TextGrid
Virtuelle Forschungsumgebung
für die Geisteswissenschaften

Optimierung der TextGrid Architektur und Infrastruktur

(Report R 4.5.2)

Version 27. Februar 2014
Arbeitspaket 4.5
Verantwortlicher Partner GWDG

TextGrid

Virtuelle Forschungsumgebung für die Geisteswissenschaften

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

Projekt: TextGrid – Institutionalisierung einer Virtuellen Forschungsumgebung in den Geisteswissenschaften
BMBF Förderkennzeichen: 01UG1203A
Laufzeit: Juni 2012 bis Mai 2015

Dokumentstatus: Final
Verfügbarkeit: Öffentlich
Autoren:

Fatih Berber, GWDG
Maximilian Brodhun, SUB Göttingen
Stefan E. Funk, DAASI International GmbH
Peter Gietz, DAASI International GmbH
Ubbo Veentjer, SUB Göttingen
Philipp Wieder, GWDG

Revisionsverlauf:

Datum	Autor	Kommentare
25.10.2013	Philipp Wieder	Erste Version
10.02.2013	Fatih Berber	Zweite Version
14.02.2013	Maximilian Brodhun	Kommentare und leichte Korrekturen
17.02.2013	Stefan E. Funk	Kommentare und leichte Korrekturen
19.02.2013	Maximilian Brodhun	Leichte Ergänzungen
19.02.2014	Stefan E. Funk	DARIAH-Part ergänzt
20.02.2014	Peter Gietz	Ergänzungen im Gesamttext
26.02.2014	Fatih Berber	Status Quo TG-crud
27.02.2014	Maximilian Brodhun	Status Quo TG-search
27.02.2104	Philipp Wieder	Endredaktion
27.02.2014	Stefan E. Funk	Formatierung letztes Kapitel, Einbinden des DARIAH-Repositoriums-Dokument in den Anhang

Inhalt

1. Einleitung

TextGrid[1] stellt eine produktive Forschungsumgebung für die textorientierte Geisteswissenschaften und zunehmend auch für andere Disziplinen bereit. Ziel der aktuellen Förderphase ist es, diese Umgebung zu verstetigen und nachhaltige Konzepte und Strukturen zu erarbeiten, die Forscher auch in der Zukunft bei ihrer Arbeit bestmöglich unterstützt.

Eine wichtige Säule bei der Verstetigung von TextGrid ist die Middleware, welche die Basis für die Forschungsumgebung darstellt. Hier ist es notwendig, die existierende Architektur und Umsetzung in regelmäßigen Abständen mit den Wünschen der Fachwissenschaftler, Nutzerzahlen, Dienstgütedaten und aktuellen technischen Entwicklungen abzugleichen und zu bewerten, sowie die Resultate mit Hinblick auf den technischen Regelbetrieb zu betrachten. Bei einer solchen Betrachtung zeigen sich in der Regel Lücken, die mittels einer sogenannten Gap Analysis zu erfassen sind. Die Kosten und Zeitaufwände für das Schließen dieser Lücken sind anschließend zu erheben und eine Risikoabschätzung einzelner Lücken bezüglich deren Umsetzung zu machen. Ausgehend davon lässt sich eine Roadmap für die Umsetzung formulieren. Grundsätzlich geht es dabei vornehmlich um eine Verstetigung der bisherigen Funktionalität und nicht um die Entwicklung neuer Funktionalitäten.

Zu Beginn der Förderphase wurde die wurde die angesprochene Gap Analysis durchgeführt und in Bericht R 4.5.1, „Architekturskizze inklusive Identifikation der Lücken und der Optimierungsmöglichkeiten“ dokumentiert. Im Folgenden wurde mit der Optimierung von TextGrid begonnen. Der vorliegende Bericht beschreibt die bis dato umgesetzten technischen Verbesserungen und setzt sie in Bezug zu den identifizierten Lücken.

Der Bericht ist wie folgt strukturiert: Anschließend an diese Einführung, die im nächsten Abschnitt kurz die Ziele des für den technischen Regelbetrieb und die Architektur zuständigen Arbeitspaketes beschreibt, werden in Abschnitt 3 kurz die Kernkomponenten von TextGrid beschrieben, gefolgt von einer kurzen Zusammenfassung der Evaluierung der TextGrid 2.0 Architektur in Abschnitt 4. Dies ist notwendig, um das in Abschnitt 5 beschriebene Hochverfügbarkeitskonzept in Bezug zur Middleware zu setzen und in Abschnitt 6 die bisher durchgeführten Arbeiten zu dessen Umsetzung zu beschreiben. Abschnitt 7 gibt dann abschließend einen Einblick in die technischen Aspekte der Zusammenarbeit zwischen den Vorhaben TextGrid und DARIAH-DE, die ebenfalls eine Verstetigung und Optimierung der Angebote für die Digital Humanities Communities zum Ziel hat.

[1] http://www.textgrid.de/

2. Ziele des Arbeitspaketes AP 4.5

Das Ziel des Arbeitspaketes AP 4.5 „Technischer Regelbetrieb, Architektur, innovatives Evolutionspotential“ ist gemäß Vorhabensbeschreibung

„[...] *die technische Verstetigung der TextGrid-Infrastruktur sowie die Vorbereitung des Regelbetriebs des TextGridRep bei der GWDG. Vorbereitend wird zunächst die Architektur der TextGrid-Middleware evaluiert und eine Architekturskizze einschließlich Dokumentation erstellt. Hier kann ein großer Teil der bereits vorhandenen Dokumentation nachgenutzt werden. Je nach Ergebnis dieser Analyse werden gegebenenfalls Code- und Infrastrukturanpassungen erforderlich sein – dies möglichst unter Beibehaltung der Middleware- API, die vom TextGridLab und anderen Clients genutzt wird.“*

Dabei geht es unter anderem darum, die TextGrid-Architektur und -Infrastruktur stetig zu optimieren. Dazu werden sowohl die Architektur als auch die Infrastruktur ständig analysiert und evaluiert und die Ergebnisse mit den Ergebnissen des *technology watch* abgeglichen. Aus den gesammelten Erkenntnissen werden entsprechende Strategien entwickelt die einen ausfallsicheren und performanten Betrieb von TextGrid und den assoziierten Dienstangeboten zum Ziel haben.

3. TextGrid-Middleware

Die TextGrid-Middleware besteht aus den Hautkomponenten TG-auth*, TG-crud und TG-search. Diese stellen die grundlegenden Funktionen bereit, die für das TextGrid Repository[2] benötigt werden, und bilden somit das Rückgrat der virtuellen Forschungsumgebung für die Geisteswissenschaften. Daher ist es essentiell, dass diese Komponenten nicht nur zuverlässig arbeiten, sondern dass sie im gleichen Maße eine gleichbleibende Performanz bei steigenden Nutzerzahlen garantieren. Zudem muss sichergestellt werden, dass die TextGrid-Middleware technologisch den Anschluss behält und keine teuer zu pflegende oder bereits abgekündigte Software Bestandteil der Middleware bleibt. Um auf die Komponenten detailliert in Bezug zum Hochverfügbarkeitskonzept im vierten Abschnitt eingehen zu können, werden die Hauptkomponenten und ihr Status Quo kurz erläutert.

