TextGrid – Installation eines Datengrid-Knotens

Version – 16.03.2009

Arbeitspaket – AP3

verantwortlicher Partner – SUB Göttingen

TextGrid

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

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:
Stefan Funk, DAASI
Peter Gietz, DAASI
Martin Haase, DAASI
Wolfgang Pempe, Saphor

Inhaltsverzeichnis

Einleitung

In diesem Dokument wird beschrieben, wie ein weiterer DatenGrid-Knoten für TextGrid eingerichtet werden sollte. Zielgruppe sind Administratoren oder technische Mitarbeiter von Institutionen, die entweder Grid-Speicherplatz zur Verfügung stellen möchten oder ihre Datenarchive in TextGrid einbinden möchten.

Als Hintergrund einige Worte zur Architektur von TextGrid: Der TextGrid-Client (TextGrid-Lab) greift auf die TextGrid-Middleware zu, die zusammen mit den GridKnoten und Archiven das TextGrid-Rep(ository) bildet. Zusätzlich gibt es noch Services („Tools“) in TextGrid, die aber nicht im eigentlichen Sinne Grid-Services sind und somit auch keine Grid-Rechenknoten benötigen, sondern im Prinzip auf beliebigen Servern im Netz laufen können. TextGrid ist somit ein Projekt, in dem zunächst ein Storage-Grid aufgebaut wurde und bislang noch kein Compute-Grid.

Die DatenGrid-Knoten von TextGrid werden allesamt von einem zentralen TextGrid-crud Service bedient (Create-, Read, Update- und Delete-Service), der den Clients – beispielsweise dem TG-lab – lesenden und schreibenden Zugriff auf die gespeicherten TextGrid-Objekte ermöglicht.

1 Einrichtung eines TextGrid Daten-Knotens

Die TextGrid-Middleware greift via GAT[1] auf Datei-Funktionen des Globus-Toolkits[2] zu, um TextGrid-Daten im Grid zu verteilen. Ein TextGrid-Datenknoten muss deshalb über eine Globus-Toolkit-Installation verfügen, die im Folgenden beschrieben wird.

1.1 Globus Toolkit

Hier sollen in Kürze die wichtigsten Daten für eine Installation des Globus Toolkit (im Folgenden GT) genannt werden. Allerdings geben wir vor allem Hinweise auf weiterführende Quellen, anstatt den durchaus mannigfaltigen und ausführlichen Dokumentationen zum GT eine weitere hinzuzufügen.

1.1.1 Voraussetzungen

Für die Dateifunktionen wird der Service GridFTP des GT verwendet. Er benötigt in der Firewall die folgenden ein- und ausgehenden offenen Ports: 2811 und 20000 bis 25000.[3]

Für GT können verschiedene Betriebssysteme eingesetzt werden. Offiziell unterstützt werden vom D-Grid Integrationsprojekt (DGI) Scientific Linux 4.5 und 5.1 sowie SLES 10.[4] Dessen ungeachtet sind auf den Seiten des Globus-Projekts[5] jedoch viele weitere Konfigurationen als Binärpakete erhältlich, so u.A. zu RedHat Advanced Server 3 und 4, Fedora Core 4, Debian Etch, SuSE 9, Mac OS X, Solaris 9 und 10, und AIX 5.3. Als Hardware sind x86er Architekturen am besten geeignet (32bit oder 64bit).

Als weitere Voraussetzung werden Server-Zertifikate für das Grid benötigt. Im D-Grid werden Zertifikate vom DFN oder FZK akzeptiert.[6]

Mit der Teilnahme an TextGrid (und damit an D-Grid) wird auch die Einwilligung in das

---

[1] http://www.gridlab.org/WorkPackages/wp-1/, aber auch die diesbezüglichen Seiten des DGI-Projekts unter http://www.d-grid.de/index.php?id=80 (letzter Zugriff 20.02.2009)
[2] http://www.globus.org/toolkit/ (letzter Zugriff 20.02.2009)
[3] http://www.d-grid.de/index.php?id=309 (letzter Zugriff 20.02.2009)
[4] http://dgiref.d-grid.de/wiki/Operating_systems (letzter Zugriff 20.02.2009)
[5] http://www.globus.org/toolkit/ (letzter Zugriff 20.02.2009)
[6] http://www.d-grid.de/index.php?id=304 (letzter Zugriff 20.02.2009)

Betriebskonzept gegenüber dem DGI notwendig.[7]

1.1.2 Globus Toolkit und der TG-crud Service

Zunächst gibt es nur eine Instanz des TextGrid crud-Service, der für eben diese Funktionen Methoden zur Verfügung stellt. Eine Verteilung der TextGrid-Daten findet über dessen Replika-Management statt. Über das Replika-Management kann konfiguriert werden, auf welchen Grid-Knoten die TextGrid-Daten repliziert werden. Die Konfiguration des TG-crud Replika-Managements ist weiter unten beschrieben.

