Available online at www.sciencedirect.com

ScienceDirect

Computer Speech & Language 45 (2017) 326–347

www.elsevier.com/locate/csl

Multilingual processing of speech via web services

Thomas Kisler[a], Uwe Reichel[b], Florian Schiel[a,*]

[a] *Institute of Phonetics and Speech Processing, Ludwig Maximilian University of Munich, Schellingstr. 3, 80799 Muenchen, Germany*
[b] *Magyar Tudományos Akadémia, Nyelvtudományi Intézet, 1394 Budapest, P.O. Box 360., Hungary*

Received 29 August 2016; Accepted 6 January 2017

Abstract

A new software paradigm 'Software as a Service' based on web services is proposed for multilingual linguistic tools and exemplified with the BAS CLARIN web services. Instead of traditional tool development and distribution the tool functionality is implemented on a highly available server that users or applications access via HTTP requests. As examples we describe in detail five multilingual web services for speech science operational since 2012 and discuss the benefits and drawbacks of the new paradigm as well as our experiences with user acceptance and implementation problems. The services include automatic segmentation of speech, grapheme-to-phoneme conversion, syllabification, speech synthesis, and optimal symbol sequence alignment.
© 2017 Elsevier Ltd. All rights reserved.

Keywords: Web service; Speech processing; Automatic segmentation; Grapheme-to-phoneme; Software as a service

1. Introduction

The advent of fast and ubiquitous internet connections over the last decade has resulted in a general paradigm shift in the use of software tools. Traditionally tools have been designed as stand-alone programs that required special attention (design, implementation, maintenance) for a (restricted) number of operation systems (OS) as used by the scientific community. This involved considerable overhead in development, as distinct versions of the same program code had to be implemented and maintained for each different OS. For their part, end users had to keep up with necessary OS updates and had to reinstall the software after major system upgrades. Work on both ends, in development and in usage, can be reduced by emerging web technologies and standards (Anthes, 2012), e.g. when a client application is written in a language that runs out-of-the-box on standard web browsers (e.g. in JavaScript) or when a web service uses commodity software to provide a standardized interface to a tool installed on a server. Furthermore, the user support (if there is any) requires significantly less effort, be it in the form of a help desk, a user forum, a collection of Frequently Asked Questions (FAQ) or the maintenance of user manuals. This is mostly due to the fact that the most frequent end user problem, the installation process on their local computer, becomes obsolete.

The same paradigm shift is underway for specialized software tools for speech science, with the difference that these particular tools are often available for free for researchers. While there are some widely-used speech tools

* Corresponding author.
E-mail address: kisler@phonetik.uni-muenchen.de (T. Kisler), uwe.reichel@nytud.mta.hu (U. Reichel), schiel@phonetik.uni-muenchen.de (F. Schiel).

http://dx.doi.org/10.1016/j.csl.2017.01.005

such as Praat (Boersma, 2001) that were developed as stand-alone systems, the number of purely web-based tools and services is growing, mainly within the large infrastructure initiatives ‘Common Language Resources and Technology Infrastructure’ (CLARIN, Hinrichs & Krauwer, 2014) and ‘Digital Research Infrastructure for the Arts and Humanities’ (DARIAH, Romary & Chambers, 2014), but also many individual systems, most of them subsumed under the label ‘Digital Humanities’ (for a good and up-to-date overview see the proceedings of the conferences organized by the Alliance of Digital Humanities Organizations, http://adho.org/).

In this paper, we describe a set of web services at the Bavarian Archive for Speech Signals (BAS) CLARIN centre in Munich that were developed for the multilingual processing of spoken language, i.e. speech signals. We also discuss and present web services that deal with symbolic data, where such data are associated to spoken content in some way (e.g. a phonological labelling, a text transcription of a recording, a syllabification of a phonetic symbol string, etc.). The paper is structured as follows: First, we define best practice in offering tool services to the speech science community. We propose a traditional server/client architecture with a strict distinction between a server-based back end implementation and a web interface that acts as an interactive front end to the back end services. Second, we present examples of public web services based on this architecture that have been implemented at the BAS CLARIN centre at the University of Munich[1]; for each service we describe the technique applied in the back end, the usage of the web service, evaluation data (where applicable) and an exemplary use case. Third, we demonstrate the possibility of combining basic web services into more complex processing units, and describe two examples implemented at the BAS. The last two sections are dedicated to a critical evaluation of the new paradigm as well as some of our experiences with the proposed architecture (including user statistics and feedback), and plans for the future.

2. Web services as fundamental processing units

2.1. Web services and the web service interface

We maintain a traditional client-server architecture which ensures highly decoupled system components. The web services encapsulate the functionality of tools that are installed on the server. In the following we call the server-side also the ‘back end’. Fig. 1 shows the back end components, the tool wrappers and their relation to each other. The web services exhibit a defined interface to the outside world and can therefore been seen as an Application Programming Interface (API), which is accessible by the clients in the form of *other tools, programs or web services* . The advantage in separating the storage and processing of data (back end) from the graphical user interface GUI (client) is that both can be maintained and developed independently. Indeed a complete makeover of the front end (as described in Kisler et al., 2012), did not necessitate any change to the back end. The functionality of each web

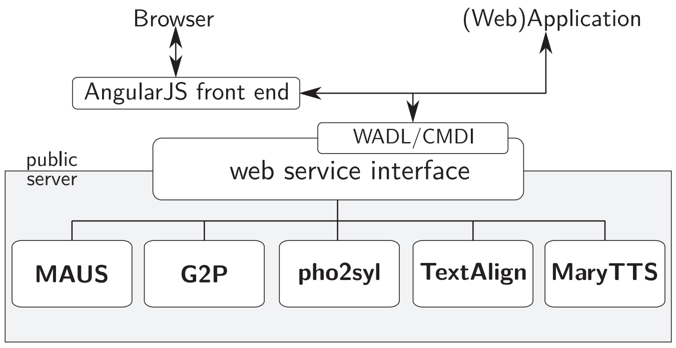

Fig. 1. Architectural overview of the BAS CLARIN web services: the five tools are implemented on a public back end server and presented via a standardized web service interface to the Internet. Users or applications can access the public specification (WADL/CMDI) of the interface to perform calls to the web services or use the web front end via a standard web browser.

[1] http://hdl.handle.net/11858/00-1779-0000-0028-421B-4.

service and the corresponding web service interface are documented using standardized public metadata (see Section 2.3 for details).

2.2. Implementation details of the back end

The following short paragraph provides some technical details of the current back end server implementation. These are merely presented here as one possible way to achieve the back end functionality as described so far, and will be very likely be replaced by more advanced techniques in the future. Nevertheless, these details may give the interested research software engineer some inspiring insights into the technical setup of the current BAS back end; the paragraph can be skipped by readers who are not interested in implementation details without any harm of understanding the following sections.

The back end uses an Apache HTTP server as proxy to redirect calls to an Apache Tomcat servlet container, without exposing the Tomcat's server address to the outside world. This is done mainly for maintenance reasons, so that the URL of the web services and interfaces can be kept fixed, even if the Tomcat location has to be changed. The services themselves are implemented in a RESTful way (REST stands for Representational State Transfer) in Java[2] using "Jersey", which is the reference implementation for the 'Java API for RESTful Web Services' (JAX-RS)[3]. From a process-oriented point-of-view, the services provided are best described as RESTful remote procedure calls (RPC). We use the HTTP methods GET and POST as envelopes for the RPCs and return the response in Extensible Markup Language (XML). The response format in XML allows effective communication of results, warnings and errors to the calling instance.

