READ

RECOGNITION & ENRICHMENT OF ARCHIVAL DOCUMENTS

D4.1
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	31.12.2016
Actual date of delivery	28.12.2016
Date of last update	21.12.2016
Deliverable number	D4.1
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	Martin Majek
Keywords	Transkribus

Contents

1 Executive Summary **4**

2 Service Maintenance **4**
2.1 New features implemented during READ 4
2.1.1 Fulltext Search 5
2.1.2 Enhanced Export Functionality 5
2.1.3 Table Editor 6
2.1.4 Login via OAuth2.0 6
2.2 Source Code Management 6

3 Architecture **7**
3.1 Improvements in READ 8

4 Hardware **8**

5 Conclusion and Outlook **9**

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 will ease 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.
This deliverable is divided into three parts: the first deals with service maintenance, the second part describes the improved architecture of the platform, and the last part provides some details on currently used hardware resources and additions in the near future.

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.
When the project started, Transkribus had a user count of 2828 (as of 1.1.2016). During the first year, the dissemination activities led to 2082 new users that could be acquired (28.11.2016). The increase in the number of users also is reflected in the amount of work put into the task of user support. UIBK received over 284 feature requests and bug reports in 2016 and to the date of this writing 10.601 users visited the Transkribus website[1]. The Transkribus user interface client was downloaded 6552 times at all, while 2510 of those downloads happened in the first year of READ.
Also the service uptime could be maintained at a high level: in 2016, the platform was unavailable for 7 hours due to scheduled maintenance. One incident with a malfunctioning storage system caused an unscheduled downtime of 17 hours; an issue which will be reflected in the migration strategies in 2017. All in all, service availability can thus be stated with 99,73%.

2.1 New features implemented during READ

Besides the general bug fixing and service maintenance, a large number of feature requests have been incorporated. While this was a constant effort in maintaining Transkribus, some features that enhance the usability significantly, are worth to be noted

[1] https://transkribus.eu

here.

2.1.1 Fulltext Search

During the first year of READ, the platform was extended with Apache Solr, a fulltext search engine that allows for indexing and retrieval of fulltext as well as filtering the results by certain aspects of the metadata. Therefore, a Solr indexing schema was developed that is able to handle the relevant fulltext and metadata from Transkribus. The TranskribusSearch project utilizes the Java API of Solr in order to facilitate the indexing and retrieval of data and is built into the TranskribusServer application (see Figure 2). All the retrieval functionality is exposed via the Transkribus REST service which allows to access it from external applications, such as the Transkribus graphical user interface. The latter also provides a reference implementation of a search interface, utilising the features Apache Solr provides.

2.1.2 Enhanced Export Functionality

Due to the fact that our service platform has grown in number of users as well as in the amount of available features during the first year it was important to keep the export up to date and close to the user requirements. The status at the beginning of READ was that we had following export formats available:

- METS/Page XML
- TEI
- PDF
- RTF

Now after one year we have an ongoing support of these formats and replaced RTF with DOCX. For the Mets export additional Alto files can be produced now. Two new export formats are available as well - Excel for tag export and Excel for table export. More information is below. Maybe the most important enhancement is the possibility to blacken out sensible information, e.g. for privacy protection which is always an issue when publishing documents. Another user requirement concerns treating tags. Since marking tags is an essential task in transcribing text the export of these tags needs to be considered as well. So far we have several possibilities for tag export: First of all the user can highlight the tags in the PDF export with colored lines, this means each tag is mapped with a different color and gets underlined with this color during export. Moreover, the tags are added as a list appended at the end of the document and sorted after the type of each tag. For the PDF export we have not only the text layer under the image but also extra text pages behind each image page as a uniform and readable presentation. For the docx export, handling abbreviations can be rendered in different ways: keep abbreviation tags as they are or add the expansion after the abbreviation or substitute the abbreviation with the expansion. Tags in general get exported as index entries in an index list at the document end. For a tag only export we provide an Excel

where each tag type gets stored in an extra sheet with a combined sheet containing all tags at the first page. This leads to a good starting point for further operations. The integration of the table editor (see section 2.1.3) into the service platform results in enhancing the export as well. For further usage it is very helpful to get an Excel document containing all the defined and recognized tables. Since the platform should be as universal as possible the support of right to left writing is now provided as well. Other export features developed in the first year of READ are creating a title page - with metadata information like title, writer and so on and also the editorial declaration - the possibility to create a ZIP file containing all exported formats and choosing different transcription versions. Of course, all of the described export settings are configurable. The latest work in the export field was to offer a server-side export. The big advantage of that is to avoid computationally- and main memory intensive jobs at the user side, at least for very big or huge amounts of documents. This was one of the latest step to offer all the main features as web services for machine to machine communication which is already a very imporant use case for many partners and users.

