Roadmap Integration Grid / Repository (R 1.2.1)

Version – 13.12.2010
Work Package – 1
Responsible Partner – SUB Göttingen

TextGrid

Virtual Research Environment for the Humanities

Ein Projekt im D-GRID

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung

Project: TextGrid – Virtual Research Environment for the Humanities
Funded by the German Federal Ministry of Education and Research (BMBF) by
Agreement: 01UG0901A
Project Duration: June 2009 – May 2012

Document Status: Living Document
Distribution: public
Authors:
Andreas Aschenbrenner (SUB)
Jörg-Holger Panzer (SUB)
Wolfgang Pempe (SUB)

Table of Contents

1. Introduction

As one of the Community Grid projects of the first D-Grid call, TextGrid established a Virtual Research Environment for the Humanities in Germany. This VRE is part of a larger infrastructure enabling the collective utilisation and exchange of data, tools and methods – with a strong focus on text-oriented research. The central component of this infrastructure is a grid-based repository ensuring the sustainable availability of and access to research data, the so-called *TextGrid Repository*. This repository has proven itself as a stable and reliable storage device. While currently providing Bitstream Preservation on a very basic level, TextGrid aims for a repository solution that supports the curation and long-term preservation of research data. With the proposal for the second project phase (2009-2012), TextGrid committed itself to implement the long-term storage architecture to be developed by the WissGrid project, another D-Grid project bringing together the five academic communities of the first D-Grid call to develop generic solutions and blueprints for academic grid communities and users. This report gives an overview of the concepts and technical solutions developed by WissGrid – as far as TextGrid is concerned – and describes the technical context and measures for the implementation of these solutions with the TextGrid infrastructure. For a better understanding of the chosen implementation strategy we'll start with an overview of the basic concepts in TextGrid.

Since the interaction with WissGrid and the implementation of the WissGrid solutions is a highly dynamical process, this report is considered to be a **living document**, to be updated (more or less) regularly.

2. TextGrid Basics – Concepts and Infrastructure

2.1. The Logical View

2.1.1. Project and Aggregation

Starting the *TextGrid Lab*, the eclipse-based TextGrid client application, one of the first tools the TextGrid user is confronted with is the omnipresent *Navigator* (figure 1). Offering the same (and some additional) functionality as an ordinary file browser, this tool provides access to the objects in the grid.

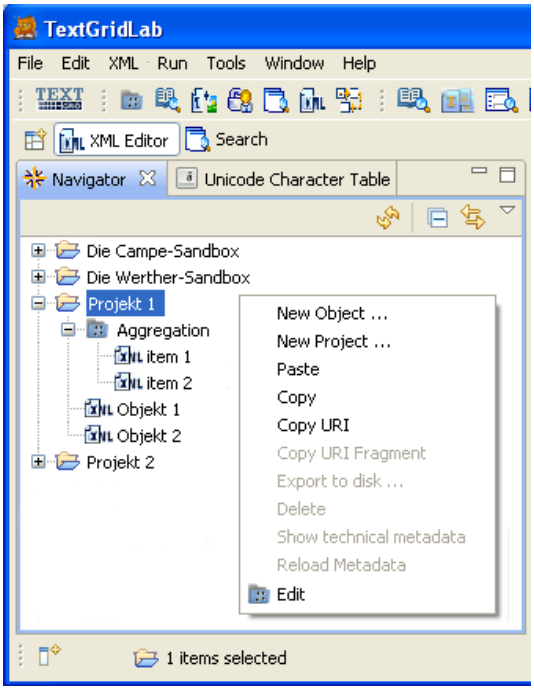

Figure 1: The TextGrid Navigator

The top-level elements of the hierarchical structure displayed by the Navigator are the so-called *Projects*. The *Project* serves as container for the (role-based) rights management – comparable with the *Bucket* in Amazon S3[1] or the *Context* in eSciDoc[2]. Any TextGrid object belongs to a project. TextGrid users can start a new project in order to create new or copy existing objects. The creator of a project can select other TextGrid users, associate them with the project and assign specific roles to them. A *Project* consists of an arbitrary number of objects (*Simple Object*) and *Aggregations*, (virtual) folders, that can contain objects and other *Aggregations* in turn. *Aggregations* are also objects, though specialised ones. Every object is coupled with a set of metadata.

2.1.2. Object Types: Item, Edition, Work and Collections

In order to avoid redundancy, misspellings etc. and to keep the management of bibliographic metadata straightforward and simple, bibliographic metadata is spread on three types of Objects: *Item*, *Edition* and *Work* – inspired by the FRBR model[3]. While – in terms of metadata – *Item* is the most generic object type that is not necessarily a bibliographic object in a narrower sense (could also be an aggregation), *Edition* and *Work* objects must conform to a specific metadata sub-schema. While *Edition* is modelled as an *Aggregation* (with a specific set of metadata), a *Work* object is modelled as an empty *Simple Object* with work-specific metadata. The connection between instances of these three object types is modelled via relations between *Item* and *Edition* (*aggregates*), and *Edition* and *Work* (*isEditionOf*), cf. figure 2:

[1] http://docs.amazonwebservices.com/AmazonS3/latest/dev/index.html?UsingAuthAccess.html
[2] https://www.escidoc.org/JSPWiki/en/ContentModel
[3] http://www.ifla.org/en/publications/functional-requirements-for-bibliographic-records

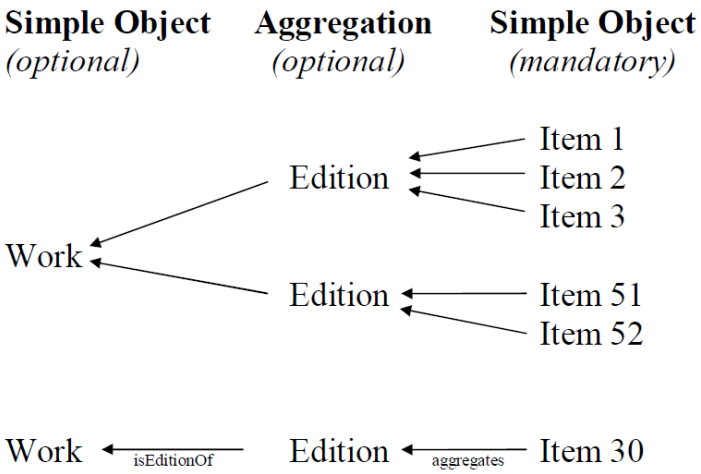

Figure 2: Bibliographic Hierarchy.