2.3. Usage of web services

The basic idea is that all programs that can create HTTP requests[4] can easily access the web services (see example below and in Section 3). This not only includes all modern programming languages, but also command line tools for at least three of the major operating systems used on Desktop computers and Laptops, namely Windows, MacOS and Linux. One example for such a tool is curl[5], an open source Windows and Linux command line tool. Using this tool to access the services is very similar to using services installed on a local machine, with the obvious difference that the computer requires an internet connection. This requirement comes with the benefit of being independent from the processing power of the client machine. To allow the user to easily create requests, all services are described in standardized metadata (in our case both, the Web Application Description Language (WADL)[6] and the Component Metadata (CMD) infrastructure). The WADL description is used for the syntactic description of the services and it contains the necessary envelope (POST or GET) to make a successful call. The CMD instance used for the BAS web services[7] is structured following the CLARIN core data model for web services (Windhouwer et al., 2012) and provides the semantic description of the web services and provides a human readable explanation of all parameters. The CMD allows the user to understand the parameters and form queries for the web interfaces. The two metadata descriptions, WADL and CMD instance, allow for automatic service invocation by other applications and the automatic generation of documentation. For instance, the help page,[8] providing information about all available web services including their respective parameters and example calls, is generated automatically from the two. Finally, our own back end server utilizes these metadata descriptions to perform checks on the passed parameters automatically and to provide the user with meaningful error messages that contain alternatives to possibly incorrectly formed arguments.

As a typical example of the integration of web services into other software, the annotation software ELAN (Wittenburg et al., 2006) embeds the web service 'runMausBasic' through their plug-in mechanism: the application passes the user provided signal and annotation file, and calls the web service at the back end server to obtain the

2 https://www.java.com/.
3 https://jersey.java.net/.
4 as specified in RFC 2616 https://www.w3.org/Protocols/rfc2616/rfc2616.html.
5 https://curl.haxx.se/.
6 http://clarin.phonetik.uni-muenchen.de/BASWebServices/services/application.wadl.
7 https://clarin.phonetik.uni-muenchen.de/BASRepository/WebServices/BAS_Webservices.cmdi.xml.
8 http://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help.

segmentation and labelling of the currently annotated signal. The result is immediately integrated back into the application data and therefore saves the user the need to perform those tasks manually (Kisler et al., 2012).

Human users making use of our web services directly have to form a query using a tool like curl. For example, a curl call to the current version of the web services that generates a segmentation and labelling based on a signal file and the orthographic transcription would be as follows:

```
curl -v -X POST
-H 'content-type: multipart/form-data' \
-F LANGUAGE=deu-DE \
-F SIGNAL=@filename.wav \
-F TEXT=@filename.txt \
'https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runMAUSBasic'
```

The web service is called using a POST envelope (-X POST), specifying the structure of the call to be a "multipart/form-data" (-H 'content-type: [. . .]), followed by some service-specific parameters and the URL of the web service. The parameters passed with the "-F" option are a signal file denoted by 'WAV filename' (SIGNAL=@[. . .]), a text file denoted by 'Text Filename' (TEXT=@[. . .]) and the instruction to use German (LANGUAGE=deu-DE) as the processing language. The web service returns a standardized XML file containing the URL to the result file, a success/failure flag, and an error/warning message (if applicable), e.g.:

```
<WebServiceResponseLink>
  <success>true</success>
  <downloadLink>
  https://clarin.phonetik.uni-muenchen.de:443/BASWebServices/data/....emuR
  </downloadLink>
  <output></output>
  <warnings></warnings>
</WebServiceResponseLink>
```

Even in the case of successful execution, the service might still issue warnings in the "warnings" element. Warnings indicate smaller problems or incompatibility of options, which do not lead to a termination of the execution. It is necessary to communicate these warnings because they might indicate that the result is not what the user would expect and contain useful information on what caused the warning. In the case where the service execution was unsuccessful, the field "output" should yield further information regarding the cause of the error.

2.4. Web interfaces

Two tasks occurred frequently: extending the web interface to incorporate new services and changing/adding parameters that control the back end behaviour. The first task was simplified by modularising the web interfaces which consist of the same basic building blocks (e.g. file upload, display of file content, listing results, etc.). One popular framework that supports the definition of such building blocks is AngularJS[9] which follows a modified Modal View Controller (MVC) pattern. In contrast to its name, we used TypeScript, a typed version of JavaScript, to work with the AngularJS framework. In AngularJS, building blocks are called directives and are implemented as custom HTML tags. After they have been defined, directives can be used from within the HTML markup and can be put together to combine the functionality of different smaller parts.

The second frequently occurring task involved changes to the web interface parameters. The parameters that control the behaviour of certain services, especially the language parameter, are subject to frequent changes e.g. when the user requests new options that control the underlying functionality or when wrapped tools are extended to new languages. This task was simplified by extracting the relevant information from the CMDI metadata description in which all parameters are specified (so that the user or other callers can form correct calls to the services). This

[9] https://www.angularjs.org.

procedure means that changes to the parameters in the metadata description are automatically visible in the interface. No adaption of the program code is necessary and people without programming skills can perform the task of changing and extending the web service parameters.

2.5. Processing infrastructure

Since the web services have the advantage that processing is not done on a local machine, they can also be used on slower clients. In the case of laptops, tablets and other battery-driven devices, web services do not affect battery life, which would be the case if the CPU-intensive processing were to be done locally. The disadvantage is that there has to be an internet connection available during the entire processing time. Most researchers have access to at least moderately fast internet connections and soon this will hopefully also be the case for field researchers. Currently, the back end server consists of two Intel®[10] Xeon (X5650) processors. These allow for up to 24 virtual tasks to run simultaneously (by providing 12 real cores via Intel® Hyper Threading). The web interfaces that allow for multiple file uploads use this infrastructure and process multiple files at the same time (for one user). In the case where multiple users process multiple files at the same time, a queuing mechanism ensures that users have equal access to processing resources and limits the overall usable resources to avoid overloading the server. The queuing mechanism works in a Round Robin fashion and is depicted in Fig. 2. For each user U_n there is a unique queue Q_n in which the user's tasks are placed. Suppose, for example, that there are already two users U_1 and U_2, with their respective queues Q_1 and Q_2. A new user U_3 arrives and is assigned queue Q_3. If the last task was drawn from Q_2, the next task will be drawn from Q_3. The next time a task is finished, the new task will be drawn from Q_1 and so forth. This ensures that processing time is allocated to users who arrive later, even if one user already uses all available processing resources by trying to process more tasks than being processable simultaneously. Because it is difficult to estimate the processing time for any one task in advance, one shortcoming is that this mechanism does not protect against very long single tasks that block processing resources for a long time.

3. The multilingual BAS CLARIN services

In this section, we describe a number of web services provided by the BAS CLARIN centre. For each service we outline the functionality of the encapsulated tool, a typical use case employing the current web interface, details about usage of the web service via a REST3 call to the back end, and the results of a validation of the service (where

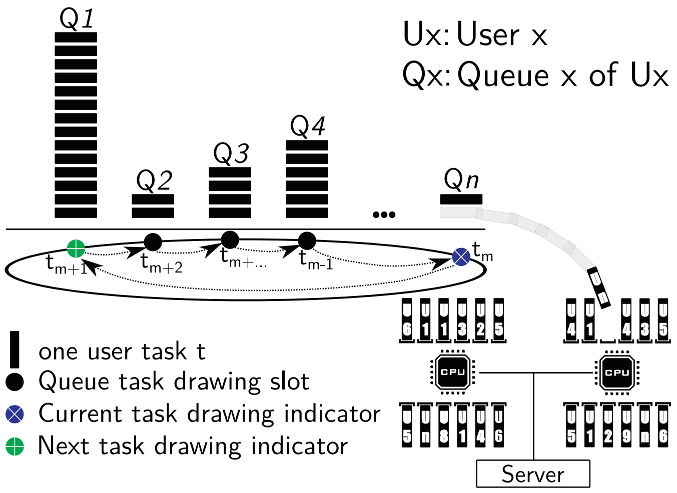

Fig. 2. Round Robin styled queuing mechanism of the back end: processing resources are fairly distributed to user queues to prevent one user with a very large queue from blocking other users (see text for details).

