Proceedings

Service-oriented Architectures (SOAs) for the Humanities: Solutions and Impacts

Joint CLARIN-D/DARIAH Workshop at Digital Humanities Conference 2012
# Table of Contents

Keynote Eric Nyberg.  
A Service-Oriented Architecture for Rapid Development of Language Applications ........................................... 1

Dorothee Beermann, Pavel Mihaylov and Han Sloetjes.  
Linking annotations Steps towards tool-chaining in Language Documentation .................................................. 2

Andre Blessing, Jens Stegmann and Jonas Kuhn.  
SOA meets Relation Extraction: Less may be more in Interaction ........................................................................ 6

Michael Scott Cuthbert, Beth Hadley, Lars Johnson and Christopher Reyes.  
Interoperable Digital Musicology Research via music21 Web Applications ......................................................... 12

Emanuel Dima, Erhard Hinrichs, Marie Hinrichs, Alexander Kislev, Thorsten Trippel and Thomas Zastrow.  
Integration of WebLicht into the CLARIN Infrastructure .................................................................................. 17

Rüdiger Gleim, Alexander Mehler and Alexandra Ernst.  
SOA implementation of the eHumanities Desktop .............................................................................................. 24

Thomas Kisler, Florian Schiel and Han Sloetjes.  
Signal processing via web services: the use case WebMAUS ............................................................................. 30

Chiara Latronico, Nuno Freire, Shirley Agudo and Andreas Juffinger.  
The European Library: A Data Service Endpoint for the Bibliographic Universe of Europe ......................... 35

Przemyslaw Lenkiewicz, Dieter van Uytvanck, Sebastian Drude and Peter Wittenburg.  
Advanced Web-services for Automated Annotation of Audio and Video Recordings ........................................ 39

Scott Martens.  
TüNDRA: TIGERSearch-style treebank querying as an XQuery-based web service ........................................... 43

Christoph Plutte.  
How to Turn a Desktop Application into a Web-Interface?  
Archiv-Editor as an Example of Eclipse RCP and RAP Single Sourcing .............................................................. 51

Thomas Zastrow and Emanuel Dima.  
Workflow Engines in Digital Humanities ........................................................................................................... 56
Keynote: Eric Nyberg (Carnegie Mellon University, Pittsburgh)

A Service-Oriented Architecture for Rapid Development of Language Applications

Abstract

In this talk we present recent work on the design and development of a distributed, service-oriented architecture for language application development that defines atomic and composite web services along with support for service discovery, testing and reuse. The architecture provides an open advancement framework for component- and application-based evaluation; this framework enables rapid identification of the best-performing component assemblies for new applications, thus contributing to more effective investment of resources in research and development of language technologies in new problem domains.
Linking annotations
Steps towards tool-chaining in Language Documentation
Dorothee Beermann, Pavel Mihaylov, Han Sloetjes
Norwegian University of Science and Technology (NTNU), Trondheim, Norway
Ontotext, Sofia, Bulgaria
Max Planck Institute for Psycholinguistics, Nijmegen, The Netherlands
dorothee.beermann@hf.ntnu.no, bin@bash.info, han.sloetjes@mpi.nl

Abstract
The strong point of a virtual research environment (VRE) is that it facilitates a collaborative approach to data management and creation. Ideally, such a VRE allows its users to combine best-for-the-task tools in a simple but efficient manner. This paper presents work in progress which aims to improve combined system use in Language Documentation. Exploiting the strength of the multi-media annotation tool ELAN and the online multi-lingual database TypeCraft, we describe how the annotation of audio, video and text resources can be enhanced by presenting technology which supports shared annotations and collaborative editing online. We examine data mobility problems by presenting a use-case, and discuss how a workflow typical for data processing in Language Documentation can be improved substantially through minor but user-oriented development.

Keywords: virtual research environment, language documentation, user-driven software development

1. Introduction
The strong point of a virtual research environment (VRE) is that it facilitates a collaborative approach to data management and creation. Ideally, such a VRE allows its users to combine best-for-the-task tools in a simple but efficient manner. This paper presents work in progress which aims to improve combined system use in Language Documentation. Exploiting the strength of the multi-media annotation tool ELAN and the online multi-lingual database TypeCraft, we describe how the annotation of audio, video and text resources can be enhanced by a technology which supports shared annotations and collaborative editing online. We examine data mobility problems encountered in this technology. The combined used of TypeCraft and ELAN is already practised, but it comes at present at a high cost. We present the technical changes necessary to improve the workflow for a combined system use. Some of the solutions presented here are already implemented while others still need to be finished.

2. Short system descriptions
2.1 TypeCraft
TypeCraft (TC) is an online database featuring a tabular editor for the manual creation of Interlinear Glossed Text (IGT). The core application is wrapped into a customised mediawiki (TC-wiki) which serves as a general entrance port and collaboration tool. Below we provide an overview of the application’s main functionalities:

Annotation
- Manual import of continuous text or sentence collections
- Tabular interface for morpheme level glossing, automatic sentence break-up

- Drop-down reference lists for annotation and flexible insertion and deletion of words and morphs
- Semi-automatic annotation and easy access to relevant information such as gloss definitions and an ontology of grammatical concepts from the annotation interface

Collaboration
- Graded access, individual work spaces
- TC texts and phrases have their own URI and thus can be acquired and exchanged freely online
- User groups can share data
- Collaborative editing of TC-wiki pages

Data export
- Export of annotated data to Microsoft Word, Open Office and LATEX for paper publications
- Print-friendly versions of the TC web pages including exported database material
- Export of XML for automatic data processing

TC search is another strong point of the service that can be highlighted here. It allows complex searches on several tiers so that word and morpheme queries can be freely combined with a search for specific glosses or combinations of glosses, co-occurring either in a phrase, on a word, or on a morpheme. Search is graded, and can target the user’s own data, as opposed to group data where search serves to establish inter-annotator consistency.

2.2 ELAN
ELAN is a standalone tool for the manual annotation of digital audio and video recordings. It offers generic media annotation functionality and most of its current features are not specifically designed for research in linguistics, but are useful in other areas of research as well. The same holds for its data model, with tiers as
containers for annotations and the possibility of defining hierarchical relations between tiers and, ipso facto, between annotations. ELAN is flexible enough to suit the needs of various types and conventions of annotation, and EAF, the XML data format native to ELAN, is supported as an import and export format by a growing number of tools and frameworks. ELAN runs well on Windows and Mac OS, while on Linux the situation concerning AV playback needs improvement. Though most of the Java code is platform independent, for media playback, ELAN preferably uses the frameworks that are readily available on a system. ELAN allows to link multiple videos to an annotation document, it can visualize the waveform of audio and the curves of time-series data. In addition it offers several views and editors for annotations.

For documentary linguists, exchanging data with e.g. Toolbox or FLEX is and has been the most common way to create annotations that are both time-aligned and linguistically rich. Although in the meantime work has started on extending ELAN with modules that add semi-automatic interlinear glossing functionality to the program, it is not available yet, and it will not make existing interoperability features superfluous.

ELAN, as a single user desktop application, historically focuses on improving and streamlining local media handling and local data processing. Collaboration is only possible by exchanging files between members of a team. Although some actions can be performed on multiple files (importing and exporting data, searching etc.), which can be seen as operations on a local corpus, most of the work is done on a per document basis. Only recently the first steps have been made to extend the locally available processing power and algorithms by calling web services. There are two main categories of services that are of interest to ELAN and its users, and the one of interest in this context is a service that works on text, applying parsers and taggers to the input returning new layers of annotations.

2.3 Synergy

The goal of the present project is to integrate annotated data from both ELAN and TC in order to produce a richer set of annotations on the same source material. ELAN will be used for audio and video annotation, while TC will provide interlinear glossing on the language contained in the audiovisual resources. Through the present development, the user will be able to start annotating in either program and then export data and continue annotating in the other program. The data exchange will be done in a standardised XML format improving the already existing TC XML export. It is within the reach of present ELAN development to provide public URIs referring to audio- and video source material. While ELAN's local handling of audio- and video material is state of the art within the present set of technologies, TC is designed for the creation, retrieval and sharing of linguistic data in the form of IGT which is the most common data format in linguistics. The IGT is important since it often is the only structured data available for less-resourced or endangered languages. For these reasons it is important that the digital management of such data finds solid support also in emerging VREs where multi-media annotation, such as facilitated by ELAN will be the other central theme. Designed for the normal linguist, TC can offer simplicity of method to its users together with those features in which online services excel: linking of resources, general openness and a peer-centered design. Complex annotation of media, however, is best handled locally, therefore, a synergy between online and off-line tools is the way we go.

3. A Use-case

The Paunaka Documentation Project (PDP) is funded by ELDP and located at the University of Leipzig. PDP's work comprises the compilation of a corpus of audio and video recordings. The project data is managed with ELAN and archived with ELAR. For the actual data analysis PDP uses both ELAN and TypeCraft where 60% of the data receives a morphological analysis using TypeCraft. Some members of PDP have used ELAN and Toolbox already in earlier projects, while the present project considered three different glossing tools, namely Toolbox, FLEX and TypeCraft. The two main reasons why the project selected TC were that, first of all, TC allows incremental annotation, which is particularly necessary in the beginning of analytic work, and secondly, because TC is an online service and therefore allows the exchange of project data under distributed work.

The project loads all primary data to ELAN where transcriptions and translations are created in one swoop together with the segmentation of the audio. We will come back to primary data processing in ELAN in section 4 since it crucially effects data export. To combine the use of ELAN with the use of an Interlinear Glosser is not new to some project members. In prior projects, Toolbox files had to be exported back into ELAN, not for further morphological processing, but as a prerequisite for archiving (a requirement imposed by some project funding agencies).

Turning now to the combined use of TC and ELAN, the PDP reports that the export of data to TypeCraft is not only tedious but in addition sentence indices and ELAN's time codes 'get lost'. Below we reproduce a project internal description of the present data export procedure used by the PDP:

1. in ELAN select File->Export As->Tab-delimited
   Text. Select the line to be exported - make sure not to select any other option
2. save
3. open the export in Word
4. use search & replace to delete the line initial
5. if Word recognises the special symbols, copy the text directly to TypeCraft
else:
    identify the document for Toolbox, e.g., by id, open the document in Toolbox, then copy to TypeCraft selecting 'text' as the format.

The present data-flow has two major problems: (i) it is too round-about to be effective and (ii) crucial information for the linking of media and text annotation gets lost.

4. **Combined system use**

In section 2, we have identified where our tools are complementary and where their strong points are, while in section 3, we discussed present problems for data exchange. In spite of these problems, the Paunaka project's decision to combine the use of ELAN and TypeCraft buttresses the usefulness of a working environment that allows to integrate multi-media annotation with linguistic online services. Under the combined use of our tools, TypeCraft will continue to provide the collaborative work environment and access management system as a service online. For our users that means that with a login to TC, the system allows online morphological glossing and collaborative editing, as well as export of formats not available in ELAN. However, accurate media handling over internet is still problematic, and ELAN, as a local tool, mostly has direct access to the media, which improves the speed and accurateness of the segmentation process. Therefore media handling will in our project remain local. In order to make a combined and flexible use of online service and local tool possible, we need to provide for a seamless data-flow between the online and off-line system. To this end, already implemented changes and planned development will be described in the next section.

4.1 **System adaptations**

TC was designed as a system to be accessed by humans. Thus, the existing login and export facilities were not directly suited for the exchange of data between TC and ELAN. However, MediaWiki and the existing XML export schema provided a solid basis for developing the initial TC-data-exchange API. By making the web service API available on TypeCraft, we allow users to log in and to retrieve text documents that they have access to, in order to download them and time align them in ELAN. To this purpose the XML schema was extended with several items one of which is the item **Listing of texts without phrases.** This feature will enable TC users to choose transcribed texts (their own or data that they share with a group of other users) for import into their local ELAN where these texts can be aligned with the corresponding media material so that further annotations, important for cross-media comparison and analysis, can be added.

To enable the communication between the two systems we use a simple RESTful (Richardson and Ruby) web service. Access control is provided by enabling the MediaWiki remote API and the exchange of a session id. The following commands regulating export from TC to ELAN have become available to the users of both systems:

- list texts
- export text with 1 to n phrases
- export text with all phrases
- export 1 to n phrases

In ELAN a menu has been added for accessing known web services, such as TC. For the combined system use of TC and ELAN that means that the user can now log in to TC from ELAN. After login, s/he will be presented with a list of available texts from TypeCraft. Users may have an audio and/or a video already loaded in ELAN or they may download a speech recording directly from TypeCraft. Depending on the text downloaded from TC, ELAN will create one or more tiers to accommodate the imported information. It is crucial for the whole process that the ids of the phrases are retained, otherwise it will be next to impossible to cross-corelate annotations in ELAN with those in TC, effectively turning the import into a one way process.

The described export of already existing TC-data to ELAN will be of particular importance for those TC-users that host speech recordings and corresponding texts on TypeCraft. A combined use of both systems can now provide a better integration of the user's resources and an improved workflow. Text annotations coming from TC can, using ELAN, be confronted with the original recording, and it now becomes possible to correct already existing annotations. Central for connected text is time alignment. Moreover, using ELAN, the user can choose the size of the interval to be time-anchored. Finally, users can enrich already existing TC annotations by adding cross media annotations in ELAN, something that will allow them to identify linguistically important formatives across media.

Turning now to the import of ELAN data to TypeCraft, we have described a laborious version of this exchange in section 3. Paramount for the exchange is that the user of ELAN does not lose information crucial for the alignment of audio/video and text annotation, leading to unwanted separation of recordings and the connected text. In order to facilitate the process, we have updated the TC XML by the following additional items:

(i) **Introduction of the notion Speaker at the phrase level**

So far TC only allowed the storage of speech recordings and transcripts for safe-keeping, yet, their connected processing has not been possible. Neither was it possible in TC to identify speakers or align speaker and utterance. This has now become possible through the present development.

(ii) **Introduction of offset and duration at the phrase level**

This information assures the alignment of text and
audio/video. For the online services we do not plan a visualisation of time-overlap.
In the Paunaka use-case, the project started data processing in ELAN. Focusing only on the here relevant aspects of the project's workflow, this means that intervals were created and first segmentations were made. Project members then create several note tiers, corresponding to the Toolbox 'nt'. In a first go-through of the material, the researchers use the first note tier to report unknown words or morphemes, or other phenomena that do not allow the initial translation of the material. A common procedure, which we assume is quite general for group work, is to leave comments about initially troublesome features or pointers to interesting grammatical formations so that they at a later point can be selected for integration into the grammar that the project plans to write. Next to the translation and the note tier, it is not unusual to create a loan word tier. A typical working procedure in the early work with ELAN is that transcriptions are corrected together with native speakers, and that notes concerning these corrections are entered on separate tiers, reflecting the comments of different informants. Through the present development we have made sure that the information resulting from the initial processing of data in ELAN is mostly preserved under data exchange. This means that next to the transcribed text and its translation, the speaker identification and the time alignment, also the information coming from note tiers is preserved. However, for the time being, we will have to write all notes, independent of their type, to the already existing note tier in TypeCraft.

We should again mention that the project does not plan the export of audio and video data to TC, that is, the processing of media information will continue to happen locally. Yet, there are ways to make also this data available online for representational purposes, something we will not discuss here further.

In summary, limited reconfiguration using a simple RESTful web service in addition to an improved TC-XML format makes it possible to allow users of our systems to map ELAN tiers (or tier types) to TC tiers and vice versa, to import TC texts into ELAN together with their corresponding speech recordings. Most importantly time alignment information and speaker information is now retained when data is shipped from one system to the other. Manual export and import of the type described by the PDP is through the development presented here no longer necessary.

5. Outlook
Software development profits from an active and articulated user community. Through an approach that targets problems that have become apparent through use-case analysis, small but efficient changes can be made. However, to move towards linguistic VRE, more is needed and mostly these are standards and more

standards. With the CLARIN endorsed Metadata Framework (Broeder et.al, 2010) metadata standards have already been addressed. The same is true for a general XML structure for IGT (Bird et.al, 2003). A true challenge is still linguistic modeling. The GOLD ontology, as an RDF resource, is the most suitable starting point for the present project. However, a description of the development of ontology integration, as well as of an improved data parsing and search functionality is well beyond the scope of this paper.

6. Acknowledgements
We would like to thank the Paunaka Documentation Project and especially Lena Terhart for discussion and comments. Paunaka is a South Arawakan language spoken in Bolivia. More information about the Paunaka project can be found on TypeCraft under: Paunaka Documentation Project.

7. References
SOA meets Relation Extraction: Less may be more in Interaction

Andre Blessing, Jens Stegmann, Jonas Kuhn
Institute for Natural Language Processing (IMS)
Universitaet Stuttgart
Pfaffenwaldring 5b, 70569 Stuttgart, Germany
andre.blessing, jens.stegmann, jonas.kuhn@ims.uni-stuttgart.de

Abstract
This paper presents a prototype system which is based on RESTful web services and implements an interactive relation extraction system: IMS-ADAPT. Our aim is to show how the dynamic interaction between such software and human users from the digital humanities can be brought about. Specifically, the interactive aspect here consists of (a) allowing users to define their own relation types which can be extracted from any kind of text provided, (b) providing means for the user to rate the quality of the results delivered by the system and (c) using those ratings in order to retrain the underlying models and therefore deliver improved results. This is crucial for the digital humanities because of the diversity of textual domains for which no adequately pre-trained models are available. Technically, the setup of IMS-ADAPT involves a (re-)trainable web service on top of a web service processing chain that brings about automatic linguistic annotation on several levels. However, the complexity and the results of the latter are hidden from the user. Furthermore, no special software is needed in order to interact with the system: any HTML 5 compatible web browser is sufficient. The web services used are available as part of the CLARIN research infrastructure. Aspects of the RESTfulness of the services and the SOAness of the architecture are discussed.

Keywords: RESTful, Interactive, Web service, relation extraction

1. Motivation

1.1. Dynamic Interaction
SOAs and web services tend to neglect issues in the dynamic interaction between users and software. The main reason for this is that web services are a means for the communication between computers. The user’s role is usually restricted to searching the suitable services for the task at hand. He may also provide the necessary information in order to start a series of calculations. But from then on, the progress of things will rather be determined by the web services alone. Of course, the services may interact with each other in the sense of sharing intermediate results, for example, but usually no further interaction on behalf of the user will be involved until he is presented with the final results of the calculations.

However, interaction can be of profit for both the users of an infrastructure as well as for the providers of the tools. This holds especially concerning the domain of the digital humanities, e.g., the vast possibilities associated with the improvement of automatic annotations of textual data via domain adaptation. Many textual resources in the digital humanities belong to very specific domains (e.g., German patent specifications from the 19th century) where pre-trained models will usually not be available and the application of generic models available tends to deliver rather disappointing results. The idea is to make use of the direct feedback of users: false positives and false negatives could be marked for just what they are by expert human annotators. This would help with both the current results for the expert who provides the feedback, if retraining can be applied, as well as with future processing, as the provider of the tools can make use of the improved models created during the process. The latter would also help all future users of the tool, of course.

1.2. Less may be more

The CLARIN infrastructure (Vradi et al., 2008) provides many services which can be combined into linguistic processing chains or pipelines, c.f. the services available via the WebLicht platform (Hinrichs et al., 2010) for specimens of RESTful web services or GATE (Cunningham et al., 2011) services for ones that comply with the SOAP approach. However, this diversity of possible configurations may seem daunting for researchers with a background in digital humanities. Of course, the availability of GUI-based platforms or the visualization of results as implemented in WebLicht helps, but still the linguistically motivated outputs of the services may often be too technical to understand for non-linguists with an interest in very specific research questions. Our framework shows how the infrastructure can be used as a kind of black-box where the user will only be confronted with a reduced output in the sense of the specific relation data he is interested in (and which he therefore can interpret easily), while still making use of the additional information provided below the visible surface.

2. Background

We will not introduce new methods for linguistic processing or relation extraction in this paper. We rather combine already existing methods via web service implementations and show how they can be used in an interactive way. Nevertheless, we do provide a very brief introduction to the methods used and give some pertinent references.

2.1. Automatic Linguistic Annotation

Our application is an automatic relation extraction system (Zhao and Grishman, 2005). The performance of such systems can be improved if deep linguistic analysis is used (Blessing and Schütze, 2010a). Such analysis needs several
processing steps on different levels of linguistic analysis as morphology, syntax and semantics. Tools involved may be tokenizers, lemmatizers, (dependency) parsers and named entity recognizers among others.

2.2. Relation extraction
Most analysis tools are entity-centric which means that the user has to provide a specific term which he is looking for. The tool then returns a set of results concerning that term (e.g. web search engines). If a user were to search for a higher level concept like “all the people who were born in Berlin” or “all academic researchers who were influenced by a specific person” however, relation extraction would be the preferred method in order to enable such queries. In relation extraction, so called extractors classify if two mentions\(^1\) describe the queried relation in the current context\(^2\) or not.