3.1. Hauptkomponenten der TextGrid-Middleware

3.1.1. TG-auth*

TG-auth* ist die zentrale Infrastruktur, die die notwendigen Funktionalitäten für (föderierte) Authentifizierung und Autorisierung sowie für die Zugriffskontrolle zur Verfügung stellt. TG-auth* bietet hierzu verschiedene Web-basierte und Web-Service-basierte Schnittstellen und verwendet konsistent als Datenbank die hochperformante Open-Source LDAP-Implementierung OpenLDAP.

3.1.2. TG-crud

TG-crud ist die Verwaltungseinheit der TextGrid-Datenobjekte. Ein TextGrid-Datenobjekt besteht aus den Daten selbst und den zugehörigen Metadaten. Identifiziert werden die Datenobjekte durch die TextGrid-URI, welche durch die Komponente TG-noid erzeugt werden. Als Verwaltungseinheit nimmt TG-crud Anfragen von Klienten an und bearbeitet diese. Klienten können die Erstellung (**C**reate), das Lesen (**R**ead), das Aktualisieren (**U**pdate) und das Löschen (**D**elete) eines Datenobjekts anfragen. Beim Erstellen eines Datenobjekts sendet der Klient die Daten und die Metadaten an TG-crud. Dieser speichert die einzelnen Bestandteile eines TextGrid-Datenobjekts auf die verschiedenen Speicher-Backends in der Middleware. In der Middleware sind folgende Instanzen existent: TG-auth (tgextra-crud), Sesame (RDF-Datenbank, Triple Store), ElasticSeach (Indexdatenbank) und der StorNext-Knoten. Beim erfolgreichen Zugriff auf alle diese Instanzen sendet TG-crud eine Antwort zurück, andernfalls eine Fehlermeldung.

3.1.3. TG-search

Die Kernaufgabe der Suchfunktion ist durch die Middlewarekomponente TG-search implementiert. Dabei kann mittels TG-search über die Volltextdateien gesucht werden, welche im TextGrid Repositorium als TEI-Dokumente gespeichert werden. Des Weiteren besteht die Möglichkeit gezielt auf den Metadatenfeldern zu suchen. Auch eine kombinierte Suche in Volltext und Metadaten ist möglich. Diese Funktionen sind sowohl für die Suche in der Web-

[2] Webseite TextGrid Repository, http://www.textgridrep.de

präsenz des Repositoriums gegeben, als auch für die Suche durch die Anwendung TextGrid-Lab. Im Kontext dieser Anwendung ist TG-search weitergehend für die Navigation innerhalb der Projekte zuständig.

TG-search arbeitet auf zwei voneinander getrennten Such-Indizes. Zum einen existiert ein dynamischer Speicher für Dokumente an denen aktuell gearbeitet wird und die nicht veröffentlicht sind und zum anderen ist ein statischer Speicher vorhanden, der die öffentlich zugänglichen Dokumente bereitstellt.

Die Metadaten und die TEI-Daten sind im JSON Format in einer ElasticSearch Indexdatenbank abgelegt. Die Daten für den öffentlichen und den nicht öffentlichen Bereich sind dabei in zwei getrennten Indizes gespeichert. Die Umstellung auf ElasticSearch ist eine der Neuerungen für die bereits angesprochene Verstetigung der TextGrid-Infrastruktur.

Zur Realisierung der Navigation über Projekte besteht eine Schnittstelle zu der Middlewarekomponente TG-auth*, mit der die Ergebnisse von TG-search abgeglichen werden, um festzustellen, ob der Nutzer oder die Nutzerin die notwendigen Rechte besitzt. Für zusätzliche Informationen für die Navigation, wie zum Beispiel den Wert der letzten Revision, nutzt TG-search den RDF-Speicher Sesame. An diesen werden Anfragen in der Abfragesprache SPARQL gestellt.

4. Zusammenfassung des Berichts R 4.5.1

Der Bericht R4.5.1[3] analysiert die Architektur, basierend auf welcher die TextGrid-Version 2.0 implementiert wurde. Dabei wurden die gesamte Architektur und alle ihre Komponenten auf den Prüfstand gestellt, Lücken sowie Verbesserungspotential identifiziert und Lösungsvorschläge für die Optimierung der Architektur und der Implementierung gemacht.

Hier werden knapp die Resultate für die Hauptkomponenten der TextGrid-Middleware, TG-auth*, TG-crud und TG-search, vorgestellt. Auf diese Weise können Hochverfügbarkeitskonzept und die integrativen Arbeiten, die die Ergebnisse der letzten Monate darstellen, bestmöglich in Bezug gesetzt werden.

TG-auth*

Tabelle 1 Gegenüberstellung der in R 4.5.1 aufgezeigten Lücken und deren Optimierungsmöglichkeiten in TG-auth*

Problem	**Optimierungsmöglichkeit**
Performanzprobleme bei vielen und komplexen schreibenden Zugriffen.	Aufbau von redundanten TG-auth* Instanzen mit Spezialisierung (nur schreibend, lesend und schreibend, nur lesend).

[3] R 4.5.1 „Architekturskizze inklusive Identifikation der Lücken und der Optimierungsmöglichkeiten“, http://www.textgrid.de/fileadmin/user_upload/TextGrid_R451_Architekturskizze.pdf

TG-crud

Tabelle 2 Gegenüberstellung der in R 4.5.1 aufgezeigten Lücken und deren Optimierungsmöglichkeiten in TG-crud

Problem	**Optimierungsmöglichkeit**
Speicherung der TextGrid-Objekte über eine JavaGAT-Implementation im Speicherbackend. JavaGAT wird nicht weiter gepflegt und stellt damit ein potentielles Risiko im Bezug auf die Nachhaltigkeit von TextGrid dar.	Wechsel der Speicherschnittstelle unter Berücksichtigung der DARIAH Storage-API.

TG-search

Tabelle 3 Gegenüberstellung der in R 4.5.1 aufgezeigten Lücken und deren Optimierungsmöglichkeiten in TG-search

Problem	**Optimierungsmöglichkeit**
Hohe Ausfallrate der eXist Datenbank	Umstellung auf eine ElasticSearch Indexdatenbank
Geringe Performanz bei der Suche	Umstellung auf eine ElasticSearch Indexdatenbank

5. Hochverfügbarkeitskonzept

Zur Steigerung von Performanz und Stabilität sowie um den steigenden Nutzerzahlen gerecht zu werden wurde ein Hochverfügbarkeitskonzept für TextGrid ausgearbeitet. Bei der Erstellung des Konzepts am 24. Oktober 2013 in Göttingen wurden alle Komponenten der TextGrid Architektur (vgl. R 4.5.1) auf ihren Einfluss auf die Performanz und Stabilität untersucht und Empfehlungen für das weitere Vorgehen gegeben.

Die Untersuchung hat für die einzelnen Komponenten der TextGrid Middelware folgende ergeben:

- Proxy (HTTP / HTTPS): nginx ist prinzipiell auch als Load-Balancer auf n Instanzen der Middleware nutzbar. Dies sollte getestet werden.

- TG-crud: Der Dienst kann parallelisiert laufen.