[10] www.intel.com.

applicable). All web services described here are accessible via an interactive web interface.[11] However, the design and structure of the web interface are subject to ongoing improvements and there may be discrepancies between the use cases described in this paper and newer versions of the web interface. Since users often process more than a single speech recording or annotation file, most services are able to process large groups of files in batch mode; the user simply drags the files from his local file system onto a drop area within the web interface. Most of the services are language-specific; since regional variants of the same language can deviate considerably, we use a more fine-grained descriptor based on the RFC5646 standard; this allows us to encode languages dependent on standardized country and region codes (see table in Appendix A; e.g. ‘eng-US’ for American English, ‘gsw-CH-SG’ for Swiss German (’gsw’) spoken in Switzerland (’CH’) in the region of St. Gallen (’SG’)).

3.1. Munich automatic segmentation (MAUS)

Automatic segmentation and labelling into phonetic units, sometimes in its simpler form also referred to as ‘forced alignment’, is an important prerequisite needed for several tasks, such as linguistic/phonetic analysis, media indexing and training of automatic speech recognition/speech synthesis (Wester et al., 1998; Pearson et al., 2000; Schmid, 2002; Wang et al., 2004; Almpanidis & Kotropoulos, 2008; Jarifi et al., 2008; Jiahong & Liberman, 2008; Gorman et al., 2011; Bigi 2012; Lubbers & Torreira, 2016; McAuliffe et al., 2016). The service described here is the re-implementation of the Munich AUtomatic Segmentation system MAUS as described in (Schiel 1999, 2004, 2015). Based on a given canonical phonological pronunciation, MAUS delivers the most likely phonetic labelling and segmentation of the corresponding speech signal. Currently, this service supports 23 languages or language variants (see language table for all services in Appendix A).

3.1.1. The technique behind the service

What sets MAUS apart from other forced aligners is its two-step modelling approach: prediction of pronunciation and signal alignment (see Fig. 3). In the first step, MAUS calculates a probabilistic model of all possible pronunciation variants for a given canonical pronunciation. This is achieved by applying statistically weighted re-write rules to a string of phonological symbols. The language-specific set of re-write rules is learned automatically from a large transcribed speech corpus. The pronunciation variants, together with their conditional probabilities are then transformed into a Markov process, in which the nodes represent phonetic segments and the arcs between them represent transition probabilities (see an example in Fig. 4). Thus each path through the Markov model represents one possible pronunciation variant of the canonical pronunciation given the language and the corresponding set of re-write rules. It is not a trivial task to summarize results of multiple re-write rules which can overlap in context or even in target/replacement strings into a single consistent statistical Markov model (see Schiel, 2015 for a detailed outline of the basic principle).

In the second step, this Markov model is passed together with the (pre-processed) speech signal to a Viterbi decoder (Young et al., 1996) which calculates the most likely path through the model, and - by means of backtracking this path - the most likely alignment of nodes to segments in the signal. The pre-processing of the speech signal is a standard feature set of 12 MFCCs + Energy and their first and second time derivative, as frequently used in automatic speech recognition (Young et al., 1996). The segmental accuracy is constrained by the step size to a range of about 10 ms. Since the required processing power increases exponentially with the complexity of the Markov model and is closely related to signal length (Pörner, 2016: 53), it is advisable to process only signals below a duration of around 10 min.

3.1.3. A use case

A user wants to test the hypothesis that in certain unstressed syllables of a German dialect the phone schwa is often not realized. The spoken corpus consists of 500 recordings of the dialect plus orthographic transcriptions to test this hypothesis.

The user prepares 500 BAS Partitur Format (BPF) files[12] with the same base name as the sound files containing the canonical pronunciation of each recording (using for instance a pronunciation dictionary or the service G2P

[11] http://hdl.handle.net/11858/00-1779-0000-0028-421B-4.

[12] http://www.bas.uni-muenchen.de/forschung/Bas/BasFormatseng.html#Partitur.

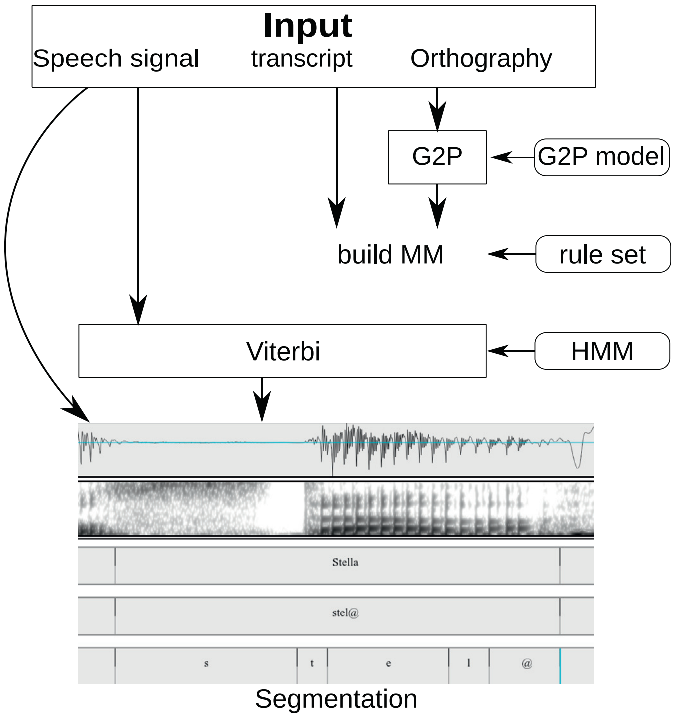

Fig. 3. Work flow of the Munich AUtomatic Segmentation system (MAUS): based on a phonetic transcript (or orthographic transcript translated by a grapheme-to-phoneme system), a language-specific Markov Model (MM) is created and then decoded (Viterbi) on the speech signal; back-tracking the best path through the MM yields a phonetic segmentation and labelling.

described in the next section). The user then selects the service 'WebMAUS Multiple' from the web interface, drags all 500 sound and 500 BPF files to the drop area, presses the upload button and selects the following options:

Language = German
Output Format = emuDB
Output Symbols = IPA

and presses the run button at the bottom of the interface.

The back end processes all file pairs and logs the progress in the status window of the web interface. If everything works well (status window is green), the web interface will display links to the calculated annotation files and also a link to a ZIP package containing the complete Emu Database (Winkelmann, 2015).

The user tests the hypothesis by loading the resulting Emu Database in the R programming environment using the 'emuR' library (Winkelmann, 2015) and queries a regular expression with and without the schwa symbol on the phonetic level of the database. A simple count of results can confirm or reject the hypothesis.

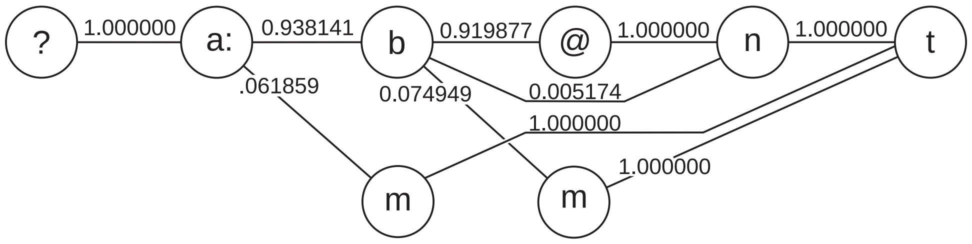

Fig. 4. Example of an a priori pronunciation Markov Model of MAUS for the German word 'Abend' ('evening'); nodes represent phonetic symbols (encoded in SAM-PA), arcs represent transition probabilities; each path through the model represents one possible pronunciation together with the accumulated a priori probability.