Für die Zusammenarbeit zwischen dem TextGrid crud-Service und dem neu eingerichteten Globus Toolkit sind im Globus Toolkit keine weiteren Einstellungen vorzunehmen.

2 Einspielen großer Datenmengen

Das TextGrid-Repository kann vom TG-Lab bequem mit Daten gefüllt werden, jedoch ist ein komfortables Einspielen großer Datenmengen mit dem Lab bisher nicht automatisiert möglich.

Zum einen benötigt jede Datei, die in das TextGrid-Repository eingespielt werden soll, eine entsprechend vollständige Metadaten-Datei. Zum anderen gibt es bei zusammengehörigen Datenbeständen wie zum Beispiel Wörterbüchern, Textkorpora oder weiteren verschiedenartig strukturierten Beständen Abhängigkeiten zwischen den einzelnen Dateien, beispielsweise zwei Versionen eines logisch identischen Objekts, wie ein Scan einer Buchseite als TIFF-Datei und der Volltext eben dieser Seite als TEI-Datei. Diese Abhängigkeiten sollen idealerweise in den Metadaten der TextGrid-Objekte abgebildet werden, um effektiv mit den Daten arbeiten und Beziehungen abfragen zu können.

In den TextGrid-Metadaten der TEI-Datei des oben genannten Beispiels kann nun der XML-Tag

```
<relation>
      <isAlternativeFormatOf>
            textgrid:uri-to-existing-TIFF-image
      </isAlternativeFormatOf>
</relation>
```

gesetzt werden, dessen Inhalt die URI eines bereits eingespielten TIFF-Images enthält. Diese Relationen-Daten werden in der Relationendatenbank gespeichert. Sie können so für die Recherche und für die Visualisierung von Beziehungen zwischen TextGrid-Objekten genutzt werden.

Um solche Informationen nicht per Hand und für jede Datei einzeln erfassen und speichern zu müssen, wird ein Tool benötigt, das die vorhandenen Strukturen der Daten nutzt, um automatisiert die Relationen zwischen den Dateien und auch andere Metadatan wie Titel, Autor, etc. zu erfassen und mit ihnen eine TextGrid-Metadaten-Datei sinnvoll zu füllen. Aus den Dateien und ihren zugehörigen Metadaten werden so TextGrid-Objekte, die dann zu guter Letzt in das TextGrid-Repository eingespielt werden.

2.1 koLibRI – kopal Library of Retrieval and Ingest

Mit der *kopal Library of Retrieval and Ingest (koLibRI)*[8] existiert ein solches Tool, welches im

[7] http://www.d-grid.de/uploads/media/D-Grid-Betriebskonzept.pdf (letzter Zugriff 20.02.2009)

Projekt kopal[9] entwickelt wurde. koLibRI ist ein Framework zur Integration eines Langzeitarchivs wie dem IBM Digital Information Archiving System (DIAS)[10] in die Infrastruktur einer Institution. Insbesondere organisiert koLibRI das Erstellen und Einspielen von Archivpaketen in DIAS und stellt Funktionen zur Verfügung, um diese abzurufen und zu verwalten.

Mit *koLibRI* kann ebenfalls der für das Einspielen benötigte Workflow abgebildet und eine große Menge an Dateien als Eingabe verarbeitet werden, die bereits vorhandene Infrastruktur kann nachgenutzt und für den TextGrid-Import erweitert werden. So können mit relativ wenig Aufwand die einzelnen Aktionen bis hin zum Speichern im TextGrid-Repository per TG-crud-Client modular mit Hilfe von Java-Klassen implementiert werden. Projektabhängige Konfiguration und Anpassung des Workflows an vorhandene Datenbestände ist somit möglich – abhängig von der Struktur des Datenbestandes und/oder gewünschtem Komplexitätsgrad der zu erstellenden TextGrid-Objekte.