2.1.3 Table Editor

The table editor was created to be able to digitize the vast amount of table data that is present in the Trankskribus database of documents. The challenge was to be able to represent (and in a further step to export) the logical structure of the table on the one hand but also aid the automatic table recognition process carried out by CVL on the other hand. Thus, a convenient table editor where one is able to draw the exact borderlines of table cells as well as represent text inside the table cells had to be created. Also, the PAGE file format had to be extended in order to be able to store all necessary data generated during that process.
In its current status, the table editor is still in an "alpha" version, meaning that it is already usable and included in the latest version of the program but still, some convenience features are missing and further testing and bug fixing has to be performed.

2.1.4 Login via OAuth2.0

In order to ease the registration process, an OAuth2.0 client was implemented that allows to register via an external user authentication process. As reference, login with Google accounts was integrated. For 2017, it is planned to extend this with means to login with Facebook and ORCID[2] accounts.

2.2 Source Code Management

While the source code of all the Transkribus applications resided in a Subversion repository on a server of the University of Innsbruck during the TranScriptorium project, a more sophisticated system was needed in order to allow access for all the READ partners, manage access rights and collect all contributions in one place. Therefore, a Transkribus

[2]http://orcid.org/

organization was created on Github[3] and all source code (besides some legacy components) was moved there. However, some of the code is not publicly available due to either its work in progress status or, in case of the TranskribusServer, due to proprietary dependencies, such as Oracle DB. Github turned out to be a valuable project management tool, offering issue management and basic documentation facilities.

3 Architecture

From the architectural viewpoint, the Transkribus system started with a rather simple client-server scheme, which is outlined in Figure 1. The central component of the plat-

