die Darstellung von der Verfügbarkeit entsprechender Schriften abhängt) Unicode-Zeichen auch jenseits der Basic Multilingual Plane zu erhalten.

3.6.7 Benutzeroberfläche

Die Benutzeroberfläche sollte sich an gängigen Usability-Kriterien orientieren (vgl. etwa ISO 9241, Teil 10) und für Textwissenschaftler, Bearbeiter und technische Betreuer unterschiedlicher Computerexpertise benutzbar sein. Eine Anlehnung der GUI an die von der Textverarbeitung her bekannten Oberflächen wird angestrebt, soweit dies mit der spezifischen Funktionalität eines XML-Editors vereinbar ist. Eine enge Integration in die *Rich Client Platform* und mit den anderen Tools der TextGrid-Workbench ist verpflichtend.

3.6.8 Nichtfunktionale Anforderungen

3.6.8.1 Einzuhaltende Gesetze und Normen, Sicherheitsanforderungen, Plattformabhängigkeiten

Die Software soll die gängigen Standards im XML-Bereich nicht verletzen und Unicode-Daten berücksichtigen. Sie muss auf allen Plattformen laufen, auf denen die *Rich Client Platform* läuft, mindestens auf aktuellen Linux-, Macintosh- und Windows-Plattformen.

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

Nicht anwendbar.

3.6.9 Technische Produktumgebung

3.6.9.1 Software

Der Client wird im Rahmen der *Rich Client Platform* auf der Basis von Eclipse implementiert, siehe Abschnitt 3.2. Eine Serverkomponente ist nicht vorhanden.

3.6.9.2 Hardware

Es gelten die Anforderungen der *Rich Client Platform*, siehe Abschnitt 3.2. Aus Performancegründen ist einigermaßen aktuelle Hardware wünschenswert.

3.6.9.3 Produktschnittstellen

Der XML-Editor implementiert, soweit vorhanden, die Eclipse- bzw. RCP-eigenen Schnittstellen, um eine möglichst enge Integration in die *Rich Client Platform* bzw. mit anderen auf dieser Plattform aufbauenden Tools zu realisieren.

Anhänge

Anhang 1: Arbeitsplan AP 2

Milestones/ Reports	Monate	Tool	Tool-Typ	Verantwortlich	DAASI	FH Worms	IDS Mannheim	Saphor	SUB Göttingen	TU Darmstadt	U Trier	U Würzburg	fachl. Betreuung	altes Tool / Kommentar
		PM insges.*			7	2	14	37	10	24	26	3		
M 2.1	1-6	**Tokenizer**	**s**	Saphor			x	x	x	x	x		IDS, Trier	
		Workflow-Editor	**i**	DAASI	x				x	x			TUD	Editor techn. Workflow
M 2.2	7-12	**Lemmatisierung**	**s**	IDS Mannheim			x	x			x		IDS, Trier*	* nicht Neuhochdeutsch
		XML-Editor	**i**	TU Darmstadt					x	x			TUD	
		Rich Client Platform (GUI)	**i**	SUB Göttingen					x	x			SUB	
R 2.1	12	**Dokumentation**												
M 2.3	13-24	**Recherchetool**	**i**	Saphor			(x)	x		x			IDS, TUD	Query Interface, Text Retrieval
		Streaming-Editor I (XSLT-Komponente, u.a. für Adaptor-Manager)	**s**	Saphor	x			x					TUD, Worms	
		Metadaten-Annotation	**i**	SUB Göttingen			x		x	x	x		SUB	
		Datei- / Rechtemanagement	**i**	SUB Göttingen	x				x					Editor admin. Workflow
		grafischer Link-Editor	**i**	TU Darmstadt						x		x	TUD, WÜ	Link-Editor
		Bild-Segmentierung	**s**	TU Darmstadt						x		x	TUD, WÜ	Link-Editor
		Link-Editor Text		Saphor				x		x			TUD	Link-Editor
		Bibliographietool	**i**	DAASI	x			x			x		Trier	
		Sortieren	**s**	FH Worms		x	x				x		IDS, Trier	
R 2.2	24	**Dokumentation**												
M 2.4	25-30	**Streaming-Editor II**	**s**	Saphor				x		x			TUD	
		Kollationierung	**s**	U Trier						x	x	x	Trier, WÜ	
M 2.5	31-36	**Text Publisher (Print)**	**s**	FH Worms		x		x						
		Text Publisher (Web)	**s**	Saphor				x						
		OCR	**s**	U Würzburg					x		x	x	SUB, Trier, WÜ	
R 2.3	36	**Dokumentation**												