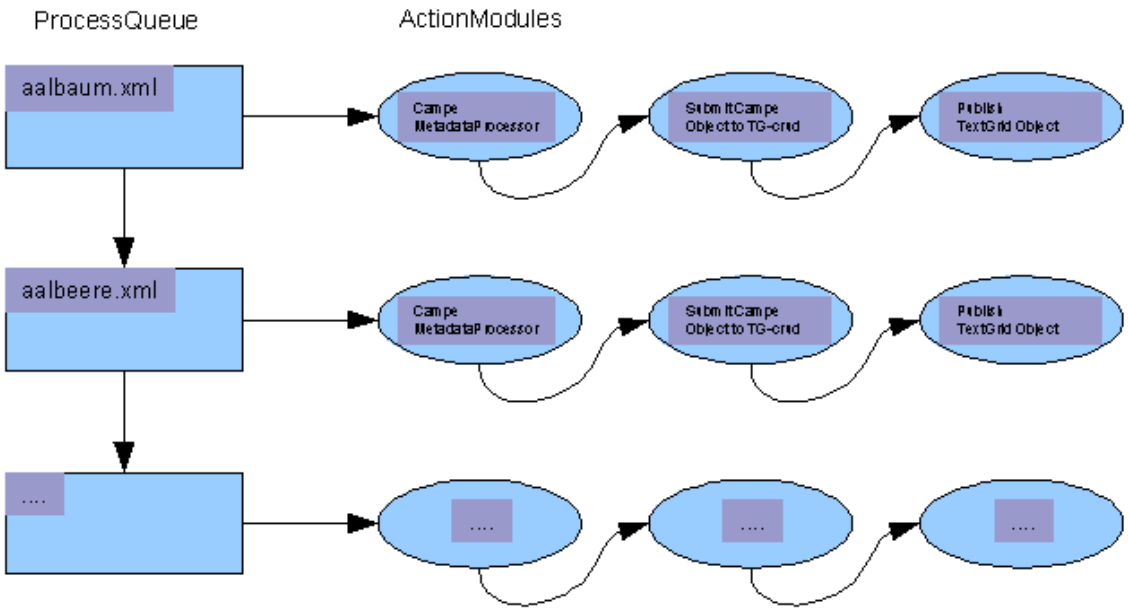
Mit Hilfe der kopal Library for Retrieval and Ingest wurden Teile des „Wörterbuch der Deutschen Sprache“ von Joachim Heinrich Campe in das TextGrid-Repository eingespielt. An diesem Beispiel wird die Funktionsweise von koLibRI nachfolgend erläutert. Tiefer gehende Informationen zu koLibRI und weitere technische Details, die hier nicht erwähnt werden, können in der koLibRI-Dokumentation nachgelesen werden, da sie für ein Einspielen in TextGrid nicht notwendig sind.[11]

Die koLibRI-Infrastruktur nutzt prinzipiell vier Konstrukte, um Workflows abzubilden und zu verarbeiten:

- Zunächst sammelt der sogenannte *ProcessStarter* die einzuspielenden Dateien/Daten ein, die als kleinste Einheit für den TextGrid-Ingest definiert wurden – für TextGrid sinnvollerweise eine einzelne Datei (zum Beispiel eine TIFF- oder TEI-Datei).
- Jede Einheit wird vom ProcessStarter an die *ProcessQueue* angehängt, die dann der Reihe nach (oder auch nebenläufig) abgearbeitet werden.
- In den *ActionModules* werden einzelne Aufgaben implementiert, die für ein jedes Objekt in der ProcessQueue durchgeführt werden sollen. Für den TextGrid-Ingest wurden im Rahmen des TextGrid-Projekts unter anderem ein Metadaten-Extraktions-Tool (für Campe-Metadaten) sowie ein TextGrid-crud-Client (universell für TextGrid nutzbar) implementiert. Auch einige der mit koLibRI gelieferten ActionModules und ProcessStarter können für einen TextGrid-Ingest von Nutzen sein – zum Beispiel ein Modul, das mit Hilfe von JHOVE[12] technische Metadaten aus einzelnen Dateien extrahiert oder ein Modul, das Dateien auf UTF-8-Konformität prüft
- Die Reihenfolge, in der die ActionModules für jedes dieser Objekte in der ProcessQueue verarbeitet werden, wird als *Policy* konfiguriert.