Why does this work? The German language is fully supported within the MAUS system. This means that MAUS can automatically determine from the speech signal whether the schwa was realized or not.

3.1.3. The web service runMAUS

The same result can also be obtained by calling the web service ‘runMAUS’ from within a simple script looping over all 500 file pairs using a curl command as follows:

```
curl -v -X POST -H ‘content-type: multipart/form-data’ \
-F LANGUAGE=deu-DE \
-F OUTSYMBOL=ipa \
-F SIGNAL=@<WAV file name> \
-F OUTFORMAT=emuR \
-F BPF=@<BPF file name> \
‘https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runMAUS’
```

where ‘WAV filename’ refers to the name of a recording and ‘BPF file name’ to the name of the corresponding BPF file, both stored on the user’s local computer.

For each call, the web service will return the standardized XML file with an URL to the result file as described in Section 2.

3.1.4. Validation

Validating automatic segmentation services requires a manually verified segmentation and labelling for each supported language as the ground truth (e.g. Räsänen et al., 2009). Since this is not feasible for all languages supported by MAUS, to date only two languages have been validated empirically: German (Kipp et al., 1997) and Scots English.

We distinguish two quality measures for automatic segmentation: the match of the phonetic transcript to the transcript of the ground truth compared to the average match of human experts (LabelMatch), and the average deviation of segment boundaries in the MAUS output compared to average deviations between human labellers in the ground truth (BoundaryMatch). The latter is expressed by counts of segmental differences larger than an arbitrarily set threshold (here: 20 ms) in corresponding segments.

For spontaneous spoken German, the ratio of MAUS-to-ground-truth agreement and the inter-labeller agreement of three human expert labellers (LabelMatch) was estimated at 97%, based on a sub-portion of the German Verbmobil corpus (approximately 5000 segments). Thus MAUS reaches about 97% of the average labelling performance of human experts.

Regarding segmental accuracy (BoundaryMatch), the rate of boundaries with less than 20 ms deviation between three human labellers was determined (93%) and compared to the same rate when measuring the deviation between MAUS and human labellers (84%). Thus, the segmental performance in relation to that of human labellers is about 90% (Kipp et al., 1997).

The MAUS performance on Scots English has been validated by the CLARIN F-AG 6 group in 2015 (internal project report[13]). In this case, the evaluation was not done to test the performance of the segmentation and labelling of an existing language model, but rather to determine how well an existing language-specific model works on a new variety of that language without its own model: the Standard Southern British model was used to segment and label Scots English (i.e., there was a data − model mismatch) and the accuracy was evaluated. For both the transcription and segmental accuracy metrics, MAUS had an error rate twice that of human experts. This poor performance illustrates the importance of adapting MAUS to language variants wherever possible.

3.2. Automatic text to phoneme conversion (G2P)

This service allows textual input (connected text or lists) to be transformed into the most-likely standard pronunciation combined with a number of (optional) annotations. Since there is not a one-to-one mapping between

[13] http://www.clarin-d.de/de/konferenz-abstracts/403-reichhaltige-phonetische-annotation-ice-scotland-corpus (German abstract).

grapheme and phoneme representation, this is not a trivial task for most languages. G2P converters are needed for speech synthesis, for aiding manual and automatic transcription of spoken text and for the generation of pronunciation dictionaries based on text collections (to name but a few possible uses). Currently the G2P service supports 24 languages or language variants (see language table for all services in Appendix A).

3.2.1. The technique behind the service

Based on the taxonomy proposed by Bisani & Ney (2008) for distinguishing between global probabilistic, local, and analogy-based methods, our approach can be characterized as local classification, that is, the classifier locally maps a letter within its current environment to the corresponding target phoneme. Depending on the availability of a pronunciation dictionary for training, the G2P converter is either realized as a C4.5 decision tree chain (Quinlan, 1993)[14] or as an expert-crafted mapping table.

Each decision tree is trained on a grapheme window and a phoneme history of certain lengths. Following van den Bosch & Daelemans (1993), we start with the most specific (largest) context, and the feature windows are shortened successively until the output of a tree in the chain is a phoneme (Reichel & Kisler, 2014).

For some languages, further trees were trained on an extended feature set including the part of speech (POS) label of the word and morphemic information (cf. Fig. 5). The underlying part of speech tagging, that like the TnT tagger (Brandt, 2000) employs suffix information, and the morphological analyses are explained in detail in (Reichel, 2012a). It has been shown by Reichel & Schiel (2005) that performance can be improved by incorporating these additional features.

In the application these trained classifiers are used for all words whose pronunciation cannot be derived by dictionary lookup.

For languages with no available pronunciation dictionary (e.g. Georgian) or with transparent grapheme-phoneme mappings (e.g. Spanish), tables were created by experts defining context-sensitive G2P mappings.

The G2P conversion is further extendable by syllabification (see Section 3.3), alignment (see Section 3.5) and word stress assignment. Word stress is assigned by Bayes classification within a learning-by-analogy framework. This approach is motivated by phonetic pseudo word production studies (Dohmas et al., 2014) in which subjects take over stress patterns of "phonetically similar" words of their language. Analogy-based word stress assignment has been implemented by Daelemans et al. (1994) assigning to new words the word stress of those stored entries that have the most similar syllable pattern. In addition to their approach, we combine different degrees of abstraction from the original transcription to an integrating model. In the training phase, the transcriptions in the lexicon are mapped onto several more abstract patterns (e.g. on a lower abstraction level each consonant is mapped to C, and on a higher level all C sequences are reduced to a single C). The probability distribution of word stress positions is then estimated for these patterns. In the application phase, an incoming word is mapped onto the same abstract representations as in the training phase. A Bayes classifier predicts the most probable stress position based on the stress distri-

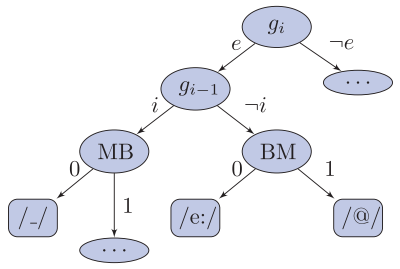

Fig. 5. Decision tree snippet for German G2P trained on an extended feature set accounting for morpheme boundaries MB and bound vs. free morphemes BM. An example path through the nodes g_i, g_{i-1}, MC and /@/ is to be read the following way: map letter ‘e’ to phoneme Schwa /@/, if it is not preceded by letter ‘i’ and occurs in a bound morpheme. To give two examples: The letter gi=‘e’ in the German word ‘rannte’ (‘ran’, simple past of ‘to run’) is preceeded by gi-1 =‘t’, thus not ‘i’, and occurs in an inflectional ending for temporal past marking which is a bound morpheme. In this context it is mapped to a Schwa. In ‘bieten’ (to offer), it follows gi-1 =‘i’ without an intervening morpheme boundary (MB= 0). In this context it serves as a lengthening marker for ‘i’ without being mapped on a phoneme itself which is indicated by the empty phoneme /_/.

[14] http://www.rulequest.com/Personal/.

butions for each of the abstractions weighted by the mutual information between the corresponding operation and word stress. Further details on this procedure can be found in (Reichel & Kisler, 2014).