2.3. Research Infrastructures: CLARIN
One of the features of the CLARIN infrastructure is that it provides automatic linguistic annotation tools as web services concerning the different levels of linguistic analysis mentioned above. In particular, many such web services are integrated into CLARIN-D’s WebLicht platform. The available services can be easily integrated into specific applications like the interactive relation extraction tool IMS-ADAPT that is our focus here.

Advantages of using the infrastructure include: sustainability of the tools and resources, reproducibility of the results, availability of high computing power and the avoidance of local downloads and installations among others. Most parts of the infrastructure are available to academic users already.\(^3\)

---

\(^1\)Mentions are named entities mostly, but they can also be common nouns like a profession, for example.

\(^2\)In our case we will conceive of the context as just the current sentence.

\(^3\)Legal restrictions may apply for specific resources.

---

3. The Interactive Text Analysis System

3.1. Overview: The Architecture
Figure 1 shows the architecture of IMS-ADAPT. The RESTful web services are the core components of our system. Each of the services has a specific role in processing the given textual input. The relation extraction system takes the aggregated results as input in order to calculate a feature representation. This representation is the input of the classifier which decides which relation type the given mentions matches.

3.2. The Aspects involved

3.2.1. Linguistic annotation
The system is generally open to any kind of textual input. In the prototype we implemented two interfaces for text input. The first provides a simple text area in the web browser in which the user can write the text he wants to analyse. The second is a little more sophisticated and integrates all Wikipedia articles of the German edition in our prototype. The Java Wikipedia Library (Zesch et al., 2008) allows the extraction of structured and unstructured information from Wikipedia dumps. In this work we are only interested in raw text which equates unstructured data and category information. Other structured data like headings, hyper-links, template information are not used at this stage.

We use a mixture of services which are already part of WebLicht and some additional services to annotate the data.

- Sentence Splitter and Tokenizer segments the raw input text in sentences and tokens.
- Part-of-Speech Tagging (Schmid, 1995) annotates each token with its POS-tag.
- Dependency Parser (Bohnet, 2010) finds for each token its unique head token including a dependency label.
• Named Entity Recognizer (Faruqui and Padó, 2010) annotates named entities in the sentences which consist of sequences of one or more tokens.

3.2.2. Candidate selection
We use the UIMA (Ferrucci and Lally, 2004) framework for the next step. The advantage of UIMA is the simple integration of different annotation layers. That means that there is no restriction with regard to overlap or multiple annotations. That allows us to easily integrate all the annotations attained, even if some of the layers have different annotations. Since our classification is implemented as a web service in UIMA, we had to implement a wrapper that converts TCF data (Heid et al., 2010), which is the output from the WebLicht web services, into the UIMA type system.

Our service uses a simple heuristic to find relevant mentions which are used for the classification. To use all binary token combinations, including the possible multi-word expressions, is not feasible. The output of the named entity process provides a good baseline, but it has no perfect coverage. Additionally, we take common nouns and pronouns into account for the candidate selection. In future versions, we may make use of more sophisticated mention detection approaches as described in Florian et al. (2010).

3.2.3. Relation classification
ClearTK (Ogren et al., 2008) provides many wrappers for machine learning inside an UIMA chain. In particular, it provides several feature extraction classes which are highly expandable. We created extractors for each of the previously described annotation layers, populating a feature space for each pair of the candidates (Zhou et al., 2005). A maximum entropy classifier (McCullum, 2002) is used to classify such representations into relation types.

3.2.4. (Re-)training
One distinctive feature of IMS-ADAPT is the aspect that the system can be tuned to the user’s needs. Users are not only able to extract new relation types, but they can also do corrections regarding the results where necessary. This means initiating a retraining process on the fly.

3.2.5. Annotation
The annotation can be brought about interactively by clicking on the corresponding mentions in the text and choosing the correct relation type from a list. Each annotated instance is stored persistently including the sessionID. This is important for the retraining since the complete training data can be generated. Another advantage is that if the underlying pipeline were to be changed or new features were to be used, the processing and feature extraction could also be redone. Therefore a sticky bit is used to flag the consistency of the pre-stored features.

3.2.6. Rating
All positively classified relations are presented within a table. The user interface provides the opportunity to rate each of the results. Therefore the user can validate or change the classified relation type.

3.2.7. Model management
Due to the retraining mechanism, not one, but many models are used in this application. Each retraining creates a new model, so they have to be managed.

3.2.8. Ajax interaction
The web application is implemented via the JQuery framework which simplifies the Ajax interaction and it provides widgets for the presentation of the data.

Each user interaction is mapped to a specific web service call. They can be divided into calls which modify the model and calls which query for new results. We use four variables to model all states. (i) the sessionID to identify the session, (ii) a hashCode which is a fingerprint of the current text, (iii) sentenceNumber which is displayed in more detail, and finally (iv) a modelID of the used parser model.

3.3. An Illustrative Example
We illustrate our system by a concrete example.

The remaining figures show screen shots from a step by step runthrough, working with an example that involves the relation between famous scientific advisors and their doctoral students.

Figure 2: STEP 1: Annotation of the relation between Kurt Gödel and Hans Hahn as positive instance.
The user starts from scratch by initiating a new session, which means that the system has no knowledge about any kind of relations yet. The user can insert new relation instances into the system by selecting them in the text (Figure 2). Each instance is stored persistently in a relational database, so the user may stop his session and continue it at a later time. The user uses three sentences\(^4\) for the training. Those sentences are the input for the linguistic annotation processing chain which is part of Figure 1. For the user, however, this processing chain is a black box, i.e., he does not need to know anything about NLP processing. The user has to annotate at least one positive (Figure 2) and one negative (Figure 3) relation instance. In Figure 4 the gray box on the right side shows a table which includes all positively classified pairs of the extraction. In this case the classifier produces many false positive instances, however. This problem can be corrected by the user. In our case the user corrects three of them before he initiates the retraining mechanism. Now there is only one positively classified instance left. All other instances are classified as negative and are not visualized. Remember, the classifier is now trained on one positive and three negative instances. To further improve the classifier, the user annotates the instance involving Carl Menger and Louis Cauchy as positive (Figure 5). The result of the retrained extraction shows that the classifier again produces many false positives. Until now the user has only evaluated his process against the training set. In order to show that the system can also act on sentences which are not part of the annotation, two other sentences\(^5\) are used for means of evaluation. The user is not allowed to rate and modify the results for the evaluation set. It shows that the learned classifier produces a false positive on the evaluation set also. Finally, the user corrects the falsely positive instances of the training set. Then he gets two truly positive instances. This is no big surprise because both are part of the annotated training set already. Altogether, this set now contains 2 positive and 8 negative instances. Finally, the user considers his evaluation set again. The results are presented in Figure 6. Now the trained classifier extracts both relations correctly. No false positive instances are extracted.

\(^4\)Kurt Gödel earned his doctorate under Hans Hahn. Carl Menger studied with Hans Hahn and earned his doctorate in 1924 at the University of Vienna. Bruno studied under Louis Cauchy, Charles Hermite was his classmate.

\(^5\)Riemann attended lectures on partial differential equations by Peter Gustav Dirichlet. Kurt Alder received his Ph.D. under Otto Diels.
tative evaluation (precision, recall, F-measure) at this point. For the given small domain, the extractor delivers promising results, but the trained model is not yet very robust. Our empirical investigations showed that a minimum of 100 instances are needed to get a more complete extractor. In the future we will also integrate an automatic annotation method (Blessing and Schütze, 2010b) to annotate relation instances in the given texts during the training phase. This may be helpful in creating a base system that already provides an extraction model on a high confidence level.

We have described experiments with German texts here only. However, the general approach does not depend on the specifics of a particular language and can hence be applied to other languages as well. Of course, this requires that the components of the linguistic annotation processing chain are available. Our plan is to do further experiments with Chinese, French, English and German.

The main goal is to bring relation extraction to the digital humanities in a way that will be useful and non-complicated for the target group (hence, both interaction and “less may be more”). Nevertheless, research in relation extraction may also benefit from such an interactive tool, since it will help to explore new domains in a quick manner. In contrast to the normal approach, such a tool enables easy evaluation at the instance level while typically evaluation is mostly done in a quantitative manner without considering single instances.

Aspects of the CLARIN research infrastructure that may be exploited in future versions are persistent identifiers (PIDs) (ISO TC37/SC4, 2011) for means of reference to the resources involved, as well as the results produced. CMDI metadata (Broeder et al., 2010) should also be provided. It would be desirable to make both the results and the improved models available to the scientific community. Options here include personal workspaces of the users involved or the repositories of CLARIN resource centres (Wittenburg et al., 2010).

Both the SOAIness of complex software architectures (Melzer, 2010) and the RESTfulness of web services (Burke, 2009) are somewhat debated topics. The very term “service-oriented architecture”, for example, is regarded as an industry buzzword by at least some authors. It is also often understood to presuppose the use of SOAP web services or RPC-style architectures. However, this need not be the case. (Richardson and Ruby, 2007) introduces the concept of a “resource-oriented architecture” which makes use of services that adhere to RESTful principles in a strict sense. This involves, for example, sticking to the original semantics of HTTP methods (GET, HEAD, PUT, POST, DELETE) for what they call the method information (what to do with the data) and to URI paths for the Scoping information (which part of the data set to operate upon) and not to mix these two aspects up. Please note that the “RESTful” web services in IMS-ADAPT do not adhere to those strict constraints at the moment. Hence, our system would rather qualify as a REST-RPC hybrid architecture in Richardson’s and Ruby’s sense. However, this may apply to the whole WebLicht platform for the time being.

As mentioned above, we are still in the process of testing and developing the system and there is still ample room for improvement, e.g., usability of the GUI, refactoring the code, scalability of the system etc. Therefore, the software will probably not be released to end users before the beginning of 2013.

5. Conclusions

We presented a prototype system which is based on RESTful web services and implements an interactive relation extraction system: IMS-ADAPT. It shows how the dynamic interaction between such software and human users from the digital humanities can be brought about in a suitable way.

6. References


Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In Chu-Ren Huang and Dan Jurafsky, editors, COLING, pages 89–97. Tsinghua University Press.

Daan Broeder, Marc Kemps-Snijders, Dieter Van Uytvanck, Menzo Windhouwer, Peter Withers, Peter Wittenburg, and Claus Zinn. 2010. A data category registry- and component-based metadata framework. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapis, editors, Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC’10), Valla, Malta, may. European Language Resources Association (ELRA).


Radu Florian, John F. Pitrelli, Salim Roukos, and Imed Zitouni. 2010. Improving mention detection robustness

---

6 Some videos can be found at clarin01.ims.uni-stuttgart.de/demo/videos.


Marie Hinrichs, Thomas Zastrow, and Erhard Hinrichs. 2010. WebLicht: Web-based lrt services in a distributed escience infrastructure. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC’10), Valletta, Malta, may. European Language Resources Association (ELRA).


Peter Wittenburg, Nuria Bel, Lars Borin, Gerhard Budin, Nicoletta Calzolari, Eva Hajicova, Kimmo Koskenniemi, Lothar Lemnitzer, Bente Maegaard, Maciej Piasecki, Jean-Marie Pierrrel, Stelios Piperidis, Inguna Skadina, Dan Tufis, Remco van Veenendaal, Tamas Vradi, and Martin Wynne. 2010. Resource and service centres as the backbone for a sustainable service infrastructure. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC’10), Valletta, Malta, may. European Language Resources Association (ELRA).


Interoperable Digital Musicology Research via music21 Web Applications

Michael Scott Cuthbert, Beth Hadley, Lars Johnson, Christopher Reyes

Department of Music and Theater Arts
Massachusetts Institute of Technology
Cambridge, Mass., USA

{cuthbert, bhadley, larsj, chrisrr}@mit.edu

Abstract

Digital humanities practices applied to musical scores have the potential to open up vast new datasets and avenues for research in musicology and are beginning to transform the field of musical research. Yet beyond the common difficulties of all digital humanities projects, significant problems arise in digital musicology that are unique to the structure of musical scores and the lack of available tools for manipulating scores. Performing analysis tasks often requires specialized tools that have high barriers to entry, such as compiling, choosing a particular operating system, and converting data between divergent formats. The “weapps” module of the open-source music21 toolkit provides the architecture to connect various digital musicology projects. It makes standard but time-consuming musicological tools available to less technologically sophisticated users while providing tremendously varied developmental options to technically-inclined researchers. The authors propose a JSON format for encoding both score data and manipulations to/analysis of scores that can easily be used by backend systems besides music21, whether specialized for musical analysis or for other digital humanities and machine learning tasks. The article ends by stressing, with examples, the continued need for standalone musical analysis systems even in a world of easily available web systems.

Keywords: Digital Musicology, scores, music web systems

1. Background: Digital Musicology today

Musicological research, particularly of western classical music, has long relied on the intense study of small numbers of individual works looking for particularly distinctive, inspiring, or unusual moments in single scores. Comparative research among scores or repertories has been out of favor since the middle of the century (Cook, 2004) because of inaccuracies (particularly a bias towards western, often Germanic, forms) and an inability to cope systematically with large corpora. Computational approaches to repertories have been embraced in the past twenty years by several projects, but they have not been the norm in musicology due to the difficulty in obtaining computer-encoded versions of scores and in particular the absence of easy-to-use software packages for examining, analyzing, and manipulating these scores.

Music21 (Cuthbert & Ariza, 2010) is an open-source object-oriented toolkit built in Python for digital and computational musicology. The toolkit builds on the strengths of earlier applications, such as the Humdrum toolkit (Huron, 1997), but adds to it an object-oriented framework that allows users to find desired data more quickly and easily. First released in 2008 for all standard operating systems (including Windows, OS X, and Unix variants), the music21 toolkit is now in its fourteenth release and the first non-beta version was released on June 14, 2012. The rapid adoption of music21 for use by computational musicology projects has made it close to a new de facto standard for computer-aided work, but difficulties in increasing its use among less technically minded musicologists has necessitated recent work in building web applications to take advantage of its power while making it simpler to use and eliminating the need for installation.

2. The Present and Future Need for Web Applications in Digital Musicology

Over the past fifteen years, web applications have dominated the field of computational musicology tools by providing musicologists with immediate access to music datasets and simple analytical tools. However, without providing an infrastructure for customization, research is commonly limited to the materials provided through the site, leaving little room for creative development and investigation.

We take the project Kernscores (Sapp, 2008) as exemplifying both the great potential and binding limitations of current musicological web application systems. Like most digital musicology sites, it uses URL-encoded commands accessed via websites to transform data into a variety of musical formats and give the results of simple analytical processes such as key analysis or piano roll diagrams of the pieces. These analyses have great potential, yet the currently available methods come with significant drawbacks. The most obvious is that the tools can only be applied to the scores made available by the developers—a problem shared with nearly all similar sites. These scores need to be encoded in formats that are either not in general use (e.g., Humdrum/Kern) or cannot represent standard notational symbols that are important to researchers and performers (e.g., MIDI which stores the notes D\textsuperscript{4} and E\textsuperscript{4} as the same pitch and cannot encode tempo markings such as allegro moderato). More significantly for developers, the URL-encodings are not documented and the code for the backend systems are generally not released, making it impossible for outside developers to expand the system.
Although music21 has been designed to be easy enough for a professional musicologist without previous programming experience to learn to use in a few weeks, even this requirement presents too high of a bar for many users. Web applications offering even simple commands that process user-uploaded data and return results designed for users to view or hear without further computational processing can be incredibly valuable to researchers of all technical backgrounds.

Additionally, a service-oriented architecture (SOA) allows more advanced web developers to easily integrate complex computational methods into their own web applications. Web applications are currently being produced for many platforms, and the easy integration of computational back-end tools would make such applications even more powerful.

Finally, computer scientists working on improving generalized algorithms for classification of data are another untapped audience needing web applications for musical scores. A researcher wishing to see if her algorithm for clustering data can also work on musical scores will seldom have time or expertise to learn a specialized system for feature extraction of musical data; she and her team will be searching for already created sets of feature data (such as the Million Song Dataset gives for audio data (Bertin-Mahieux, et. al, 2011)) or a way of easily obtaining these features from data gathered from other sources. A service-oriented architecture is critical for the needs of researchers only tangentially connected to digital musicology. Such a web architecture would allow this researcher to leave specialized feature extraction tasks to musicological experts and focus on her own expertise in algorithmic design.

3. Music21 Web Applications

Since its conception, music21 has provided a modular infrastructure for manipulating and analyzing scores. This makes it ideal for providing the link between accessible web environments and sophisticated music research. Beginning with the 1.0 release, music21 includes a module designed for developing a service-oriented architecture utilizing the full suite of analysis tools provided by music21. The webapps SOA eliminates many hurdles to utilizing the music21 toolkit by placing it in a web-based setting, yet still provides users and developers unparalleled freedom.

Music21 web applications import and export data in a variety of formats, catering to a wide range of user communities. Computer-aided musicology has always depended on utilizing various data formats to encapsulate the vast variety of information extracted from music queries. For example, music21 web applications export textual and numeric data in formats ranging from simple text or JSON, to .csv and spreadsheet formats, to graphical plots. It supports numerous music notation formats, including MusicXML and Lilypond as well as MIDI and even Braille translation. Additionally, these web applications can take advantage of being embedded in modern web browsers by enabling live, editable notational output through the Noteflight (Berkovitz, 2008) Flash-based plugin and manipulable high-quality Canvas and SVG graphics through the open-source VexFlow (Cheppudira, 2010) JavaScript library. Users can run web applications using the 10,000 scores in the music21 corpus or assemble their own corpora. Providing such versatility to users ensures a broad compatibility with other music-based websites and independent stand-alone music applications.

Music21’s implementation of the VexFlow JavaScript library is particularly important for future adoption of web applications for musical scores. Prior to the creation of VexFlow, no freely available way of rendering musical data on the Internet as a viewable score was feasible. Previous attempts such as the Mediawiki extension to Lilypond (www.mediawiki.org/wiki/Extension:LilyPond) posed serious security hazards and required translating existing MIDI, MusicXML, and other score files into a new format. With music21’s adoption of VexFlow and the SOA, any Internet user can render a data file in one of numerous formats as a score for viewing within a web page or other JavaScript/HTML5-compatible application. Future work on this module will add JavaScript callbacks from the VexFlow code to the music21 SOA enabling interactive musical markup, annotation, and editing.

4. Example Uses of Music21 Web Applications

The music21 service-oriented architecture can be used for a variety of purposes. Applications can be developed in which a simple click of a button can trigger advanced analysis routines. For example, commands easily automated via music21 webapps include output of range and key data, detection of contrapuntal anomalies such as parallel and direct fifths, transformation of a collection of pieces to the same key or meter, and various feature extraction methods. One commonly used method of music21 is the “chordify” command which takes in an entire score, measure range, or collection of parts, and reduces it to a series of chords representing the music sounding at each moment in the score. This reduced score is much easier to understand at a quick glance than a full score. The tremendous modularity innate in music21 methods and objects allows identification and analysis of music scores not possible via static interfaces similar to previous musicology sites where both user input and analysis tools are limited.

For the advanced user, the music21 service-oriented architecture may be used as a platform upon which more complex web applications may be built. An example demonstrating the versatility of the webapp architecture
coupled with the interoperability offered by the toolkit is a tool we created for analyzing a student’s music theory assignment for contrapuntal writing errors (See Figure 1). Using the music21 webapp architecture, the student’s assignment passes easily from third-party notation software to analysis methods within the toolkit that identify areas of concern in the work. The tool then returns a pre-graded score, either to the student or the professor, along with text describing each error. Of particular interest to educators is the automatic identification of violations of common-practice rules of counterpoint, such as motion by parallel fifth or dissonant harmonic intervals. In developing this app, we extended and customized the existing music21 methods of analysis, creating specialized music21 objects to encapsulate individual elements within the score, such as linear segments, vertical slices of simultaneously sounding objects, and two by two matrices of notes. Elements identified as errors were colored, and text output further explained the algorithm’s observation (such as between which notes the parallel fifths exist, or the name of the dissonant interval). This data is packaged into a JSON data structure and provided directly to the client (either a web browser or the open-source MuseScore notation software (Brontë, et. al., 2008) completing the service to the user. This service-oriented architecture for music is under consideration to become the backbone for music courses in the developing MITx/EdX open educational platform.

The core of the module involves two objects: an Agenda, and a CommandProcessor. An Agenda object is a dictionary-like structure that specifies data input, requested commands, and a desired output format. A CommandProcessor object takes an Agenda, parses the data input into a format compatible with music21, safely executes the commands, and generates the output.

These objects are used in a server application compliant with the Python WSGI interface, a portion of which is shown below. This application can be enabled on an Apache/modWSGI server by adding a few lines to the httpd.conf, as Figure 2 demonstrates.

```python
from music21 import *
agda = webapps.makeAgendaFromRequest(requestInput, environ)
processor = webapps.CommandProcessor(agda)
processor.executeCommands()
(responseData, responseContentType) = processor.getOutput()
```

**Figure 2**: Code for setting up a music21 web application.

The code shown is representative of the steps involved in processing a request. First, the POST data and GET data from the request are combined into an Agenda object. The post data can be url-encoded form data, multipart form data, or a JSON string. In this way a single mount point can be used to serve a variety of request types.