Besides *Item*, *Edition* and *Work*, the TextGrid metadata schema[4] provides for a fourth object type, the *Collection*. Not unlike the *Edition* object, a *Collection* is also modelled as an *Aggregation*, but with a more generic, non-bibliographic metadata set. For instance, a *Collection* can be used to aggregate non-bibliographic objects (or mash up bibliographic and non-bibliographic *Items*) and assign them to a common temporal or spatial scope – or even provide a description for this selection.

2.2. The Physical View

The world is flat. Ever was. The same goes for TextGrid. Hierarchies are only built by the application logic of the upper levels of the TextGrid infrastructure, primarily by the *TextGrid Lab*:

- The *Projects* are modelled according to information extracted from the rights management system (OpenRBAC / LDAP)
- *Aggregation Objects* (generic ones as well as *Editions* and *Collections*) have no content except a list of URIs of the "contained" objects

Out of this information the Navigator builds its tree view.

Physically, on (grid) file system level, TextGrid objects consist of file pairs (tuples), one file for the metadata and one for the object itself (resp. the content), such as:

```
textgrid:1234ab.1.meta
textgrid:1234ab.1
```

This brings us to the identifiers used in TextGrid. TextGrid-URIs are NOIDs[5] with a `textgrid:` prefix and an ascending numerical suffix indicating the revision number. A TextGrid object is identified and referenced (metadata and elsewhere) by its URI e.g. – to stick with the example: `textgrid:1234ab.1` for the first, `textgrid:1234ab.2` for the second revision and so on. Per default, the *Navigator* displays only the latest revision of an object. The latest revision of an object can also be referenced by a logical URI without suffix: If `textgrid:1234ab.2` the newest/latest revision of an object, calling or referencing `textgrid:1234ab` will produce the same physical object. This concept is similar to the eSci-

[4] http://www.textgrid.info/schemas/textgrid-metadata_2010.xsd
[5] https://confluence.ucop.edu/display/Curation/NOID

Doc solution[6] – except the underlying terminology (TextGrid: *Revision*, eSciDoc: *Version*) and the fact that creating a new revision in TextGrid is a deliberate action to be performed by the user while in eSciDoc versions are generated automatically with every update of an object.

A possible use case for using logical URIs is to use them as references in a *Collection Object*, so that this *Collection* contains always the latest version of the respective objects. As every *Aggregation*, the content of a *Collection Object* consists only of a list of the URIs of the contained objects – formatted in RDF as a very simple Resource Map according to the OAI-ORE specification[7]:

```
<rdf:RDF xmlns:ore="http://www.openarchives.org/ore/terms/"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description xmlns:tei="http://www.tei-c.org/ns/1.0"
        rdf:about="textgrid:26dt.1">
    <ore:aggregates>textgrid:21sn</ore:aggregates>
    <ore:aggregates>textgrid:21sp</ore:aggregates>
    <ore:aggregates>textgrid:21sq</ore:aggregates>
    <ore:aggregates>textgrid:21sr</ore:aggregates>
    <ore:aggregates>textgrid:21ss</ore:aggregates>
    <ore:aggregates>textgrid:21st</ore:aggregates>
            ...
            ...
    <ore:aggregates>textgrid:21w4</ore:aggregates>
    <ore:aggregates>textgrid:21w5</ore:aggregates>
    <ore:aggregates>textgrid:21w6</ore:aggregates>
  </rdf:Description>
</rdf:RDF>
```

‘External’ Objects

There is also a mechanism to include objects in TextGrid that are stored or ‘hosted’ outside the *TextGrid Rep*. It is possible to perform a metadata-only ingest with metadata provided by or harvested from other repositories. In such a case, TextGrid includes the URL of this object in the (TextGrid-)metadata and creates an otherwise empty object. In this way, TextGrid users are enabled to work with ‘external’ objects for instance by embedding them in *Collections* or perform queries across the metadata of both ‘internal’ and ‘external’ objects.

Physical Type	Metadata Type
Aggregation	Item Edition Collection
(Simple) Object	Item
Ø	Work Item (External Object)

2.3. Search Index and Baseline Encoding

Since performing queries directly on data stored in a potentially distributed and replicated grid environment is neither quick nor straightforward, all relevant information is gathered in a search index consisting of an XML database (metadata, aggregation content, XML-encoded

[6] https://www.escidoc.org/media/docs/ges-versioning-article.pdf
[7] http://www.openarchives.org/ore/

object content) and an RDF triple store. The latter one is fed with relations such as *isDerivedFrom*, *isAlternativeFormatOf*, *hasSchema*, *aggregates*, or *hasAdaptor*. The latter one needs some explanation: To enable structured search and processing capabilities across XML data in the TextGrid Repository, TextGrid developed the so-called Baseline Encoding, a text type-specific encoding which is based on the TEI P5 standard[8]. The transformation of project-specific XML into baseline-XML is performed by a so-called *Adaptor* (i.e. an XSLT stylesheet) with every write and update operations on an object with XML content – provided the *hasAdaptor* relation is set. The baseline instance of an object is kept only in the search index, not in the grid.

2.4. The Three Pillars of TextGrid

The main constituents of the TextGrid middleware are the three utilities TG-auth*, TG-search and TG-crud (figure 3) – implemented as web services.

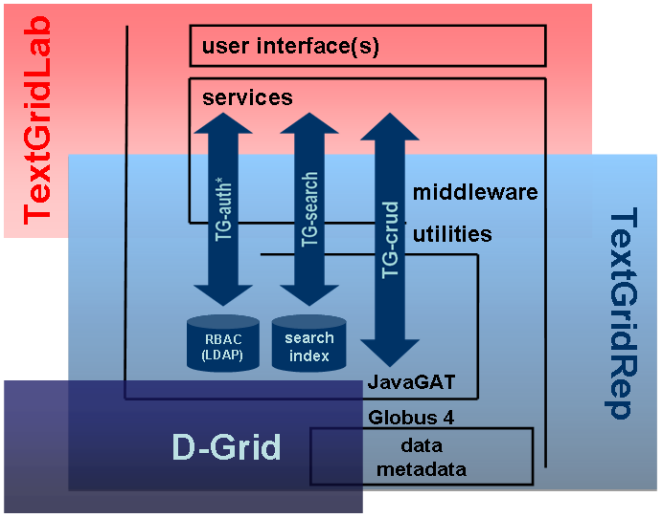

Figure 3: The TextGrid Utilities

TG-auth* covers two aspects. With "N" replacing the asterisk, it provides for autheNtication of users in the TextGrid environment. With "Z", it serves as an authoriZation engine. As already mentioned, for authorization, TextGrid uses a role-based access control solution called OpenRBAC[9] where permissions are stored in an LDAP database. With logging in to TextGrid, a session id is generated, which is passed around between the utilities and services, to be used to check permissions with TG-auth*.