- TG-noid: Dieser Dienst erzeugt TextGrid URIs und verwaltet deren Locking, dabei nutzt er eine BerkeleyDB. Es ist davon auszugehen, dass er nicht per se clusterfähig ist. Daher ist zu untersuchen, ob und falls ja, wie dies zu erreichen ist.

- TG-search: Diese Komponente von TextGrid arbeitet stateless und sollte gut zu clustern sein. Zudem greift TG-search nur lesend auf Sesame, TG-auth und ElasticSearch zu. Das Gesamtsystem sollte trotzdem getestet werden.

- TG-auth: Die Komponente ist in PHP realisiert und nicht zustandsbehaftet. Ihre Funktion sollte mit geclustertem LDAP getestet werden.

- Datenbanken:

    - ElasticSearch (Indexdatenbank): Diese Suchmaschine ist clusterfähig ausgelegt. Hier gilt es primär, die Konfiguration zu planen und zusätzliche Knoten für die HA-Lösung bereit zu stellen.

    - Sesame (RDF-Datenbank): Da diese Datenbank vermutlich nicht clusterfähig ist, muss a) untersucht werden, ob es andere Installationen gibt, die dieses doch realisiert haben und b) sollten andere Lösungen wie Virtuoso oder Fuseki betrachtet werden.

    - LDAP für User mit hdb-backend: Potentiell wird dieser LDAP durch die DARIAH Lösung ersetzt.

    - LDAP für RBAC mit hdb-backend: LDAP-Clustering (Multi-Master) ist möglich und sollte entsprechend konfiguriert und getestet werden.

- TextGrid Identity Provider (IdP): Clustering ist mit Shibboleth möglich, allerdings sollte auf neue Version Shibboleth 2.3 gewartet werden, da dort Clustering eingebaut ist.

- TG-publish: Diese Komponente ist zustandsbehaftet (z.B. UUID für Statusabfragen, etc.) und daher in der aktuellen Implementierung nicht clusterfähig. Daher wird zunächst nur eine Instanz von TG-publish betrieben und entsprechende Anfragen weitergeleitet. Langfristig sollte die Clusterfähigkeit in die Weiterentwicklung einfließen.

- Monitoring: Umsetzung der Monitoringinfrastruktur ist in Zusammenarbeit mit DARIAH weiter zu entwickeln.

- Externe Services: Hier sind keine Probleme durch den Proxy zu erwarten.

- Aggregator: Die Komponente ist stateless, daher sind keine Probleme zu erwarten.

6. TextGrid Infrastruktur

Ziel der aktuellen Arbeiten in Arbeitspaket 4.5 ist, neben der Weiterentwicklung einzelner Komponenten der TextGrid Middleware, die Umsetzung des Hochverfügbarkeitskonzeptes. Dazu wird in diesem Abschnitt erst einmal der aktuelle Status beschrieben und dann die Planung für die Umsetzung des Hochverfügbarkeitskonzeptes.

6.1. Status Quo

Das StorNext-Speichersystem der GWDG bildet zusammen mit drei physikalischen Rechnern die gegenwärtige Repository-Infrastruktur von TextGrid. Auf den physikalischen Rechnern übernehmen virtuelle Rechner den Betrieb von TextGrid. In Abbildung 1 sind die physikalischen Rechner als blaue Kästen und die virtuellen Rechner als weiße Kästen dargestellt. Zusätzlich wurden zwei weitere virtuelle Rechner in der Compute-Cloud der GWDG[4] eingerichtet, welche als Testsysteme für Weiterentwicklungen genutzt werden.

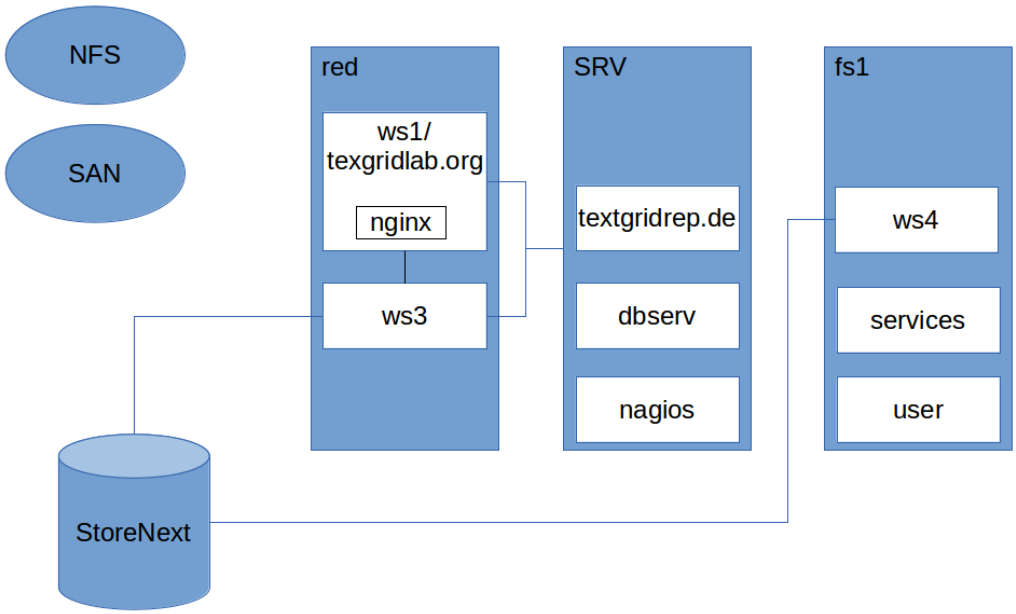

Abbildung 1 – Die aktuelle TextGrid Infrastruktur

6.2. Umsetzung des Hochverfügbarkeitskonzeptes

Das Hochverfügbarkeitskonzept sieht vor, für den Produktivbetrieb zwei hochperformante virtuelle Maschinen aus dem VMware VSphere-Cluster der GWDG einzurichten. Dieses ist

[4] http://www.gwdg.de/index.php?id=2684

über zwei Standorte verteilt, über ein redundantes Netzwerk angebunden und erhält seine Daten von einem synchron gespiegeltes NetApp Metrocluster. Um die Betriebslast adäquat auf beide Maschinen zu verteilen, soll ein Load-Balancer eingerichtet werden. Technologisch soll dies durch die Software nginx realisiert werden, oder, falls es sich als sinnvoll erweist, durch einen Hardware-Load-Balancer.