Figure 3 shows an example of the typical JSON formatted input to the webapp interface. This text encodes commands to use music21 parse a Bach chorale from the corpus, transpose that chorale by a perfect fifth, then return the chordified score in VexFlow format. Should the user wish to view their score in a different music21-supported output format, such as MusicXML, Braille, Lilypond, or MIDI, only a one-word change to this JSON format is necessary.

```json
[ "dataDict": { "workName": { "data": "'bwv7.7'" } },
  "commandList": [
    { "function": "corpus.parse",
      "argList": [ "workName" ],
      "resultVar": "chorale" },
    { "caller": "chorale",
      "method": "transpose",
      "argList": [ "'p5'" ],
      "resultVar": "choraleTransposed" },
    { "caller": "choraleTransposed",
      "method": "chordify",
      "resultVar": "choraleChordified" } ]
```

5. Service-Oriented Architecture in music21: the webapps library

To enable development of interoperable webapps utilizing the full suite of computational tools, the music21 toolkit includes an extensive service-oriented architecture. It consists of Python classes and functions used to parse a server request, execute the desired commands, and return content to the user in an appropriate format. The flexible nature of the architecture allows it to use a single URL to handle any requests to the server wishing to use music21. These requests can come from a variety of sources, including HTML form POSTs, AJAX requests, or even web requests from a plugin in an open source notation application. The commands used by the requests can either be commands built in to music21 or custom commands created by the user.

A full suite of computational tools, the P5.js library and the MuseScore browser plugin, are utilized to develop these webapps. A webserver, either Apache/modWSGI or NGINX (includes an extensive service-oriented architecture. It consists of Python classes and functions used to parse a Bach chorale from the corpus, transpose that chorale by a perfect fifth, then return the chordified score in VexFlow format. Should the user wish to view their score in a different music21-supported output format, such as MusicXML, Braille, Lilypond, or MIDI, only a one-word change to this JSON format is necessary.

**Figure 1**: Screenshot displaying the use of this webapp embedded as a plugin for the open-source notation software MuseScore used as part of an automatic “pre-grading” system for music theory teaching. A full video showing this demonstration is available at http://www.youtube.com/watch?v=5VBfag3YWls.
Figure 3: An example JSON request to return a Bach chorale (BWV 7 movement 7) as a chordal reduction, transposed up a perfect fifth as a VexFlow Canvas.

If an appName is specified in one of the request fields, additional data and commands are added to the agenda. This flexibility allows for the creation of applications in which the majority of the commands are specified by the server and only a subset of the data is specified by the user for each request. For instance, by specifying a “featureExtractorApp,” as the appName, each request would only need to include the name of the feature they would like to extract and the zipfile containing the scores, without explicitly needing to specify the individual commands necessary for feature extraction and machine learning of musical data (Cuthbert, Ariza & Friedland, 2011).

The command processor then takes the agenda and parses its input data into primitives or music21 objects. Although most of the values arising from POST and GET fields start as type string, the processor will determine if the string was intending to be a number, boolean, list, etc. and save its value accordingly. Additionally, music21 is compatible with a wide variety of symbolic music formats (MusicXML, Humdrum/Kern, abc, MIDI, etc.) and can convert fields of those types into corresponding music21 objects.

Next, the command processor executes the commands specified by the agenda. To avoid the security risk of executing arbitrary code while still maintaining the flexibility of the architecture, the server checks that each requested command is allowed to be executed on the server and only interacts with a set of variable bindings internal to the processor.

Finally, the processor generates the output of the results. The elements of the Agenda specify the output format which can be of a wide variety of types, including an html page with a score displayed in an SVG or Flash embed, a downloadable MusicXML or comma-separated value file containing analysis results, or simply the raw JSON of selected variables that can be decoded using JavaScript in a client HTML page.

A video demonstrating this system is viewable at http://ciconia.mit.edu/feature-extraction.wmv and the software itself is at http://ciconia.mit.edu/music21/featureapp/uploadForm. Examples of sample webapps are available at http://ciconia.mit.edu/music21/webapps/client/.

6. Cloud Computing and Web Services

Repertorial analysis requiring the best analytical methods might run hundreds of times per score on a corpus of tens of thousands of scores. The music21 service-oriented architecture provides the infrastructure necessary to command complex and computationally intensive analysis. However, such tasks might take hours to run and provide little to no real-time feedback during processing. Thus, it has become apparent that integrating more powerful processing power would make music21 webapp services even more accessible. Our recent work has included research into providing cloud computing functionality to music21 analysis routines via Amazon Web Services and the Python map-reduce module, mrjob (Yelp, 2009)

Any webapp routine that can be abstracted into multiple independent tasks benefits greatly from the additional computing power provided through cloud computing. Processing time can be greatly decreased by implementing a standard MapReduce algorithm (Dean & Ghemawat, 2004) to distribute processing of hundreds or thousands of files over a network of independent computers. The Python library mrjob accesses Amazon Web Services and can be utilized to prepare MapReduce algorithms employing music21 analytical methods. Due to the modularity of the music21 service-oriented architecture, webapps can be developed to provide quicker access to music21 processes via the Amazon Cloud. These webapps would route input data from the user, such as a corpus of music files, establish an SSH connection with EC2 instances provided by Amazon, deploy the job specified, and wait while the data is processed. The resulting output would be passed back to the web interface and displayed to the user in a fraction of the time it would take the user to run the same analysis algorithm on a local computer. After implementing this process in a test run examining bass motion over thousands of popular music leadsheets we recorded promising improvements in the time taken in processing many scores.

By adding the component of cloud computing to our already existing service-oriented music21 architecture, the limit of computational power and time is tremendously alleviated. Integrating cloud computing into a pre-existing web service allows musicologists great freedom in both developing and running research studies.

7. Limitations of Web-systems and the Co-existence of Stand-alone systems in Digital Musicology

While web-based applications will open up many new avenues for research and data exchange, downloadable applications to be run on individual users’ systems will need to continue to be developed. To start, unless a system is implemented entirely in JavaScript, users’ queries need
to be parsed and understood by a traditionally based backend system. As long as such an engine exists, there is little to be gained by limiting programmers’ access to this backend, and continued development of server-based systems demand tests that can be executed outside the web system. More complex queries that nest the filtering of musical objects and annotations are much more easily created with short scripts that have direct access to the musical objects. For instance, the research question “does Mozart cadence on first-inversion triads more often on strong beats vs. weak beats in his sonatas written earlier in his life?” is easily answered in music21 by writing a short module using nested “if,” “break,” and “getElementsByTagName()” statements. A similar web query would be so complex that the designing the command would be a more difficult process than installing the system and writing a script by hand. A researcher must carefully evaluate the advantages to developing a web-based application versus stand-alone scripts, depending on their individual goals, technical background, and time constraints. In addition, while HTML5 simplifies many programming tasks and moves them from the server to the client, it does not contain support for microphone or MIDI input without external plugins (usually Adobe Flash). Thus for many realtime audio and musical applications, standalone versions of the software are needed.

Security and privacy concerns are two other factors to consider when evaluating whether to develop a web-based platform or stand-alone application. Complex queries may require access to the file system or generate huge temporary files, both of which can introduce security holes. Users may not want to trust their private research data to be uploaded to a web server not under their control. This desire may seem paranoid when the only data are musical scores, but music21 can also correlate score data with physiological response data from listeners and reported musical preferences, all of which could be used to deanonymize survey data. Thus, both security and privacy concerns promote continued development of stand-alone applications.

8. Conclusion and Future Work

Fundamentally, the goal of the music21 service-oriented architecture is to provide researchers from a wide range of technical backgrounds and disciplines access to powerful musical analysis tools conveniently, efficiently, and quickly. Future development includes expanding the webapp infrastructure to implement a larger suite of customizable music21 features along with improved computational power via the Amazon Cloud. Modules within the toolkit that require extensive external dependencies, such as “Gregorio” the LaTeX chant notation software, can be adapted to use the SOA to render the notation on a properly equipped external server. Work in the near future will also include extensions to our VexFlow web architecture to enable interactive annotation and editing of SVG-rendered musical scores. The possibilities of service-oriented architectures in computational musicology toolkits such as music21 are only beginning to be tapped. In the near future music web applications will be among the most important contributors to the exciting cross-disciplinary advancements emerging in digital humanities.

9. Acknowledgements

The development of music21 and its web applications has been funded by a NEH Digging into Data Challenge Grant as part of the ELVIS project and by the Seaver Institute. Additional support has been provided by the School of Humanities, Arts, and Social Sciences at MIT and support for this presentation has been provided by the Germany Seed Fund of the MIT International Science and Technology Initiatives.

10. References

Integration of WebLicht into the CLARIN Infrastructure

Emanuel Dima, Erhard Hinrichs, Marie Hinrichs, Alexander Kislev, Thorsten Trippel, Thomas Zastrow

University of Tübingen
Seminar für Sprachwissenschaft
firstname.lastname@uni-tuebingen.de

Abstract

This paper describes the ongoing work to integrate WebLicht into the CLARIN infrastructure. It introduces the CLARIN infrastructure for scholars in the humanities and social sciences as well as WebLicht - an orchestration and execution environment that is built upon Service Oriented Architecture principles. The integration of WebLicht into the CLARIN infrastructure involves adapting it to the standards and practices used within CLARIN, including distributed repositories, CMDI metadata, and persistent identifiers.

Keywords: WebLicht, CLARIN, SOA

1. Motivation

The ESFRI project Common Language Resources and Technology Infrastructure (CLARIN)\(^1\) has the goal of providing an integrated and interoperable research infrastructure for scholars in the humanities and social sciences. One of the main challenges in constructing this CLARIN infrastructure lies in the integration of language data and of services that allow scholars to search, annotate, enrich, and visualize language data on a large scale. CLARIN aims to make such language processing tools and services available for novice users, eliminating technical obstacles that are often encountered. In addition, these tools and services should be capable of operating on data from various sources and should be combinable into complex processing workflows.

2. CLARIN Metadata

An important part of the CLARIN infrastructure involves creating language processing tools and making them available to the larger academic community. Equally important, however, is devising a framework which allows these tools to be discovered and used with minimal effort on the end-user’s part. This framework relies heavily on the use of a common metadata format. This section describes the metadata within CLARIN which enables WebLicht (Henrich et al., 2010) to discover language tools and create processing chains for annotating text corpora. Section 3. describes in detail how WebLicht will use CLARIN’s metadata for building processing chains.

2.1. Persistent Identifiers

A persistent identifier (PID) is a unique character string used to identify a digital object (ISO 24619:2011, 2011). PIDs can be used as references to digital objects in a wide range of documents. The URL of the actual digital object can be obtained simply by resolving its PID by means of a designated service. PIDs are widely applied within the CLARIN metadata to refer to webservices, resources, etc.

2.2. CMDI

The CLARIN project utilizes the CMDI\(^2\) (Component Metadata Infrastructure) metadata format, which provides a unified library of building blocks (called components). These components can be reused to compose a wide range of increasingly complex metadata components. Each of the elementary components is normally linked to a field descriptor in a registry of concepts and terminology widely used by the linguistic community, such as ISOcat\(^3\) (ISO 12620:2009, 2009). The CLARIN infrastructure provides the CMDI Component Registry\(^4\), in which arbitrarily complex metadata components can be added for any resource type, reusing existing components (Broeder et al., 2010). The advantage of this building-block approach is that it imposes no limitations on the expressiveness of the metadata. CLARIN requires that every resource is described with CMDI metadata, and that this metadata is assigned a PID and made available for harvesting via the OAI-PMH protocol.

2.3. Center Resource Repositories

Each individual CLARIN center will operate its own repository for storing its resources and metadata. Currently there are nine CLARIN centers, located at academic and research facilities. When a center wishes to inform the wider CLARIN community about one of its resources, it makes the corresponding metadata (and optionally the resource itself) publicly available in its repository. This metadata can be harvested via the standard OAI-PMH protocol, enabling integration of the centers’ tools and resources into a broad range of applications. For example, WebLicht must employ a harvester to gather CMDI metadata about available WebLicht-compatible webservices.

2.4. CLARIN Center Registry

In order to ease the discoverability of resources within the CLARIN infrastructure, the center registry has been put in

---

\(^{1}\)CLARIN: http://www.clarin.eu

\(^{2}\)CMDI: http://www.clarin.eu/cmdi

\(^{3}\)ISOcat: http://www.isocat.org

\(^{4}\)CMDI Component Registry: http://catalog.clarin.eu/ds/ComponentRegistry/#
place. The CLARIN Center Registry stores CMDI metadata describing each of the nine CLARIN centers. This metadata includes the location of each center’s repository, as well as the center’s type (A, B, or C), contact information, CLARIN compliance status, etc. Figure 1 shows how the center registry, together with the center repositories, will be utilized by WebLicht to collect webservice metadata. An integral part of this webservice metadata is so-called orchestration metadata, which specifies detailed input and output descriptions as well as webservice parameters (see section 3.3.). This orchestration data is the key component for constructing workflows in WebLicht.

3. WebLicht

WebLicht is a service orchestration and execution environment built upon Service Oriented Architecture principles. Although it is currently mostly used for incremental automatic annotation of text corpora, other uses such as audio data processing are under development. In order to fulfill the integration requirements the following components are necessary:

- A harvester for aggregating metadata about distributed webservises
- An orchestration engine for building valid webservice workflows
- A user-friendly web application for construction and invocation of workflows, which relies on the harvester and the orchestration engine

The orchestration engine and the web application are already a part of the current version of WebLicht, while the harvester is presently under development.

3.1. Webservises

A WebLicht-compatible webservice is simply a synchronous REST-style webservice. The client establishes an HTTP connection to the webservice and initiates a POST request containing the input data. Depending on the webservice, the client can also set various processing parameters using the query string in the URL. The webservice processes the input data synchronously and returns the result as output data, using the same HTTP connection. Currently there are approximately one hundred WebLicht-compatible webservises located across Europe. The data format used by the majority of WebLicht-compatible webservises is a valid XML format, TCF (Text Corpus Format), fully compatible with the Linguistic Annotation Format (LAF) and Graph-based Format for Linguistic Annotations (GrAF) developed in the ISO/TC37/SC4 technical committee (Ide and Suderman, 2007). A set of webservises allow conversions between TCF and most of the other formats used in the linguistic community.

3.2. Harvesting of Webservice Metadata

Since webservice metadata is distributed over all the CLARIN center resource repositories, there is no central location from which to obtain webservice metadata. This necessitates the collection of webservice metadata (called harvesting) from each resource repository. Using information from the CLARIN center registry, a harvester is able to target a specific metadata set contained in the resource repositories. Based on given requirements, the harvester will locate the appropriate sets within each center resource repository, read the sets via the OAI-PMH protocol, and aggregate the results.

For example, the WebLicht harvester must periodically check all the center repositories for WebLicht-compatible webservice metadata sets and cache them, as shown in Figure 1. With the help of the harvested metadata, WebLicht can assist the user in building webservice workflows in a type-safe manner. A webservice workflow is a sequence of webservises that are executed in succession, each one receiving as input the output of the previous one. Using the descriptions of the input and output in the metadata for each webservice, the system can guarantee the correctness of a webservice workflow from a data perspective.

3.3. Orchestration Metadata and Chaining

Each webservice is described in the CLARIN repositories by a piece of metadata. This contains information about the webservice creators, access rights, development status, description, input and output format specifications, registration date, URL, etc. An important part of the metadata is the so-called orchestration metadata, which is used by orchestration engines to match webservises to a given input data and to create workflows.

In the CLARIN CMDI Component Registry, an extendable core webservice metadata format is specified. By design, this core format can be easily extended to describe a wide variety of webservises. Such an extended format describing WebLicht-compatible webservises is registered in the CMDI Component Registry, which elaborates on the orchestration metadata in the core format. The orchestration metadata in this extended format will be referred to in this paper as orchestration metadata. The purpose of this orchestration metadata is to describe the profile of the data
that a webservice accepts and the profile of its output, implicitly defining a type system. This I/O data specification is designed to be as straightforward and generic as possible. It is domain neutral and independent of the actual data format used by the web services, but at the same time simple to understand, use and generate. The orchestration metadata is a list consisting primarily of data features, each feature containing zero or more values. The input description specifies what properties input data must have in order to be correctly processed by the webservice. The output description specifies properties of the output data that is generated by the webservice.

The input specification asserts that the "type" feature of the input data must be "application/msword" or "text/plain". The "type" feature in this case designates the data mime type. Similarly, the "lang" feature must be either "de", "en", or "es", which indicates the language of the input document. Additionally it can be seen that a query string is assigned to each feature value which will be passed to the tool as part of the URL.

The CMDI metadata in Listing 1 also specifies that the output will be a TCF file with version 0.4, a language identical to the language of the input text document, and a text layer (represented in the TCF file as a special node). Table 2 shows the summary of output features for our example.

### Table 1: Input Specification Summary for Example 1

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value(s)</th>
<th>URL Query String</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>application/msword, text/plain</td>
<td>informat=doc, informat=plaintext</td>
</tr>
<tr>
<td>lang</td>
<td>de, en, es</td>
<td>language=de, language=en, language=es</td>
</tr>
</tbody>
</table>

### Table 2: Output Specification Summary for Example 1

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value(s)</th>
<th>Input Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>text/tcf+xml</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>lang</td>
<td>de, en, es</td>
<td>lang=de, lang=en, lang=es</td>
</tr>
</tbody>
</table>
of the output TCF document will depend on the language of the input text document.

The output specification can operate in two different modes: addition or replacement. Using the addition mode means that the declared output features for the webservice are added to the ones already in the input data. This mode is used when webservices simply augment the original input data, such as in a case where an additional annotation layer is added to an input TCF document.

In the example above (Listing 1) the replacement mode is used as declared by the element ReplacesInput. A value of "true" indicates that the webservice completely transforms the input data, so that the profile of the output data is determined solely by the declared output features of the webservice.

Listing 2 shows another example, illustrating the use of the addition mode. It describes a part-of-speech tagging tool which augments the input TCF document with a part-of-speech annotation layer.

The input specification for this example is summarized in Table 3. In particular, it tells us that the input format must be a TCF document (version 0.4) in one of four languages (de, en, fr, or it), containing a `tokens` annotation layer. In this case, no URL query string is required by the webservice.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value(s)</th>
<th>URL Query String</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>text/tcf+xml</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>tokens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lang</td>
<td>de</td>
<td></td>
</tr>
<tr>
<td></td>
<td>en</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>it</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Input Specification Summary for Example 2

Since the ReplacesInput element has a value of "false", the output specification for this tool, summarized in Table 4, must not explicitly specify the type, version, or language of the output document, as all the input features will be retained in the output. Thus only the additional annotation layers must be specified. In this case, `lemmas` and `postags.tagset` layers are added, where the value of the `postags.tagset` feature depends on the input language. Namely, if the input document is in English the `penntb` tagset will be used for part-of-speech tagging, whereas the `stein tagset` will be used for French and Italian documents, and the `stts tagset` will be used for German documents. This example illustrates the use of input references to create complex output/input dependencies.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value(s)</th>
<th>Input Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>lemmas</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>postags.tagset</td>
<td>penntb</td>
<td>lang=en</td>
</tr>
<tr>
<td></td>
<td>stein</td>
<td>lang=[fr, it]</td>
</tr>
<tr>
<td></td>
<td>stts</td>
<td>lang=de</td>
</tr>
</tbody>
</table>

Table 4: Output Specification Summary for Example 2

The feature and value name semantics are not yet formally specified. However, consistency of the feature names is necessary for interoperability of webservices within the same domain. In order to ensure this consistency, the feature and value names must be kept in a concept registry such as ISOcat, which is widely used in the CLARIN infrastructure. The use of these standardized feature and value names will then be applied to currently registered WebLicht-compatible webservices and enforced when registering a new webservice.

This system of data-describing assertions makes chaining possible (even before any webservices have actually been invoked). Chaining is the process of automatically find-
ing appropriate webservices when the characteristics of the data to be used as input are known. Each time a webservice is selected, the characteristics of the resulting data are recomputed and used to generate a new set of suitable webservices. Automatically finding new webservices to be applied in the chain is now just a matter of filtering through the list of the available webservices.

### 3.4. Example Processing Chains

The WebLicht architecture allows a high degree of flexibility with respect to webservice processing chains. For example, Figure 2 shows a small subset of possible processing chains available for the German language. Each circle represents an individual webservice (labeled with the institute which created it and an identification number) and the rectangular boxes represent linguistic annotation states. An annotation state indicates which linguistic annotation layers are present in the data document. Often it is possible to reach an annotation state through different processing chains. This allows users to compare results from different tools and to pick-and-choose those tools that they deem best suited to their needs.

Figure 3 shows the connectivity of all of the webservices currently available in WebLicht for the German language.