For German and English, the stress learning-by-analogy is preceded by a compound decomposition step based on the morphological analyses (described in Reichel 2012a). From this decomposition, a metrical tree is induced based on the relative coherence of neighbouring compound parts (Reichel 2012b). Coherence is measured in terms of likelihood ratio. Within these trees, principles of metrical phonology (Liberman & Prince, 1997) guide the way to the stressed compound part, which is then further processed as described in the preceding paragraph.

For all languages, a basic initial text normalization step transforms digit sequences to cardinal numbers. For German and English, an extended text normalization module identifies and expands 22 non-standard word types (this is an extension of the taxonomy proposed by Sproat et al. (2001), for ordinal numbers, acronyms, date expressions, URLs, etc.).

3.2.2. A use case

A user wants to transform a German text into a dictionary comprising for each item the canonical transcription in IPA, syllabification, word stress, the POS label, and a morphological segmentation. In addition, each item should be normalized to include the expansion of number expressions, abbreviations, etc. The user uploads the input text via the web interface into the service ‘G2P’ and selects the following options:

Language = German (DE)
Input format = txt
Output Symbol = ipa
Feature set = extended
Output format = extlex
Word stress = yes
Syllabification = yes
Text normalization = yes

The result file is a semicolon-separated table of sorted lexical entries, one per row. The columns contain: the word, its transcription (here encoded in IPA, including lexical stress markers and syllable boundaries), its POS label (STTS[15] for German, Penn tagset[16] for English), and the morphological segmentation with the corresponding morpheme classes.

To give an example: The input text “Am 28.4. werden wir ihren Hochzeitstag feiern” (“on April 28th we will celebrate their wedding day”) will be converted to the following table:

```
achtundzwanzigsten-vierten;ʔ ˈ a x . t ʊ n t . t s v a n . t s ɪ ç s . t ə n . f iːɐ . t ə n;NN;acht und zwan zig st en vier t
en;CARD KON CARD SFX KOMP INFL CARD ORD INFL
Am;ʔ ˈ a m;APPRART;am;APPRART
feiern;f ˈ aɪ . ɐ n;VVINF;feier n;NN INFL
Hochzeitstag;h ˈ ɔ x . t s aɪ t s . t aː k;NN;hoch zeit s tag;ADJ NN FG NN
ihren;ʔ ˈ iː . r ə n;PPOSAT;ihr en;PPOS INFL
werden;v ˈ eːɐ . d ə n;VAFIN;werd en;V INFL
wir;v ˈ iːɐ;PPER;wir;PPER
```

3.2.3. The web service runG2P

The result described above can also be obtained by calling the web service ‘runG2P’ from within an application or from the command line using curl as follows:

```
curl -v -X POST -H ‘content-type: multipart/form-data’ \
   -F i=@<TXT file name> \
```

[15] http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/TagSets/stts-table.html.
[16] https://www.cis.upenn.edu/~treebank/.

```
-F lng=deu-DE \
-F iform=txt \
-F oform=extlex \
-F featset=extended \
-F syl=yes \
-F stress=yes \
-F nrm=yes \
-F outsym=ipa \
'https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runG2P'
```

The web service will return the standardized XML file with a URL to the result file, a success/failure flag, and an error/warning message (as described in Section 2).

3.2.4. *Validation*

An exhaustive validation of results for all supported languages would include all additional submodules from text normalization, part of speech tagging, morphological analysis, syllabification and word stress assignment and is beyond the scope of the current presentation. We thus restrict this description to a sample of languages and modules. In (Reichel & Kisler, 2014), a comparative G2P 10-fold cross-validation on six languages yielded mean transcription word error rates of 8% for German, 14% for British English, 1% for Hungarian, 3% for Italian, 4% for Dutch, and 1% for Polish. These results are to be seen as lower performance boundaries related to out-of-vocabulary cases, since in application the conversion is preceded by a lexicon lookup. For POS tagging of German, Reichel & Shigemori (2008) report an accuracy of 96.4% on held-out data. For the morphological analysis of German, Reichel & Schiel (2005) report both a segmentation and classification accuracy of 91.6% (given an average number of 2.67 morphemes per word).

3.3. *Automatic syllabification (Pho2Syl)*

The 'Pho2Syl' service allows for the syllabification of canonical as well as spontaneous speech transcriptions and can either be synchronized to word boundaries or neglect them when applied to connected speech. It is currently available for the same 24 languages supported by the G2P web service (see language table for all services in Appendix A).

3.3.1. *The technique behind the service*

As with G2P also Pho2Syl is based on local classification in terms of tree classifiers following Pearson et al. (2000). For languages where sufficient training data are available, the syllabification of a phoneme sequence is carried out by C4.5 decision tree classifiers that decide for a phoneme window of a certain length whether or not a syllable boundary is to be placed in the mid of the window (Reichel, 2012a). For languages for which no training data are available, boundaries are placed in front of each sonority minimum and their locations are subsequently adjusted if the language-specific syllable phonotactics are violated (see e.g. Vennemann (1988) for general considerations on sonority and phonotactics).

3.3.2. *A use case*

The user wants to add a syllable tier to a BPF file of British English data that already contains a MAU tier with the phone segments (e.g. generated by WebMAUS, see Section 3.1). The sampling rate of the corresponding speech recording is 44,100 Hz. The user wants to account for the finding that syllable and word boundaries are not necessarily synchronous in spontaneous speech.

Via the web interface, the user uploads the input BPF file and selects the following options:

Language=English (GB)
Tier name=MAU
Word synchronous=no
Output format=bpf
Sample rate= 44,100

The result will be again a BPF file with an additional MAS tier of the same type as the MAU tier, i.e. including time and (potentially multiple) word reference.

An example for a one-to-many assignment of syllables to words in spontaneous speech is the syllabification of German “. . . Termin am . . .” (“. . . date at . . .”). The corresponding MAU input tier section is given by:

```
MAU: 141280 1119 29 t
MAU: 142400 479 29 E
MAU: 142880 479 29 6
MAU: 143360 959 29 m
MAU: 144320 1119 29 i:
MAU: 145440 799 29 n
MAU: 146240 799 30 a
MAU: 147040 639 30 m
```

It contains one entry per sound segment specifying its onset sample index, its duration in samples, the word reference, and its transcription (Schiel et al., 1998). Due to the omission of the initial glottal stop in word 30 ” am”, Pho2Syl suggests a word boundary crossing spontaneous speech syllabification as follows:

```
MAS: 141280 2079 29 tE6
MAS: 143360 2079 29 mi:
MAS: 145440 2239 29,30 nam
```

3.3.3. *The web service runPho2Syl*

From the command line, the user obtains the same output using the curl command as follows:

```
curl -v -X POST -H ‘content-type: multipart/form-data’ \
  -F i=@<BPF file name> \
  -F lng=eng-GB \
  -F tier=MAU \
  -F wsync=no \
  -F oform=bpf \
  -F rate= 44100 \
  ‘https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runPho2Syl’
```

3.3.4. *Validation*

For the same reasons that apply to ‘G2P’, validation is possible only for a selection of languages. In a comparative 10-fold cross validation, Reichel & Kisler (2014) report syllabification word error rates of 1.5% for English, 0.1% for Italian, and 3% for Dutch.

3.4. *Speech synthesis (MaryTTS)*

The speech synthesis service of BAS CLARIN can be used to generate German spoken output based on orthographic input. The user may choose between 4 different standard voices (2 female + 2 male). The technique for creation and application of the synthetic voices are based on the MARY text-to-speech system (see e.g. Schroeder et al., 2011).