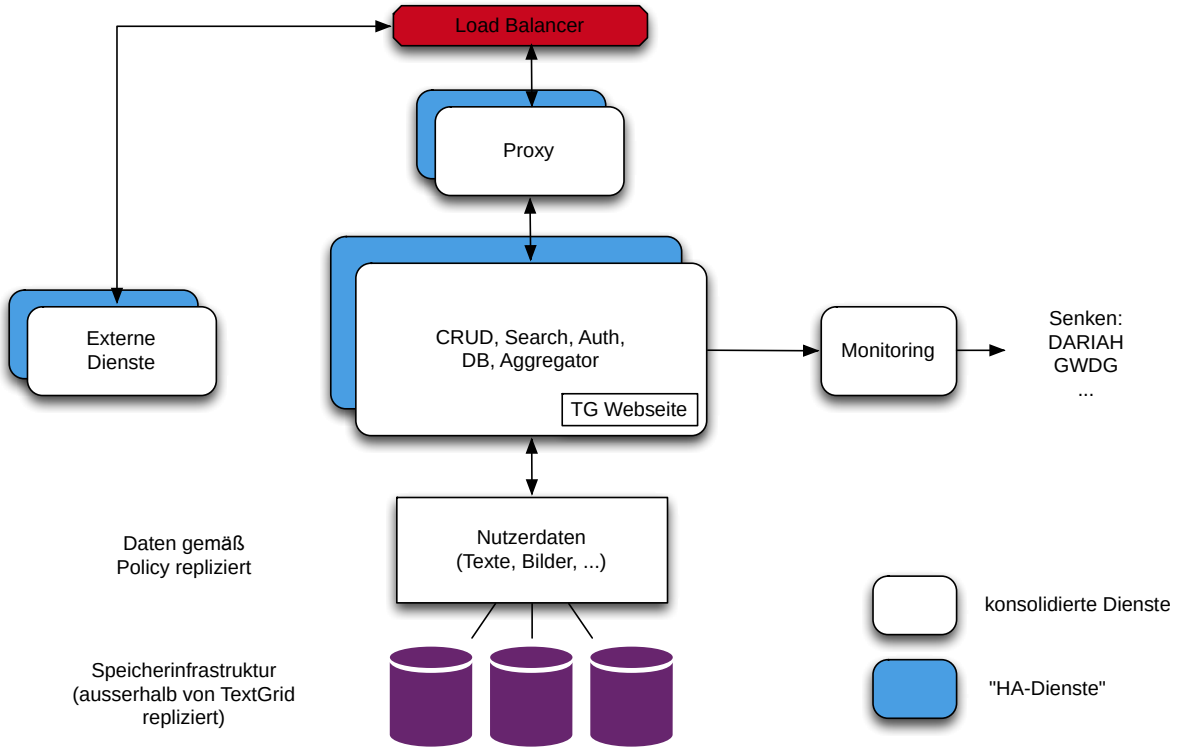

Abbildung 2 – Das Hochverfügbarkeitskonzept für TextGrid

Im Anschluss an die Untersuchung der einzelnen Komponenten von TextGrid (siehe hierzu Abschnitt 5) und nach weiteren Evaluierungen wurden Strategien entwickelt, die dazu beitragen eine hochverfügbare Gesamtarchitektur zu erstellen und zu etablieren. Bezüglich der Parallelisierung der Middleware-Komponenten können daher folgende Aussagen festgehalten werden:

- TG-crud ist grundsätzlich problemlos parallelisierbar, allerdings muss die Möglichkeit der Parallelisierung von TG-noid in diesem Zusammenhang genauer untersucht werden, bzw. eventuell eine Neuimplementierung ins Auge gefasst werden.

- TG-search ist nach den durchgeführten Änderungen im Backend (Ersetzung von eXist-DB durch ElasticSearch) problemlos parallelisierbar.

- TG-auth ist ebenfalls problemlos parallelisierbar.

- TG-publish wird noch untersucht werden müssen, wie es optimal in eine redundante Middleware-Architektur integriert werden kann.

Bezüglich der Parallelisierung der eingesetzten Datenbanken stellt sich die Lage wie folgt dar:

- ElasticSearch ist mit seiner einfachen Parallelisierbarkeit problemlos,
- Sesame wird tatsächlich relativ schwierig zu parallelisieren sein. Deswegen werden andere RDF-Datenbanken auf Parallelisierbarkeit evaluiert, um Sesame evtl. abzulösen.
- OpenLDAP besitzt ein stabiles Replikationsprotokoll, wodurch die Parallelisierbarkeit problemlos gegeben ist.

Das Hochverfügbarkeitskonzept ist in **Fehler! Verweisquelle konnte nicht gefunden werden.** schematisch dargestellt. Über einen Load-Balancer können Anfragen auf beliebig viele Middleware-Proxy-Instanzen verteilt werden. Die Proxies leiten die Anfragen dann an die jeweilige Instanz der Middleware weiter. Der Storage ist wiederum unabhängig von den Middleware-Instanzen hochredundant verfügbar. Alle Komponenten dieser Architektur werden über das DARIAH-Monitoring-System überwacht, sodass beim Ausfall einer Instanz schnell reagiert werden kann. Durch die Redundanz ist der ungestörte Betrieb auch bei Ausfall von Instanzen gesichert.

Bei dem Speichersystem soll StorNext durch die DARIAH[5]-Storage-Lösung abgelöst werden (siehe Abbildung 3). Der DARIAH-Storage wird aus einem Verbund von den Rechenzentren GWDG (Göttingen), RZG (München), KIT (Karlsruhe) und Jülich gebildet. Die Daten in diesem Verbund werden anhand festgelegter Regeln repliziert. So sind die Daten bei einem Ausfall des Speichersystems eines Rechenzentrums weiterhin erreichbar bzw. zumindest Wiederherstellbar aus diesen Replikaten. Dies erforderte eine Erweiterung der TG-crud Komponente, damit die DARIAH-API bedient werden kann, über die Daten in den DARIAH-Storage gelangen. Die Erweiterung von TG-crud durch eine DARIAH-API-Storage-Interface-Implementierung ist bereits durchgeführt, wird in den Testsystemen eingerichtet und dort getestet und für einen Produktivbetrieb fertiggestellt.

[5] https://de.dariah.eu/

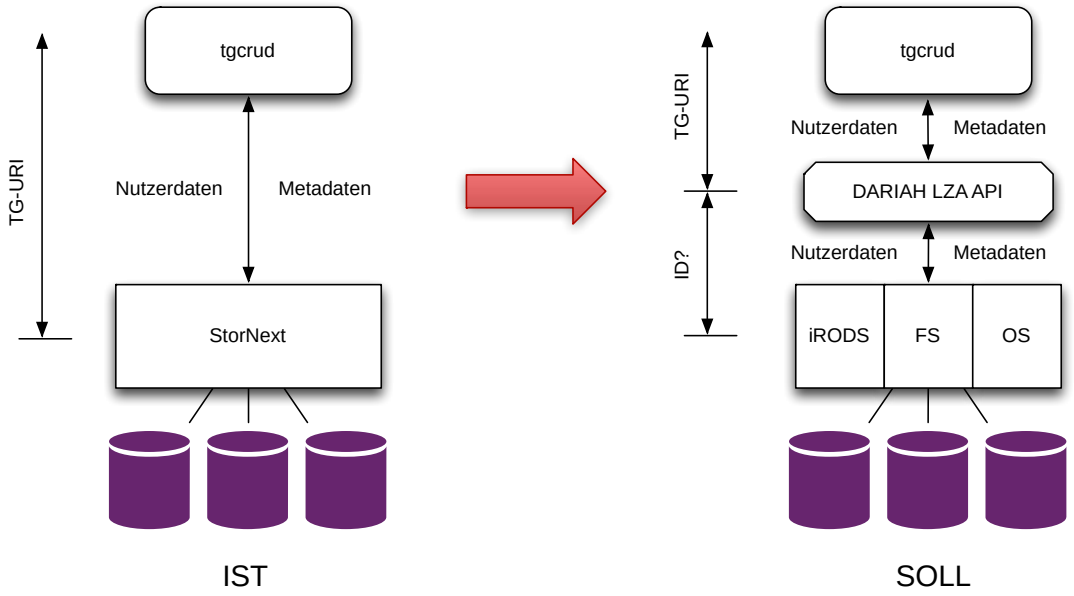

Abbildung 3 – Anpassung des Storage Backends für TextGrid

6.3. Roadmap und Risikoanalyse

In einem ersten Schritt wird eine produktive virtuelle Maschine im VMware vSphere-Cluster aufgebaut. Alle für den Betrieb essentiellen TextGrid-Middleware-Dienste werden auf diesem Server eingerichtet.