### 4. Conclusion

We presented the requirements of integrating WebLicht into the CLARIN infrastructure and described the ongoing work to fulfill them, including harvesting CMDI metadata from the CLARIN center registry and the distributed center resource repositories. In addition, the importance and utility of the CMDI metadata within WebLicht was described. Once integration is complete, WebLicht-compatible webservices will be easily discoverable and thus the entire CLARIN developer community will be able to utilize them (also directly - without using the WebLicht web interface). As for users, they will benefit from immediate access to new tools as soon as they are registered in any CLARIN center.
Figure 3: Complete Set of Webservices for German
5. References


SOA implementation of the eHumanities Desktop

Rüdiger Gleim, Alexander Mehler, Alexandra Ernst

Goethe-University Frankfurt
Senckenberganlage 31, D-60325 Frankfurt, Germany
gleim@em.uni-frankfurt.de, mehler@em.uni-frankfurt.de, ernst.alexandra@gmail.com

Abstract
The eHumanities Desktop is a system which allows users to upload, organize and share resources using a web interface. Furthermore resources can be processed, annotated and analyzed in various ways. Registered users can organize themselves in groups and collaboratively work on their data. The eHumanities Desktop is platform independent and runs in a web browser. This paper presents the system focusing on its service orientation and process management.

Keywords: Digital Humanities, Web Services, Asynchronous Processes, TEI P5

1. Introduction
Developments in text-technology and digital humanities in recent years document a strong shift towards web services and applications. The diversity of research interests is reflected by the rich applications, platforms and frameworks being developed. Textgrid (Kerzel et al., 2009) is a research association which aims at supporting access and exchange of resources in the humanities. Textgrid provides the Textgrid Lab\(^1\) as a platform to work on resources. The project eAqua\(^2\) (Büchler et al., 2010) aims at knowledge extraction based on processed antique sources and offers classic scholars convenient means to work on the results. Furthermore, digital editions of works offer additional value by allowing users to browse and query meta data and to perform full text search on resources. For example, the Heinrich Heine Portal\(^3\) provides a digital edition of the works of Heinrich Heine (Füllner and Fournier, 2002). The letters of Vincent van Gogh have also been published as a digital edition (Jansen et al., 2010). Inscriptiones Graecae\(^4\) is a project at the Berlin-Brandenburgische Akademie der Wissenschaften which provides access to Greek inscriptions. In addition, there are many more projects not explicitly named here. Research associations, notably CLARIN\(^5\) (Tamás Váradi and Koskenniemi, 2008) and DARIAH\(^6\) (DARIAH, 2010), focus on creating common infrastructures and improving exchange and interoperability of services.

Desktop applications continue to do well in specific domains. Web applications on the other hand are typically not designed as mere web portations, but provide additional value. Some principal advantages are the ease or elimination of software installation as well as platform independence. But the true benefits are the means to share and use resources and services online. Web applications and services thus allow for collaborative work on research artifacts in a way which would hardly be possible otherwise.

The concept of Service Oriented Architectures (SOAs) has been proposed as a means to integrate isolated services in order to provide more sophisticated functionality. The general idea of SOAs is described as the orchestration of services rather than the service itself. How a SOA is implemented in a specific domain is not standardized, but has to be designed according to the requirements of the respective application area. In this article we describe how the eHumanities Desktop (Gleim and Mehler, 2010) adopts the concept of Service Oriented Architectures to provide a platform for research in the digital humanities.

2. SOA implementation of the eHumanities Desktop
The eHumanities Desktop is a system which allows users to upload, organize and share resources using a web interface. Furthermore resources can be processed, annotated and analyzed in various ways. Registered users can organize themselves in groups and collaboratively work on their data. Figure 1 shows a screenshot of the eHumanities Desktop’s web client in action. It is platform independent and runs in a web browser.

2.1. Overview of Architecture
The functionality of the overall system is realized by means of application modules. The development of new modules is mainly driven by research projects. Developers can benefit from the eHumanities Desktop framework by using its APIs to manage access permissions, storage of resources and incorporate other modules, applications or remote webservices to perform their task. Thus typical aspects like user- and rights management as well as storage handling are already taken care of by service modules. The developer of

\(^1\)http://www.textgrid.de/1-0.html
\(^2\)http://www.equa.net
\(^3\)http://www.hhp.uni-trier.de/Projekte/HHP/start
\(^4\)http://telota.bbaw.de/ig/
\(^5\)http://www.clarin.eu
\(^6\)http://www.dariah.eu
a new application can concentrate on the new functionality. An application module typically consists of a client module which provides a user interface and a service processor on the server which executes incoming commands. In the following section we will describe the system architecture in more detail. We start with an overview that provides a general picture of the components and technologies being used. Then we describe the process of executing a service request in the eHumanities Desktop.

The service processors implement the functionality of the system and are accessible by a web API. They are implemented in Java and run as part of a web servlet on Apache Tomcat. Each processor covers a certain area, for example resource management, annotation, text preprocessing, categorization and the like. In order to accomplish their task, service processors can use the eHumanities Desktop APIs to access the master data, storage and annotation system. In addition they can call local applications on the servers or other web services on the internet. The primary means to access the web API is using the eHumanities Desktop Client. Nonetheless service requests can also be triggered by other means, for example PHP, C++ or Java applications using HTTP Post/Get. Thus the eHumanities Desktop can also be used by other services and applications.

The system APIs give access to resource and user management, annotations and storage. Since we intend to focus on the service orientation of the system we do not describe them in depth here. The repositories, documents, users and resources are managed in a master database. Figure 3 shows the class diagram of the master data. It is currently implemented based on the graph database Neo4j\(^8\) in combination with Lucene\(^9\) for indexing.

The storage API manages access to the document contents. The way a document is stored in the system is not fixed. By implementing an interface, developers can rather add new technologies as required. How a

---

\(^7\)http://www.sencha.com/products/extjs

\(^8\)http://neo4j.org/

\(^9\)http://lucene.apache.org
new document is stored is decided based on its file format (or MIME-type\textsuperscript{10}): For example a TEI P5 document will be stored in the TEILex DB, a binary file in a Hadoop/HDFS\textsuperscript{11} cluster and even MySQL databases can be imported as documents. From the user perspective the way a document is stored does not make any difference. The annotation API supports RDF to annotate resources in the system.

2.2. Processing of Service Requests

This section describes the process of retrieving, executing and responding to a service request. Figure 4 illustrates this approach. At the beginning of processing a call, an incoming service request is checked for its authentication. A login command would trigger a new session in the session handler and return a unique session key to authenticate subsequent requests. The next step is to parse the request and to determine which service processor is in charge. This is determined by checking the prefix of the incoming command string. A failure in one of these checks will trigger an error message to the client.

Service requests as well as responses are normally formatted in the JavaScript Object Notation Language (JSON)\textsuperscript{12} and transmitted via HTTP. This is due to the primary means of accessing the system via the eHumanities Desktop Client which is widely written in JavaScript. As already noted any other means to issue HTTP Get/Post can be used as well.

Next, transactions on the underlying databases are started to ensure that any changes to the system during the processing can be rolled back in case of an error. Then an instance of the proper service processor is created, registered in the process manager and started. The process manager is an essential component to keep track of which user runs which processes at a given time. Every process has a process id, name and owner. Furthermore the time when the process was triggered is stored and its current progress is tracked. The web API allows users to list their current processes and cancel them if desired. Canceling a process is interesting for long running processes such as complex analyses. However the user can only trigger a process to stop. The process has to regularly check whether it should continue or if the user wishes to cancel it. The same applies to updating the current progress – a task only the process itself can answer. A hard stop of a process would be possible but could lead to unpredictable results and data corruption. Therefore this option is currently switched off – only “soft” stopping is supported as described. Figure 5 shows a screenshot of the graphical frontend of the process manager with two indexing tasks running.

Figure 5: Screen of the Process Manager showing two asynchronous tasks of the user.

The service processor checks permissions of the requesting user on resources which have been passed as parameters. In case of a failure the client is informed accordingly. Then the computation itself is performed. A command processor can use the eHumanities Desktop’s APIs to accomplish its task, as for example to import a TEI P5 document into the TEILex database and link its tokens to the lexicon entries. This task can be quite complex and time consuming, depending on the size of the document. For direct feedback to the client the process would take too long and cause a timeout on the client side, which waits for a response. In such cases a developer of a service processor can choose to execute the task in a thread asynchronously. This means that the task is executed as a separate thread. After the thread has been triggered control is given back to the system which returns a message to the client that the process as been started. The transactions and resources remain open and the process manager keeps the process as well. The client can track the progress of the processes, safely log out and check for the results later by logging in again. When an asynchronous process has finished it triggers the steps to unregister it from the process manager and commit transactions. These steps are performed automatically when a synchronous task was completed. A drawback of HTTP is that the server can not actively inform the client when a process has finished. Thus a response message is only sent for synchronous tasks.

2.3. Use Case: Representing TEI P5 and Lexica with TEILex

To give an example of a typical application module and its components we now describe the TEILex subsystem. It is one of the most recent additions to the eHumanities Desktop. It provides a seamless integra-

\textsuperscript{10}http://www.iana.org/assignments/media-types/index.html
\textsuperscript{11}http://hadoop.apache.org/hdfs/
\textsuperscript{12}for more informations about JSON see http://json.org
The integration of lexica with text collections encoded in TEI P5. This is achieved by linking lexical information of text tokens directly with lexicon entries. The integration of lexicon and TEI representation enables the user to systematically correct erroneous linguistic annotations in preprocessed texts and to perform disambiguation. Figure 6 shows a screenshot of the lexicon browser which exemplifies dynamic linking of word forms between text and a lexicon. The visualization uses color codes to inform the user about ambiguities. Uniquely identified word forms are marked green. In this example the word form “spiritus” is highlighted in yellow to state that multiple lexicon entries of this word form exist, that belong to the same lemma. If the item were orange, it would mark an ambiguity that affects different lemmata.

The system allows users to edit and expand lexica as well. Figure 7 exemplifies how a new word form can be created for a previously selected lemma. Both lexicon and text representation are designed to be dynamic and extensible by users. This is a demanding task because highly dynamic databases are more difficult to optimize for efficient reading and writing. Since an import may take minutes or even hours, depending on the size of the document set, this task is done asynchronously. A user can trigger the import of a set of documents and log out. Later he or she can log in again and check on the progress by opening the process manager (see figure 5).

The TEILex is a good example of how different service modules can be put together to achieve an advanced functionality in terms of a SOA. A preprocessor module can be used to convert plain text documents into TEI P5 documents which can be imported into the TEILex system. The TEILex API can then be used by service modules to implement services for client modules or other external clients. The Lexicon Browser module serves as an interface to the lexica, while another module, called HSCM (Historical Semantics Corpus Management), (Jussen et al., 2007; Mehler et al., 2011) offers an interface to query and explore text corpora. Both modules interact on the client side to lookup words or keywords in context. And both use the same mechanisms on the server to accomplish their tasks.

Figure 4: UML Action Diagram of how service requests are processed.

Figure 7: Screenshot of the Lexicon Browser exemplifying the creation of a new word form.
3. Conclusion

We have described the eHumanities Desktop as a service-oriented architecture. It supports the development of services related to managing, browsing and analysing resources in the humanities. Special focus was given to the way service requests are handled and the way long running tasks can be managed asynchronously. Current development outside of project specific development is targeting the annotation system and improving the TEILex system. The eHumanities Desktop is open to scientists who are interested in testing and working with the system.

4. Acknowledgements

The eHumanities Desktop core is being developed in the lab for text-technology (computer science) at Goethe University while active development on new services and extensions is done with and by research projects. We gratefully acknowledge the support by the Leibniz project “Politische Sprache im Mittelalter”\textsuperscript{13} of Bernhard Jussen and the projects of the LOEWE Schwerpunkt “Digital Humanities”\textsuperscript{14}, both at Goethe University.

5. References


Rüdiger Gleim and Alexander Mehler. 2010. Computational linguistics for mere mortals – powerful but easy-to-use linguistic processing for scientists in the humanities. In Proceedings of the 7th Interna-

\textsuperscript{13}https://semantik.geschichte.uni-frankfurt.de/

\textsuperscript{14}http://www.digital-humanities-hessen.de/


Signal processing via web services: the use case WebMAUS

Thomas Kisler¹, Florian Schiel¹, Han Sloetjes²
Institute of Phonetics and Speech Processing¹, LMU München, Max-Planck-Institute for Psycholinguistic², Nijmegen
(kisler,schiel)@phonetik.uni-muenchen.de¹, han.sloetjes@mpi.nl²

Abstract
CLARIN aims at providing a technical infrastructure for language resources and language processing tools. In this context we present the design of a web service and a corresponding interface to provide easy access to an formerly only locally executable application called Munich AUtomatic Segmentation (MAUS). MAUS enables a scientist to automatically label and segment a spoken utterance into word and phoneme units given some text representation of the spoken content. The original software package has been embedded into a set of CLARIN conform webservice. Using a standard description format the interactive usage of those services, but also the access from within other services or chaining engines like WebLicht is made possible. As an example for the integration of the web service in another application, we show the WebMAUS integration into ELAN and into an easy-to-use web interface.

1. Introduction
The working draft of the HTML5 standard had a big influence on the possibilities of web interfaces and applications. Though almost everything has been possible before, the new standard makes many things much easier and defines functionality in a way that all browsers could implement it. And especially the new standard and the big hype around web 2.0 in general changed the visibility of web applications.

Services commonly referred to as web services are a convenient way to make software applications freely available without the hassle to install software on the local computer. Applications under development may be tested by a broader individual users via appropriate web interfaces or in batch mode from the command line or by other applications calling the web service as a “helper application”.

In this contribution we present a first example of a linguistic web service that automatically segments and labels a spoken utterance into its phonemic contents, implemented as a CLARIN¹ conform web service, as well as examples for a web interface and the usage as a helper application to the well-know ELAN tool (Wittenburg et al., 2006).

2. Munich AUtomatic Segmentation (MAUS)
Language resources often contain some kind of speech recording, either as a single or multiple channel sound file or as the soundtrack of a video recording. Many analyzes require some kind of alignment of a symbolic representation (annotation) of the recorded speech act (i.e. an orthographic or phonetic transcript) to the corresponding parts of the speech signal. Such an alignment is usually called a segmentation and labeling (S&L). While the orthographic transcript of the truly spoken content can be obtained with the aid of transcribers (sometimes even via mechanical Turk) at moderate effort, the S&L at word level or even worse the S&L on smaller linguistic units such as morphs, syllables or phones is expensive, time-consuming and error prone.

There are several ways to automate the S&L process. A simple way is the so called forced alignment to a given sequence of phones using speech recognition technology such as Hidden Markov Modeling (HMM). Here the aligner has the task to find the best partitioning of the speech signal given a fixed sequence of phonetic symbols and a set of pre-trained statistical models for each phoneme class of the language. Forced alignment works very well granted that the signal is of moderate good quality and the truly spoken phones are known a priori (which is usually not the case because given the spoken words usually only a citation form of pronunciation can be calculated automatically which deviates from fluent spoken speech).

The Munich AUtomatic Segmentation system (MAUS) extends the basic aligner concept by modeling a statistical space of possible pronunciation variants for a given orthographic input (Schiel, 1999; Schiel, 2004). Figure 1 shows a simple example for the German word ‘Abend’. The hypotheses space is calculated for each individual text input based on a machine-learned statistical expert system of pronunciation. Combined with HMM technology the MAUS can thus not only find the best segmentation but at the same time the most likely sequence of truly spoken phones in the speech signal (see Figure 2 for MAUS S&L result as viewed in praat).

On a subset of spontaneous German speech in the Verb-mobil corpus (Burger et al., 2000) the MAUS technique yielded about 97% of the average interlabeler agreement of three trained phoneticians working on the same task (Schiel, 1999).

MAUS is implemented as a system of UNIX script files and C++ binaries that can be run on Linux and Windows platforms. It requires as input the speech signal and some form of either orthographic or phonological transcript of the spoken utterance. The result is stored in either BAS

¹http://www.clarin.eu/
Partitur Format BPF (Schiel et al., 1998), praat TextGrid\(^2\) or Emu (Bombien et al., 2006) compatible annotation format files. MAUS currently (version 2.30) supports 7 languages: German, English, Hungarian, Italian, Estonian, Spanish and Dutch (the first 4 including tokenizing, text normalization and text-to-phoneme conversion). A vast number of options allow the user to control the S&L process as well as the form of output formatting and the statistical modeling of the pronunciation variation (e.g. by learning new pronunciation models or formulating explicit phonological pronunciation rule sets). The MAUS freeware package can be downloaded from the Bavarian Archive for Speech Signals\(^3\).

3. WebMAUS Services

As a contribution to the “Common Language Resources and Technology Infrastructure” (CLARIN) project, we developed web services that provide the functionality of the aforementioned MAUS tool. This web services allow access to MAUS without the need of installing it locally on the users machine, might the “user” be a human or another web service that wants to access it. The CLARIN view on web services is rather a process oriented view, especially when regarding the efforts that are put into implementing work flow engines, that allow the chaining of different web services (Boehike, 2010). That was one of the main reasons we decided to implement the web services as RESTful remote procedure calls (RPC).

We wanted to use standard technologies to provide an easy-to-use and easy-to-understand interface to our web services. Therefore some of the HTML operations serve as an envelope for the data necessary to our web services. Through the “overloading” of POST we can achieve customized behavior without breaking the RESTful idea (Richardson and Ruby, 2007). The web service wrappers around MAUS are implemented in Java using the “Java API for RESTful Web Services” (JAX-RS) reference implementation Jersey.

To provide a machine readable format, we described our web services in the standard description format for RESTful web services, the Web Application Description Language (WADL). A WADL file contains the technical aspects of a certain web service and therefore allows programs or chaining engines to generate the call to the WADL described web service automatically.

In addition to the technical WADL description, the CLARIN meta data infrastructure (CMDI\(^4\)) format gives us the possibility to describe our web services also semantically. Each web service is described in the CMDI file, based on com-

\(^2\)http://www.praat.org/
\(^3\)http://www.bas.uni-muenchen.de/forschung/Bas/software/
\(^4\)http://www.clarin.eu/cmdi
ponents that are registered in a public component registry (Broeder et al., 2010). The harvester for CMDI metadata that are built to harvest the metadata of data can then also be used to harvest the metadata about the available web services in the CLARIN infrastructure (Van Uytvanck et al., 2012). Those two descriptions can then be used to manually or automatically integrate WebMAUS to any infrastructure that also understands these descriptions. Example for those Service Oriented Architectures (SOA) are WebLicht (Hinrichs et al., 2010) and Taverna (Hull et al., 2006).

So far we provide a variety of different RPCs that offer access to two MAUS modes, a simple basic and a general functionality. The “runMAUSBasic” takes a plain text and a signal file as input and returns a Praat compatible TextGrid file for a number of languages\(^5\). In this case the web service is extended by a language specific tokenizer and text normalization followed by a statistical-driven text-to-phoneme algorithm (Balloon (Reichel, 2012)) which converts the arbitrary input text into an ordered sequence of canonical pronunciations. Other than the language no parameterization is allowed. The more powerful “runMAUS” call requires a signal file and a BAS Partitur Format file (BPF) containing a tier with already tokenized and phonemic encoded speech. In contrast to the simpler “runMAUSBasic” this web service allows the caller to provide the full set of optional parameters of the MAUS tool. With those parameters the MAUS call is fully customizable and therefore appropriate for advanced users. By incorporating both calls into simple script loops vast amounts of data can be processed automatically (batch processing). Other web service calls provide documentation and information about language specific phoneme sets used by the MAUS services.

4. WebMAUS Interfaces

As a show case for our developed web services and as a user interface for technical unfamiliar users we developed a few state-of-the-art web interfaces.

A web interface should provide an intuitive, natural web-compliant way of processing the data. We based the interface on recent technologies like the upcoming HTML5 standard and jQuery. The working draft of the HTML5 standard provides a variety of features that emerged from the widespread use of HTML and were missing in older standards. Those are audio players, advanced forms and also desktop-like behavior, like drag and drop (W3C, 2012). jQuery, as an abstraction to JavaScript, hides much of the counter-intuitive aspects of JavaScript for the developer and provides a rich set of functions for GUI design and Ajax (Asynchronous JavaScript and XML) calls.

A big advantage of those web technologies is, that they can be run from within every standard compliant browser. Unfortunately, HTML5 is no standard so far and only a handful browsers implement the working draft HTML5 in a broader sense. It is to hope that once it is released that future generations of browsers will implement the standard in a more compatible way, than this was the case the past decades. Until now we propose to use Mozilla Firefox or Google Chrome, since both of these browsers implement a big subset of the specifications of the draft. Both of them are freely available and Firefox furthermore is open source, so that theoretically everybody has access to our web interfaces. As a demonstration for our “runMAUSBasic” and “runMAUS” functionality, three interfaces have been developed\(^6\).

For the “Basic MAUS” interface only a plain text file and a signal file with the utterance are necessary. This is the easiest to use interface and useful for single signal and text files. The visual feedback presented to the user is responsive and entertaining, and results can be stored in a widely used standard format (Praat TextGrid). The user may select between different languages, for which the BPF generation is implemented, since text normalization and grapheme to phoneme conversion are already available. The second interface “General MAUS” gives access to the general MAUS interface, takes a signal file and a BPF file as inputs, and provides a set of options to the user which she might change to achieve better results. The user may choose between 7 languages and three different output formats\(^7\). The last web service, and the most interesting one especially for processing large amounts of data, is “Multiple MAUS” (see Fig. 3). It not only allows the full customization as does the general interface, but also for uploading several files via drag and drop. The combined size of files is limited to 300 MB and the interface has been successfully tested with a set of 600+ files (300+ signal-text pairs).