---

[8] http://kopal.langzeitarchivierung.de/index_koLibRI.php.de (letzter Zugriff 20.02.2009)
[9] http://kopal.langzeitarchivierung.de/ (letzter Zugriff 20.02.2009)
[10] http://www-5.ibm.com/nl/dias/ (letzter Zugriff 20.02.2009)
[11] http://kopal.langzeitarchivierung.de/index_koLibRI.php.de (letzter Zugriff 20.02.2009)
[12] JHOVE – JSTOR/Harvard Object Validation Environment: http://hul.harvard.edu/jhove/ (letzter Zugriff: 20.02.2009)



Der koLibRI Campe-Workflow

2.1.1 Installation und Konfiguration für TextGrid

Eine für TextGrid vorkonfigurierte Beispiel-Installation von koLibRI kann im TextGrid Subversion-Repository[13] heruntergeladen werden, die auf den folgenden Seiten detailliert erläutert wird.

In dieser Anleitung wird eine Installation unter Eclipse beschrieben, ein Betreiben von koLibRI auf der Konsole ist ebenfalls möglich, hierzu kann das Startskript „kolibri“ oder „kolibri.bat“ genutzt werden. Es müssen lediglich die benötigten Bibliotheken in das Skript eingebunden werden.

Zwei Beispiele für das Einspielen großer Datenmengen stehen für einen TextGrid-Ingest momentan zur Verfügung. Zum einen ein einfaches Beispiel, das alle Dateien aus einem Verzeichnis unter Nutzung eines Hotfolder-Mechanismus' einsammelt, die Metadaten aus einer Metadaten-Vorlagendatei bezieht und beides als ein TextGrid-Objekt im TG-rep speichert. Zum anderen ein etwas komplexeres Beispiel, bei dem die Metadaten-Extraktion und die Relationen zwischen den einzelnen Dateien eine Rolle spielen, weshalb es hier auch näher erklärt wird.

Folgende Schritte müssen für eine erfolgreiche Installation zunächst durchgeführt werden:

1. Auschecken des koLibRI-Clients als Eclipse-Java-Projekt:

   `https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgcrud/clients/tgcrudclient_kolibri/`

2. Auschecken der Axis2-Bibliotheken in das Verzeichnis `./lib/axis2` des koLibRI-Clients:

   `https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/lab/core/external/org.apache.axis2.full/lib/`

[13] `https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgcrud/clients/tgcrudclient_kolibri/` (letzter Zugriff: 11. Februar 2009)

3. Einbinden aller Bibliotheken des koLibRI-Clients und der Axis2-Bibliotheken (zu finden nun im Verzeichnis `./lib`).
4. Erstellen einer Run-Configuration in Eclipse im ausgecheckten Projekt:
    - Main Class: `de.langzeitarchivierung.kopal.WorkflowTool`
    - Argumente: `-c ./config/textgrid/campe_config.xml`

Bis auf die folgenden Einstellungen, die unter anderem von der TextGrid-Authentifizierung abhängen, sind alle nötigen Dinge für einen Campe Test-Ingest bereits vorkonfiguriert. In der Datei `./conf/textgrid/campe_config.xml` müssen noch alle mit `***` gekennzeichnenden `<value>` Tags mit sinnvollen Werten gefüllt werden, das sind im einzelnen:

- `<field>projectId</field>` – Die Projekt ID des Projekts, in das eingespielt werden soll.

  Beispiel: `<value>TGPR123</value>`
- `<field>rbacSessionId</field>` – Die RBAC Session ID, die nach dem Einloggen in das TextGrid-Lab vergeben wird.

  Beispiel:

  ```
  <value>
        3Gg52c1u08JiO3uKHX75FACrCix7jY3wokF8i2CIQS9xtnbx05Sin5eQTcktwc
  </value>
  ```
- `<field>rbacSessionIdEnv</field>` – Alternativ kann auch auf eine Umgebungsvariable verwiesen werden, die, sofern sie eine RBAC Session ID enthält, bevorzugt genutzt wird.

  Beispiel: `<value>RBAC_SESSION_ID</value>`

2.1.2 Einspielen der Campe-Daten – Programmablauf

Wird das koLibRI-WorkflowTool nun mit dieser Konfiguration gestartet, passiert im einzelnen folgendes:

1. Der ProcessStarter `textgrid.CampeIngest` wird aufgerufen, um die ProcessQueue zu füllen, diese wird dann der Reihe nach (oder auch parallel[14]) abgearbeitet. Die Klasse `de.langzeitarchivierung.kopal.processstarter.textgrid.CampeIngest` ist, wie bereits erwähnt speziell, speziell an die Daten des Campe-Wörterbuchs angepasst.