3.4.1. *The technique behind the service*

The MARY text-to-speech (MaryTTS) system is a server-client based synthesis system developed by Schroeder and colleagues (Schroeder et al., 2011). We used this open source system to train four different synthetic voices based on the German BITS-US synthesis speech corpus (Ellbogen et al., 2004) in two technical variants: a standard unit selection TTS and a HMM based TTS, resulting in eight different voices.

For a detailed and up-to-date description of MaryTTS please refer to the corresponding github documentation.[17] Several standard techniques to modify the synthesized speech, e.g. 'vocal tract scaling', 'fundamental frequency scaling', 'chorus effect' etc., are available.

3.4.2. A use case

For a perception experiment, a user requires spoken stimuli that are uttered in a reproducible manner and contain the same words in different contexts.

The user selects the page of the 'MaryTTS' service in the web interface, enters the text „Hallo, Welt“ (“hello world”) in the text field, selects a suitable male voice in the pull down menu below (e.g. 'bits1unitselautolabel'), and presses the button 'SPEAK'.

The service plays the synthesized speech. The user downloads the resulting audio file to her local computer by right-clicking on the link 'Save audio file'.

3.4.3. The web services runTTS and runTTSFile

The same result can be achieved by the following curl command issued from the command line:

```
curl -v -X POST -H 'content-type: application/x-www-form-urlencoded' \
   'https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runTTS?\
   AUDIO=WAVE_FILE&INPUT_TYPE=TEXT&\
   INPUT_TEXT=Hallo+Welt&VOICE=bits1unitselautolabel&\
   OUTPUT_TYPE=AUDIO'
```

As an alternative, we also offer a web service 'runTTSFile' which allows a text file to be uploaded as input instead of encoding the text in the URL:

```
curl -v -X POST -H 'content-type: multipart/form-data' \
   -F AUDIO=WAVE_FILE -F INPUT_TYPE=TEXT \
   -F INPUT_TEXT=@<TXT file name> -F VOICE=bits1unitselautolabel \
   -F OUTPUT_TYPE=AUDIO \
   'https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runTTSFile'
```

This service is for instance of value in creating a large set of spoken stimuli from a list of sentences.

3.5. Text alignment (TextAlign)

The 'TextAlign' service can be applied to any kind of symbolic data, e.g. letters, phonemes, words, or prosodic labels. It not only supports alignment of sequences from the same symbol inventory as in phoneme-phoneme alignment, but also alignment across different inventories (e.g. grapheme-phoneme) due to a flexible choice and training of empirically based cost functions. The service is not bound to any language. It not only returns the alignment result but also cost functions for future usage.

3.5.1. The technique behind the service

The aligner considers the alignment of two symbolic sequences v and w as a task to transform v into w by a minimum sum of edit costs, which is known as the Levenshtein distance (Levenshtein, 1966). This minimization task is solved algorithmically by means of dynamic programming (Wagner−Fisher algorithm; Wagner & Fischer, 1974) using the standard basic edit operations substitution, insertion, and deletion. The aligner supports several types of edit costs. If v and w stem from the same common vocabulary (as is the case with e.g. phoneme-phoneme alignment within a language) a naive cost function could be employed punishing all

[17] https://github.com/marytts/marytts.

operations other than zero substitutions by 1. A more flexible approach for any *v*-*w*-pairing is to define edit costs in terms of conditional probabilities reflecting symbol co-occurrences. These probabilities are estimated as described in (Reichel, 2012a) and include the probabilistic estimation of insertions and deletion cost. This approach allows for a uniform handling of all edit operations instead of using heuristics for deletions and insertions (see Kondrak, 2002 for an overview over such heuristics).

3.5.2. A use case

The user has a table of canonical and spontaneous speech transcription pairings (e.g. generated by runG2P and WebMAUS, respectively). The task is to align the transcription pairs and to train a cost function that can be subsequently used to calculate distances between canonical and spontaneous speech realizations in data sets not large enough to train their own cost function.

Via the web interface, the user uploads the table file and selects the following option:

```
Cost function=intrinsic
```

The service delivers a zip archive with two files, one containing the aligned transcription pairs, the other the optimized cost function calculated on the input data.

Subsequently, to align and determine the edit distance in new transcription pair data based on the optimized cost function, the user uploads the corresponding table file and the cost function file obtained in the previous step. The user then chooses the following options:

```
Cost function=import
Display costs=yes
```

The service now returns a table with the alignment result, including the edit distance for each transcription pair.

3.5.3. The web service runTextAlign

The user may obtain the same output from the command line using curl for the first step as follows:

```
curl -v -X POST -H ‘content-type: multipart/form-data’ \
-F i=@ali_in.csv \
-F cost=intrinsic \
‘https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runTextAlign’
```

The zip archive to be downloaded contains the cost function file ‘ali_in.19779.1467746108.cst.csv’ which can be used in future applications as follows:

```
curl -v -X POST -H ‘content-type: multipart/form-data’ \
-F i=@ali_in.csv \
-F cost=import \
-F costfile=@ali_in.19779.1467746108.cst.csv \
-F displc=yes \
‘https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runTextAlign’
```

3.5.4. Validation

Due to the large flexibility of this service, we restrict the validation report to two results reported by Reichel (2012b). The evaluation of 450 German and English word types with differing canonical and spontaneous transcriptions yielded a word error rate of 10.64% for grapheme-to-phoneme and of 1.31% for phoneme-to-phoneme alignment.

4. Combined services

One advantage of web services is the possibility of easily combining services into more complex processing constructs or processing chains. For example, the Weblicht system developed by the University of Tübingen, Germany[18] constructs chains of text-processing modules by chaining calls to distributed web services. The Weblicht chainer automatically checks the input/output formats of each module in the chain to ensure a syntactically and semantically consistent processing pipeline (Hinrichs et al., 2010).

At BAS, we utilize the modular approach of web services to provide some popular mini chains (hereafter 'combined services'), where the output of a first web service is fed back as input to a second web service. We present two examples: automatic segmentation based on orthographic input and pre-processing of so-called chunk segmentations. For the sake of brevity, we only present the calls to the web services, but the corresponding web interfaces are also available.

4.1. Automatic segmentation based on orthography (WebMAUS basic)

Users without linguistic expertise often lack the understanding and skills to produce the canonical pronunciation transcript (encoded in SAM-PA) that is required as input for the MAUS segmentation tool. Although this input can be produced from an orthographic transcript using G2P, this would require two interactions via the web interface, including two uploads of files and selection of the appropriate options (which can be a source of error, when non-matching options are selected within the two services). We therefore offer a combined web service called 'runMAUSBasic'[19] to allow the user to run both services in a consistent order.

As the technique and usage of the component parts of this combined service have already been described in the previous sections, an example of a REST call to this mini chain of services will suffice for our purposes:

```
curl -v -X POST -H 'content-type: multipart/form-data' \
  -F LANGUAGE=deu-DE \
  -F TEXT=@<TXT file name> \
  -F SIGNAL=@<WAV file name> \
  'https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runMAUSBasic'
```

As described earlier, 'WAV file name' refers to the signal file containing the recording, while 'TXT file name' refers to the text file (UTF-8 encoded) that contains the orthographic transcript of the recording. This call produces a Praat compatible TextGrid file (Boersma, 2001) that contains the result of the automatic canonical transcription (in the tier 'KAN') and the result of the MAUS segmentation and labelling (in the tier 'MAU').

4.2. Preprocessing of chunk segmentations (ChunkPreparation)

We frequently observe that users apply the automatic segmentation and labelling MAUS to very long recordings of e.g. more than 10 min and up to several hours in a single recording. Since the processing time of MAUS increases exponentially with time length of recording (Pörner, 2016), recordings longer than 20 min should not be processed. Consequently – and also to prevent our web services against denial-of-service-attacks – we limit the maximal upload size of any recording.

