READ

RECOGNITION & ENRICHMENT OF ARCHIVAL DOCUMENTS

D4.3
READ Platform and Service Maintenance

Philip Kahle, Sebastian Colutto, Günter Hackl, Günter Mühlberger
UIBK

Distribution: http://read.transkribus.eu/

READ
H2020 Project 674943

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 674943

Project ref no.	H2020 674943
Project acronym	READ
Project full title	Recognition and Enrichment of Archival Documents
Instrument	H2020-EINFRA-2015-1
Thematic priority	EINFRA-9-2015 - e-Infrastructures for virtual research environments (VRE)
Start date/duration	01 January 2016 / 42 Months

Distribution	Public
Contract. date of delivery	30.06.2019
Actual date of delivery	10.07.2019
Date of last update	10.07.2019
Deliverable number	D4.3
Deliverable title	READ Platform and Service Maintenance
Type	Demonstrator
Status & version	FINAL
Contributing WP(s)	WP4
Responsible beneficiary	UIBK
Other contributors	All partners
Internal reviewers	Gundram Leifert, Hervé Dejean
Author(s)	Philip Kahle, Sebastian Colutto, Günter Hackl, Günter Mühlberger
EC project officer	Christopher Doin
Keywords	Transkribus

Contents

1 Executive Summary

This deliverable outlines the progress of task 4.1, READ platform and service maintenance, which involves activities such as updating background systems, bug and error handling, system migration (to larger servers according to the expanding network), user support, and similar activities.
In the first year of the project, activities were focussed on improving the overall user experience by fixing existing bugs in the system and extending functionality. On the other hand, the architecture of the platform was overhauled with a focus on scalability which eases the addition of hardware resources in the future. Furthermore, a migration of parts of the system to the University of Innsbruck computing center aimed at providing improved availability of the platform.
In the second year the main goal in this task was to improve existing functionality; only urgently needed features have been added. Also the overall architecture designed in the first year has proven to be robust and was just slightly adapted where shortcomings were revealed. Due to this, the platform could be scaled hardware-wise as planned.
In the last one and a half years of the project the system was augmented with two additional server machines, containing 8 GPUs each, and the server-side software was adapted to manage those resources appropriately. Most other actions taken in this task aimed at maintaining availability and throughput of the platform despite the increasing number of users.
This deliverable is divided into three parts: the first deals with service maintenance, the second part describes the changes in software and architecture of the platform, and the last part provides some details on currently used hardware resources.

2 Service Maintenance

The starting point of the READ platform was the existing Transkribus system, developed in the TranScriptorium project. Transkribus allows a user to ingest sets of document images into the system, where they are stored persistently, transcribe and enhance them in a standardized way and, finally, export them in different formats, such as METS/ALTO, PDF, Word or Excel. Several integrated tools ease the transcription process with automated steps, e.g. finding regions and/or lines in the images or recognizing the text.

Date	**Nr. of users**	**Δ**
1 Jan. 2016	2.828	-
1 Jan. 2017	5.098	+2.270
1 Jan. 2018	8.810	+3.712
1 Jan. 2019	18.441	+9.631
30 Jun. 2019	25.080	+6.639

Table 1: The number of registered Transkribus users throughout the READ project.

The time period from the beginning of 2018 until the end of the project brought a