TG-search provides several interfaces for text retrieval and metadata search. The capabilities of searching across XML data and the semantic (RDF) index are explained in the section before.

TG-crud is a web service to **c**reate, **r**etrieve, **u**pdate and **d**elete TextGrid objects. It is the interface to storing information to the grid environment, the search indices (see above) and the role based access control system (RBAC) using TG-auth*. TG-crud also checks access permissions and ensures that the TextGrid repository stays consistent. Furthermore, it uses the

[8] http://www.textgrid.de/fileadmin/TextGrid/reports/baseline-all-en.pdf
[9] http://www.openrbac.de/en_startup.xml

Adaptor Manager to convert XML documents into the TextGrid baseline encoding, which also are stored in the XML database for efficient structural search. Additionally, the Adaptor Manager is responsible for extracting relation information from metadata, TEI files and the generated baseline-encoded files (such as contained links to other TextGrid objects and XML schema references) and storing them to the RDF triple store. As may be easily comprehensible, TG-crud bundles most of the application logic of TextGrid and its middleware.

A detailed description of these utilities is given in Report 3.5[10] (first project phase).

2.5. Rights Management Issues and Publication

The role-based rights management and the user's ability to invite others to join her *Project* according to pre- or individually defined roles is one of the greatest strengths (i.e. enabling collaboration) of TextGrid – but is also the source of some sophisticated application logic slowing down things slightly. TG-search, for instance, has to ask for every match of a user's query whether this user is allowed to "find" this object. For this (and other) reasons, there's another, freely accessible search index where published objects are fed in. Since published objects are 'frozen'. Accordingly, the now immutable object is being moved from the dynamic (frequent I/O operations) to a static grid storage instance. Being published, an object is associated with a persistent identifier (Handle[11]), and its metadata and format is being validated. Furthermore, this object must be part of an *Edition* or *Collection* to assure a certain quality level of the metadata.

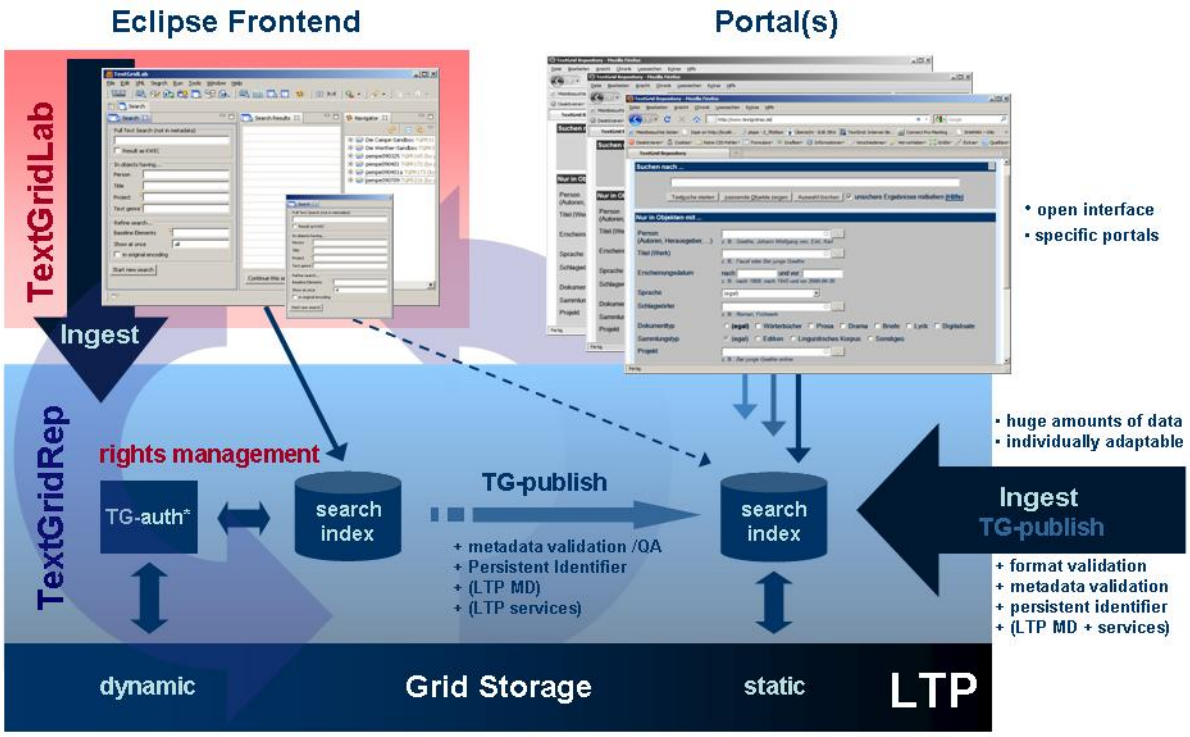

Figure 4: TextGrid infrastructure and publishing workflow

[10] http://www.textgrid.de/fileadmin/TextGrid/reports/R3_5-manual-tools.pdf
[11] http://www.handle.net

3. WissGrid and the Grid-Repository as Part of the LTP-Architecture for D-Grid

3.1. General Concept

One of the main objectives of the WissGrid project is to develop a general concept for the long-term archiving of primary research data for grid communities. **WissGrid** works together with partners in **D-Grid** and in the digital preservation community to offer Bitstream Preservation on grid resources, and to support **user communities** in content preservation and data curation through tools and policy guidance (cf. Figure 5). Its architecture is designed to be open, such that services can accommodate existing data management environments and evolve over time.