   Dieser schaut sich alle Files in den dafür konfigurierten Verzeichnissen[15] an und unterscheidet nach TIFF- und XML-Dateien. Für die TIFF-Dateien werden aus der vorhandenen Verzeichnis-Struktur Daten wie Band und Seite, für die XML-Dateien Daten aus den Dateinamen extrahiert. Diese werden in eine Struktur geschrieben, auf die die ActionModule später zugreifen können (CustomData). Die TIFF-Dateien werden zuerst in die ProcessQueue angehängt, weil diese von den XML-Dateien später referenziert werden, danach folgen die XML-Dateien.
2. Nun wird von koLibRI die ProcessQueue abgearbeitet, für jeden Eintrag die angegebenen ActionModule, die in der Policy-Datei `/conf/textgrid/policies.xml` definiert wurden. Sowohl die Policy-Datei als auch der Policy-Name sind in Campe-Konfigurationsdatei.[16]

[14] siehe `<field>maxNumberOfThreads</field>` in `./config/textgrid/campe_config.xml`

[15] siehe `<field>imageDataDir</field>` und `<field>xmlDataDir</field>` in `./config/textgrid/campe_config.xml`

[16] `<field>policyFile</field>` und `<field>defaultPolicyName</field>` in `./config/textgrid/campe_config.xml`

Dies sind im einzelnen:

3. Actionmodul `textgrid.CampeMetadataProcessor`

   Dieses ActionModule nimmt sich die vom ProcessStarter gesammelten Daten und füllt damit, je nach dem, ob es sich um eine TIFF- oder XML-Datei handelt, die entsprechenden Projekt-spezifischen Daten in die vorhandenen TextGrid Metadaten-Vorlagen[17].

   Handelt es sich um eine XML-Datei, wird kontrolliert, wird die TextGrid-URI der dazugehörigen TIFF-Datei – wenn bereits eingespielt – als alternatives Format in die Metadaten der XML-Datei übernommen.

4. ActionModul `textgrid.SubmitCampeObjectToTextgridCrud`

   Im Prinzip ist dieses Modul ein TG-crud-Client, der in ein koLibRI ActionModul eingebunden wurde. Es wird ein TextGrid-crud#CREATE durchgeführt und sowohl die TIFF- oder XML-Datei, als auch deren TextGrid-Metadaten an den TG-crud übermittelt. Einzelne Campe-Spezifika sind zum Beispiel die Extraktion der URIs der erfolgreich eingespielten TextGrid-Objekte.

5. ActionModul `textgrid.PublishTextgridObject`

   Wenn gewünscht, kann dieses Modul die erfolgreich eingespielten Objekte für die Öffentlichkeit im TG-rep als publiziert markieren. Diese Funktion ist zunächst in der Policy-Datei auskommentiert.

2.1.3 Abhängigkeiten zwischen TextGrid-Objekten

Wie schon kurz im obigen Beispiel erwähnt, können diverse Abhängigkeiten bzw. Relationen zwischen einzelnen TextGrid-Objekten im Tag `<relational>` der Metadaten eines Objekts angegeben werden. Diese Informationen werden sowohl in der XML-, als auch in der RDF-Datenbank des TG-rep gespeichert.

Als Relationen-Metadaten stehen die folgenden vier in den Metadaten eines TextGrid-Objekts zur Verfügung – jeweils mit einer TextGrid-URI als Parameter:

- `isVersionOf [URI]`

  Beispiel: Ein existierendes Dokument `[URI]` wird von einem weiteren Nutzer bearbeitet und soll als eine neue (fortgeschrittene) Version eingespielt werden, ohne das alte Dokument zu überschreiben.

- `isAlternativeFormatOf [URI]`

  Beispiel wie oben: Ein Volltext-Dokument wird als alternative Version eines XML-Dokuments `[URI]` eingespielt, oder ein PDF-Dokument als alternative Version eines TIFF-Bildes. Hier soll prinzipiell der Inhalt der Dokumente gleich sein, nur das (technische) Format ist ein anderes.