For long recordings, a simple solution is to break them (and the corresponding transcript) into smaller, treatable sub-portions called 'chunks'. Fortunately, many users already do this when transcribing the speech signal using for instance the Praat tool. Still, the separation of the recording into smaller chunks and the feeding of these chunks together with their appropriate transcriptions through G2P and MAUS are tedious tasks that requires some programming skills. We therefore offer a combined web service 'runChunkPreparation'.[20] that first transforms a csv, TextGrid (Praat) or EAF (ELAN, Wittenburg et al., 2006) file storing the chunk segmentation into BPF (tier 'TRN'), and then calls the 'G2P' service to create canonical transcripts over all chunks for MAUS processing:

[18] https://weblicht.sfs.uni-tuebingen.de/.
[19] The corresponding web interface is called 'WebMAUS Basic'.
[20] The corresponding web interface is called 'Chunk Preparation'.

```
curl -v -X POST -H 'content-type: multipart/form-data' \
   -F rate= 44,100 \
   -F tier=<praat/ELAN tier name of chunk segmentation> \
   -F lng=deu-DE \
   -F iform=<input format tg,eaf,csv> \
   -F i=@<input filename> \
   'https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/runChunkPreparation'
```

The result of this web service call is a BPF file that can then be sent together with the entire recording to the 'run-MAUS' web service.

It has been shown in several use cases that the usage of the 'Chunk Preparation' service not only allows the automatic segmentation of very long recordings but also improves the quality of segmentation considerably (simply because additional segmental information is exploited).

In a similar fashion, the experienced user may combine web services − even from different architectures into individual automated processing chains. For example, in the Australian AUSTALK project (Wagner et al., 2010), web services of BAS CLARIN were incorporated into processing scripts to allow a server-based on-the-fly automatic time alignment.

5. *Experiences with performance and user acceptance*

Performance in terms of processing time is a major issue when working with web services. On the one hand the user must have a reasonable internet connection, so that up- and downloads of large recording files occur within a feasible time frame, i.e. within the typical time-out window of an HTTP session. With ubiquitous fast internet, this issue will become less relevant, but we have found that some potential users, e.g. linguists/ethnologists/anthropologists working in the field, are less willing to take advantage of internet-based services because they (rightfully) fear drop-outs in sparsely populated areas.[21] Another performance issue is of course server power and its distribution to users as discussed in Section 2. Currently we have received only one user complaint regarding processing power, and a further analysis showed that in that one case the user simply uploaded many very long recordings in the same batch. On balance, our mid-sized server hardware and the elegant distribution schema outlined in Section 2 perform

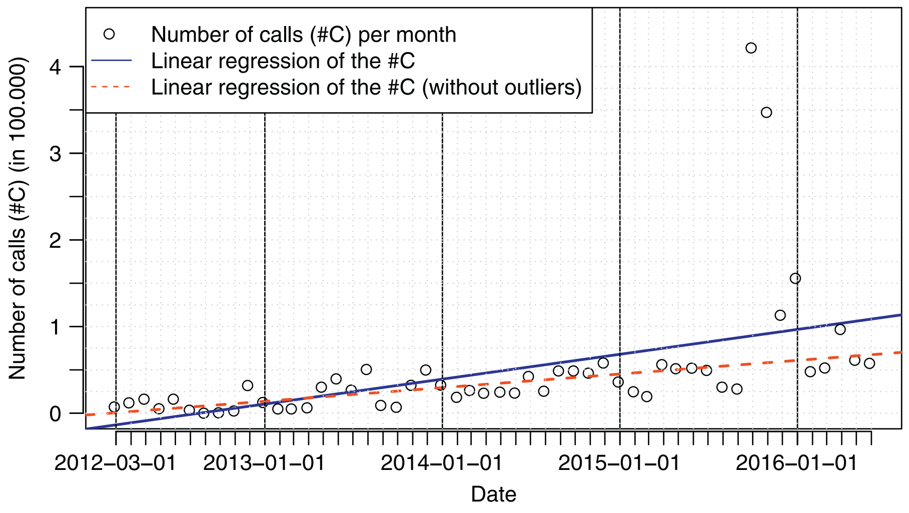

Fig. 6. Monthly user statistic between March 2012 and June 2016, with two regression lines with outliers in late 2015 (solid) and without (dashed) depicting the trend.

[21] One possible way to circumvent this problem is by supplying stand-alone versions of the web services. But as discussed in the introduction, there are many arguments against this, and therefore with one exception (MAUS) we do not provide stand-alone versions.

well. For those planning to develop a similar service architecture, it might be prudent to investigate beforehand the possibilities with the local (university) computing centre: it is often the case that server capacity can be out-sourced, which alleviates a lot of tedious maintenance tasks.

To give a general idea about usage statistics, Fig. 6 plots the number of web service calls per month over the time period that the BAS CLARIN services have existed. In the first 6 months of 2016 alone, about 450,000 web service calls were processed.

Aside from the fact that the BAS CLARIN web services are used frequently, it is a difficult task to evaluate the user satisfaction with web services and/or the web interface. Up until now, no formal evaluation (e.g. a selected user group filling out questionnaires) has taken place and the exact number of web interface users cannot be determined from the server log files, since we do not use cookies or similar techniques to track individual users. If users encounter technical problems or seek assistance in tackling a certain analysis, they sometimes contact the developers via the CLARIN help desk or directly via email. Through these two channels we registered an average of 39.4 first user contacts (threads are not counted) per year since 2012.[22] The distribution of first user contacts across the five years is as follows (2016 is estimated):

2012	2013	2014	2015	2016
23	34	41	55	(44)

The period 2012−2015 was the main development phase of the BAS CLARIN web services; in this period the number of first user contacts increased together with the growing number of REST calls (cf. Fig. 6). At the beginning of 2016, the number of user contacts started to drop while REST calls continued to increase. We interpret this as a positive sign that users require less and less assistance when using our services.

The majority of first user contacts during the first five years of operation were to do with the service WebMAUS (101), followed by 71 contacts regarding the BAS CLARIN data repository, 17 regarding the service G2P, and only eight regarding technical problems with the server, input/output formats, or problems with the local web browser.

6. Discussion and conclusions

The services described in this paper have been operational for almost five years. In this section we summarize the advantages and disadvantages of the proposed architecture according to our experience. In Section 6.3 we sketch some possible future developments.

6.1. Advantages, positive experiences

- Truly independent of operation systems; thus far less user support required.
- Only the back end server and the web interface have to be maintained in contrast to the maintenance of a large number of local installations (effort scales with number of installations).
- The possibility to provide web services that are based on software that is not public domain (e.g. proprietary code, code that is still used for research, etc.).
- Users always use the latest version of the tools.
- Users can profit from parallel computing on the back end server, saving time and local resources.
- More constructive user feedback than with traditional software models; this helps the developers to improve usability and implement new features as requested by users.
- A large part of the functionality is automatically derived from the service metadata description; thus less effort for maintaining code and databases in parallel; very fast integration of new options/features; very short development cycles.
- Access control to services (not described in this paper) allows user grouping, e.g. academic users/students that use the services for free vs. commercial users who are required to buy a user license.

[22] Including first user contacts regarding the BAS CLARIN data repository that is closely linked to the web services.