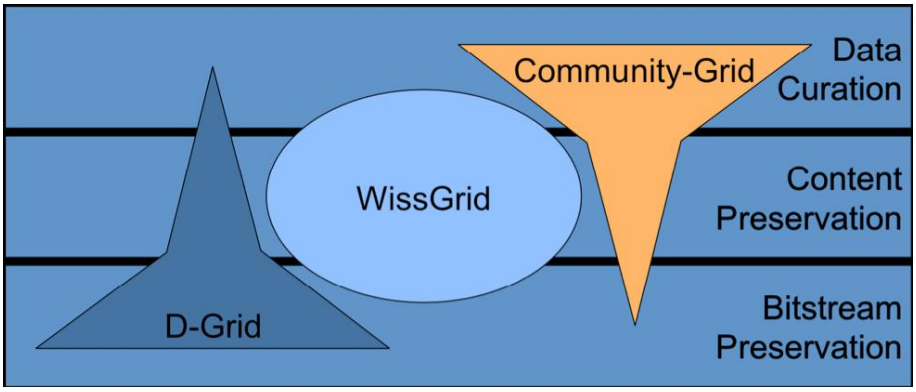

Figure 5: Data preservation in the (D-)Grid communities.[12]

Concerning digital curation, WissGrid distinguishes three levels (see figures 5 and 6) well recognised in the preservation community and following the abstraction levels of digital objects suggested by Thibodeau[13]. These levels build upon each other and may influence and support each other. The technical and organisational methods mentioned in the following are exemplary, they may change depending on the context and evolve over time.

Bitstream Preservation - the integrity of each bit: Bitstream Preservation involves the technical and organisational infrastructure for monitoring the physical stability of data, and moving data to fresh and up-to-date carriers. Components of Bitstream Preservation include durability of the hardware carriers and readers; their replication and recurrent integrity checks.
Content Preservation - processable data units: Citability and accessibility of data items goes beyond the mere availability of the item's bits and touches e.g. upon the software with which it was created and its format. Components of Content Preservation include persistent identification, technology watch, as well as preservation services such as format conversion, format validation, or emulation.
Data Curation - transporting meaning: Data Curation views the object in its intellectual context and over its whole life cycle. It aims to preserve the meaning of the digital object to foster comprehension and re-usability in the future when the original context (e.g. the creators, organisational environment) has disappeared. Components of Data Curation include data and metadata modelling, appraisal of the object's value, integration of the data item in usage environments, object versioning, access control, and others. Examples include the life cycle of

[12] WissGrid Delivearbe 3.1 „Generische Langzeitarchivierungsarchitektur für D-Grid“, figure 2, p. 8 (http://www.wissgrid.de/publikationen/deliverables/wp3/WissGrid-D3.1-LZA-Architektur-v1.1.pdf)
[13] Kenneth Thibodeau: Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years. CLIR Report, 2002. http://www.clir.org/pubs/reports/pub107/thibodeau.html

instrumental data, as it is captured, cleaned, filtered, and processed iteratively; or the enrichment of ancient texts with dictionaries and other contextual data from their time of creation.

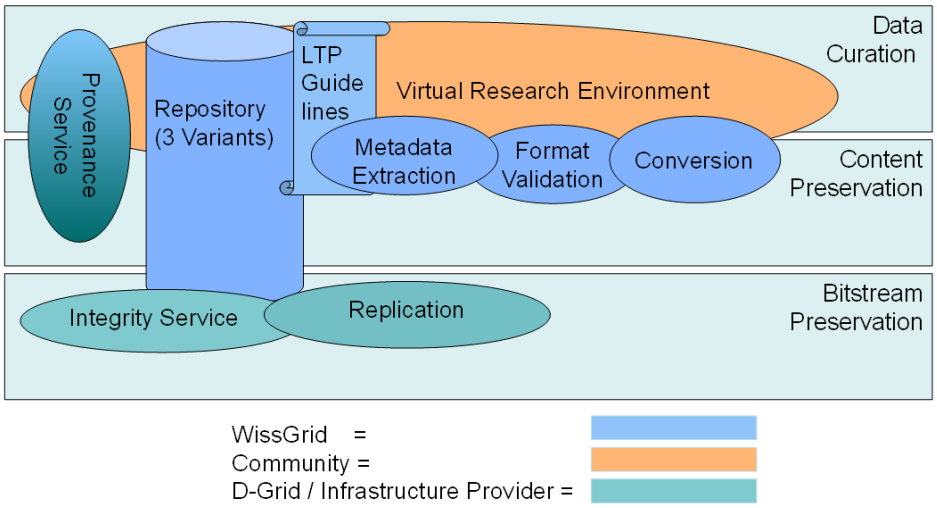

Figure 6: Curation Levels and Preservation Services.[14]

3.2. Profile for Interactive Research Environments

As part of its objective to support content preservation and data curation for research data, WissGrid analysed different integration patterns of e.g. grid technologies and preservation services. It also developed approaches for the integration of digital repositories into grid environments:

A Repositories as data backends for grid processing
B Data grids as repository storage for interactive research environments
C Virtualisation of digital objects in federated repositories

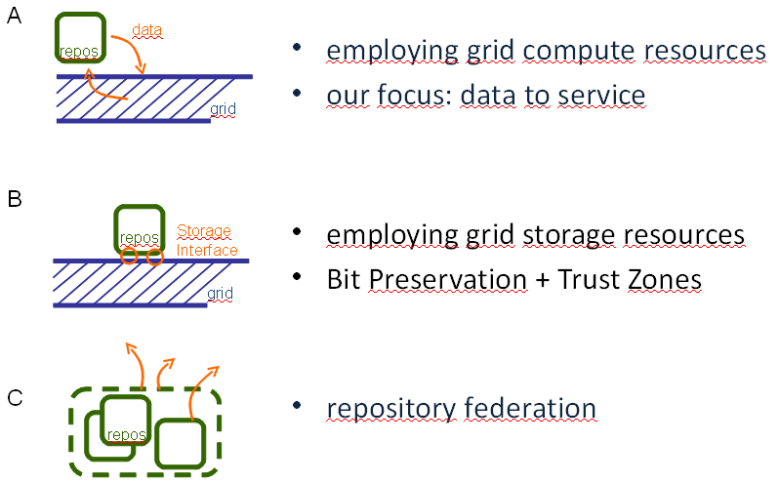

Figure 7: Grid / Repository Integration Patterns[15]

[14] WissGrid Deliverabe 3.5.2 „WissGrid-Spezifikation: Grid-Repository“, figure 3, p. 13 (http://www.wissgrid.de/publikationen/deliverables/wp3/WissGrid-D3.5.2-grid-repository-spezifikation.pdf)
[15] Andreas Aschenbrenner, „Elements of a Curation Grid“, Presentation at ISGC2010, http://event.twgrid.org/isgc2010/slides/isgc2010-aaschen_CurationGrid.pdf (slightly modified)

The application profile 'B' was developed for community grids producing their data in virtual research environments, where users work collaboratively and usually web-based on the data, and want their data to be long-term and trustworthily archived and curated in the grid. This profile provides maximum support for individually adapted data and metadata models, and the modelling of research processes and data life-cycles.