- `hasAdaptor [URI]`

  Für XML-Dateien kann an dieser Stelle die TextGrid-URI einer bereits eingespielten XSLT-Datei angegeben werden, die dazu genutzt wird, die einzuspielende Datei in das Baseline-Encoding[18] zu transformieren. Mit Hilfe von TG-search kann dann über das gesamte TextGrid-Repository einheitlich recherchiert werden. Mehr dazu weiter unten im Kapitel

---

[17] `./config/textgrid/campe_xml_metadaten-template.xml` und `./config/textgrid/campe_tif_metadaten-template.xml`

[18] Für weitere Informationen zum TextGrid Baseline Encoding siehe http://www.textgrid.de/fileadmin/TextGrid/reports/TextGrid_Kerncodierung_070615.pdf (letzter Zugriff: 20.02.2009)

„Der Adaptor Manager“.

- `hasSchema [URI]`

  Hier kann eine TextGrid-URI eines XML-Schemas referenziert werden, dem die XML-Datei entspricht, die eingespielt werden soll.

Sollen solche Relationen bereits beim Einspielen berücksichtigt werden, muss für diese referenzierten Objekte bereits eine TextGrid-URI bekannt sein. Da eine URI jedoch erst dann bekannt ist, wenn ein Objekt erfolgreich eingespielt wurde, folgt hieraus wiederum, dass nur Objekte referenziert werden können, die schon in das TG-rep vorliegen.

Bei Relationen wie `<isAlternativeFormatOf>` muss somit auf die Reihenfolge des Einspielens geachtet werden, wie das im vorliegenden Beispiel an den Campe-Daten beschrieben ist.

2.1.4 Programmierung von koLibRI ActionModules

koLibRI-ActionModules sind Java-Klassen, die das Interface `de.langzeitarchivierung.kopal.actionmodule.ActionModule` implementieren. Wichtig für den Workflow sind die Attribute `processData` und `step`, die Daten eines einzelnen Elements der ProcessQueue verwalten und den Status eines einzelnen ActionModules steuern. `processData` verwaltet auch das Objekt `customData`, das im Beispiel für die Metadatenübergabe zwischen ProcessStarter und ActionModules genutzt wird.

Die ActionModules können komfortabel in der koLibRI-Konfigurationsdatei konfiguriert werden, indem die Modulklasse **(a)** ein Klassen-Attribut mit Setter-Methode implementiert wird und **(b)** eine Sektion für dieses Modul wie folgt in die Konfigurationsdatei eingefügt wird:

```
<class name=“actionmodule.textgrid.NameDerKlasse“>
      <property>
            <field>Attributname</field>
            <value>zu setzender Attribut-Wert</value>
      </property>
</class>
```

Mehr zu der Programmierung von ActionModules und ProcessStarter, zur Einpassung vorhandener Klassen in diese oder weitere Einzelheiten zur Konfiguration und Implementation von koLibRI können in der koLibRI-Dokumentation nachgelesen werden.

2.2 Weitere Möglichkeiten des Ingest

Die Nutzung von koLibRI ist nur eine von vielen Möglichkeiten, eine große Menge an Daten in das TextGrid-Repository einzuspielen. Da ein Zugriff auf die Methoden des TG-crud per SOAP-WebService möglich ist, kann jedes andere Tool, das SOAP-WebServices unterstützt, die benötigten Workflows abbilden, die erforderlichen Daten sammeln und viele, viele, viele Daten einspielen – immer vorausgesetzt, der Zugriff ist autorisiert.

Eine Modellierung der Metadaten-Nutzung sowie das extrahieren und berücksichtigen der Struktur der vorliegenden Daten ist jedoch immer Voraussetzung für ein erfolgreiches Nutzen der eingespielten Daten.

3 Adaptoren

Adaptoren sind, wie bereits weiter oben kurz erwähnt, XSLT-Skripte, die im TextGrid-Repository als Objekte vorliegen, und die von XML-Dateien (bzw. deren Metadaten) referenziert werden können. Beim Einspielen desjenigen Objekts, in dessen Metadaten ein Adaptor angegeben ist, wird die XSL-Transformation auf das einzuspielende Objekt angewandt. Momentan werden von TG-crud nur Transformationen in das Baseline-Encoding akzeptiert, das dann zusätzlich zu dem original XML und den Metadaten im TG-rep gespeichert werden.