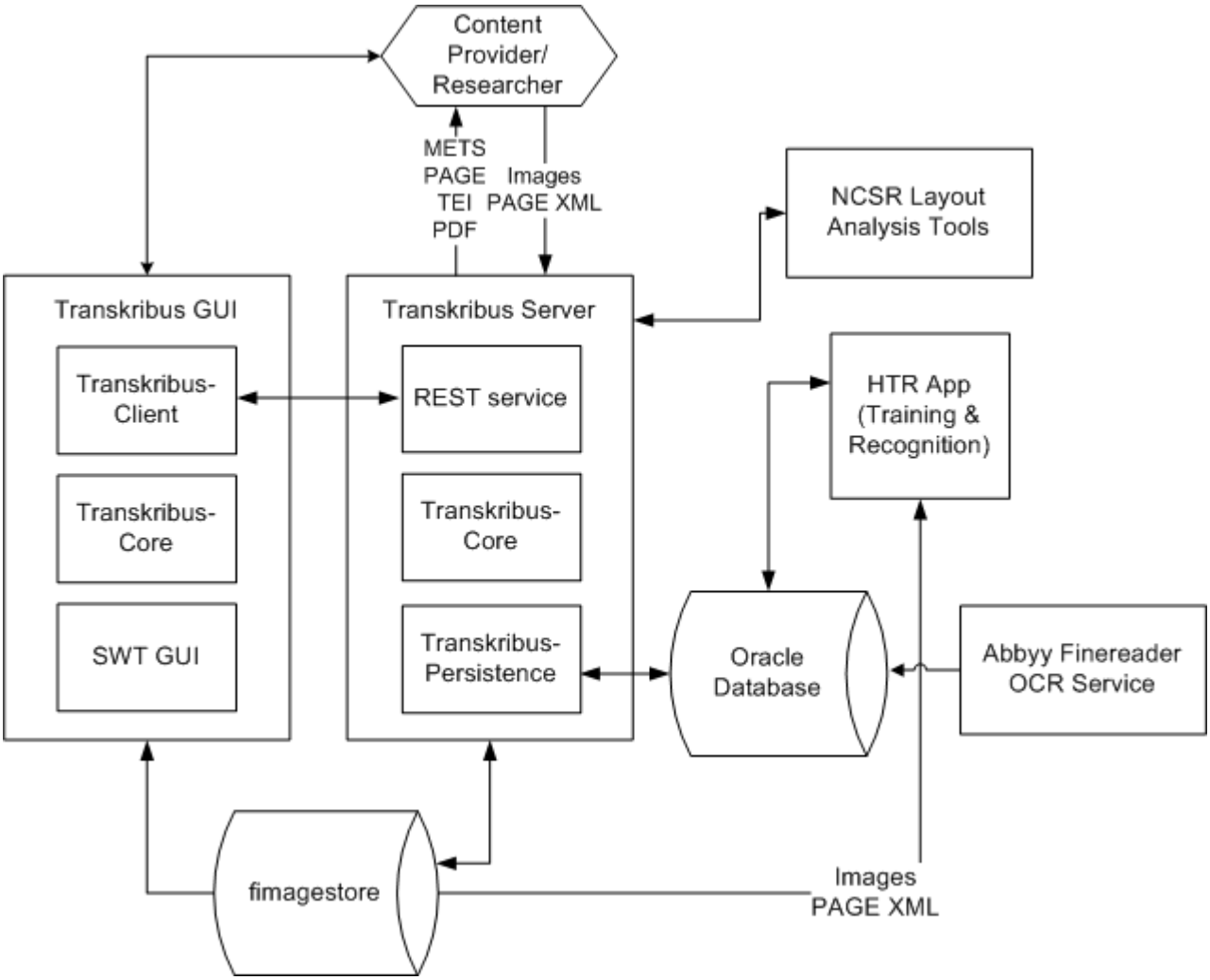

Figure 1: Initial Transkribus architecture

form is the TranskribusServer, a web application written in Java that runs within an instance of Apache Tomcat. That application deals with the complete document management workflow and handles storage of digital objects in the database and files in a

[3]https://github.com/Transkribus

separate filestore application (fimagestore). It exposes its functionalities via a RESTful API which is used e.g. by the native Transkribus graphical user interface (GUI). The Transkribus GUI consists of a client component that exposes all functionality for communication with the server's REST service and of course the user interface itself, which is based on the SWT framework[4].
An additional application, called HTR server, includes the handwritten text recognition engine provided by UPVLC during the TranScriptorium project. Due to the vast amount of processing power used for this task, this was set up on a separate machine.
All of those applications rely on the TranskribusCore package which provides the business objects and basic functionality for creating, manipulating, and storing those.

3.1 Improvements in READ

A cornerstone functionality of the server application is the job management, that is responsible for execution of all kinds of utility jobs, such as creation of documents in the platform via upload, duplication or deletion of documents, layout analysis jobs such as automatically finding lines (based on tools provided by NCSR during the TranScriptorium project), and most importantly handwritten text recognition on documents. A major flaw in this architecture was that all the workflows were executed directly in the server application which led to bottlenecks in various situations, e.g. when a user triggers the creation of a large number of documents at once. Also, the integration of tools was based on custom solutions, respecting the properties of the specific piece of software to be integrated (this topic is discussed in more detail in D4.2).
Thus, one of the first actions within the READ project was to redefine the system architecture in terms of scalability, allowing for large numbers of users triggering workflows in the platform. The outcome of these efforts is depicted in Figure 2. For scheduling workflows in the platform, the Quartz scheduler framework[5] was introduced which is able to organize vast amounts of jobs in a database. Different job queues were implemented for dealing with utility jobs, layout analysis and different types of text recognition (OCR, HTR). The previous bottleneck was overcome by scheduling all the jobs in the server component while executing them in a separate application, the TranskribusAppServer. As Quartz allows for a clustered mode of operation, an arbitrary number of TranskribusAppServer instances can be added in the future to handle increasing workloads.

4 Hardware

The Transkribus system was initially deployed on a set of virtual servers of the Institute for Databases and Information Systems (DBIS) of the University of Innsbruck. The HTR engine ran on a dedicated 8-core server acquired during the TranScriptorium project.

[4]https://www.eclipse.org/swt/
[5]http://www.quartz-scheduler.org/

Due to the increasing requirements regarding service availability, parts of the system, e.g. the TranskribusServer application, were moved to the University of Innsbruck's computing center, whilst the TranskribusAppServer(see section 3.1) now runs on the dedicated server; a setup which exploits the available resources much better. As the training of handwriting models requires high computing capacities and this is often requested by users, UIBK acquired a HP Bladecenter with 192 cores in the end of 2016 which should satisfy the demands in 2017. To the date of this writing, this machine is set up and can be implemented into the platform in the coming weeks.