Specific requirements:

- **Scalability** – repository and preservation/curation services must be able to handle both huge amounts of small objects, for instance, in the case of TextGrid, the approx. 140.000 entries of the Campe dictionary[16] as XML (TEI) files, and (usually) smaller amounts of large files, e.g. scans. To avoid unnecessary delays with the transfer of large files, it would be useful to implement the preservation (or other) services as computing services on the respective (data) grid nodes.

- **Seamless integration in existing, mostly web-based research environments** – today, interactive and collaborative research environments usually make extensive use of HTTP-/REST-based technologies. In terms of user-orientation, these technologies must be integrated.

- **Flexible modelling of metadata and object models** – users must be able to define specific metadata models and relations between objects. During the data / research life-cycle, those information may be enriched and steadily augmented (technical metadata, annotations, relations – Linked Data).

The development of web-based repositories with open and flexible (user) interfaces, supporting the handling of complex objects (like DSpace[17], Fedora[18] or EPrints[19]) is driven forward by an active and international community in the context of the Open Repositories. Since the combination of web-based repositories with data grids is already well-tried (e.g. DSpace with SRB) and there are several projects and initiatives working on interfacing Fedora with iRODS[20], WissGrid is aiming for this solution. Actually, an Akubra[21] module for iRODS was developed.

According to this profile, iRODS is used as storage infrastructure and Fedora as management layer for metadata and object models, providing several interfaces for HTTP-/REST-based access to the repository. This (front-end) architecture offers a broad range of possible community-specific extensions, for instance building upon frameworks like Muradora[22] or eSciDoc[23].

[16] Wörterbuch der Deutschen Sprache. Veranstaltet und herausgegeben von Joachim Heinrich Campe (Braunschweig 1807-1813) – 6 Volumes.
[17] http://www.dspace.org
[18] http://fedora-commons.org
[19] http://www.eprints.org
[20] http://www.irods.org
[21] https://wiki.duraspace.org/display/AKUBRA/Akubra+Project
[22] http://www.muradora.org
[23] http://www.escidoc.org/

For more information see the WissGrid deliverables available at http://www.wissgrid.de/publikationen/deliverables/wp3.html.

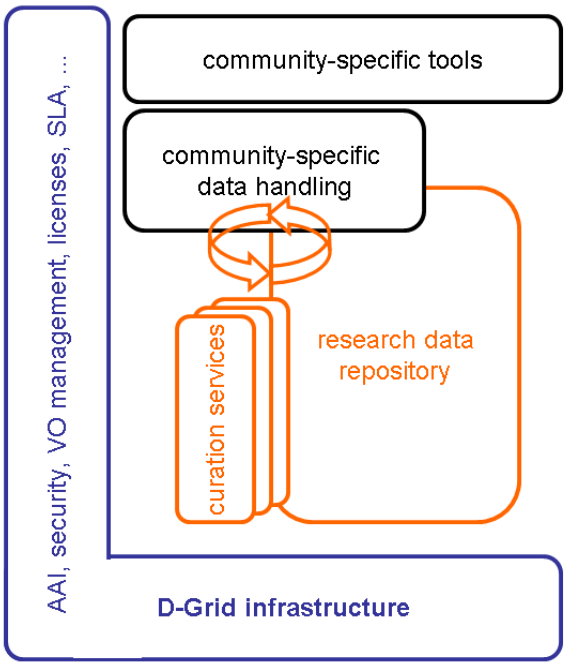

Figure 8: Grid-based Research Data Repository[24]

3.3. Integration with TextGrid

The reconfiguration of the repository infrastructure has to take care that the logical view on TextGrid and the object model as described above (section 2) remains the same and that the interoperability between the TextGrid Lab and the TextGrid is upheld. The utilities (TG-auth*, TG-crud, TG-search) should at least remain as interfaces. For reasons to be explained later, the only utility suitable to be re-implemented by means of Fedora is TG-crud.

Since the Fedora repository is considered as front-end to the iRODS data grid while WissGrid provides the connectivity layer between both components, TextGrid in the first place has to care for the migration of its middleware to a more or less Fedora-based architecture. The only exception where TextGrid has to talk directly with the iRODS back-end is the bulk ingest.

As currently (autumn 2010) the focus of TextGrid is on stabilising the infrastructure for going productive in spring 2011 (TextGrid v1.0), and there is still some technical work to be done in WissGrid, the following sections present only some basic considerations and more or less preliminary sketches. They will become more detailed with further revisions of this report.

[24] Andreas Aschenbrenner, „Elements of a Curation Grid“, Presentation at ISGC2010, http://event.twgrid.org/isgc2010/slides/isgc2010-aaschen_CurationGrid.pdf

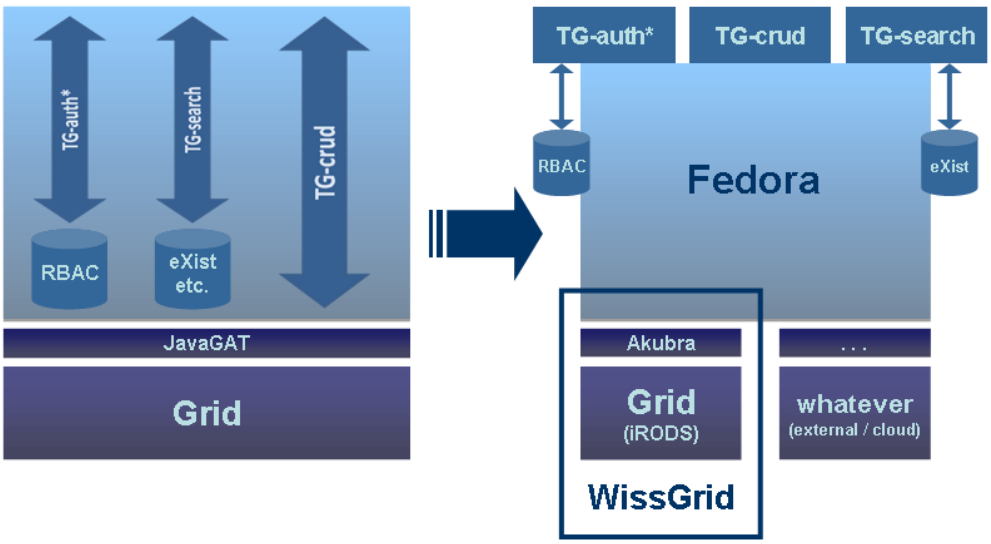

Figure 9: Migration to Fedora

3.3.1. Fedora

The first practical steps in this direction have already been taken together with TEXTvre, a sister project of TextGrid that is considered to "embed the VRE within the day-to-day research practices at the institution [i.e. King's College], and […] integrate it fully with institutional repository and data management infrastructures"[25] – where these infrastructures are Fedora-based. For this reason, in early 2010 both projects developed a first solution for the integration of a Fedora repository with the TextGrid infrastructure. Not being a particularly deep one, this solution (already in use at King's College London) can be used as starting point for further developments: Instead of writing to the Grid via JavaGAT, the TEXTvre instance (cf. figure 10) makes use of the Fedora API-M. The next step will be to implement the TextGrid object model(s) by means of the Fedora Digital Object Model and the underlying Content Model Architecture[26].