3.1 Der AdaptorManager

Prinzipiell soll sich der TextGrid-crud-Service darauf beschränken, die Metadaten der TextGrid-Objekte aktiv zu bearbeiten, beispielsweise auszulesen oder zu ergänzen; die Daten der Objekte selbst (zum Beispiel die TEI- oder andere XML-Dateien) betrachtet TG-crud nicht als XML-Konstrukt, sondern behandelt diese als Bytestream, um sie im Grid oder in diversen Datenbanken zu speichern. Für TG-crud also unterscheidet sich eine TIFF-Datei – betrachtet als Daten-Objekt – nicht von einer XML-Datei.

Der AdaptorManager wird nur dann von TG-crud aufgerufen, wenn ein Zugriff auf die Daten der TextGrid-Objekte nötig wird, zum Beispiel dann, wenn eine XML-Datei in das Baseline-Encoding umgewandelt werden soll. Er ist der Teil der TextGrid-Middleware, die eben diese Transformation durchführt, gibt die umgewandelte Datei an TG-crud zurück, der sie dann (wieder als Bytestream) weiter verarbeitet.

Aufgerufen wird der AdaptorManager ausschließlich vom TextGrid-crud-Service, wenn ein `<hasAdaptor>` Tag in den Metadaten angegeben ist und es sich bei der einzuspielenden Datei um eine XML-Datei handelt.

Der AdaptorManager führt dann die folgenden Schritte durch:

1. Falls es sich um eine XML Schema-Datei handelt, werden alle dort vorhandenen Namespaces extrahiert und in die RDF Datenbank geschrieben (`<definesSchema>`).
2. Wenn ein Zugriff auf die angegebene XSLT Adaptor-Datei autorisiert ist und die Datei existiert, wird die Transformation durchgeführt. War diese erfolgreich und das Ergebnis ist eine valide XML-Datei im Baseline-Encoding, wird diese von TG-crud in die XML-Datenbank geschrieben.
3. Die valide Baseline-Datei wird zuletzt noch nach Links auf andere TextGrid URIs durchsucht, die als `<refersTo>` Relation in die RDF Datenbank geschrieben werden.

3.2 Programmierung von Adaptoren

Wie bereits oben erwähnt, müssen XML-Objekte, um in einer Projekt-übergreifenden Suche gefunden und durchsucht werden zu können, in einer zusätzlichen, Instanz abgelegt werden, die dem Schema der Kernkodierung (Baseline-Encoding)[19] entspricht. Der AdaptorManager führt die Transformation mit Hilfe eines XSLT-Stylesheets durch. Im Folgenden sollen die für die Transformation prinzipiell notwendigen (d.h. unabhängig vom Original-/Projekt-spezifischen XML-Schema) Parameter/Einstellungen besprochen werden[20]:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

[19] http://www.textgrid.info/schemas/textgrid-baseline-2008-05-20.xsd (noch nicht endgültig).

[20] Einen umfassenden Überblick über den Standard XSLT 2.0 bietet Kay (2004).

```
      version="2.0" xmlns="http://www.tei-c.org/ns/1.0"
      xmlns:db="http://docbook.org/ns/docbook">
```

`version="2.0"`

Der vom AdaptorManager angesprochene XSLT-Prozessor Saxon(-B)[21] unterstützt seit der Version 7.0 XSLT 2.0 (derzeit im Einsatz: 9.x).

`xmlns="http://www.tei-c.org/ns/1.0"`

Der Default-Namespace wird auf `http://www.tei-c.org/ns/1.0` gesetzt, da das Schema der Kernkodierung auf TEI P5 basiert.

`xmlns:db="http://docbook.org/ns/docbook"`

Weitere Namespaces (z.B. DocBook), die in den zu verarbeitenden Daten vorkommen und bei der Verarbeitung zu berücksichtigen sind.

```
<xsl:output method="xml" omit-xml-declaration="no" encoding="UTF-8"
indent="no" normalization-form="NFKC"/>
```

`method="xml"`

Das Zielformat der Transformation ist XML

`omit-xml-declaration="no"`

Die generierte XML-Instanz soll eine XML-Deklaration (`<?xml version="1.0"?>`) erhalten.

`indent="no"`

Die hierarchische Struktur der Ziel-Instanz muss nicht durch Einrückung angezeigt werden (spart Speicher).

`encoding="UTF-8"`

Damit Textdaten in TextGrid problemlos durchsucht und verarbeitet werden können, müssen sie in UTF-8 vorliegen.