5. Calling WebMAUS from ELAN

ELAN\(^8\) (Wittenburg et al., 2006) is an audio and video annotation tool for the desktop, that is being developed by The Language Archive department of the Max Planck Institute for Psycholinguistics. It is applied in several types of research within linguistics and beyond. Annotation in ELAN is still mainly a manual process and although it offers specialized user interfaces for the before mentioned steps of segmentation and labeling (S&L), producing fairly accurately aligned, multi-layered annotations to the primary data, is time consuming (and therefore expensive). In recent years first steps have been taken to extend the tool with software components that semi-automatically segment and/or label media signals. A language independent silence recognizer was the first of such extensions. Next, in the AVATEch\(^9\) (Tschöpel et al., 2011) project the extension mechanism for recognizers was enhanced in order to accommodate extension with components created in a variety of programming languages (ELAN itself is written in the Java programming language). Because the recognizers in AVATEch run on different platforms, it seems natural to make them available as web services and that is what currently is being worked on.

The initial implementation of the interaction with WebMAUS builds on this existing extension mechanism of ELAN (therefore it is not an integral part of ELAN but a separate module). The advantage of this approach is that

---

\(^5\)currently English, German, Hungarian and Italian

\(^6\)https://webapp.phonetik.uni-muenchen.de/BASWebServices

\(^7\)BPF, TextGrid and Emu.

\(^8\)http://www.lat-mpi.eu/tools/elan/

\(^9\)http://www.mpi.nl/avatech
the WebMAUS client software can be replaced easily and repeatedly independently of the update cycles of ELAN itself. In this first implementation the “runMAUSBasic” variants for four languages are supported. The information about the web services and how to invoke them, is extracted manually from the WebMAUS WADL file and is stored in a meta data file describing the module and its parameters. Whether it is possible to build a generic web service client module that can invoke any web service that’s sufficiently described by a CMDI and a WADL file, is still unclear and to be investigated. Interaction with WebMAUS, at this stage, consists of uploading a wav file and a text (or text file) to the web service and converting the returned content to tiers. The user interface of ELAN allows the user to either select a text file from the file system, or to select a tier for upload. In the latter case ELAN converts the contents of the tier to plain text and uploads that to the web service. The results are currently returned as Praat TextGrid files. This scenario is an example of bringing the data to the algorithm and for each call the data is uploaded again. Since the “basic” WebMAUS does not support parameters to configure the workings of the service, it is not likely that the same file needs to be uploaded more than once (the result will always be the same). But for more complex processing it would be beneficial to upload the audio separately and then supply the URL of the uploaded file to the web service.

As possible enhancement of the WebMAUS client extension, supporting the general “runMAUS” service can be considered. For that purpose ELAN should ideally be able to produce BPF files. A further enhancement can be reached in post processing the results. While the fine segmentation is advantageous in many cases, it is often also required to have aligned annotations on the level of bigger units e.g. “sentence”. Joining smaller segments to build larger units (as a separate tier) could be a next step to improve the usability of the MAUS technology for ELAN users.

6. Conclusion

The described web services and interfaces have reached a stable state and are now used for the research and teaching at the “Institute of Phonetics and Speech Processing” in Munich. Especially the much more eye-candy and, for users who are familiar with the web, intuitive interface seemed to have led to an improvement in the user experience. Even students in lower semesters who are not yet used to to handle the tools and programs that are commonly used within the research community have been able to produce S&L data after been shown the interface once.

The web interface furthermore saves the hassle of local installation and permanently updating this installation and makes the tool available to a broader audience which are not able to install a set of tools that are supposed to interact which each other. People that are familiar with a command line interface can profit from the possibility to call the web services from the command line itself or scripts over tools like curl. Researchers and students that are already used to tools like ELAN can profit from the easy integration of web services into their favorite GUI. As the interface is clearly structured the integration of a web service is much easier than integrating functionality by other means (e.g. through libraries) at the expense of control. The developers of tools, too, profit from the fact that no updates of the provided functionality on the client are necessary. In a well-designed user interface additional work might be taken from the user, as it can be seen in ELAN. There the user not only is able to select a file from the file system, but also send a tier to the web service which might save him some time in creating such a text file.

7. Acknowledgments

The work of the authors was carried out within the CLARIN-D project (CLARIN-D, 2012) (BMBF-funded).

8. References


The European Library: a Data Service Endpoint for the Bibliographic Universe of Europe

Chiara Latronico, Nuno Freire, Markus Muhr, Shirley Agudo, Andreas Juffinger
The European Library, National Library of the Netherlands
Prins Willem-Alexanderhof 5 | 2595 BE Den Haag
E-mail: Chiara.Latronico@KB.nl, nfreire@gmail.com, Markus.Muhr@KB.nl, Shirley.Agudo@KB.nl, Andreas.Juffinger@KB.nl

Abstract
This paper presents the current status, on-going work, and future plans for providing a wide range of resources originating from European libraries to digital humanities research infrastructures. These resources include library catalogues, digital objects and full-text resources. We believe that this consolidation of pan-European library resources will provide new opportunities for research, and we aim to make The European Library an important asset for the digital humanities research in the coming years. This paper also presents how the Service Oriented Architecture has been used for the technical interoperability infrastructure between the national and research libraries and The European Library, and how it is being used within the internal information system of The European Library for application interoperability. Future work towards the integration into relevant SOA and research infrastructures is also presented, as The European Library plans to provide two kinds of services: resources/data access services, and library data-processing services. We further highlight the fact that the portal showcases data mining capabilities with the data at hand.

Keywords: services, interoperability, aggregation, enrichment, bibliographic data, libraries, SOA

1. Introduction
The European Library holds over 100 million bibliographic records from the 48 national libraries of Europe and leading research and university libraries. The aim of The European Library is to serve a data access point for tools and functions supporting research but also as a primary research corpus. Such tools and functions can be used to analyze the metadata of all Europe’s books and some of the digital content and texts held in the libraries. To achieve this, The European Library aims at becoming an integrated data service endpoint for DARIAH, CLARIN and other research infrastructures.

The European Library provides access to resources including the national bibliographies and other highly valuable bibliographic data sets via a single service endpoint. Note that the national bibliographies are one of the main bibliographic data sources in each country. In Europe, National Bibliographies are typically created and maintained by national libraries. Whenever a book is published in a country, it is recorded in the corresponding national library catalogue from where the national bibliography is derived. Through the combination of all national bibliographies The European Library can provide somehow the European National Bibliography. The European Library is further expanding its operations to other leading research and university libraries, within the Europeana Libraries project, which brings three major library associations LIBER1, CERL2, CENL3 and Europeana4 together.

Beyond that, The European Library also offers access to collections of digital resources and also full-text contents. These include a diverse wealth of content, such as e-theses, books, manuscripts and films, including many sizeable collections grouped around specific themes, topics or notable personalities. Examples of these are: digitised letters from prominent thinkers such as philosopher Immanuel Kant, pamphlets detailing the history of the Low Countries from 1490 to 1800, and images covering the papers, photographs and correspondence of prominent physicists such as Nobel prize winner Erwin Schrödinger. There are also digitised manuscripts created in European parts of the Ottoman Empire.

The vast majority of the bibliographic data will be in the public domain, following a decision in September 2011 by the Conference of European National Librarians (CENL, the group which owns The European Library and provides most of its data) to release its datasets under a Creative Commons Universal Public Domain Dedication, or CC0 license5.

The library domain is very diverse in data-gathering practices and data formats. In spite of many efforts that libraries undertake regarding the standardisation of cataloguing practices and bibliographic data formats, we still find very heterogeneous data in library catalogues. Cataloguing rules still leave room for different interpretations, and librarians often have to work with the limitations of the information systems that are not always up-to-date with the standards, or don’t fully enforce and validate cataloguing practices. Therefore, the same information may be represented quite differently from library to library, and even within the same library.

1 http://www.liber.eu
2 http://www.cerl.org
3 http://www.cenl.org
4 http://europeana.eu
Library catalogues have been created across several decades, also contributing to highly heterogeneous data. They contain legacy data, which was created according to older cataloguing rules in older library management systems, and may have been subjected to automatic data migration processes.

Over the years, The European Library has developed tools and knowledge on how to consolidate these bibliographic resources, and has operational processes in place for:

- Formal negotiation of content with libraries.
- Management of the rights for use of the data.
- Transfer of bibliographic data and full-text resources.
- Regular updating of the data sources.
- Processing of bibliographic data.
- Deliver of data to end-users and third parties.

The European Library is a data-focused and service-oriented organisation. To ensure that the bibliographic data can be used for research purposes, we homogenise the data for retrieval and mining purposes but also always give access to the original data for verification.

2 Service Oriented Architecture for Research Infrastructures

Although The European Library service was founded as a traditional library service, focused on making the library resources available through OPAC features, it is now aiming to facilitate the use of these resources for research, by making them available to research infrastructures and virtual research environments in the humanities and social sciences through data services.

Service Oriented Architecture (SOA) has always been central to the technical interoperability infrastructure between the national libraries and The European Library. Originally through federated search, and nowadays also for data exchange and harvesting. Furthermore the data has been provided in a number of projects to researcher and services in Europe. As shown in Figure 1 The European Library acts as the gateway between libraries and services.

One particularly successful integration project is ARROW\(^6\) (Accessible Registries of Rights Information and Orphan Works), where a single SOA infrastructure is being established to combine and access author rights information. In this project, The European Library provides Web services for the processing and use of national bibliographies during rights clearance processes, such as those of mass digitisation projects. The service from The European Library is thereby a bibliographic record clustering service for national bibliographies. The service provides a set of related publications from the national bibliographies for a deeper analysis in a rights clearance process.

As mentioned before, The European Library is on the crossing point of traditional data exchange protocols and the more state of the art SOA architectures often used in research infrastructures and modern distributed systems. Besides the “homogenization” of bibliographic information for retrieval purposes, the core service of The European Library is to serve as a gateway to such traditional federated systems. For instance, this gives a researcher access to bibliographic information originally in MARC21 available only via the Z39.50 protocol, as well as bibliographic records in TEI (Text Encoding Initiative) provided to The European Library as files. The European Library acts as a domain specific protocol normalization service as well as a data normalization service. Given that the data is aimed for the use in research, The European Library always ensures that the original data remains available and unaltered, by serving a separate normalized version to maximize the interoperability.

Service Oriented Architecture is also in place within the internal information system of The European Library, mainly for interoperability between the different services, as shown in Figure 2. This graphic shows the main components of the internal information system, and the protocols used for cross-application interoperability. The components presented in the figure are:

- REPOX is a data aggregation and management tool and is used to synchronize our bibliographic data with the partner network. In the current version REPOX provides a REST interface to maintain and exchange information about aggregators, providers and data sources configured to “harvest” data from our partners.

\(^6\) http://www.arrow-net.eu/
in the library environment. Besides the REST interface that is used for administrative tasks on REPOX, OAI-PMH is utilized to let UIM retrieve bibliographic records from REPOX.

- SugarCRM is a customer relationship management framework and we administrate all of our general information about partners with it. As our central management tool it holds information about harvesting, ingestion, enrichment or portal status as well as contact information to libraries. It communicates with UIM using a SOAP interface. Since SugarCRM is highly customizable, communication is done using key value pairs from a defined vocabulary. General harvesting information from REPOX is piped through UIM to SugarCRM. This allows us to be aware of the status of different collections from one central point of view including automatically created reports as basis for performance indicators.

- UIM is an acronym for Unified Ingestion Manager and is our core-processing unit to normalize, enrich, index, and store bibliographic data. It communicates with REPOX to ingest data into the framework with the current foremost aim to prepare it for our portal. While the actual bibliographic records are gathered from REPOX via OIA-PMH, all general administration tasks like the provider and collection information necessary to process data inside UIM is delivered via SOAP from SugarCRM. Furthermore, our GWT based frontend to manage the ingestion processes utilizes a REST service. Since we support different kinds of library format, this unit is also normalizing and harmonizing data into the same format, allowing us in the future to provide our data in a homogenized way back to different institutions helping them to access the data a unified way.

- SRU & Gateway represents a gateway for legacy communication protocols. Since for legal reasons we cannot integrate all collection from our partners in our central storage and index, we have to incorporate remote search over a gateway to deliver retrieval results for these collections. We communicate with the gateway using REST and receiving XML in SRW format.

- Portal BL represents *The European Library* business logic. It provides an internal API for access to our central repository as well as search indices, query suggestion, grouping information, etc. It uses SOAP to communicate with SugarCRM to get information about contributors and collections as well as REST to retrieve data from storage engine, search indices, etc. Although our portal is the main target for this API, we also developed an OpenSearch API that allows executing simple queries that deliver results in a simple, standard Dublin Core format.

- Confluence is used as content and document management system. We manage there information like publications, reports etc. Furthermore, we use Confluence as a front-end for editorial parts of the portal. REST calls are then used to synchronize this editorial content with the portal.

### Section 3: Future Work in the Field of SOA

*The European Library* plans to provide two kinds of services: resources/data access services, and library data-processing services.

The resource/data access services aim to provide access to the European bibliographic universe for researchers and research infrastructures. These resources will be available through linked open data (LOD) and also through the implementation of service specifications of established research infrastructures.

The library data processing services aim to provide the experience and knowledge about library specific information handling. These services will allow the provision of specialized services for researchers with respect to preparation of corpora and alike. Examples for such services are:

- Library data-format conversion
- Particular data-cleaning operations for library data
- Data normalisation
- Duplicate detection
- Named entity extraction and linking to open data sets (such as VIAF\(^7\), Geonames\(^8\), and DBpedia\(^9\))

We believe that this pan-European library resources and bibliographic processing services will provide new opportunities for digital humanities research in the coming years. Furthermore, besides researchers in the humanities we also believe that in the future our data can be exploited for data mining purposes giving more insights about the nature of bibliographic data.

### Section 4: New Portal a showcase for Data Mining Possibilities

In an effort to serve researchers in the best way possible, a brand new version of *The European Library*’s portal\(^10\) will be launched in June 2012. This new platform will showcase national and research libraries collections to a wide audience, and draw new customers to their repositories. At launch, this “union catalogue of Europe” will have about 10 million digital objects and over 100 million bibliographic records, and a full text search index of 25 million pages.

On the other hand, *The European Library* serves for libraries as a collaborative platform for data

---

\(^7\) [http://viaf.org](http://viaf.org)
\(^8\) [http://www.geonames.org](http://www.geonames.org)
\(^9\) [http://dbpedia.org](http://dbpedia.org)
\(^10\) [http://www.theeuropeanlibrary.org/beta](http://www.theeuropeanlibrary.org/beta)
enhancements, for work on ontologies, for increasing the retrievability and re-use of their collections, and for increasing the visibility of European cultural heritage.

The introduced, internal SOA based architecture in The European Library can process and aggregate bibliographic records at a rate of up to 450 records per second (>30mill/day) which supports the content strategy of aggregating all bibliographic information in Europe and beyond.

In our portal we show a couple of applications that already highlights data mining possibilities. You can see some of these in figure 3. The portal holds visualizations on year and place of publication, topics as well as language information for search results, specific collection or the whole data set.

Furthermore, a first simple duplicate detection is integrated in the new portal to improve retrieval experience by avoiding multiple entries in search results pointing to different records from different libraries representing the same manifest. The portal also shows simple normalization of named entities to open data sets (Geonames, VIAF, MACS) to further improve the user experience by interlinking authors, places of publication, etc. These features will be gradually improved over the near future and will be published as Linked Open Data.

Together with the portal an OpenSearch API will be launched. In the first release, it will provide functionality to trigger simple search queries delivering up to a maximum of 5000 retrievable records in a standard Dublin Core format. In the future, it will be extended to give access to more sophisticated functionality like faceted search, exposition of our group information like ARROW clusters as well as manifest clusters, authority information and enriched data. This allows developers in the future to build their own services or integrate our service into their own web appearance.

### 4 Conclusion

This paper presented the current status, ongoing work, and future plans for making available to digital humanities research infrastructures a wide range of resources originating from European libraries. These resources include library catalogues, digital objects and full-text resources. We believe that this consolidation of pan-European library resources will provide new opportunities for research, and we aim to make The European Library an important asset for the digital humanities research in the coming years by ensuring the integration into relevant SOA and research infrastructures.
Advanced Web-services for Automated Annotation of Audio and Video Recordings

Przemyslaw Lenkiewicz1, Dieter van Uytvanck1, Eric Auer1, Peter Wittenburg1, Sebastian Drude1

1Max Planck Institute for Psycholinguistics, Nijmegen, Netherlands
Przemek.Lenkiewicz@mpi.nl, Dieter.vanUytvanck@mpi.nl, Eric.Auer@mpi.nl, Peter.Wittenburg@mpi.nl, Sebastian.Drude@mpi.nl

Abstract
In this paper we describe currently developed web services for audio and video processing. Their purpose is to shorten the time taken by manual annotation of audio and video recordings by extracting features from media files and creating semi-automated annotations. We show that the use of such supporting services can shorten the annotation time to 30-50% of the time necessary to perform a fully manual annotation of the same kind.

Index Terms: speech segmentation, gesture segmentation, web services

1. Introduction
Annotated video and audio recordings are an important asset in numerous research areas. One of the aims of the AVATecH project [1] is to implement algorithms that allow for the automatic and semi-automatic creation of pre-annotations for the corpora, hence reducing the time needed to perform the manual annotation. Analysis of such content is a difficult task due to two factors: 1) the size of the media corpora is very significant; 2) the recordings are of very high diversity of languages, conditions and situations. This means that effective methods for automated processing of such content are not widely available or don’t exist at all. In this paper we describe the algorithms that have been developed in the project, we present the scenarios of application of such algorithms and their web-services based interface, and finally we present the initial results that they deliver.

2. Progress in the AVATecH project

2.1. General recognizers
Manual annotation can be significantly improved with incorporation of several general-purpose recognizers, which analyze the recordings and deliver assistance with the task of navigation through the media, finding relevant parts, etc.:  
- Acoustic query by example: allows to provide a short audio clip or a part of longer recording and look for occurrences of similar audio in very long segments. This allows to find relevant parts in recordings and highly improves the navigation, which is normally very difficult in audio files.
- Video shot/sub-shot detection and key-frame extraction: creating an image storyline automatically helps in navigation through unstructured fieldtrip video recordings, or experiment recordings.
- Camera motion detection: helps to find relevant parts of the recording, e.g. by labeling panning or zooming.

2.2. Audio recognizers
Audio recognizers operate on the audio stream of the recording and focus on delivering utterance segmentation of the signal, which usually is a conversation of two or more subjects. This functionality, which can be seen on Figure 1, is obtained with three main steps:
- Audio segmentation: Noise-robust segmentation of the audio stream into homogeneous regions inserts boundaries e.g. between speakers, when an utterance ends or at other significant acoustic changes. The algorithm is capable of providing fine-grained segmentation of speaker utterances. The user can control the granularity of segmentation by tuning a corresponding feedback parameter.
- Speech/no speech classification of the segments: finds audio segments, which contain human speech in a language-independent way. It also allows the user to provide several examples of human speech segments (and also examples of non-speech segments) from the given recording, so the model of human speech can be updated to better match the features of a given recording.
- Speaker diarization: A language-independent intra-document speaker clustering algorithm labels identical speakers within a single document with the same ID. The results can be used for removing the interviewer in a recording, or for extracting material from specific speakers from a recorded discussion.
- Vowel and pitch contour detection: The pitch contour recognizer can allow researchers to graphically specify typical pitch contours and search for similar patterns.

2.3. Video recognizers
Video recognizers focus on delivering a complete segmentation of gesturing action in the video recording and delivering useful analysis of such. This is performed by a sequence of following steps:
- Skin color estimation recognizer: using the information about changes in consecutive frames the algorithm estimates which parts of the frame have showed more movement than the others. Later these regions are analyzed to estimate the color ranges responsible for human skin.
- Hands and head tracking recognizer: using the information about skin color the algorithm detects the coordinates of hands and head for each frame of the video. These coordinates serve as the base to further advanced analysis: using the time information the speed and acceleration of the hands is calculated and movements that represent gesturing action are detected. The user is able to control
the sensitivity of the detection with a set of numerical parameters. Within the boundaries of movements information about each of them is calculated, in particular:

- the area covered by the movement,
- the orientation of the hand,
- and the speed of the movement.

Furthermore, using the position of the head and hands the gesture space grid according to McNeil [2] is projected over the gesturing person and all the positions of the hands are annotated according to this grid. This can be seen on Figure 2, where a result of the said recognizer is shown.