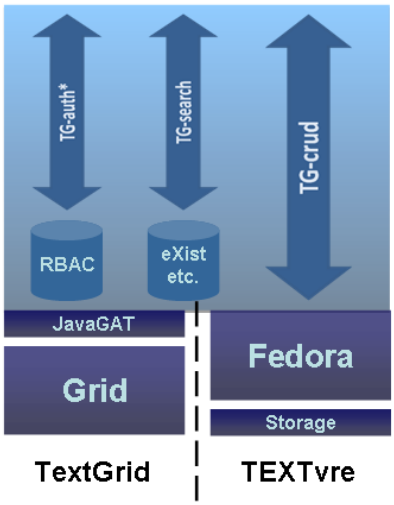

Figure 10: TextGrid and TEXTvre

[25] http://textvre.cerch.kcl.ac.uk
[26] https://wiki.duraspace.org/display/FCR30/Content+Model+Architecture

Object Modelling

Since the logical object, respectively the logical view on TextGrid objects (see section 2.1 above), is mostly modelled by the TextGrid Lab, the modelling in Fedora has to consider primarily the physical objects (cf. section 2.2):

Physical Type	Metadata Type
Aggregation	Item Edition Collection
(Simple) Object	Item
Ø	Work Item (External Object)

Modelling objects with Fedora means thinking in Datastreams. Fedora is a file-based data management system implementing and managing compound objects as Fedora Digital Objects (FDO). The structure of a FDO outlines as follows:

- **Digital Object Identifier** – a persistent, *repository-internal* identifier (PID) for the object, not to be confused with global referencing systems like Handle, URN, DOI etc. Since the NOID service deployed in TextGrid to generate TextGrid URIs (see above) can also be set up as a name resolver, the TextGrid middleware should be able to handle both types of identifiers (need to be verified, though).
- **System Properties** (Object State, the Content Model it subscribes, and other properties that are necessary to manage and track the object in the repository), and
- several **Datastreams**

Each object contains up to four reserved and an arbitrary number of user-defined Datastreams (cf. figure 11):

- RELS-EXT – describes relationships to other FDOs
- RELS-INT – Internal relationships from digital object Datastreams (optional)
- DC (Dublin Core) – metadata about the object
- AUDIT (Audit Trail) – records all changes made to the object (system-generated)
- N Datastreams – the actual content items. For the different types of these Datastreams as *Internal XML Content*, *Managed Content*, *Externally Referenced Content*, and *Redirect Referenced Content* see the official Fedora documentation[27]. If not otherwise mentioned, we usually refer to *Managed Content*: "The content is stored in the repository and the digital object XML maintains an internal identifier that can be used to retrieve the content from storage"[28].

[27] https://wiki.duraspace.org/display/FCR30/Fedora+Digital+Object+Model#FedoraDigitalObjectModel-Datastreams
[28] ibid.

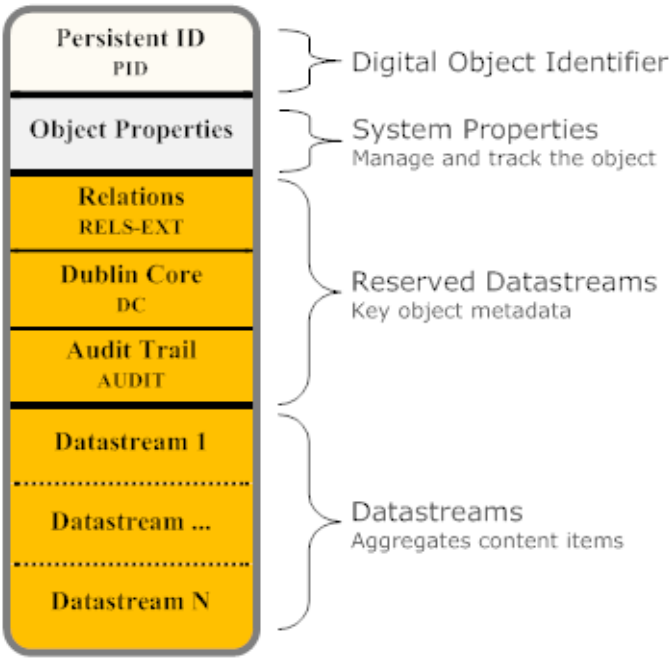

Figure 11: The Fedora Digital Object Model[29]

All this information constituting a FDO is kept in an XML document (FOXML) that is physically stored together with the related Datastreams (versions) in an iRODS data grid instance, approximately as:

```
/objectStore/FOXML
                ...
/datastreamStore/object-id/DS1/DS1.0
                                DS1.1
                                ...
```

The relations specified in the RELS-Datastreams are automatically indexed and stored in a triplestore[30] (Mulgara per default) – quite the same as in the current implementation of the TextGrid middleware (cf. section 2.3).

Furthermore, Fedora allows for the definition of so-called Content Model Objects, specifying characteristics (structural, behavioral and semantic information) for the definition of "classes" of FDOs[31].

Given these instruments, the modelling of TextGrid objects in Fedora is supported in many ways. According to the (physical) object types in TextGrid, three (single-object) Content Models have to be specified. Without going into details (which is subject to future revisions of this report) we only will have a look at the arrangement of Datastreams (DS).