In Schritt zwei werden die Maschinen aus dem VMware vSphere-Cluster auf zwei Maschinen aus der Compute Cloud der GWDG geklont, um daraus ein Testsystem zu erstellen. Dieser Schritt soll bis zum ersten Quartal 2014 abgeschlossen sein.

Bis zum zweiten Quartal 2014 wird eine zusätzliche Produktivmaschine aus dem VMware vSphere-Cluster eingerichtet. Diese wird als ein Klon aus der im ersten Schritt erstellten Produktivmaschine eingerichtet. Die Maschinen aus der Compute Cloud GWDG führen ihren Betrieb als Testsysteme der produktiven Maschinen fort.

Innerhalb des dritten Quartals 2014 soll die DARIAH-Storage-API Implementierung und die Migration der Daten aus dem StorNext in das DARIAH-Storage abgeschlossen sein.

Durch die erwähnten Umstellungen innerhalb der TextGrid-Architektur und -Infrastruktur können Risiken entstehen. Bei der Migration der Daten aus dem StorNext in den DARIAH-Storage müssen Datenverluste unbedingt vermieden werden. Weiterhin muss die Anbindung des Storage Backends über die DARIAH-Storage API zunächst ausführlich getestet werden, um Performanzeinbußen im Vergleich zum StorNext-System zu vermeiden da sonst die Leistung des gesamten Systems in Mitleidenschaft gezogen werden könnte. Falls dieser Fall eintritt müssen mögliche Alternativen geprüft werden, die es erlauben, die Integration der DARIAH-Replikationsarchitektur in die TextGrid Umgebung weiter zu testen und zu verbessern, bis diese Lösung für den Produktionsbetrieb vollumfänglich geeignet ist.

Generell gilt dieses Vorgehen für alle neu eingesetzten Technologien. Diese müssen eingehend getestet werden bevor sie in den Produktivbetrieb überführt werden können.

7. Zusammenarbeit mit DARIAH-DE

Das DARIAH-Repositorium ermöglicht es DARIAH-Nutzerinnen und -Nutzern, ihre digitalen Objekte bzw. Datensammlungen nachhaltig und sicher zu archivieren.

Um der Nachhaltigkeit Willen plant TextGrid für den Storage-Bereich, der momentan über lokale File-Adaptoren via JavaGAT und StorNext genutzt wird, zukünftig einen DARIAH-Storage zu Nutzen, der über die DARIAH-Storage-API angesprochen wird. Weiterhin werden über eine OAI-PMH-Schnittstelle (OAI Data Provider) seitens TextGrid die TextGrid-Kollektionen bei der DARIAH Collection Registry angemeldet, wodurch alle TextGrid-Daten über die Generische Suche von DARIAH-DE gefunden werden können.

Der Stand der Arbeiten seitens TextGrid ist bisher soweit gediehen, dass zum einen die Storage-Interface-Implementation (TG-crud) in einer ersten Test-Version fertiggestellt ist, und weiterhin eine OAI-ORE-Schnittstelle zum TextGrid Repository angefangen wurde zu entwickeln – ein OAI-Data-Provider – , so dass die Daten aus dem TextGrid-Repository zukünftig in der DARIAH Collection Registry registriert werden können und sobald die OAI-Schnittstelle existiert, auch per DARIAH Generic Search such- und findbar sind. Für detaillierte Informationen siehe Kapitel □.

8. Anhang: Das DARIAH-DE Repositorium – Architektur und Implementierungsplan

8.1. Einführung

Das DARIAH-Repositorium ermöglicht es DARIAH-Nutzerinnen und -Nutzern, ihre digitalen Objekte bzw. Datensammlungen nachhaltig und sicher zu archivieren. Dies ist komfortabel und intuitiv über ein Web-Interface im Browser möglich, aber auch automatisiert per API. Die Daten werden (zunächst) mit DC-Metadaten ausgezeichnet, sind nach dem Einspielen per PID referenzierbar, recherchierbar mit der Generischen Suche von DARIAH und öffentlich zugänglich.
Für das DARIAH-Repositorium sind einige Services nötig, die zu einem großen Teil von den Diensten des Projekts TextGrid nachgenutzt werden können. Für eine komfortable Nutzung eines DARIAH-Repositoriums ist unter Anderem das Konzept der Kollektion dringend nötig, so dass z.B. die Beziehungen zwischen den digitalen Objekten innerhalb einer solchen Sammlung abgebildet werden können. Weiterhin kann auf eine Metadaten-Eingabe und gerade auch auf eine Metadaten-Validierung nicht verzichtet werden, so dass ein einfacher CRUD-Service – mit nur den Funktionen CREATE, RETRIEVE, UPDATE und DELETE – nicht als ausreichend erachtet wurde.
Der technische Workflow für einen Import in das Repository läuft wie folgt (die Authentifizierung erfolgt über die DARIAH AAI und muss von allen Services bedient werden):

1. **Publish Web-Interface**

Die Nutzerin oder der Nutzer wählt über das DARIAH-publish Web-Interface einzuspielende Daten aus, versieht sie mit DC-Metadaten und beschreibt Abhängigkeiten (z.B. Sub-Kollektionen, o.Ä.). Das Web-Interface (oder ein anderer automatisierter Client) liefert die Objekte samt Metadaten per API an den DARIAH-publish-Service.

2. **DARIAH-publish Service**

Hier werden zum Beispiel die Metadaten validiert, Referenzen auf Objekte innerhalb der einzuspielenden Kollektion von Dateipfaden auf Identifier umgeschrieben, evtl. Metadaten generiert, uswusf. Schließlich werden alle Daten an den DARIAH-crud weitergegeben.

3. **DARIAH-crud Service**

Nun werden die Metadaten und Daten aller Objekte im DARIAH-Storage gespeichert, die Metadaten in die Indexdatenbank eingetragen (für einen späteren Abruf per OAI-PMH), und eine PID erzeugt, die jedes Objekt eindeutig und dauerhaft identifiziert und referenziert.

4. **OAI-PMH Service**

Der OAI-PMH Data Provider kann öffentlich angefragt werden nach neuen Datensätzen des DARIAH-Repositoriums (nach dem OAI-PMH Protokoll) und nutzt für seine Antworten den ElasticSearch-Index. So kann die Generische Suche alle Daten des Repositoriums indexieren und den Nutzerinnen und Nutzern zur Verfügung stellen.