large growth in the number of registered Transkribus users when compared to the first two years of the project (see table 1). Starting Y3 with 8.810 users, the number has reached 25.080 on the last day of the project.
While actions have been taken in the years before to scale the platform on additional hardware (see section 3), new challenges arose at peak times with hundreds of users which mostly could be mitigated.
The scheduled downtime in 2018 accounted to 10 hours and 1,5 hours in 2019[1] until the end of the project. While this also includes the standard maintenance of underlying services such as storage and database systems, in October of 2018 the amount of data threatened to exceed the capacity available and required a migration to another system. The high load on parts of the systems at the mentioned peak times nevertheless led to a fair amount of unscheduled downtime. While it is hard to measure an exact number, we estimate that at a total of 50 hours in 2018/2019 servicing of users could not be provided due to software bugs or failures in the service infrastructure, yielding an overall availability of about 99,53%[2] during the last one and a half years of the project.
However, the incidents have shown weaknesses in the software, software development methods (see section 2.1) and infrastructure which were or are object to improvements: a more modern monitoring system based on Checkmk[3] was set up in 2019 and will replace the legacy solution, based on Icinga[4], once all existing service checks have been migrated. This system will allow for a more in-depth analysis of load peaks and bottlenecks, where the legacy system's configuration proved to be too time-consuming and not flexible enough.
Also, in the beginning of 2018 some service components, such as the file server of Transkribus, still relied on the initial setup and hardware from the Transcriptorium project and could not tackle the increasing load. Those have been moved to more powerful server machines and a redundant setup is planned in order to mitigate failure.
The increased number of users of the software is also reflected in the effort put into support. 1.789 mails from users have been processed by the UIBK team that were received on the general support and bug report mail accounts.

2.1 Software Development in Year Three

2.1.1 Structure Editor

In the past few years we have received more and more requests by users for the enrichment of documents on a structural level. Thus we developed a structure editor for the Transkribus expert client (TranskribusX) in 2018. The purpose of this editor is to add structural tags, e.g. "paragraph", "heading" or "marginalia", to layout elements (i.e. a region, line or word) on document pages.
The editor can be accessed via the "Metadata" tab and its sub-tab "Structural".
The set of available tags for every installation of TranskribusX can be configured via the

[1] Scheduled downtime 2016: 7 hours, 2017: 3,5 hours
[2] Availability 2016: 99,73%, 2017: 99,73%, see D4.2 and D4.1
[3] https://checkmk.com/
[4] https://icinga.com

user interface. It is possible to select from a set of predefined tags or create new custom types.
Each different structure type has a (configurable) color associated with it that defines the color with which this element is drawn in the image canvas when the editor is selected.
The basic usage of the structure editor is easy and straighforward: the user selects one or more structure elements from the image canvas, then clicks on the "plus" button at right side of each listed structure type to assign it to those elements. Alternatively, the user can also right-click on the selected elements in the canvas and assign the structure type via the submenu "Assign structure type".
During the export of a document, structural tags are written into the "custom" attribute of the corresponding layout element. A tag called "structure" with an attribute "type" used for that purpose, e.g.: custom="structure type:paragraph".
Note also, that document pages that have been enriched by structure types on region level can consequently be used as ground-truth to train a model for an automatic structure analysis tool as described in D4.6.

2.1.2 File and Image Server

As discussed in section 2, weaknesses in the initial design emerged with the growing number of clients connecting to the backend and the increasing amount of image data ingested into the system. One of the identified shortcomings concerned the delivery of files to client applications and especially the throughput when storing new files. Both tasks are handled by a separate file and image server application (*fimagestore*), which exposes a HTTP API for uploading files that are written to a network attached storage system. For image files, different formats used for presentation are created asynchronuously and stored, e.g. a thumbnail and a lightweight JPEG representation. The original file and any precomputed derivates are then available to be retrieved via the HTTP API. Moreover, on-demand image operations, e.g. scaling or cropping, can be requested.
Several person months were spent in 2018 on an analysis, partial reimplementation and update of the file and image server application in order to improve the multithreading capabilities and fix issues that caused problems regarding performance or stability. Moreover, the original design, using a embedded database, was restricted to a single deployment while the refactored version relies on UIBK's Oracle database management system. This allows for a redundant deployment of multiple application instances for load balancing and increased failure tolerance, laying the foundation for further scaling. A separate instance can be set up to handle the preprocessing of the presentation image types exclusively and relief those deployments that serve the files to clients.
Tests were conducted running both versions of the server application, *fimagestore 1.0* and *fimagestore 1.5*, as well as the client on a Intel Core i7 3770 desktop machine. Files for both instances are stored on the NFS storage that is also used in production and have been picked from the Bentham dataset (JPEG, 5,3 MB per file). Each test simulates 10 clients accessing the server application concurrently and every client executes the same set of commands, i.e.:

- Upload: upload 8 JPEG files, access the metadata file via HTTP and delete the

files afterwards. Average time taken from 5 test runs.