Tool-Typ: „s“ = Streaming Tool, „i“ = interaktives Tool; Grün hinterlegt: gegenüber dem Antrag geänderte Tools

Anhang 2: Schnittstellen zur Middleware – File Services

Für die Anbindung eines TextGrid-Service an die TextGrid-Middleware gibt es auf Ingrid (Standort: DAASI, http://ingrid.daasi.de:8080/axis/services/*) die WebServices:

- `FileInputStreamService` zum sequentiellen Lesen von Text aus (entfernten) Dateien,
- `FileInputStreamBinaryService` zum sequentiellen Lesen von Bytes aus (entfernten) Dateien,
- `FileOutputStreamService` zum sequentiellen Schreiben von Text in (entfernte) Dateien,
- `FileOutputStreamBinaryService` zum sequentiellen Schreiben von Bytes in (entfernte) Dateien,
- `RandomAccessFileService` zum Lesen aus und Schreiben von Bytes in (entfernte) Dateien an beliebigen Dateipositionen,
- `FileService` zur Manipulation von (entfernten) Dateien/Verzeichnissen,
- `AdvertServiceService` zum Veröffentlichen und Wiederfinden von Gridobjekten (z.B. Dateien) und
- `FactoryService` zum Erzeugen von Instanzen der oben genannten WebServices.
- Noch zu implementieren: Services für Logging

Ein Schema für die clientseitige Verwendung dieser WebServices zeigt der folgende Pseudocode:

```
1. {"sessionId" => sessionId, "url" => service, ...} :=
   FactoryService.createFileInputStream(uri) // oder
   FactoryService.createFileOutputStream(uri) oder
   FactoryService.createFile(uri)
2. service.<Operation_1>(sessionId, <Args_1>)
3. service.<Operation_2>(sessionId, <Args_2>)
4. ...
5. service.<Operation_n>(sessionId, <Args_n>)
6. service.destroy(sessionId)
```

Zeile 1 erzeugt eine Instanz eines der drei WebServices `FileInputStreamService`, `FileOutputStreamService` oder `FileService`, der auf der Datei oder dem Verzeichnis uri (z.B. dem Verzeichnis any://ingrid.daasi.de//tmp/TextGrid/) arbeiten wird. Diese Instanz hat die Adresse `service` vom Typ String (oder die WSDL-Beschreibung `service?wsdl`) und ist fortan über eine SessionId `sessionId` vom Typ String ansprechbar. Zur Abkürzung bezeichnen wir eine solche WebServiceInstanz der Form `{"sessionId" => sessionId, "url" => service, ...}` mit dem Typ `WebServiceInstance` (kurz: WSI).

Die Zeilen 2 bis 5 zeigen die Verwendung der in Zeile 1 erzeugten Instanz des WebService `service`. Es werden die Operationen Operation_1 bis Operation_n aufgerufen, die jeweils als erstes Argument die SessionId `sessionId` zur Identifizierung der Instanz erhalten.

Zeile 6 ruft die `destroy`-Operation des WebServices auf und löscht somit die über `sessionId` ansprechbare WebService-Instanz. Dieser destroy-Aufruf nach der Verwendung einer WebService-Instanz ist obligatorisch, weil wir bisher keine automatische Speicherbereinigung eingeführt haben. Bisher werden alle Dateien unter dem user globus angelegt und kein Rechtemanagement durchgeführt. Es ist geplant, ein solches Benutzer- und Rechtemanagement zu implementieren.

WSDL-Beschreibungen der Services sind unter
http://ingrid.daasi.de:8080/axis/servlet/AxisServlet einzusehen.