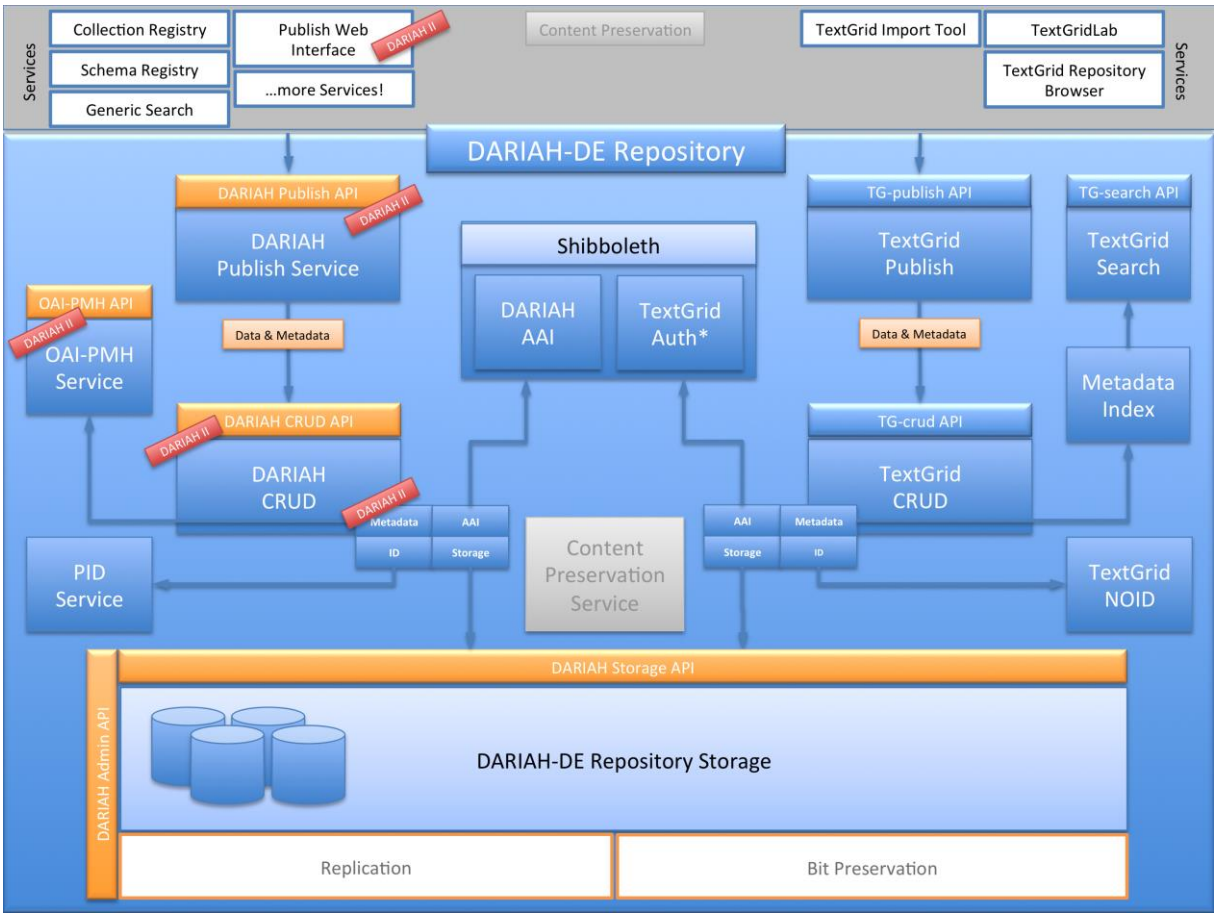

Abbildung 4 – Architektur des DARIAH-Repositoriums

8.2. TG-crud und API

8.2.1. Funktion

TG-crud ist ein Dienst, der für TextGrid die grundlegenden CRUD-Operationen über eine SOAP- und REST-API bietet (CREATE, RETRIEVE, UPDATE und DELETE), und noch einige weitere Methoden bietet. Eine umfassende Dokumentation befindet sich im öffentlichen TextGrid-Wiki[6]. Alle diese Operationen können (je nach Verfügbarkeit) auf dem dynamischen TextGrid-Repository (hier wird mit den Daten gearbeitet, z.B. aus dem TextGridLab heraus[7]) sowie auch auf dem statischen ausgeführt werden. Im statischen TextGridRep liegen alle publizierten Daten – diese sind unveränderbar, PID referenzierbar und weltweit lesbar.
TG-crud wird hauptsächlich vom TG-lab aus angesprochen, kann aber auch, sofern eine RBAC SessionID und ggf. eine TextGrid ProjektID vorhanden ist, von anderen Clients oder per Web-Browser genutzt werden.
Der TG-crud-Code[8] ist bereits weitestgehend so modularisiert, dass es für die Anbindung an eine AAI, an einen Identifier sowie an diverse Storage-Knoten bereits Interfaces gibt, die für die verschiedenen Storage-Backends, Datenbanken bzw. Identifier- und AAI-Anbindungen nahezu beliebig implementiert werden können. Es existieren bereits Implementierungen für:

---

6 TG-crud-Dokumentation im TextGrid-Wiki: https://dev2.dariah.eu/wiki/display/TextGrid/Main+Page
7 Dies ist vergleichbar mit dem DARIAH-Storage.
8 TG-crud Code im SVN: https://textgridlab.org/svn/textgrid/trunk/middleware/tgcrud/

- Storage: JavaGAT und Fedora (Storage), eXist (XML-Datenbank), ElasticSearch (Index-Datenbank), Sesame (RDF-Datenbank)
- AAI: TextGrid-RBAC (Tgextra)
- Identifier: UUID, NOID

8.2.2. TODOs

Die Metadatenverwaltung ist in TG-crud jedoch noch fest an das TextGrid-Metadatenschema[9] gebunden. Für eine Nutzung in DARIAH ist es sinnvoll, diese in ein eigenes Interface auszulagern. Es gäbe dann eine Implementierung für die TextGrid-Metadaten und eine für eine DARIAH-Metadatenverwaltung, die zunächst mit DC Simple arbeiten würde. Weitere Implementierungen sind dann jederzeit denkbar. Folgende Schritte sind notwendig:

- Entwurf des neuen Metadaten-Interfaces.
- Umschreiben des TG-crud Metadaten-Handlings auf die neue Schnittstelle.
- Implementieren des TextGrid-Metdaten-Moduls (für TextGrid).

Weiterhin muss der TG-crud-Code für die neue DARIAH-crud API vorbereitet werden, wahrscheinlich läuft das auf eine Generalisierung der API bzw. eine Modularisierung hinaus:

- Prüfen, inwieweit der TG-crud Code angepasst werden kann, um mit verschiedenen APIs auf diesen zuzugreifen. Es würde evtl. ausreichen, für eine solche Generalisierung nur die REST API zu bedienen, die SOAP API kann TextGrid-spezifisch bleiben.
- Code entsprechend anpassen.