`normalization-form="NFKC"`

Siehe Unicode Standard Annex #15 — Unicode Normalization Forms[22]. Buchstaben mit Diakritika bzw. über- oder unter-gesetzten Zeichen müssen vereinheitlicht werden, da sie auf unterschiedliche Weise realisiert – d.h. entweder zusammengesetzt oder als einzelnes Zeichen definiert – werden können.

```
<xsl:template match="*">
      <xsl:element name="{local-name(.)}"
```

[21] http://saxon.sourceforge.net

[22] http://www.unicode.org/unicode/reports/tr15/tr15-23.html (letzter Zugriff: 16.03.2009)

```
                namespace="http://www.tei-c.org/ns/1.0">
                <xsl:copy-of select="@*"/>
                <xsl:apply-templates/>
        </xsl:element>
</xsl:template>
```

Für den Fall, dass die Original-Daten oder Teile davon in einem anderen Namespace liegen, muss dieser durch den TEI-Namespace ersetzt werden, da eine Suche in den Strukturdaten sonst nicht möglich ist.

```
<xsl:template match="db:book">
        <TEI>
                <xsl:apply-templates/>
        </TEI>
</xsl:template>
```

In weiteren `<xsl:template>`-Elementen können dann die spezifischen Transformationen vorgenommen werden. Dabei ist darauf zu achten, dass valides XML im Sinne des Kernkodierungs-Gesamtschemas generiert wird, da der AdaptorManager eine entsprechende Validierung durchführt, bevor er die neu generierte XML-Instanz in die Datenbank schreibt, auf die TG-search zugreift.

4 ReplicaManagement

Das ReplicaManagement des TextGrid-Repositories wird zunächst vom TextGrid-crud Service gesteuert und auch konfiguriert. Unter Nutzung von JavaGAT[23] – über das zur Zeit alle nötigen Zugriffe auf die TextGrid DatenGrid-Knoten abgewickelt werden – wird momentan jedes Erstellen (TG-crud CREATE) und jede Aktualisierung (TG-crud UPDATE/UPDATEMETADATA) einer Datei konfigurierbar an verschiedenen Grid-Knoten gesichert. In der TG-crud Konfigurationsdatei `tgcrud.properties` wird zu diesem Zweck neben der Angabe der `DEFAULT_GAT_URI`, die den ersten Speicherort der TextGrid-Dateien angibt, ein weiterer Wert `REPLICA_GAT_URIS` definiert, der eine White-Space separierte Liste von weiteren Grid-Locations enthält. Alle schreibenden Operationen des TG-crud werden dann auf alle diese weiteren Orte repliziert.

JavaGAT bedient sich hierzu einer sogenannten logischen Datei, die zum Funktionsumfang des Globus Toolkit gehört und die von JavaGAT genutzt wird. Diese logische Datei verwaltet eine Liste von physischen Speicherorten für die TextGrid-Objekte, das heißt, für eine logische TextGrid URI – zum Beispiel:
`textgrid:Sandbox:Die+Leiden+des+jungen+Werther:20090211T013132:xml%2Ftei:1` – gibt es beliebig viele physische Repräsentationen einer Datei. Nach dem Speichern einer Datei (sei es eine Metadaten-Datei oder eine Inhalts-Datei) wird von TG-crud ein solches logisches Datei-Objekt angelegt, und alle in der Konfiguration angegebenen Grid-URIs werden als physische Speicherpfade angelegt, und letztendlich wird die Datei an all diesen Orten im Grid gespeichert.

Ein Merkmal des ReplicaManagements (und der LogicalFiles) ist, dass die Entfernung der Grid-Knoten zueinander berücksichtigt wird. Wenn beispielsweise eine Datei repliziert werden soll, wird diese von der Datei kopiert, die dem Zielknoten am nächsten ist.

[23] JavaGAT – Java GRID Application Toolkit: https://gforge.cs.vu.nl/gf/project/javagat/ (letzter Zugriff: 16.03.2009)

Zukünftig soll das TextGrid ReplicaManagement sehr viel feiner konfigurierbar sein, so dass beispielsweise (auch direkt vom Nutzer) projektabhängig festgelegt werden kann, welche Daten auf welchem Grid-Knoten wie oft repliziert werden. Außerdem sollen dann auch lesende Zugriffe mit den LogicalFiles berücksichtigt werden, so dass auch hier dem Nutzer die Datei geliefert wird, die den schnellsten Zugriff verspricht.