5 Conclusion and Outlook

Recapulating, the first year of the project can be seen as a preparation phase where things were put on the right track for the coming years. The platform is now able to handle a much higher numbers of active users, speaking in terms of system architecture as well as hardware-wise. Furthermore, the Transkribus GUI end user client application became more stable and feature-rich while many user requests where implemented in order to maintain top-notch user experience. With the newly acquired server, a valuable pool of computing power will be added to the platform at the end of the year, catering for the expected growth in the coming year.
Whilst the architectural decisions are mostly settled now, some resources will be shifted to task 4.2, tools integration, as more and more pieces of software become available from READ partners. However, the task of service maintenance is and will remain an important building block of the success of this project and UIBK will eagerly search the contact to their users.

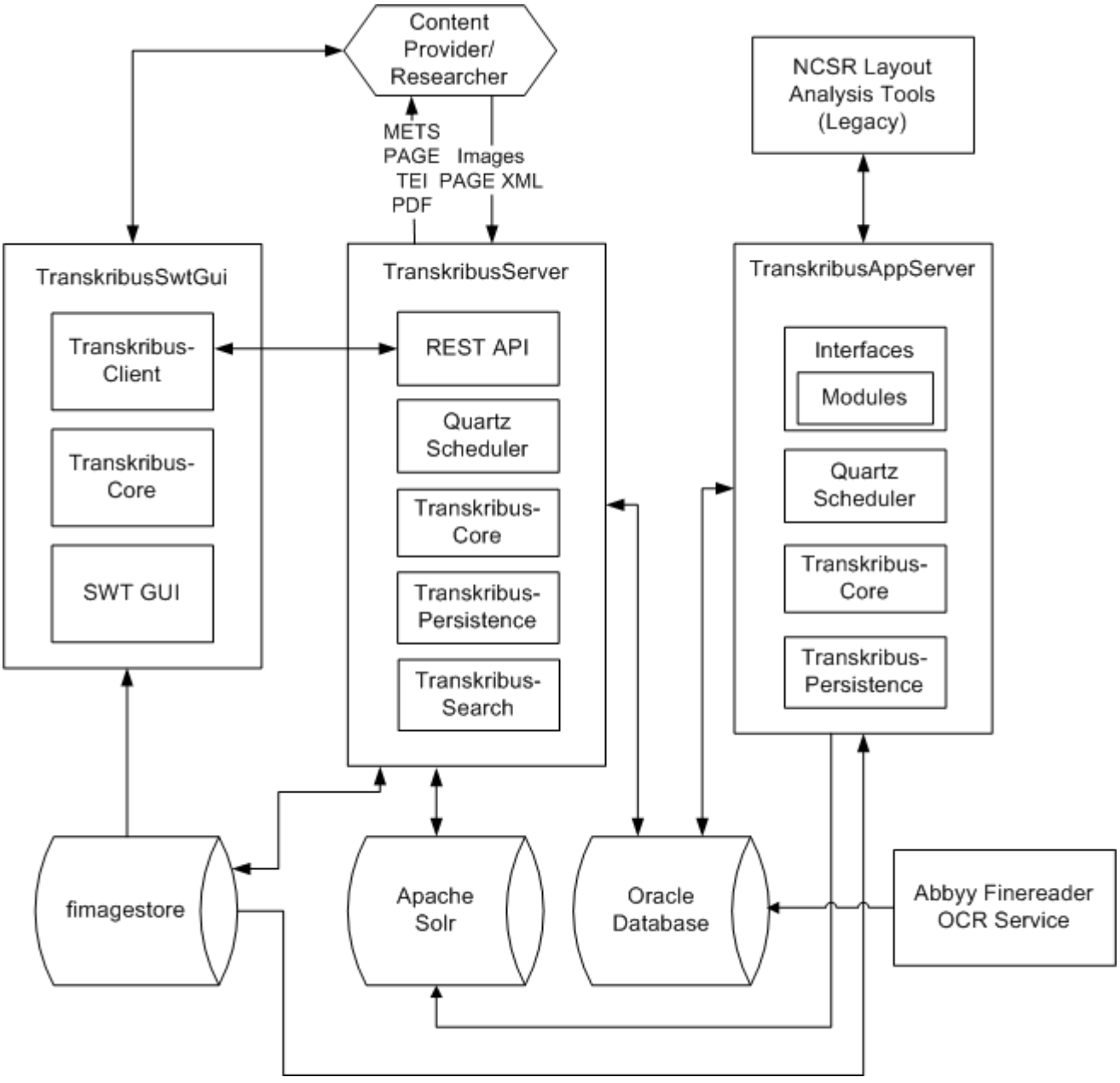

Figure 2: Current Transkribus architecture