3. Web services scenario

In the context of the European CLARIN research infrastructure [3] we are making the AVATecH recognizers available as a web service. The idea behind this is, not unlike the common UNIX philosophy, to be able to rely on a wide set of simple yet powerful modules that can be combined according to the end user’s individual needs. The output of a single module can then be fed into another module, and so on. Several frameworks to create such processing pipelines exist, like WebLicht [4] and Taverna [5]. To be able to integrate the RESTful AVATecH web services into the aforementioned frameworks, an appropriate metadata description needs to be provided for each of them. This is done in the form of WADL1 and CMDI2 descriptions.

An additional advantage, especially relevant for this case, is that processing software that cannot be freely distributed among the scientific community for local installation (e.g. for licensing reasons or hardware requirements) can be offered and used after all. Of course this also implies that the scientific end user depends more on the availability and the level of service offered by the web service hosting party. This aspect too is catered for in the CLARIN architecture. The centers, where the web services are running, are closely monitored both technically (e.g. with Nagios3-probes) and organizationally (via assessment procedures as the Data Seal of Approval4). An aspect that still has to be studied closer is the feasibility of transferring large multimedia objects via web services. CLARIN, in cooperation with the collaborative data infrastructure EUDAT5, will perform a series of pilot experiments to assess such scenarios. Some first preliminary results (using WebDAV to upload large files to a computing center) seem to be promising in this respect.

3.1. Deployment of the service

To make AVATecH recognizers available through REST6 we have used the generic RESTful web service wrapper CLAM7. It offers the functionality to connect command line tools to REST. As AVATecH recognizers take an XML blob of parameters via standard input, read input files, create output files and produce a message log on standard output, they can be considered command line tools with control pipelines. To configure a recognizer in CLAM it is necessary to provide metadata about the input and output parameters and files. In order to be uses in a

---

3. [http://www.nagios.org](http://www.nagios.org)
4. [http://www.datasealofapproval.org](http://www.datasealofapproval.org)
5. [http://www.eudat.eu](http://www.eudat.eu)
web browser, CLAM brings CSS and Javascript code to render user-friendly web pages from the underlying XML / REST data. Some small patches (tier files are XML, logs are similar to CSV) already make AVATeCH recognizers quite easy to use in a browser. For plain REST use with a "robotic" client, this is not necessary. A wrapper is used to send parameters as XML and detect global status events (failed / done) in the logs. CLAM can generate command line arguments itself, but only a bit of extra code is needed to wrap parameters into XML. Without this it would also be possible to let AVATeCH clients generate that XML, but using CLAM it felt more appropriate to keep all individual parameters explicitly described in metadata and sent separately. CLAM can be run either as a stand-alone server with a built-in web server or connected to Apache using WSGI. The latter makes it easy to install a number of different AVATeCH recognizers on the same server. Each invocation of a service in CLAM uses a separate workspace directory where services can run in an asynchronous way: After setting parameters and uploading the input files, the caller can start processing and check back from time to time to poll for a "finished" status. We extract this information from the continuous log stream of AVATeCH. When recognizers are run on the user's computer or in the old "raw TCP/IP pipeline for I/O, input and output files on a shared network drive" way of AVATeCH, the caller would have to keep the connection open until processing has finished. In that sense, using CLAM makes it easier to process large files over the network. As transfer of the files is an issue both with CLAM and in the old network drive approach, we plan to make it possible to refer to files by persistent identifiers. That way the server that will run a recognizer can pick a copy (e.g. of an input video file), which is as close as possible in terms of network connectivity. It might even have already had a mirror copy of a corpus containing that file on a local hard disk. When PIDs are not available, MD5 or SHA1 checksums can be an alternative way for referring to large media input files.

### 4. Experiments

We have performed a range of experiments tests to assess how our methods can increase the effectiveness of researchers’ work. By the measure of effectiveness we consider the difference between the time necessary to create annotation for given media with and without our algorithms. This value is not easily calculated, as the time necessary for annotating a time unit of media depends on factors like: the purpose of the recording and contents of the media; what exactly from the contents needs to be analyzed and annotated; the person performing the annotation and their expertise. In order to estimate this usefulness of our methods we have created scenarios in which researchers had to perform a number of annotation tasks to answer different research questions. The tasks have been chosen to represent a very common set of actions taken by researchers that are annotating their recordings. For each experiment the tasks have been first performed manually by several researchers and the time necessary to carry them out was measured and averaged. As another step, the same tasks have been performed with the help of recognizers. In each experiment after the automated annotation have been created a human annotator has verified them in order to perform necessary corrections, as the automated annotations were never flawless. The time to preform these corrections is measured and included in the total time of the automated annotation scenario.

#### 4.1. Audio

To test the efficiency of audio recognizer we have created the task of utterance segmentation and speaker classification. A field recording was used, representing typical noise and quality conditions for such material. One part of the file was annotated with the help of recognizers, another one (of equal length) without them. The result of manual annotation was annotated with 557 segments, whereas the scenario with recognizers one with 457. As we can see, there are some differences in the results of annotation work, however the values we have calculated for these scenarios are independent of this difference.

The results of our experiments can be seen on Table 1. Both recordings were of around 40 minutes length and after the annotation process the first one had 557 annotation blocks created, whereas the second one had 457 annotation blocks. This lets us see that the first recording had more dialogue action and therefore it resulted with more annotation blocks. However, the time necessary to create the annotation for the first file is significantly larger than for the second file, these times equal 1800 and 780 minutes respectively. As we can see, the scenario supported with the recognizers took much faster to annotate, with the annotation speed almost twice as fast – 585 blocks per minute comparing to 309 blocks per minute in the manual annotation scenario. The resulting efficiency of annotation for audio segmentation with recognizers increased by 0.49%.

#### 4.2. Video

The annotation to be done were first consulted with researchers from the field of gesture and sign language research, to match the annotations that are typically performed with this type of work. The task included labeling all the gestures in a short video recording and then annotating these segments with several features for each gesture, namely the size of the gesture, the speed of hand movement and the orientation of the hand. It can be seen that these typical annotations match the functionality of the Hands and Head Tracking recognizer, as it has been developed with close collaboration with Max Planck Researchers, in order to deliver the set of features that could be used with the most commonly performed annotation tasks. The said features (size, speed and orientation of the gesture) are usually estimated by a human annotator in a very rough way and using a very subjective scale (e.g. “large area” or “medium area”). They are also very time consuming, as each gesture needs to be spotted in the recording and then re-watched several times.

![Table 1. Results of the audio segmentation experiment.](attachment:image.png)

<table>
<thead>
<tr>
<th>Manual</th>
<th>Automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media length (minutes)</td>
<td>Annotation blocks</td>
</tr>
<tr>
<td>40.55</td>
<td>557</td>
</tr>
<tr>
<td>40.07</td>
<td>457</td>
</tr>
<tr>
<td>Annotation density (blocks/minute)</td>
<td>Annotation time (minutes)</td>
</tr>
<tr>
<td>13.73</td>
<td>1800</td>
</tr>
<tr>
<td>11.40</td>
<td>780</td>
</tr>
<tr>
<td>Av. annotation speed (blocks/minute)</td>
<td>Annotation time/media length</td>
</tr>
<tr>
<td>0.309</td>
<td>44.38</td>
</tr>
<tr>
<td>0.585</td>
<td>19.46</td>
</tr>
</tbody>
</table>

The results of our experiments can be seen on Table 1. Both recordings were of around 40 minutes length and after the annotation process the first one had 557 annotation blocks created, whereas the second one had 457 annotation blocks. This lets us see that the first recording had more dialogue action and therefore it resulted with more annotation blocks. However, the time necessary to create the annotation for the first file is significantly larger than for the second file, these times equal 1800 and 780 minutes respectively. As we can see, the scenario supported with the recognizers took much faster to annotate, with the annotation speed almost twice as fast – 585 blocks per minute comparing to 309 blocks per minute in the manual annotation scenario. The resulting efficiency of annotation for audio segmentation with recognizers increased by 0.49%.
in order to estimate the said features. Usually also some corrections are necessary, as the video annotation proceeds. Four annotation experiments were performed, 2 using recognizers and 2 without them. The resulting annotation time for the scenario with recognizers equaled only 28% of the time necessary to perform fully manual annotation. As it is possible to see, the efficiency of recognizer-supported annotation proved to be much higher comparing to manual annotation.

5. Conclusions

The experiments presented in this paper show clearly that with the support of low-level analysis of audio and video recordings the time necessary for their annotation can be significantly shortened. The resulting annotations, after necessary corrections by a human annotator, are of high quality and can be used for further research tasks.

6. References


TüNDRA: TIGERSearch-style treebank querying as an XQuery-based web service

Scott Martens

Seminar für Sprachwissenschaft, Universität Tübingen
Wilhelmstr. 19, Tübingen, Germany
scott.martens@sfs.uni-tuebingen.de

Abstract

TüNDRA - the Tübingen annotated Data Retrieval Application - is a CLARIN-D supported project to provide a general search tool for treebanks, available as a web service and running on top of scalable, off-the-shelf XQuery engines. TüNDRA’s query language is based on TIGERSearch, converting on the fly queries written in a constraint-satisfaction style into XQuery’s procedural select-and-filter style, using an intermediate formalism inspired by model theory. This approach makes it possible to do intelligent query optimization, to adapt this approach to back-ends other than XQuery and to apply it to broad classes of annotated language data rather than some specific set of treebank formalisms.

1. Introduction

Treebanks and other text corpora with complex annotation are commonplace in linguistic research, but there are few generic tools for searching in them and none that operate as web services. One of the most widely-used treebank search applications is TIGERSearch (Lezius, 2002), a freely accessible tool with a simple query syntax and a large existing installation base. However, it is only available on a limited number of platforms and is no longer actively supported by its original authors.

Offering an engine comparable to TIGERSearch as a web service makes treebank search more accessible. It eliminates problems of local installation and the limitations of desktop hardware. Using networked, distributed, robust hardware means a larger user community can search more and larger corpora. Making treebank search a reality on scalable, standard data management systems is an important part of bringing it to web services, since it makes it possible to do web search using existing, well-supported, adaptable back-end systems.

It is generally uncomplicated to encode the structural features of treebanks in XML. Although exceptions exist, treebanks can usually be treated as collections of typed feature structures - words and constituent labels with various features - organized into parent-child and sequential relations with each other. An XML document consists of tags with named attributes each of which has a value. Tags can contain other tags, and are encoded in a sequence. This makes XML a natural approach to encoding treebanks using standard, widely supported data structures, and XML is already widely used for data exchange in computational linguistics. XML-encoded materials can be searched using the XQuery language1 in a fashion analogous to the role of SQL in relational database management. XQuery has been a W3C recommendation since 2007 and is supported by a growing number of free and commercial software suites.

(TüNDRA) bridges TIGERSearch-style syntax and XQuery. The principal problem this poses is the fundamental difference in paradigms between the two: TIGERSearch queries are structured and understood in terms of constraint satisfaction, while XQuery uses a procedural select-and-filter paradigm more naturally suited to databases. TüNDRA uses an intermediate representation that reorganizes user-provided constraints into a logical form well-suited to constructing semantically equivalent XQuery statements. It can also be adapted to other select-and-filter-style query languages.

TüNDRA is being implemented on WebLicht,2 using the BaseX open-source XQuery engine3 for retrieval.

2. Brief overview of TIGERSearch querying

TIGERSearch analyzes syntax trees as a collection of nodes that have named features with string values and edges that may or may not have labels. Each sentence corresponds to a single rooted tree. A system of secondary edges is also supported, connecting nodes separately from the normal tree structure and allowing some support for more powerful directed graph structures.

Terminal nodes in the tree have a fixed ordering corresponding to the word order of the sentence the tree represents. Edges may “cross” - there is no requirement that constituent nodes and their edges be organized so that trees are projective.

Figure 1 is an example of a tree from the TÜBa-D/Z treebank (Hinrichs et al., 2004) stored and displayed in TIGERSearch.

Query 1. An example TIGERSearch query compatible with Tündra.

```xml
#1:[cat="NX"] > #2:[lemma="das"] & #1 >HD #3:[pos="NN"] & #2 . #3
```

1. http://www.w3.org/TR/xquery/
2. http://weblicht.sfs.uni-tuebingen.de/englisch/weblicht.shtml (But see also Hinrichs et al. (2010).)
TIGERSearch provides a language for querying on the features of nodes, on the parent-child and sequential relationships between them, and on various other predicates on nodes. For example, Query 1 is an example of a TIGERSearch query that finds all sentences containing a noun phrase (NX in the TüBa-D/Z markup scheme)\(^4\) that has an immediate child nodes a node with the lemma “das”, followed by an ordinary noun (NN) which is also the head of the noun phrase (indicated by the edge label HD).

Query 1 should be interpreted as matching all sentences where the following constraints hold:

1. There is a node \#1 with the cat attribute value NX.
2. There is a node \#2 with the lemma attribute value das.
3. There is a node \#3 with the pos attribute value NN.
4. Node \#1 is the parent of node \#2 (indicated by the “>” relation).
5. Node \#2 is the head of the constituent node \#1. (indicated by HD).
6. Node \#1 is the parent of node \#3.
7. Node \#2 is immediately followed by \#3 (indicated by the “.” relation).

In queries, variable names are prefixed with the “#” character and they make it possible to query for complex structures. Users can perform queries on a number of structural relations and conditions other than immediate parent/child relations and attributes, including queries relating to siblings, surface order, ancestry at arbitrary distances, and a number of other features.

For a more detailed description of TIGERSearch query syntax and semantics, see König et al. (2003).

3. XML representation

TüNDRA employs an XML encoding to store treebanks for access by the BaseX XQuery engine. This XML structure is very compact and makes it straight-forward to devise XQuery functions equivalent to user-provided constraints.

\[<\text{pp y="st1"}>\]
\[<\text{c n="0" c="VROOT" s="0" f="4", e="root"}>\]
\[<\text{c n="1" c="SIMPX" s="0" f="3", e="--"}>\]
\[<\text{c n="2" c="LK" s="0" f="0", e="--"}>\]
\[<\text{c n="3" c="VFIN" s="0", e="HD"}>\]
\[<\text{c n="4" t="Vunrueten" i="Vunrueten" p="VFIN" m="3sitt" o="0", e="0", f="0", e="HD"}>\]
\[</c>\]
\[<\text{c n="5" c="MHP" s="1", f="3", e="--"}>\]
\[<\text{t n="1" c="die", l="die", p="ART", m="nsp", o="1", s="1", e="--"}>\]
\[<\text{c n="6" c="NX", s="1", f="2", e="OA"}>\]
\[<\text{t n="4" c="Veruntreute", l="Veruntreute", p="VFIN", m="3sitt", o="0", e="0", f="0", e="HD"}>\]
\[</c>\]
\[<\text{c n="10" c="NX", s="3", f="3", e="OA"}>\]
\[<\text{t n="11", c="Spendengeld", l="Spendengeld", p="NN", m="asm", o="3", s="3", f="3", e="HD"}>\]
\[</c>\]
\[<\text{c n="12", t="?", l="?", p="\%", m="\%", o="\%", s="\%", f="\%", e="\%"}>\]
\[</c>\]
\[</pp>\]

Figure 2: TüNDRA XML representation of the same sentence as in Figure 1.

Figure 2 is the same tree as Figure 1 in the XML encoding used in TüNDRA for the TüBa-D/Z treebank. The semantics of the two representations are the same with respect to the characteristics and relations that are accessible in TIGERSearch: Nodes in the treebank correspond to tags in XML and parent-child relations in the treebank correspond to parent-child relations between tags. Node attributes are represented by XML tag attributes, including edge labels.

For compactness, and because human-readable XML is not a major consideration in this application, attribute and tag labels are one or two letters long. In Figure 2, the tag `<c>` indicates a non-terminal node - a constituent in the syntax tree - while `<t>` indicates a terminal node, and `<pp>` is a container tag for a sentence. Similarly, the attribute `c`, used only for tags `<c>`, indicates the category of a constituent node, while the `p`, `t`, `l` and `m` attributes indicate respectively the part-of-speech, token, lemma, and morphological categories of terminal nodes. The attribute `e` indicates the label on the edge between the node each tag corresponds to and its parent in the syntax tree.

The attribute `n` is a numerical attribute that uniquely identifies each node in the treebank.

The other numbered attributes - `o`, `s` and `f` - provide information about the sequence of constituent and terminal nodes. Since trees can be non-projective, the order of tags in XML cannot adequately encode the sequence of elements in the corresponding syntax tree, and these numerical attributes are used instead.

Similarly, other attributes are used to encode secondary edges and their labels.

This XML format is not intended as a data exchange format, and has not been designed for human readability or
accessibility in mind. It is particular to the T˚Ba-D/Z tree-
bank and other treebanks encoded in similar ways. The
T˚uNDRA approach is readily adaptable to other treebanks
with very different structures and annotations, but requires
modifying this XML representation. It is not a very compi-
cated format and this project includes developing convert-
ers too and from various standard treebank formats.

4. XQuery and optimization

XQuery operates using select-and-filter processes similar to
those used in standard database query languages like SQL,
and its syntax reflects this paradigm. Query 2 is an example
of an XQuery statement intended to search treebank data
that has been encoded as described in Section 3..

Query 2. An example XQuery statement.

```xml
for $x in //*[@c="NX"]
where $x/*[@p="ART"]
return $x
```

Query 2 is interpreted in a procedural way as:

1. Select all tags that have an attribute c with the value
   NX (a noun phrase in the T˚uBa-D/Z formalism) and
   assign them to variable $x$.
2. Loop through all the tags in $x$, and filter out all those
   that do not have a child node with the attribute p and
   value ART (articles).
3. Return those that remain.

A major consequence of the select-and-filter paradigm is
that there are many XQuery statements that produce equiva-
 lent results but that have radically different run-times. Compare Query 2 with Query 3. Query 3 selects all tags
with a p attribute ART and then filters the result based on
whether its parent has the c attribute NX. If the number
of nodes with the part-of-speech tag ART is significantly
smaller than the number of constituents with the category
NX , then Query 3 will run proportionately faster.

Query 3. An alternative XQuery statement semantically
identical to Query 2 but likely faster.

```xml
for $x in //*[@p="ART"]
where $x/parent::*[@c="NX"]
return $x
```

Selecting tags in order, starting with the least frequent, and
performing the most aggressive filtering before the least ag-
gressive, improves query performance dramatically. And,
conversely, failing to optimize can slow run-times down, in
the worst case exponentially.

This problem makes any simplistic approach to converting
TIGERSearch-style syntax into XQuery routines too ineffi-
cient for use as a web service. An adequate solution must
take query optimization into account.

5. T˚uNDRA Intermediate Query
Representation

5.1. Model Theory basis

To solve the problem of producing reasonably efficient
XQuery statements in response to TIGERSearch-style
queries, T˚uNDRA uses an intermediate representation that
sits between user input and XQuery generation. This rep-
resentation - T˚uNDRA Intermediate Query Representation
(TIQR) - employs a model theory-based formalization of
the query.

Martens (2011) argues that language data as a whole - in-
cluding all varieties of corpus annotation - can be under-
stood in terms of finite model theory. Using model theory,
it is possible to make generalizations about the properties
different linguistic theories and productively compare
them. Model theory is the study of mathematical and ab-
stract structures through a hybrid of set theory and formal
logic. (Chang & Keisler, 1977) It has found a number of
applications in database theory, computational complexity
theory, and elsewhere in computer science. (Makowsky,
1992; Gr¨adel, 2003; Libkin, 2009) In linguistics, it is most
associated with Montague Grammar, but a number of for-
ternal treatments of HPSG have also deployed model theory.
(Montague, 1973; Sag & Wasow, 1999; Richter, 2007)

In model theory, a structure $S$ is a triplet $[D, \Sigma, I]$ where:

- $D$ is the domain of $S$, consisting of a set of discrete
  items.
- $\Sigma$ is a set of functions and relations that take as argu-
  ments members of $D$, $\Sigma$ and $D$ can be combined with
  conventional logic operators to form predicates in
  a first-order logic.
- $I$ is an interpretation. It assigns to each predicate
  formed from $\Sigma$ and $D$ a boolean value of either True
  of False.

From this perspective, we can formalize a tree $T$ in a tree-
bank as a structure $\Sigma$ whose domain $D_T$ is the nodes
of the tree, and where ancestor-descendent relations, order-
ings, and features are represented as predicates formed us-
ing functions and relations in $\Sigma_T$. The interpretation $I_T$
determines which combinations of members of $D_T$ and $\Sigma_T$
are true of any particular tree and which ones are not.

A query $Q$ can then be understood as a structure $\Omega$, with its
own domain $D_Q$ and interpretation $I_Q$ but with the same $\Sigma$
as the treebank being queried.

Query 1, for example, is a query on the attributes and re-
relationships between three nodes - #1, #2 and #3 - so its
domain can be represented as:

$$D_1 = \{n_1, n_2, n_3\}$$

And its interpretation $I_1$ is a list of predicates that take
members of $D_1$ as arguments, as in Table 1.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>False</td>
</tr>
</tbody>
</table>

These same predicates can be viewed as a graph, as in Fig-
ure 3. As long as no relation or function in $\Sigma$ takes more
than two arguments, this model theory formalism is equiv-
alent to a graph. However, the formalism itself is able to
support more complex functions.
A tree $T$ in a treebank is a match for query $Q$ if there is a mapping from $D_Q$ - the domain of $Q$ - to $D_T$, the domain of $T$, such that every predicate in the interpretation $I_Q$ remains true under the interpretation $I_T$. The query is satisfied by every such mapping.