[29] https://wiki.duraspace.org/display/FCR30/Fedora+Digital+Object+Model
[30] https://wiki.duraspace.org/display/FCR30/Digital+Object+Relationships
[31] https://wiki.duraspace.org/display/FCR30/Content+Model+Architecture

Simple Object Type

There is no more need for managing file pairs (at least at Fedora repository level) as described above (section 2.2) since content and metadata can be managed as two Datastreams of one single FDO (figure 12). Possibly a task for the Adaptor manager is the mapping of TextGrid metadata to the required DC Datastream. Since the latter contains only a minimal set required by the repository system, the TextGrid infrastructure must be aware that the really relevant metadata is stored in a particular Datastream distinct from the DC DS. The TextGrid-specific relations can be stored within the RELS-EXT Datastream.

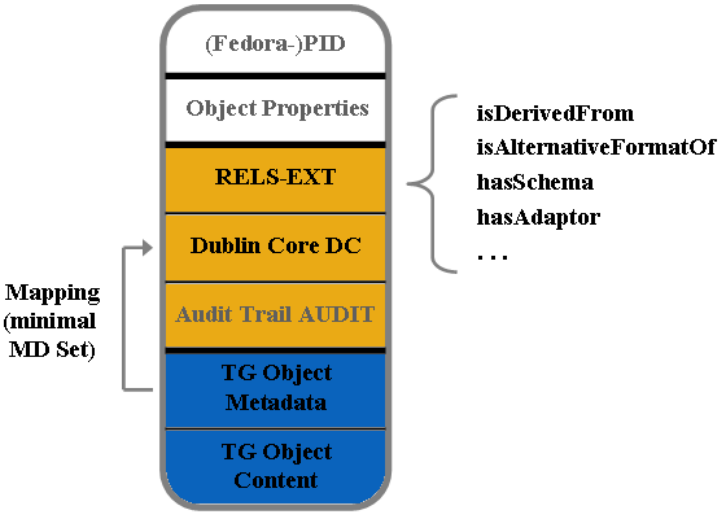

Figure 12: TextGrid Simple Object Type

Aggregation Type

Aggregations can be implemented in (at least) two ways: Since the current implementation keeps the information about the aggregated objects in the object's content (see example above, section 2.2), this pattern can be transferred 1:1 as a corresponding DS (figure 13, left). The other possibility would be to extract the `ore:aggregates` terms and insert them as relations (RDF triples) into the RELS-EXT DC (figure 13, right). Since the technical effort for the implementation of both variants seems to be almost equal, the eventual implementation must take into account the performance of the corresponding search (XMLDB vs. triplestore).

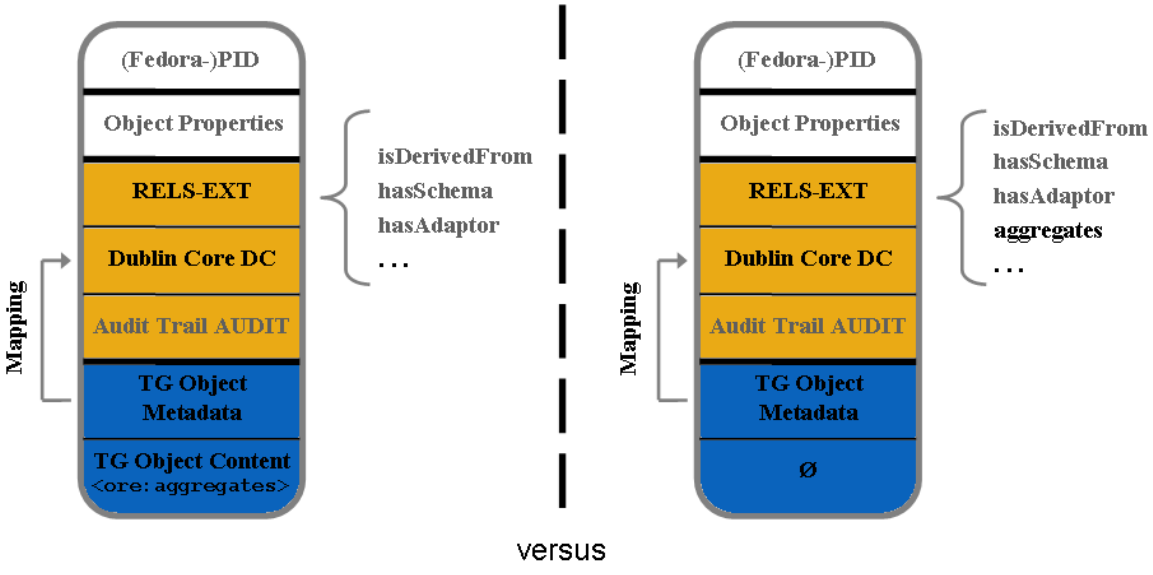

Figure 13: TextGrid Aggregation Type – Implementation Variants

Empty (Ø) Type

The only TextGrid-specific DS for *Work* is the metadata DS. In the same way *'External' Objects* could be handled. But since Fedora allows for specialised Datastreams for *Externally Referenced Content*, it would perfectly make sense to differentiate between the two cases and introduce a separate Content Model for *'External' Objects* with an additional Datastream for the reference(s) to the remote object content.

Open Issues

The TextGrid rights and user management will be kept outside the Fedora repository infrastructure for several reasons:

- Since the TG-search will not – or not deeply – be implemented with Fedora (see below), and each query has to contact the RBAC to determine whether the current user is allowed to see (and find) an object, it is easier and (presumably) faster to do this directly.
- It is not determined yet in which way user information should be passed to the iRODS system, whether the Fedora repository has to pass through user credentials as certificates, or whether the Fedora instance acts as a user of its own.
- The TextGrid Project is modelled according to information extracted from the RBAC system. It would be difficult to re-implement this functionality with Fedora components.
- The new Fedora Security Layer (FeSL) is still considered as being experimental[32] - so it would be better to wait for a production release.

For the time being, TextGrid will interact with the Fedora repository with one single administrator user and TG-auth* will remain as it is. This solution requires only minimal changes in the TextGrid infrastructure since TG-crud already interacts with the grid with a robot certificate.

The same goes more or less for TG-search. As one of the strengths of TextGrid is the possibility to search across XML content via XPath expressions or user-defined XQueries (BTW all search requests are internally modelled as XQueries – except SPARQL for RDF queries) and the Fedora Generic Search Service (GSearch) hitherto only supports the implementation of full text search engines like Lucene, SOLR and Zebra, TG-search has to deploy further on its eXist XMLDB (with its own Lucene instance) and its own XQuery-based application logic. But there is no reason not to make use of the Fedora-internal triplestore for RDF queries.