- Display: Retrieve the precomputed reduced-quality JPEG file for 100 previously uploaded images.
- On-demand conversion: Request an on-demand conversion (scale to 120x120 pixels size, PNG output) for 20 files.

Table 2 outlines the time in seconds (average from 5 test runs) measured until all the clients completed their command queue. The numbers show that a single file server instance now can serve an increased number of clients with images in a timely manner while uploads to the file server from the Transkribus Server and worker machines benefit from the improved multithreading.

Test	**fimagestore 1.0**	**fimagestore 1.5**
Upload	62,37	10,38
Display	19,45	6,76
On-demand conversion	40,56	11,55

Table 2: Time in seconds until all 10 clients complete their command queue

While the Transkribus desktop application accesses images and XML files exclusively via the file server's proprietary HTTP API, the IIIF image API[5] provides many benefits for web-based user interfaces as developers can choose from a variety of ready to use image viewers that load images dynamically and thereby save bandwidth on the client and improve the user experience crucially. Therefore, the file server setup of Transkribus is augmented with an installation of the Cantaloupe image server[6] that provides this type of API for serving the images stored in the system to respective applications.

3 Architecture

In year one, the server application of Transkribus was split into two parts: one web application (TranskribusServer) that serves the data to clients via its REST API and a worker application (TranskribusAppServer) responsible to handle heavy workload tasks such as layout analysis or text recognition. Moreover, the Quartz scheduler framework was used to implement queues for specific tasks and executing the jobs from those queues. As the number of different job tasks grew this year, it became clear that a more flexible system is needed to cater for the specific needs in the Transkribus platform. Thus, a custom-made solution was put into place which also allowed to further modularize the platform: with the current system it is possible to implement applications to handle specific job types or even just one of those, e.g. keyword spotting requests are dealt with in an application including that specific module. Once the workload in a module reaches a critical level, it is possible to deploy and start more instances of the module; the

[5]https://iiif.io/api/image/2.1/
[6]https://cantaloupe-project.github.io/

system automatically distributes the workload among them. While the scalability was already vastly improved in year one, the modular approach allows to scale the system according to the current needs and available hardware. An overview of the platform architecture is shown in figure 1. In year three, the integration of URO's next iteration HTR technology (see D7.9) required to provide access to any graphics processing units (GPU) and their compute capabilities. In this turn, the modular system described above gained further configuration options: a number of available GPUs can be assigned to a module, which then manages those resources, i.e. allocates them fully or partially for an applicable job depending on the configuration.