TIQR is an implementation of this model theoretic formalism, with the addition of some special extensions for handling disjunctions (logical OR’s) and negation in queries.

### 5.2. Queries with disjunctions

TuNDRA, like TIGERSearch, supports boolean AND and OR operations on the relations, predicates, attributes, and attribute values in queries. We will define simple queries, like Query 1, as queries that have no disjunctions (no constraints joined together by the boolean OR operator) or negation (which will be addressed in the next section). For simple queries, the intermediate representation consists of a list of predicates generated from the user’s input as in Table 1.

Extending TIQR to queries with disjunctions requires sorting query predicates into a hierarchy of containers, within which all predicates in the same container must be simultaneously satisfied, and joining these individual interpretations using formal logic operators.

Query 4 is an example of a query with disjunctions, with variable names standing in for node specifications:

**Query 4.** A query that uses disjunctions and parentheses.

These relations cannot be satisfied by a single interpretation, as in the previous section, because if some predicates are satisfied, others do not need to be satisfied. The most compact way to resolve this problem is by extending the formalism of model theory to support logical predicates on multiple interpretations over the same domain.

In this extended formalism, a structure is defined by

$$\mathcal{S} = [D, \Sigma, \mathcal{I}]$$

where $\mathcal{I}$ is predicate formed by multiple interpretations and their logical connectors.

As an example, given Query 4, we set its domain to:

$$D_4 = \{n_1, n_2, n_3, n_4, n_5, n_6\}$$

We then break Query 4 up into multiple interpretations as in Table 2.

And the extended interpretation $\mathcal{I}_4$ is a formal statement combining these interpretations:

$$\mathcal{I}_4 = I_a \land (I_b \lor I_c) \land (I_d \lor I_e)$$

Alternately, the extended interpretation $\mathcal{I}_4$ can be seen as a kind of tree, as in Figure 4.

Each interpretation in the extended interpretation - each node in Figure 4 - is equivalent to a simple query. We can, therefore, handle queries with disjunctions as disjunctions of simple queries.

### 5.3. Queries with negation

TuNDRA’s treatment of negation is substantially different from TIGERSearch. Given a query of the form:

**Query 5.** #1 $\neg$ #2

TIGERSearch matches only sentences that contain a node matching #1 and a node matching #2, but only where #1 is not a parent of a match for #2. In TuNDRA, negation
Figure 4: A hierarchy of interpretations (from Table 2) representing Query 4.

is existential, that-is-to-say, the same query will match sentences that contain a match for #1 that is not a parent of a node matching #2, without regard to whether any other nodes in the sentence match #2.

Queries containing negation are processed analogously with queries using disjunction. Query 6 is an example of a query that uses several kinds of negation permitted in TüNDRA.

**Query 6. An example of a query with negative operators.**

\[
\neg \text{ [word="Hamburg"] } \land \neg \text{ [cat=VP] } \land \text{ [cat=NX] } \neg \text{ [word=Berlin] }
\]

The query is interpreted as matching any sentence that:

- does not contain any node with the word attribute "Hamburg",
- contains at least one node that does not have a cat attribute with the value VP,
- contains at least one node that has a cat attribute with the value NX, but only if that node is not the parent of a node with the word attribute "Berlin".

Given, for Query 6, the domain:

\[\mathbb{D}_6 = \{ n_1, n_2, n_3, n_4 \}\]

We break Query 6 up into a number of interpretations, as in Table 3.

These are then joined into an extended interpretation as:

\[I_6 = I_a \land \neg I_b \land \neg I_c \land \neg I_d\]

This too can be interpreted as a tree, as in Figure 5.

---

**Table 3: A set of interpretations corresponding to Query 6.**

\[
\begin{align*}
I_a &= \{ \exists n_1 \text{ attr_word_Hamburg}(n_1) \} \\
I_b &= \{ \exists n_2 \text{ attr_cat_VP}(n_2) \} \\
I_c &= \{ \exists n_3 \text{ attr_cat_NX}(n_3) \} \\
I_d &= \{ \exists n_4 \text{ attr_word_Berlin}(n_4) \\
&\quad \text{ parent}(n_3, n_4) \}
\end{align*}
\]

---

**6. Transforming queries into XQuery**

Transforming simple queries - queries without disjunctions or negations - into XQuery statements amounts to treating the intermediate representation as a graph and walking the graph. To convert the TIQR form of Query 1 (Table 1) into an XQuery statement, we use the following procedure:

1. Select the domain member that is likely to be present in the treebank the smallest number of times, based on knowledge of the contents of the treebank. In this case, select node2, because the number of instances of the lemma “das” is likely to be smaller than the number of ordinary nouns (node3) or noun phrases (node1).

2. Construct an XQuery statement that selects all tags matching that node, using all node attribute predicates that refer to that node:

   for $node1$ in //*[@l="das"]

3. Select the root of the sentence containing each match. The root node of every sentence is indicated in TüNDRA XML by the tag label pp, so we need only identify an ancestor of the selected node with that label:

   for $sent$ in $node1/ancestor::pp

4. Select a node that is in a predicate with the first node
and construct a for statement that selects the corresponding tags in relation to the tags matching the first node:

```xml
for $node2 in $node1/parent::*[@c="NX"]
```

5. Repeat steps 4 and 5 until all nodes have been processed:

```xml
for $node3 in $node1/parent::*[@p="NN"]
```

6. Construct an XQuery where statement that restricts query matches to those that match the remaining predicates:

```xml
where $node2/@f=(($node3/@s - 1)
```

7. Return the matching sentence:

```xml
return $sent
```

This amounts to an optimal walk through the graph in Figure 3, using knowledge about the corpus to drive a heuristic that attempts to identify the best path. The path through the graph, in turn, tells TüNDRA in what order nodes must be selected and filtered to make a good XQuery statement with the same semantics as its TIQR source.

For more complex queries - those with disjunction and negation - the TIQR representation consists of a predicate combining individual simple queries. Constructing an equivalent XQuery statement is a walk over the logic tree implied by the TIQR form (as in Figures 4 and 5), constructing an XQuery statement for each included simple query, and then joining them using appropriate XQuery syntax.

### 7. Implementation

Figure 6 is a schematization of the TüNDRA architecture. Corpora must be prepared in advance by converting them into the appropriate XML format (cf. Section 3.) and loading them into BaseX. User queries are parsed by code constructed with the JavaCC parser generator, and the intermediate representation, path discovery, and XQuery statement constructor, are all implemented in Java.

XQuery statement generation currently uses very simplistic heuristics driven only minimally by corpus data: TüNDRA uses a list of attributes known to have very diverse values, i.e. word, lemma, and others indicative of specific lexical items, to determine which query nodes should be selected first. This is enough to produce very large improvement in performance over using no optimization.

The TüNDRA query syntax differs in a number of ways from that of TIGERSearch. Some of these differences are trivial - like that TüNDRA uses Unicode regular expression syntax because that is the standard for XQuery statements - but others are more substantial. The largest differences are those affecting the interpretation of negation in queries (cf. Section 5.3.), but other differences exist as well. The motivation underlying these changes is, in part, that many of kinds of queries that TIGERSearch cannot run are relatively simple to implement in XQuery. But, a major motivation is the desire to follow a principle of least surprise in design: The behavior of the query system should match user expectations as much as reasonable, especially since the purpose of TüNDRA is to make the system widely accessible. Any query language complex enough to be useful will inevitably surprise the user with some unexpected result, but we can at least try to minimize that effect for simple queries.

Consider the TIGERSearch query below:

**Query 7. A TüNDRA query with different semantics than in TIGERSearch.**

```xml
[cat="NX"] !>* [pos="ART"]
```

Query 7 is an attempt to identify noun phrases ([cat="NX"]) that do not have as descendents (!>* ) any articles ([pos="ART"]). There is no reason to expect this query to exclude sentences that have no articles at all. TüNDRA does not exclude them, at the cost of not being semantically identical to TIGERSearch.

A TIGERSearch equivalent query is still possible in TüNDRA, should users require it, but is slightly more complicated. To find all sentences that contain both an article and a noun phrase that does not dominate an article, one can enter:

**Query 8. Equivalent TüNDRA query.**

```xml
[pos="ART"] & [cat="NX"] !>* [pos="ART"]
```

### 8. Comparison to other research

With respect to other treebank search applications, few are available that have the functionality of TIGERSearch or target the same kinds of users and treebanks. Many do not have large installed bases and are often bound to particular kinds of markup, used in particular treebanks. The kind of generality TüNDRA can provide is not present in most treebank search tools.

Except for ANNIS, none appear to offer web service functionality.

Tgrep2, and applications built around it, do not use indexes, limiting their scalability, and search only projective trees in the Penn Treebank format. They do not support any more complex markup.

The ICECUP corpus exploration program and its accompanying query scheme support typed feature structures and labeled dependencies, but are only available on the Windows operating system and are very strongly oriented towards the markup in the ICE-GB corpus.

PML-TQ (Pajas & Stˇep´anek, 2009) queries corpora encoded in the Prague Markup Language (Hana & Stˇep´anek, 2012), an open standard for treebank exchange. PML-TQ has a sophisticated query system and an adaptable underlying XML-based data format. It does not run on standard back-ends, nor is it provided as a web service.

ANNIS uses a TIGERSearch-inspired query language, like TüNDRA, and searches treebanks in the PAULA format.

---

---

5. http://javacc.java.net/


(Chiarcos et al., 2008), which is used in a number of corpus annotation applications. It uses PostgreSQL as an underlying database engine and can operate as a web service. The TüNDRA approach differs primarily in separating the back-end entirely from the user’s query language, resulting in a more compact and flexible scheme. The results of the A2 project at Universität Tübingen, particularly the fsq query language (Kepser, 2003) and MonaSearch (Kepser, 2005; Maryns & Kepser, 2009) are similar to the TüNDRA approach in their use of finite structures and restricted second order logics as a query formalism. However, they do not use indexes or underlying databases and in their current implementations have limited ability to query complex structures like non-projecting trees and non-tree (secondary) edges.

A major difference between TüNDRA and related projects is its design philosophy. The logic of offering search as a web service puts TüNDRA’s focus less on logical completeness and concision and more on accessibly providing users with a tool that efficiently does what they want it to do. Consequently, it has a sketchy and highly amendable XML storage format, so that it can readily support future annotation needs, and a simple and powerful intermediate formalism so that new query primitives can easily be introduced. Using XQuery means that querying new annotation requires little more than finding a way to do it in XQuery.

### 9. Future Work

We are in the process of bringing this application out of the prototype stage and making it available through WebLicht. The application will accept user queries and display matching trees promptly (as soon as the first one is found) in a graphical display.

In its current form, TüNDRA supports all queries that TIGERSearch supports, along with some that it does not. However, it does not yet support a complete set of predicates and relations to match the full power of the TIQR intermediate representation and XQuery.

One consequence of the difference in the semantics of negation between TüNDRA and TIGERSearch is the need to introduce a mechanism to invert relations. For example:

**Query 9.** #1 !> #2

Query 9 finds sentences with a node matching #1 that does not immediately dominate a node matching #2. However, there is presently no way to query for a sentence where there must be a node matching #2 which may not be dominated by a node matching #1. The same problem arises with other operators. The TüNDRA framework and XQuery are more than powerful enough to handle queries of that type, but devising a user-transparent extension to the query language is less obvious.

Negating large substructures is also an extension to TüNDRA that is not formally complicated, but difficult to make user-transparent.

One solution under consideration is to employ a graphic query construction interface, along the lines of the one already available in TIGERSearch. Questions of usability are central to application development of this kind, and the progressive improvement of TüNDRA in response to its future user community is a continuing and open-ended commitment.

The heuristics currently used to devise efficient XQuery statements are very simplistic and could certainly be improved. For relatively small corpora, they are reasonably good, but growing treebanks constitute a good motivation to improve them.

---

9http://tcl.sfs.uni-tuebingen.de/a2/
Another direction for future work is using other backends than XQuery. The TIQR formalism is of sufficient generality that it can be adapted, with some effort, to alternative underlying database systems such as SQL and the W3C’s proposed SPARQL standard.10

10. Conclusion

The basic outline of TüNDRA - compact XML encodings for treebanks, intermediate logical representations, and knowledge-driven query optimization - are sufficiently general and powerful that they can be readily adapted to novel treebank and annotation standards. TüNDRA’s intermediate query representation is formally powerful enough to support very diverse corpora and very rich data structures. The use of XQuery ensures compatibility with commercial and open-source XML software into the foreseeable future. Furthermore, the TIQR intermediate representation offers a generalized solution to transforming satisfaction-style treebank queries into efficient database calls.

The application currently targets the TüBa-D/Z and corpora with similar styles of annotation, but the framework is readily applicable to other treebanks and annotation styles, including, with some adaptation, semantic, pragmatic, morphological, and other classes of annotation. The functionality to support full graphs and even hypergraphs is already present in the intermediate representation’s formalism - as long as a user-acceptable querying syntax can be devised. The broad outline is theory-independent and therefore fits well into a web services infrastructure oriented towards the provision of flexible tools that are not tied to specific linguistic theories.

This is an extensible framework with the potential to provide treebank and annotated corpus search to a wide audience, working with data sources and applications well beyond those currently envisioned.

11. Acknowledgements

This work is supported by CLARIN-D (Common Language Resources and Technology Infrastructure, http://de.clarin.eu/), funded by the German Federal Ministry for Education and Research.

12. References


http://www.w3.org/2009/sparql/wiki/
How to Turn a Desktop Application into a Web-Interface?
– Archiv-Editor as an Example of Eclipse RCP and RAP Single Sourcing

Author: Christoph Plutte
Institution: Berlin-Brandenburg Academy of Sciences and Humanities
Research Project: DARIAH-DE - Digital Research Infrastructure for Arts and Humanities
Address: Jägerstraße 22/23, 10117 Berlin, Germany
Tel: +49 / (0)30 / 20370-685
Mail: plutte@bbaw.de

Abstract
This paper discusses the Archiv-Editor as an example of how a Eclipse Rich Client Platform based desktop program can be turned into a web-interface and made available as a web-service. Parallel development of desktop and web interfaces as well as the transformation of an already existing desktop application is a capability for which there is a great demand. With the Rich Ajax Platform the Eclipse Foundation provides a powerful framework to turn an Eclipse Rich Client Platform into a web-interface. The Archiv-Editor is a multilingual desktop application created for working with prosopographical data and the Person Data Repository. As the Person Data Repository and the Archive-Editor do not require a specific structure of statements and information, they are open to a wide variety of research projects in the Humanities and offer the infrastructure to combine and integrate data from divergent fields and research perspectives. The software architecture Eclipse RCP was chosen in order to combine desktop and web functionality. Eclipse RCP allows a compiling of sources as a RAP web application, thus making the Archiv-Editor available online.

Keywords: interface design, desktop to web, prosopographical data

1. Starting point
The Archiv-Editor\(^1\) is a software tool with a multilingual user interface created for working with prosopographical data and the Person Data Repository (PDR), a server-client repository system started in 2009 by the DFG-Project Person Data Repository\(^2\) at the Berlin-Brandenburg Academy of Sciences and Humanities (BBAW). Further development of the Archive-Editor has become part of the DARIAH-DE initiative for digital infrastructure in 2011. The aim of the Person Data Repository is to provide a decentralized software system for research institutions, universities, archives, and libraries that allows combined access to biographic information from different data pools. It is especially designed to meet the requirements at the BBAW where diverse research projects in the Humanities collect personal data throughout their work. This data gathered from multiple research perspectives according to distinct research questions is usually stored in completely different formats, ranging from text documents to relational databases, and can hardly be maintained after the research project has finished. Since 2009, the Person Data Repository has provided a productive solution for storing, searching and exchanging person data, including long term maintenance and tools such as the Archiv-Editor, for working on the data and collecting new data. The development of the PDR and the Archiv-Editor are based on the experience with the first version of the Archiv-Editor, which was developed in 2007 by TELOTA (The Electronic Life of the Academy)\(^3\) at the BBAW for the historical research project Preußen als Kulturstaat to investigate archival materials (Czmiel, Holtz 2007). Concepts and approaches that had proved to be useful were generalized and implemented to be thoroughly customizable and extendable.

At the beginning of the PDR project evaluations of repository software and prosopographical databases were made (Körner, Plutte, Roeder, Walkowski 2010) that led to the decision to construct a new system and a new data entry interface rather than extending any existing software. Although this was an expensive approach, the experiences with the first version of the Archiv-Editor were very promising, and a redevelopment of the software could lead to a perfectly customized program.

1.1. Enfolding the Heterogeneity of Data
The scientific cultures and disciplines at the BBAW and the kind of personal data and perspectives for which it is gathered differ greatly. The PDR has therefore chosen to use a very open data model based

---

\(^1\) http://pdr.bbaw.de/software/ae
\(^2\) http://pdr.bbaw.de/english

---

\(^3\) http://www.bbaw.de/en/telota
on XML that does not narrow the type of information stored, but allows the retention of any kind of statement about persons. These statements can then be classified according to customizable semantic, time and spatial dimensions (Walkowski 2009). This approach does not define a person as a single data record, but rather as compilation of all statements concerning that person. It is thus possible to display complementing as well as contradicting statements in parallel, a capability which meets one of the basic challenges of biographic research. To enable a hierarchical and complex classification of proper names and notions, words can be marked with a TEI\(^4\) compatible and fully customizable mark-up. Thus a very high level of atomization and complexity is implemented.

1.2. Usability
The complex data model of the PDR is diametrically opposed to the usability of the Archiv-Editor. The first step to usability was to encapsulate the XML – users do not see or edit XML. Since early 2011 however, when productive work with the Archiv-Editor began, it has become clear that for better usability it is not enough to just encapsulate XML: Research projects demanded to narrow the data entry fields and to enhance user guidance. In contrast to the – greatly appreciated – complexity and openness of the PDR data model, a familiar database interface such as a formula with certain fields (Name, Profession etc.) was desired. These adjustments were required both in the desktop program for offline work as well as in an online version for web access.

2. From the Desktop…
In order to combine these divergent requirements, the Archiv-Editor is designed modular. The Eclipse Rich Client Platform\(^5\) was chosen because of its extensibility and because it allows plug-ins for individual project requirements. While the PDR provides a solution to combine heterogeneous data and different research perspectives through a complex data model, the Archiv-Editor provides several general solutions, plus individual adaption, without cutting back the interoperability and exchangeability of the data.

Research is often done in archives where no internet access is available. In order to bring the data entry process as close as possible to the actual research and archival material, one goal was to provide an offline client interface equipped with a local database that contains a partial copy of the central repository. This feature was proven by the first version of the Archiv-Editor to be very useful. Therefore the development of the current version of the Archiv-Editor started with the desktop version, based on RCP. A data synchronisation process allows to commit the locally gathered data into the central repository and to update the database. Furthermore, the p2 provisioning system\(^6\) allows automated software updates provided

---

\(^4\)http://www.tei-c.org

\(^5\)http://wiki.eclipse.org/Rich_Client_Platform

\(^6\)http://www.eclipse.org/equinox/p2/
through a central update site.

3. … to the Web

To provide greater flexibility and independence from a locally installed Archiv-Editor, the web version of the Archiv-Editor is currently being developed and tested and will be published in the next months. It enfolds the same functionality as the desktop version and brings it into the browser. Researchers will then be able to work with the desktop version and with the online version. The development costs for the online version are much lower than they would be, for example, if an independent web-platform was programmed. This is because not only the logical parts of the desktop program can be reused for the online version, but the view components as well. Furthermore, upcoming extensions and new features do not have to be developed twice for desktop and web but can be integrated into both version with little effort.

3.1. Technical Details

The framework being used is the Eclipse Rich Ajax Platform\(^7\) (RAP) that allows to compile a RCP plug-in application against a RAP Target and to create a deployable web application that runs in a web application server such as Apache Tomcat.\(^8\) RAP is a powerful framework that supports both the reusability of the RCP source code and the extensibility of RCP through plug-ins. RAP uses RWT as widget toolkit through which SWT-widgets are compiled and shown in a browser. RWT supports most of the SWT widgets, though not all of their functionality is yet implemented. Though the Netbeans Platform\(^9\) also provides a plug-in based software architecture, which is comparable to RCP, it does not provide a framework to compile the graphical user interface into a web-application.

The graphical components of RWT are rendered on the server-side and then send to the client to be shown in the browser. This makes the presentation and style of the GUI independent from the client rendering, as with CSS for example. The clear advantage is that the layout of the Interface always looks the same and that no Java-plug-ins or other extensions of the browser are required. The disadvantage is that there is a great deal of rendering on the server, which consumes resources, and that interaction with the server is needed for almost all kinds of user interaction with the GUI.