[9] TextGrid-Metadatenschema: http://textgridlab.org/schema/textgrid-metadata_2010.xsd

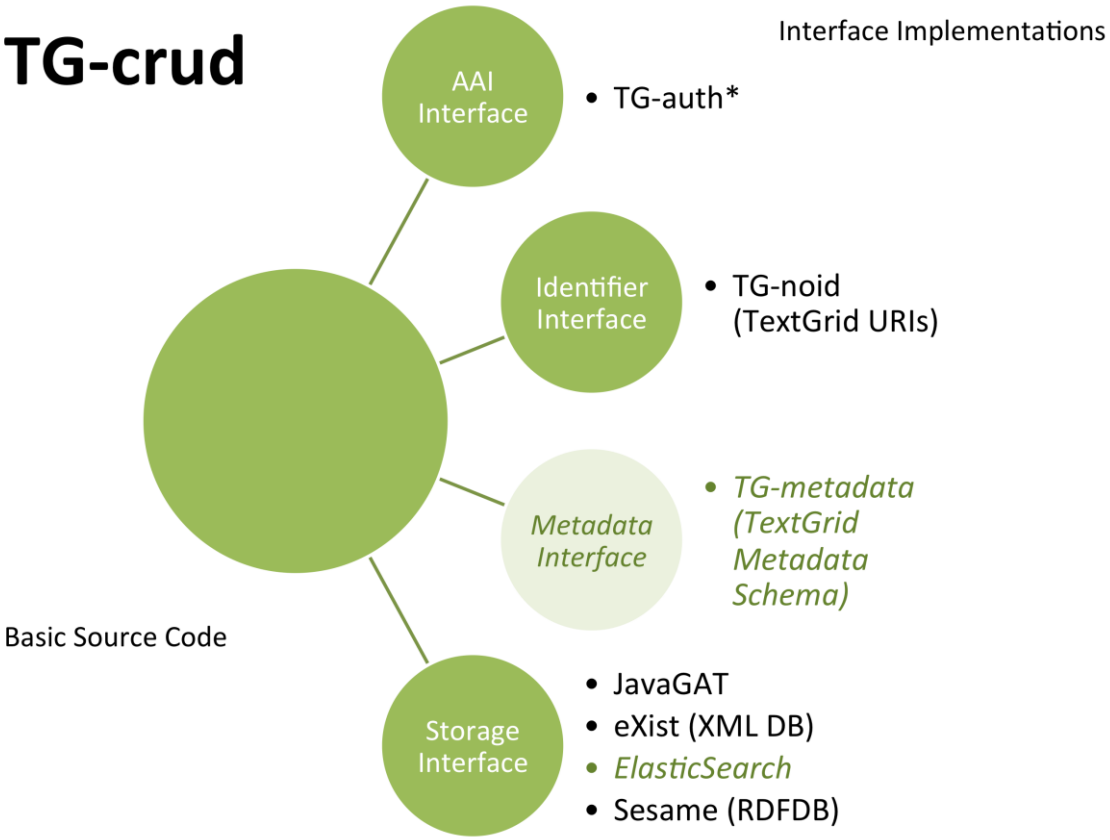

Abbildung 5 – TG-crud Interfaces und Implementierungen

8.2.3. Geschätzte Dauer

2-3 PM

8.3. DARIAH-crud und API

8.3.1. Funktion

Der DARIAH-crud soll grundlegende Operationen auf dem DARIAH-Repositorium ausführen, ebenso wie TG-crud auf dem TextGrid-Repository, also zunächst CREATE, RETRIEVE, UPDATE und DELETE. Im Gegensatz zum TG-rep soll das DARIAH-Repositorium nur als statisches Repositorium aufgebaut werden (für alle dynamischen Dinge ist der DARIAH Storage nutzbar). Welche Funktionalitäten des TG-crud also von der DARIAH-crud API angesprochen werden sollen, hängt zunächst vom DARIAH Publish Service ab.

8.3.2. TODOs

- DARIAH-crud API muss definiert werden.
- Möglicherweise evtl. simpler als die TG-crud API (weil nicht dynamisch).
- Evtl. nur REST (kein SOAP) nutzen, das vereinfacht die Sache (TG-publish API und Service ist derart implementiert), gemeinsamer Codestamm muss dann beide APIs bedienen können (siehe auch TG-crud).

- Storage-Implementierung für DARIAH-Storage muss programmiert werden (evtl. von TextGrid nachnutzen, TG-crud wird per DARIAH Storage API auf den DARIAH-Storage zugreifen).
- Storage-Implementierung für Metadaten-Index implementieren, evtl. kann das ElasticSearch-Modul von TextGrid nachgenutzt werden. Ein solcher Index wird für den OAI-PMH-Service benötigt.
- Aufsetzen einer ElasticSearch-Datenbank und evtl. einer Sesame-Datenbank.
- AAI-Implementierung muss entworfen und programmiert werden für Zugriff auf die DARIAH AAI.
- ID-Implementierung muss programmiert werden: evtl. EPIC PIDs. Da es nur um die Publikation und den Zugriff auf die Daten im DARIAH-Repositorium geht, könnten (auch DARIAH-crud-intern) evtl. EPIC PIDs genutzt werden. Ein Locking der Operationen des DARIAH-crud ist eigentlich nicht nötig, da nur CREATE and RETRIEVE erlaubt sind (?).
- Entwurf der DARIAH-Metadaten-Struktur.
- Implementieren eines DC Metadaten-Moduls (für DARIAH).