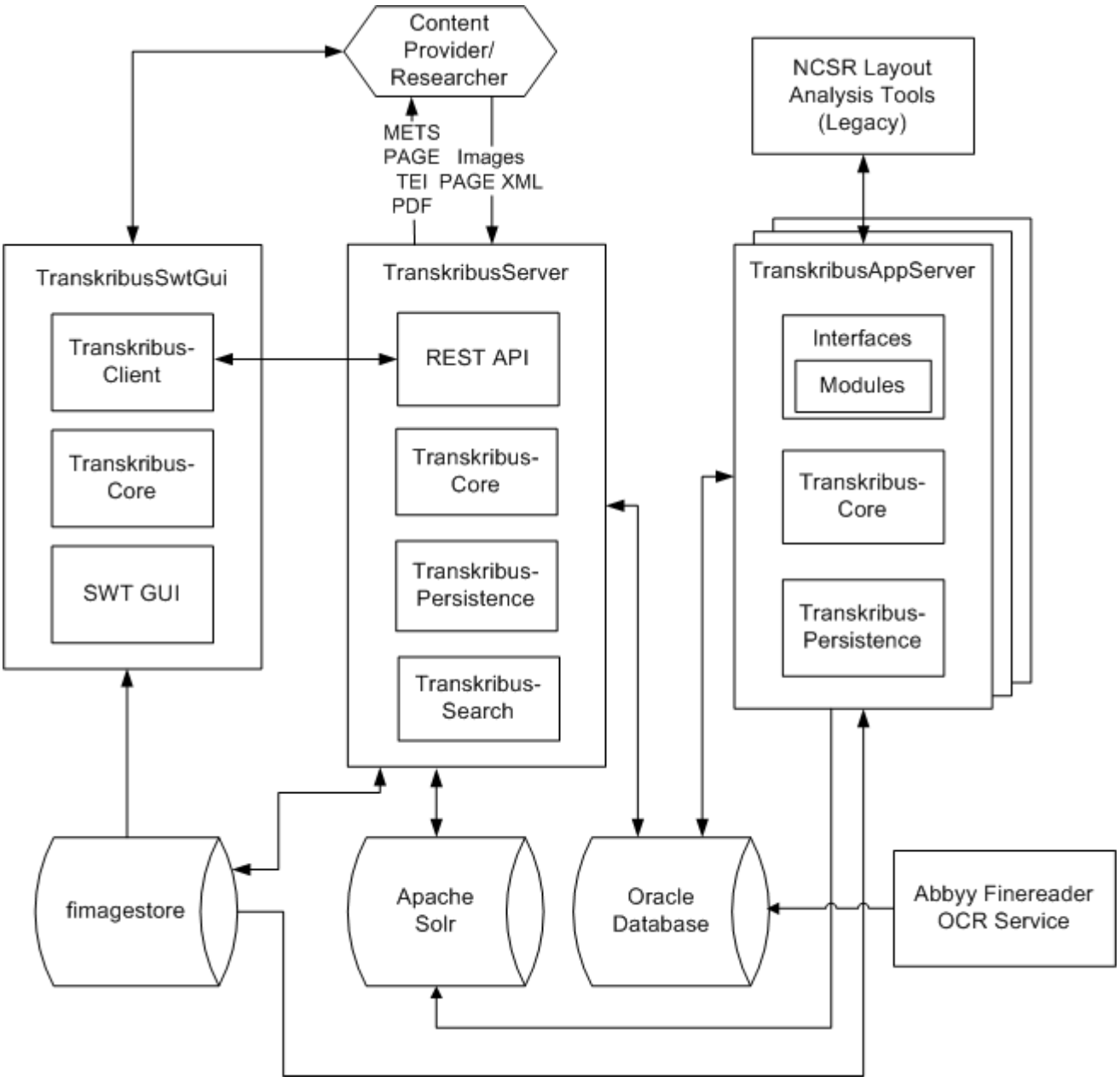

Figure 1: Current Transkribus architecture

4 Hardware

While in the beginning of the project all platform components ran on virtual machines (VM) provided by the Institute for Databases and Information Systems most of them have been migrated to VMs at UIBK's IT department during the first year. An HP Bladecenter including 16 servers with 12 cores each was acquired in the end of the first year and dedicated to running heavy processing tasks in the platform. To the date of this writing, 13 of those machines have been added to the Transkribus platform, running mostly instances of TranskribusAppServer but also other server applications such as Apache Solr and the *fimagestore* file server. Three machines remain detached for offline batch processing.
In 2018 two additional server machines, each with 8 Nvidia 1080ti GPUs, have been added to the platform. While one of those machines is currently still used for testing and offline processes, the second one runs the URO HTR+ training processes.

5 Conclusion

While the effort put into this task could be reduced to some degree during the second year, the increase of active users and also in the platform's code complexity required more resources than expected in the third year and until the end of the project. More effort had to be put in testing new code, adding tests for and refactoring of legacy code as well as the analysis and monitoring of the system.
Although, this slowed down the development process including the integration of services and tools (see D4.6), the system now can cope with larger numbers of users and strategies for further scaling are in place or at least planned in detail.