Deeper Integration – Modularization of TG-crud

In contrast to the other utilities, the Fedora-based re-design of TG-crud is indeed an option. As mentioned above (section 2.4), TG-crud bundles most of the middleware's application logic and performs many atomic actions like generating and assigning TextGrid URIs, extracting information out of (XML) object content and metadata, passing (and deleting) the information to the indices (XMLDB, triplestore), performing I/O operations on the grid etc.

In terms of maintainability and – to be verified – performance it would be useful to implement these components as autonomous modules. Making use of the Fedora messaging framework[33] (i.e. the Java Messaging Service), it should be possible to call the individual modules – if applicable asynchronously – when needed (figure14). For instance, an object can be stored quickly in the repository while the modules responsible for creating a baseline-encoded in-

[32] https://wiki.duraspace.org/display/FCR30/Fedora+Security+Layer+%28FeSL%29
[33] https://wiki.duraspace.org/display/FCR30/Messaging

stance of the object and feeding the search indices can perform their tasks with a certain delay.

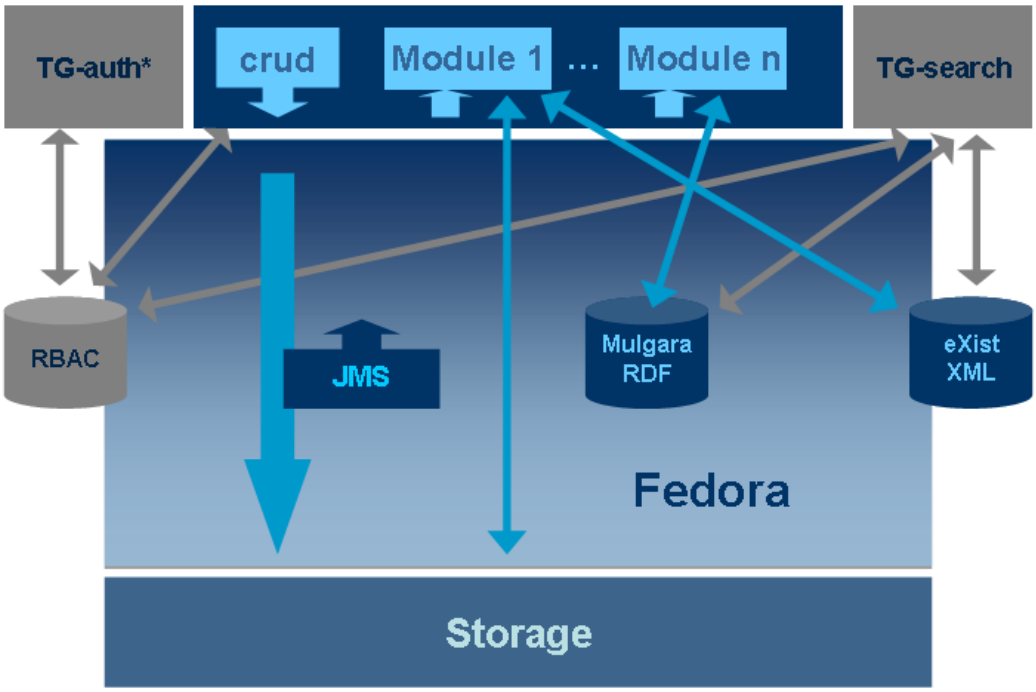

Figure 14: Re-Design of TG-crud

Projects, Revisions and the eSciDoc Solutions

Some of the features provided by eSciDoc, which is build upon Fedora, find an almost perfect match in concepts developed by TextGrid. The most striking example is the logical Content Model (figure 15), where the *Context* corresponds to the TextGrid *Project*, *Container* is the equivalent of *Aggregation*, and *Item* matches the *Item*. Another example is the versioning concept that is almost equivalent to the *Revisions* in TextGrid (see section 2.2 above).

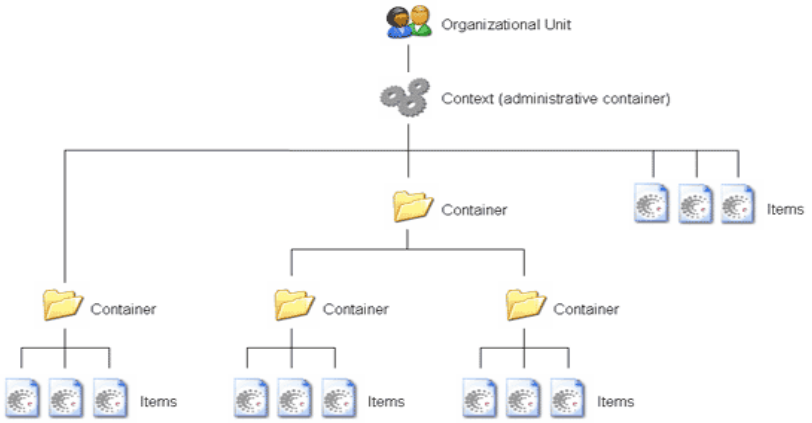

Figure 15: The eSciDoc Content Model[34]

However, the general question how and to what extend TextGrid can make use of eSciDoc code and solutions is not easily to be answered, as there are also some significant differences between the two infrastructures, such as rights management and search concepts. These ques-

[34] https://www.escidoc.org/JSPWiki/en/ContentModel

tions will be dealt with in the TextGrid work package 1.3 “Cooperation with eSciDoc” during the next months. (More to come with the next versions of this report.)

3.3.2. Grid Storage: iRODS

iRODS[35] is a data grid software system implementing a Rule Engine which allows for expressing data management and preservation policies as sets of rules defining actions (so-called Micro-Services) to be executed under specified conditions, for instance at certain events or regularly, according to user-defined intervals. For instance, there are Micro-Services for data replication, checksum verification (on ingest or regularly), metadata extraction, remote service calls etc. It is also possible to write and implement new Micro-Services.

As already mentioned (section 3.2), according to the WissGrid implementation, the Fedora repository is connected with the iRODS server via Akubra, its default low-level storage interface[36], with an Akubra-iRODS module. It is also possible to add other modules, such as an Akubra-GridFTP module.

In terms of scalability and performance, data-intense ingests with huge amounts of small files or smaller amounts of large files should not be performed via Fedora but directly via iRODS. Both the *icommands* command-line utilities and the Jargon Java API provide for performance optimisation through multi-threading. With ingest completed, a callback mechanism is triggered invoking certain Micro-Services which perform the necessary actions (query iCAT, create/modify FOXML) to register the modifications with the Fedora repository.

[35] http://www.irods.org
[36] https://wiki.duraspace.org/display/AKUBRA/Akubra+Project