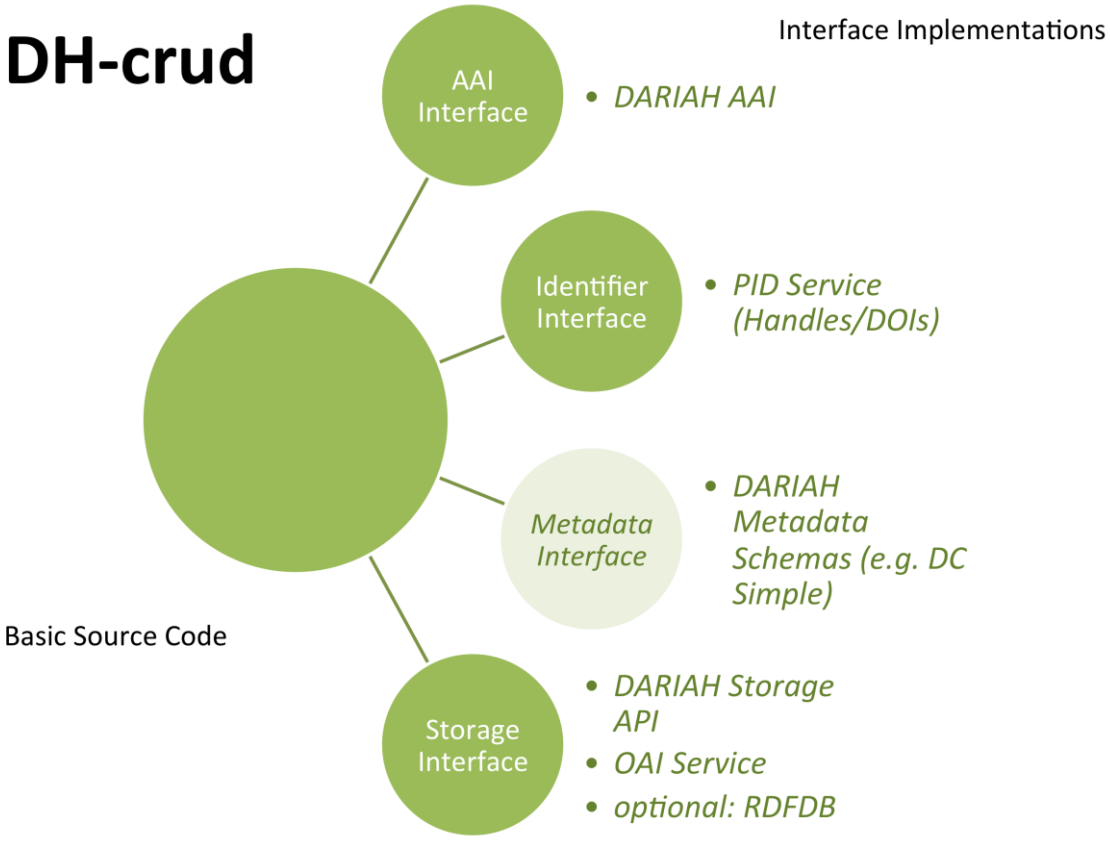

Abbildung 6 – TG-crud Interfaces und Implementierungen

8.3.3. Geschätzte Dauer

5-7 PM

8.4. OAI-PMH Service (Data Provider)

8.4.1. Funktion

Der OAI-PMH-Service des DARIAH-Repositoriums soll die Daten für die Generische Suche von DARIAH liefern, so dass alle im Repositorium verfügbaren Inhalte such- und findbar sind. Dafür muss beim Einspielen der Daten per DARIAH-crud dieser alle relevanten Metadaten (auch Daten wie Volltexte, etc.?) in einen Index schreiben, auf den der OAI-PMH Data Provider schnell zugreifen kann und dementsprechende OAI-Anfragen bedienen kann. Sowohl die Generische Suche wie auch der TextGrid Suchdienst TG-search nutzt eine ElasticSearch-Datenbank als Datenindex und beides ist bereits produktiv im Einsatz.

8.4.2. TODOs

- Aufsetzen einer ElasticSearch-Datenbank.
- Implementieren eines OAI-PMH Data Providers, der seine Daten von einem ElasticSearch-Index bekommt, den der DARIAH-crud schreibt (Nachnutzung eines Java OAI-Frameworks sollte möglich sein, so müssen nur die Daten für die Response geliefert werden, nicht aber das OAI-PMH Protokoll implementiert werden).

8.4.3. Geschätzte Dauer

1-2 PM

8.5. DARIAH Publish Service und API

8.5.1. Funktion

Der DARIAH Publish Service soll Daten von diversen DARIAH-Publish Clients (hauptsächlich dem Publish Web-Interface, weitere – möglicherweise automatisierte – Clients) annehmen und zum Beispiel die Metadaten validiert, Referenzen auf Objekte innerhalb der einzuspielenden Kollektion von Dateipfaden auf Identifier umgeschrieben, evtl. Metadaten generiert, uswusf.

8.5.2. TODOs

- Anforderungen an einen DARIAH-publish Dienst definieren
- Nachnutzung des TG-publish Codes (TG-publish API und TG-publish Service), dieser basiert auf koLibRI-Code
- DARIAH-publish API muss entwickelt und implementiert werden (evtl. Nachnutzung der TG-publish API)
- Neue Module:
- Anbindung an den DARIAH-crud
- PID-Modul (EPIC2), evtl. mit neuem PID-Wrapper (wird dann auch von TextGrid benötigt)
- AAI-Modul benötigt
- ElasticSearch-Modul benötigt(?)

8.5.3. Geschätzte Dauer

6-8 PM

8.6. Publish Web-Interface

8.6.1. Funktion

Über das Publish Web-Interface wählen die Nutzerinnen und Nutzer einzuspielende Daten aus, versehen sie mit DC-Metadaten und beschreiben bestehende Abhängigkeiten (z.B. Unter-Kollektionen, o.Ä.). Das Web-Interface (oder ein anderer automatisierter Client) liefert die Objekte samt Metadaten per API an den DARIAH-publish-Service.

8.6.2. TODOs

- Entwerfen einer GUI, die die TG-publish API anspricht
- Sammeln von Anforderungen an eine solche GUI (evtl. Ableiten von TG-publish GUI im TextGridLab oder Nachnutzen von DAWA und anpassen an die DARIAH-publish API?)
- Metadaten müssen eingegeben werden können (für DARIAH zunächst DC)
- Implementieren für Liveray-Portal und/oder Web-Browser

8.6.3. Geschätzte Dauer

2 PM

8.7. High-Availibility und Parallelisierung

8.7.1. Funktion

Alle DARIAH-Dienste, die zum Kern des Repositories gehören, sollten durch ein HA-Konzept abgesichert sein. Dazu gehören (im Sinne der TextGrid HA) zunächst der DARIAH-crud, der OAI-PMH-Service sowie der Identifier-Service (z.B. der GWDG PID-Service). TG-publish ist zunächst nicht in den Kern des TextGrid HA-Konzept aufgenommen werden, da TextGrid (TG-lab und TG-rep mit allen Kernfunktionen) auch ohne TG-publish laufen. Ein Ausfall von TG-publish legt also nicht den TextGrid-Betrieb lahm. Bei dem DARIAH Publish Service kann das anders sein.

8.7.2. TODOs

Aufbauend auf den Erfahrungen der TextGrid HA kann DARIAH eine HA für die Kernkomponenten des DARIAH Repositoriums installieren. Folgende Dienste müssen auf eine Parallelisierung sowie eine HA geprüft werden:

- Die DARIAH-crud HA kann direkt vom TextGrid-Konzept übernommen weden (falls es Anpassungen des TG-crud für eine HA geben muss, werden diese von TextGrid entwickelt).
- Für weitere DARIAH-Dienste muss ein HA-Konzept entwickelt und umgesetzt werden: OAI-PMH, PID-Service (EPIC2), DARIAH-publish, …

8.7.3. Geschätzte Dauer

Hängt sehr von den Diensten und dem Konzept ab, sollte aber mit den Vorarbeiten von TextGrid nicht viel sein (sofern der Dienst eine Parallelisierung überhaupt erlaubt, evtl. kümmern sich auch die Dienstebetreiber darum).

8.8. Priorisierung bei der Implementation

- Erheben von Nutzerinnen- und Nutzeranforderungen für das Publish Web-Interface
- API-Entwurf (DARIAH-publish und DARIAH-crud)
- Publish Web-Interface-Entwicklung im DARIAH-Portal
- TG-crud erweitern (Metadatenmodularisierung)
- DARIAH-crud implementieren
- OAI-PMH Data Provider implementieren (mit ElasticSearch)
- PID-Service einbinden
- DARIAH-publish Service implementieren

Parallel hierzu wird für das Thema High-Availibility und Parallelisierung ein Konzept mit einer Reihenfolge erstellt (siehe TextGrid HA-Konzept[10]). Dies sollte dann bei der weiteren Priorisierung berücksichtigt werden.

8.9. Fragen

- Wie präsentieren wir die Daten im DARIAH-Repositorium? Macht das die Generische Suche?
- Wie gehen wir mit Kollektionen um (alle Services!)? Bitte Anwendungs-Szenarien aufzeigen!

[10] TextGrid HA-Konzept: https://dev2.dariah.eu/wiki/display/TGINT/High+Availability+Konzept