Initially, the Archiv-Editor was designed as a pure desktop program because the off-line modus was an indispensable feature for the work in archives. However, already very early in the planning phase of the software and its plug-in components, the development of a RAP based online version was

---

taken into consideration. Thus the separation of functionality, components and the graphic presentation, was guided by the future need to separate specific desktop functionality and web functionality. This was an important advantage as the development of online version began, although both versions were not actually developed simultaneously but one after the other. As the final steps to finish the online version reach completion, parallel development of both versions will be done. The Archiv-Editor RAP uses more than 95% of the source code of the desktop version. Some of the functionalities such as MouseMoveListeners are not available in RWT and are outsourced into plug-ins that are not included in the RAP build. Furthermore, RWT does not yet thoroughly support PaintEvents and therefore painting had to be retailored for RWT compatibility. To ensure usability the Archiv-Editor encapsulates XML and makes it editable through a graphical XML-Editor, based on a SWT-widget that is not yet implemented in RWT. As an alternative a Vex\textsuperscript{10} based XML-Editor was developed, because the Vex Widget for visual XML editing proved to be powerful, easy to use and RWT compatible. Only minor changes in the source code of Vex were necessary to compile it against the RAP target.

3.2. Data storage

The online version of the Archiv-Editor does not work directly on the central repository but uses a custom database as an intermediate similar to the desktop version. Thus changes in the data do not directly affect the centrally stored data. Only after synchronisation, during which conflicts between different versions of the same data object are handled and shown to the user, is this accomplished. The resource costs for maintaining individual database storage for each user are not very high at the moment, this approach, however, allows users to thoroughly customize their own online workspace. With future development of a granular rights management and publication system, users will be able to upload their own private data objects into their Archiv-Editor RAP, work on their data, export it into PDF or HTML or commit it to the central data repository they are related to.

3.3. Archiv-Editor as a Service

Equipped with a separate and individual web-storage for the data a user works with, the Archiv-Editor is offered as a web-service to registered users. It is planed to allow unregistered users to use the Archiv-Editor. They would be considered temporary users and would have their own individual database, but without the authorisation to commit their data into the central repository without prior reviewing. Furthermore, import functionality will be enhanced in the future and made available in the online version. This would allow users to import their own data temporarily, work on that data, visualize it through visualisation tools in the Archiv-Editor, and to export it later. Thus the online version of the Archiv-Editor can be developed into a service not only for registered users, who use it for editing their project data, but also as a more general service for temporary users working with prosopographical data.

4. Conclusion

The discussed example of the Archiv-Editor and it’s transformation into a web-application is not a general answer to the need to bridge desktop and web applications. The program is developed on a specific architecture and its conclusions are only partially relevant to software using other frameworks than Eclipse RCP.

4.1. Extensibility and Cooperation

Like the Archiv-Editor many desktop software and Virtual Research Environments in the Digital Humanities are based on the Eclipse RCP framework, e.g. TextGridLab\textsuperscript{11}, Pacx\textsuperscript{12}, Edirom\textsuperscript{13}. Program components and plug-ins can be shared between these platforms, and it is therefore very helpful when DH infrastructure projects choose this architecture. Furthermore, projects with this architecture can not only share plug-ins for desktop installation, but for web versions as well (when they are compiled against a RAP target and deployed). Such an approach seems to be of high interest for many projects in this field. In Digital Humanities numerous web-services are already in use and some can be combined into pipelines. The above mentioned desktop programs already use certain functionality of different web-services, for example the Archiv-Editor uses three services (for the extraction of dates and place names from texts and other tasks). Thus the online version is a combination of web-services providing a very complex and customizable service. With the Eclipse RCP framework it is possible and relatively easy to use the already existing code (if it is RCP based) to create large and complex services and offer them as web-services with user interfaces.

4.2. Technical Conclusion

The RAP framework is far less powerful than the RCP framework for desktop application. Some important widgets are still missing, as well as a number of features of existing components. There are not many software projects that use RAP and the documentation of the framework is often very sparse. However, the RAP framework is stable and comprises many features and components from the

\textsuperscript{10}http://wiki.eclipse.org/Vex
\textsuperscript{11}http://www.textgrid.de
\textsuperscript{12}http://pacx.sourceforge.net
\textsuperscript{13}http://www.edirom.de
desktop widget toolkit (SWT), workarounds for missing components can be developed and separated into specific plug-ins. If an existing RCP application is transferred into a web-application, this framework offers a good and powerful option which is not only cheaper than the development of a separate web based GUI, but also allows further parallel development of both versions. This reduces future cost of porting features from one version to the other. For RCP projects that are about to design their plug-in architecture and could develop an interest in a future online version, it would be very useful to take the RAP compatibility of components into account at an early stage.

5. Bibliography


Workflow Engines in Digital Humanities

Thomas Zastrow, Emanuel Dima
Department of Linguistics, University of Tubingen
Wilhelmstr. 19, 72074 Tubingen, Germany
E-mail: firstname.lastname@uni-tuebingen.de

Abstract
Scientific workflow engines can be used to combine tools and resources from the user’s local computer with the ones offered via a network like the Internet. This results in advantages for the end users, who do not need to download, install and configure the tools and resources they need for their everyday work. In digital humanities, workflow engines can be embedded into existing eScience infrastructures. This paper describes three workflow engines (Kepler, Taverna and Weblicht) which can be used in a broad range of scientific research environments. They are compared to each other under aspects of flexibility, interoperability and end user experience.

Keywords: scientific workflow system, web service, service oriented architecture

1. Introduction
The Personal Computer is an essential tool for the day-to-day work of many researchers in the humanities. But even with permanent and high-speed Internet access, local computers are used mainly for working with locally installed resources and tools. This has the disadvantage of having to download, install and configure every necessary tool individually on the user’s local machine. Due to the fact that most of the tools that are of interest for humanity researchers are developed independently of each other, interoperability between these tools is not guaranteed. Different data formats and different text encodings make it difficult to exchange data between these tools. Locally installed tools can only make use of computing capabilities of the local computer, which are often not sufficient enough for complex tasks or for the processing of massive data sets.

The paradigm of Service Oriented Architectures (Perrey, 2003; Henrich et al., 2010) now allows the execution of tools and access to remote resources over the Internet in a convenient way. It is no longer necessary to download and install tools that are available as services in a SOA. Such tools can make use of the capabilities of high performance computing centers (HPC) and the SOA can assure the interoperability between the integrated tools by using compatible data formats and text encodings.

Today, not all tools and resources are integrated into a SOA. This makes it necessary to combine local tools and datasets with remotely available web services. Workflow engines (Cucin et al., 2008) are designed to break up the natural barrier between a researcher’s local computer and Service Oriented Architectures. Both resources and tools can be chained together in such a transparent way, that it makes little difference if they are locally installed on the user’s own computer or somewhere on the internet in a computing center (see Figure 1). A workflow engine provides a schematic representation of the data “flowing” through channels between processing gates, which can be either local or remote tools. With a graphical user interface, researchers can create and execute individual workflows within the workflow engine.

Let us consider the case of a humanity researcher who needs a list of lemmas and their frequency in a specific text. With a workflow engine, he can create a workflow that lemmatizes a text and applies statistical methods such as computing frequencies from the annotated data. This workflow can be then executed on the input text and will output the desired results. Any workflow is independent of the data it processes, so once a workflow is created, it can be reused for other input texts and will work just the same. This gives the researcher the power of creating his own custom tools by combining the functionality of existing ones. Even more, the researcher has the option of sharing the workflow he just created with other researchers, either directly by exporting it as a file or by uploading it in a public database of workflows, where it can be searched for and used by anyone.

A lot of different kinds of workflow engines are available (Yu et al., 2005). In this paper, we will concentrate on three of them which are often used in scientific environments and which offer different functionalities and user experiences: Kepler and Taverna, which have a

---

1 Most Service Oriented Architectures use web services for accessing tools and resources. The W3C defines web services as " [...] a software system designed to support interoperable machine-to-machine interaction over a network."
http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

2 This paper will not include workflow systems based on the well-accepted standard in business domain, BPEL. Originally designed for business workflows, in the last years BPEL has
generic character and which can be used in a broad range of scientific environments. In addition, WebLicht as a more specialized workflow engine for linguistic workflows is discussed in comparison to the more generic ones mentioned before. In the following, all three workflow engines are compared from the perspective of their usefulness in the scope of digital humanities.

2. Kepler

Kepler\(^3\) is a workflow management system derived from the Ptolemy II actor-oriented software framework\(^4\). The Kepler project was originated by UC Davis, UC Santa Barbara, and UC San Diego. It is open-source, freely available under the BSD license, Java based and maintained for all major software operating systems (Windows, OSX and Linux).

Kepler has native support for grid technologies and is used in the earth sciences domain and for phylogenetic analyses, modeling sensor networks, wildlife conservation analysis, environment monitoring, synthetic biology etc.\(^5\) It also provides direct access to EarthGrid, a collection of ecological, biodiversity and environmental data and analytic resources.

The Kepler system provides a graphical user interface (the designer, see Figure 2) that allows for designing workflows and provides feedback on their execution. Underneath the user interface, the run-time engine executes the workflows started from the designer or from the command line.

A workflow is built from basic components, called actors and one special component, the director. Just as in film production, the actors take the instructions from the director and then act based on their own script. Out of the box Kepler comes with a selection of the most used directors, but accessing other needed directors is also possible. Depending on the director that is used in a workflow, parallel execution of the workflow elements is possible.

By default the system includes hundreds of ready-to-use actors that can be easily customized. One of the available customization facilities is provided by parameters, which constitute settings for each actor. For example, the File Copy actor takes a source file path and a destination path and copies the source to the destination. This actor has parameters that specify whether the user wants to overwrite the destination file if it already exists, or just append data to it; or whether the copy should work on directories as well as on single files.

Actors also have one or more ports for data flow. The input ports transfer data from the system to the actor while the output ports transfer the processing results from the actor back to the system. These data ports are typed: each piece of data (token) must have a type and an error is generated if during runtime the actor receives a data token with the wrong type.

The designer allows the creation of illegal workflows, where an output port of one actor is connected to an input port of another actor although the types of the ports are not compatible. These errors are typically discovered during the testing phase of the workflow.

Semantic annotations are part of the metadata associated with every Kepler component. These annotations are ontology-based and facilitate the exploratory search for new components. Also, they allow for validation of the workflow from a semantic perspective.

The Kepler system encourages modularity: A complex workflow can be split into separate, simpler and possibly reusable sub-workflows (also called composite actors). All actors, sub-workflows and workflows can be saved as files and directly shared with colleagues or even uploaded to the Kepler’s Component Repository, a centralized server where all these components can be searched and reused.

The native workflow description language, Modeling Markup Language (MoML) is inherited from Ptolemy II. Kepler can execute any MoML file that contains a valid description, based and facilitate the exploratory search for reusable sub-workflows.

3. Taverna

Taverna is a workflow management system\(^6\) created by the myGrid team\(^7\). It is cross-platform (using the Java environment) and open source under the LGPL license. The Taverna community is primarily focused on the Life Sciences. The software has been used in biology, chemistry and medical imaging but also in astronomy, for text and data mining, digitization and document and image analysis. The Taverna homepage\(^8\) lists over 350 projects, most of which are using the standalone desktop client.

Taverna is also being used in some eHumanity projects like PANACEA\(^9\) or the Dutch part of CLARIN\(^10\).

The Taverna suite is centered on the Taverna Engine, the module that executes a given workflow. This module is used by the Taverna Workbench, the standalone desktop client application and also by the Taverna Server, which

---

\(^3\) https://kepler-project.org/
\(^4\) http://ptolemy.eecs.berkeley.edu/ptolemyII/
\(^5\) https://kepler-project.org/users/projects-using-kepler
\(^6\) http://www.taverna.org.uk/
\(^7\) http://www.mygrid.org.uk/
\(^8\) http://www.taverna.org.uk/introduction/taverna-in-use/
\(^9\) http://www.panacea-fr.eu/
\(^10\) http://www.clarin.nl/
allows the remote enactments of workflows. In addition, the same engine can be used for executing workflows from the command line. The Taverna model of execution is based on lambda calculus. Within the limits of this model, the engine is able to execute workflows or parts of workflows in parallel.

A Taverna workflow can integrate various kinds of resources and tools. These include:

- (Structured) data from the user’s local computer, for example spreadsheet data
- Any accessible web service, implemented in REST- or SOAP-style
- Access to already integrated and specialized web service repositories like for example Biomart or SOAPLab installations
- Other TCP/IP based server software can be accessed from within Taverna. For example, the R software for statistical computing and graphics can be used directly via the Rserve server software
- Local services are built-in functions of Taverna. Mostly, these are specialized functions with a small scope, like for example reading a text file
- Taverna makes a differentiation between domain services and shim services. Domain services perform a scientific function and are generally provided by third parties while shim services are only used to achieve interoperability between individual domain services, when the data types and formats are not compatible (for example, converters from one data format to another). Single services in Taverna can be fed with data via so called input ports. After a service was executed, an output port wraps the data coming back from the service.

While a Taverna workflow is being executed, the provenance data of each run can be recorded. After the workflow completion the intermediate data values can be inspected, which is a useful feature for debugging the workflows. The provenance data is by default kept in memory but can be exported in either OPM or JANUS format.

Nested workflows are also possible in Taverna. They behave just like a regular service, except that the inner workings of the nested workflow are exposed to the workflow designer.

Sharing workflows is made easy by projects like MyExperiment, which are collecting workflows and allow for researchers to submit and search for workflows in the domain they are interested in.

### 4. WebLicht

WebLicht, short for Web-based Linguistic Chaining Tool, is a service orchestration and execution environment for incremental automatic annotation of text corpora, built upon the principles of Service Oriented Architectures (Himrichs et al., 2010). In contrast to the general purpose workflow engines presented above, WebLicht was started as a specialized linguistic workflow system. The system was developed primarily in Java at the University of Tübingen, but the integrated web services are distributed over the whole CLARIN-D project.

The WebLicht ecosystem consists of a set of web services, each one associated with a corresponding piece of metadata, a set of decentralized repositories storing the metadata and a web application with a user-friendly graphical user interface. Beside this graphical user interface, WebLicht’s web services can be executed from the command line or being embedded into other (web-) applications.

By convention, the WebLicht web services follow a simple protocol: the input data is directly streamed to the web service which synchronously processes the data and returns the result with the corresponding mime type using the same network connection. WebLicht is independent of the data format which is used to send the data from one web service to the next one. De facto, most of the web services are using the Text Corpus Format (TCF) (Heid et al., 2010) which is also able to store the provenance data of the actually executed workflow.

Each web service is associated with a piece of metadata in the Common Metadata Infrastructure (CMDI) format (Broeder et al., 2011). Depending on the implementer of the web service, the metadata is stored in one of the nine CLARIN-D repositories and openly available to everyone. The repositories are periodically harvested and the metadata compiled by another web service, the chainer, which offers ways of forming chains of web services.

---

13 Biomart is a repository of biological databases and tools: http://www.biomart.org/
12 SOAPLab is a software for wrapping command line applications into web services (http://soaplab.sourceforge.net/soaplab2/)
11 http://www.r-project.org/
14 http://www.rforge.net/Rserve/
15 The MyExperiment project is located under http://myexperiment.org/ Here, ca. 2,400 workflows, mainly from natural sciences can be found, some of them using other workflow engines than Taverna. The Panacea project runs also a MyExperiment portal with linguistic motivated workflows: http://myexperiment.elda.org/home
The metadata contains the orchestration data, that describes the types of input the web service is expecting and the results of its processing. The metadata also contains the Persistent Identifier (ISO/DIS, 2009) of the web service. This persistent identifier allows for the creation of provenance data for each file that is processed in a WebLicht chain.

![Figure 4: The WebLicht web application](image)

An important part of WebLicht is the web application that allows the researcher to:
- Select the input data
- Create chains of tools for processing the data
- Analyze, visualize and locally store the results.
This is the only part of WebLicht visible to a novice user and consequently it was optimized for ease of use, simplicity and minimization of the user’s possibility for error.
WebLicht is secured by the CLARIN Identity Federation\textsuperscript{16}, so that only users from the academic community have access to it.

5. Interoperability of Workflow Engines

All three workflow engines are able to integrate external web services into their workflows. But web services can be implemented in different ways, depending on which protocol they are using, how they are described with metadata, etc.
The provenance data of the three engines are created in different formats (JANUS, OPM, CMDI). An interoperable standard that serves all use cases does not exist so far, but a conversion from one format to the other is straightforward.
Beside the interoperability of the workflow engines, another important aspect is the interoperability of the data format(s), used by the underlying web services themselves. If web services are requiring incompatible input- or output formats, chaining them together into a workflow makes no sense.

5.1. Integrating WebLicht Web Services into Taverna and Kepler

Both Kepler and Taverna are generic workflow engines that can integrate web services of various kinds. On the other hand, WebLicht is a complete Service Oriented Architecture where the web services are built in a modular and flexible way. This makes it possible to embed WebLicht’s web services into Kepler or Taverna:
- In Kepler, a Kepler actor is integrating the WebLicht web services. The actor has parameters that specify the web service to be called and its input and output data.
- In Taverna, WebLicht web services can be integrated in two ways: first, the SOAPGate web service in WebLicht allows to access individual web services as well as predefined chains of web services. Second, Taverna’s internal REST Service can integrate individual web services from WebLicht directly.

5.2. Integrating Web Services from other SOAs into WebLicht

When integrating web services from other SOAs into WebLicht, the following requirements have to be fulfilled:
- The web service has to be implemented in RESTStyle architecture
- The metadata of the web service has to be written in CMDI format and hosted by a CLARIN center
- A PID has to be assigned to the web service
To be integrated into WebLicht, it is not necessary to use the TCF format. However, using the TCF format would make the web service compatible with most of the other web services in WebLicht.

5.3. Concrete Usage Scenario

To evaluate the three workflow engines in a realistic usage scenario in the scope of digital humanities, we designed a typical linguistic workflow, adding widely used linguistic annotations to a plain text\textsuperscript{17}. The workflow integrates six well-known and approved web services from the WebLicht SOA. Using only WebLicht web services guarantees the interoperability of them in the scope of data formats and encodings\textsuperscript{18}.
The workflow is then implemented in all of the three workflow engines. Because they are taken directly from the WebLicht environment, no adaptation has to be done for using them from within WebLicht. However, for both Kepler and Taverna, small adaptors have to be written to integrate WebLicht web services (see chapter 5.1). Once written, these adaptors can be re-used to integrate all WebLicht web services and not only the six web services

\textsuperscript{16} http://www.clarin.eu/node/3227

\textsuperscript{17} The experiment was done as part of a broader evaluation and benchmarking of web services in the context of the EUDAT project (www.eudat.eu)

\textsuperscript{18} When integrating web services from other SOAs, converters could be needed to establish compatibility between data formats and encodings
mentioned here. The execution of this workflow with the same input text gave back identical results from all three workflow engines in the expected time. Parallel execution of several workflows at the same time was not evaluated, so that there was nearly no time difference in executing the same six web services from the three different workflow engines.

6. User Experience

An important distinction is that the user experience is different in Kepler and Taverna compared to WebLicht. The WebLicht user interface is centralized, being hosted as a web application that requires no installation and zero maintenance from the end user. When creating a WebLicht chain the user is guided at each step, the available list of tools is updated on the fly so that in all cases the created chain is well formed from a type system perspective. However, the two other workflow engines do not check the type correctness of the tool chain created by the user, so that the creation of an ill-defined workflow is always possible. Table 1 gives an overview of functionality and user experience of the three workflow engines.

The availability as a web application that needs no maintenance from the end user makes it possible to integrate WebLicht easily into academic courses or seminars. Because of their generic nature, Kepler and Taverna are more complicated to use and have a steeper learning curve than WebLicht. On the other hand, they are more flexible than WebLicht when it comes to the integration of different kinds of web services.

7. Conclusion

In this paper, we have described and compared three workflow engines that are often used in scientific use cases. Kepler and Taverna have a more generic character, while WebLicht is more specialized and designed to be used with linguistic tools and resources. But the modularity and flexibility of the web service concept makes it possible to exchange web services and use them from within different workflow engines. All three workflow engines are appropriate for solving typical tasks appearing in the scope of digital humanities, this being demonstrated in a concrete usage scenario. With the help of a workflow engine, web services which are offered over the Internet can easily be combined with local tools and resources into new workflows. This results in advantages for the end users, who do not need to download, install and configure the tools and resources they need for their day-to-day work. In a SOA, interoperability can be assured by using compatible data formats and encodings. Depending on the humanity researcher’s need or necessity, he can either opt for a more generic but more difficult to use workflow engine like Taverna or Kepler, or he can use a more specialized and intuitive to use workflow engine like WebLicht. This last use case includes also the usage of a workflow engine in academic teaching, where it is necessary to have an easy to use application with no required maintenance.

Although general interoperability between workflow engines is given on the level of web services, interoperability is an issue when it comes to chain web services with incompatible input or output formats. In addition, a widely accepted and implemented standard for provenance data would increase the interoperability of workflow engines.

8. References


R. Perrey. Service-oriented architecture, in: Proceedings