6.2. Disadvantages, problems encountered

- Problems with web browsers that do not conform to standards; so far only the Google Chrome web browser seems to work under all circumstances; some implementation effort must be spent on browser conformity problems.
- Shiboleth-based protection of web services currently requires a specific behaviour of the AAI Identity Provider (IDP) that hosts the user's credentials, which is not the case for most IDPs at the moment. We therefore do not give any recommendations for user authentification to web services in this paper. Currently all BAS web services have unrestricted access. However, we expect that this will change in the near future.
- All data have to be moved from the local computer to the back end server (only very limited processing in the client); thus a heavy networking load; not always feasible for field workers; legal problems regarding sensitive or classified user data. The latter concerns the transmission itself which can be solved by using an encrypted protocol (https), and the upload and short-term storage on the back end server for the time of processing. With that regard the Conditions of Usage of the BAS web services state the following:
  "The BAS retains the right to store the results only for the technical purpose of providing the service. Stored data will not be transfered to third parties nor exploited or reviewed in any way by BAS, and are deleted automatically after 24 h"
  It is advisable for scientists to formulate access restrictions in a form that allow the usage of this kind of processing.

- Some programming skills required for sophisticated web service chains or embedding web services into applications
- Web interface attracts a new audience of potential users (non-scientific or from non-linguistic fields): this sometimes requires more effort in user support than expected.
- The possibility for quicker development cycles leads to rapid change of back end server versions; since we cannot possibly maintain all former versions and since upgrades cannot always be backward-compatible, sometimes users are not able to reproduce their earlier results at a later time.
- The client-server architecture necessarily means a single point of failure, if there are no redundant system components; if the back end server is down, the damage (in terms of lost user processing) is high.
- Transparency: not easy for users to look into the source code of the services; thus the web service documentation must satisfy as many aspects of the services as possible.
- Web services require reliable and long-term maintenance, yet commercial (or academic) users are unwilling to pay for web services; funding is a permanent problem.

Some of the listed disadvantages could be resolved by providing the complete server software including all back end services as an open source project. This would in theory allow skilled users to set up their own on-site server. There are several practical reasons not to do this: only a subset of services could be included in this open source server due to partly proprietary software components and/or speech data, and the user support would require more man power due to installation problems that we wanted to avoid in the first place by using the client-server approach.

6.3. Possible improvements and future plans

One natural goal is to enlarge the user community by extending the implemented services to as many languages as possible. As a mid-term mile-stone we target to cover all European languages as well as the top five frequently spoken world languages by the end of 2019.

Following the approach of our sister project EmuLabeller (Winkelmann, 2015) we aim to move more processing from the back end to the client (the web browser). This will lighten the network load in the case of long signal files or derived signals (e.g. spectra) which will no longer need to travel from the client to the back end and vice versa. Such an approach is especially necessary because we plan to introduce more signal processing web services in the coming years (see below).

Input/output data could be read from or stored in cloud space instead locally. We plan to investigate the best cloud storage possibilities and implement APIs for these.

The automatic segmentation and labelling of very large recordings is still a technical challenge (see Section 4.2). We plan to implement a new automatic chunk segmentation web service based on the results reported in (Pörner, 2016) that allows the user to first chunk segment very long recordings and then process the chunks in MAUS batch mode. The only limiting factor would then be the size of the signal file upload.

A number of new tools are envisaged for possible implementation as BAS CLARIN web services: the calculation of fundamental frequency tracks, formant tracks, short time energy, spectra, cepstra, and zero crossing rate; automatic geographic speaker localization based on the accent/dialect; automatic language recognition; script-based speech recording (SpeechRecorder[23] as JS implementation).

Those interested in keeping up with the latest developments may subscribe to the BAS email news service (approx. 2 emails per year) under bas-services-news@bas.uni-muenchen.de (subject should contain "MAUS/WebMAUS Mailinglist - subscribe").

6.4. *Concluding remarks*

Currently the web service based architecture (in contrast to traditionally distributed and installed software tools) as described in this contribution has been operational as a public service of the BAS for approximately five years. The growing number of service calls as well as numerous citations and user feedback can be seen as an indicator that the basic concept has been accepted by the addressed scientific community as well as some other (unforeseen) user groups. We do not claim that the proposed concept is the only possible or the optimal way to provide scientific services in the area of speech and language; some user tasks have been and will be better solved using locally installed tools. Depending on the ongoing world-wide development of network infrastructures the future will show which approach will turn out to be the longer-lasting one. The proposed model allowed us to reduce the basic maintenance effort (without new developments) for services to a level that can be covered by the existing funding for system operations, thus making it easier for an academic institution like the BAS to make these services available for a long time into the future.

Acknowledgements

This research was partly supported by the German 'Bundesminister für Bildung und Forschung' (BMBF) within the CLARIN-D project [grant number 01UG1420I]. We also like to thank all contributors to the BAS web services for their input (please refer to the complete list of credentials in the web interface).

Appendix A − Supported languages per web service.

Language\Service	MAUS	G2P	Pho2Syl	TextAlign	MaryTTS
Language Independent	x			x	
deu-DE	x	x	x		x
ekk-EE	x	x	x		
eng-AU	x	x	x		
eng-GB	x	x	x		
eng-NZ	x	x	x		
eng-US	x	x	x		
fin-FI	x	x	x		
fra-FR	x	x	x		
gsw-CH	x	x	x		
gsw-CH-BE	x	x	x		
gsw-CH-BS	x	x	x		

(*continued on next page*)

[23] http://www.bas.uni-muenchen.de/forschung/Bas/software/speechrecorder/.

(*Continued*)

Language\Service	MAUS	G2P	Pho2Syl	TextAlign	MaryTTS
gsw-CH-GR	x	x	x		
gsw-CH-SG	x	x	x		
gsw-CH-ZH	x	x	x		
hat-HT		x	x		
hun-HU	x	x	x		
ita-IT	x	x	x		
kat-GE	x	x	x		
nld-NL	x	x	x		
pol-PL	x	x	x		
por-PT	x				
ron-RO		x	x		
rus-RU	x	x	x		
slk-SK		x	x		
spa-ES	x	x	x		
sqi-SQ		x	x		

Language to RFC5646 language code mapping.

deu-DE	German – Germany
eng-AU	English – Australian
eng-GB	English – Great Britain
eng-NZ	English – New Zealand
eng-US	English – American
fin-FI	Finnish – Finland
fra-FR	French – France
gsw-CH	German – Swiss Dieth
gsw-CH-BE	German – Swiss Dieth, Bern dialect
gsw-CH-BS	German – Swiss Dieth, Basel dialect
gsw-CH-GR	German – Swiss Dieth, Graubunden dialect
gsw-CH-SG	German – Swiss Dieth, St. Gallen dialect
gsw-CH-ZH	German – Swiss Dieth, Zurich dialect
hun-HU	Hungarian – Hungary
ita-IT	Italian – Italy
kat-GE	Georgian – Georgia
nld-NL	Dutch – Netherlands
pol-PL	Polish – Poland
por-PT	Portuguese – Portugal
rus-RU	Russian – Russia
spa-ES	Spanish – Spain
sampa	Language independent

References

Almpanidis, G., Kotropoulos, C., 2008. Phonemic segmentation using the generalized Gamma distribution and small sample Bayesian information criterion. Speech Commun. 50 (2008), 38–55.
Anthes, G., 2012. HTML5 Leads a Web Revolution. Commun. ACM 55 (7), 16–17.
Bigi, B., 2012. SPPAS: a tool for the phonetic segmentations of speech. In: Proceedings of the LREC 2012, Istanbul, Turkey, pp. 1748–1755.
Bisani, M., Ney, H., 2008. Joint-sequence models for grapheme-to-phoneme conversion. Speech Commun. 50, 434–451.
Boersma, P., 2001. Praat, a system for doing phonetics by computer. Glot Int. 5 (9/10), 341–345.
Brants, T., 2000. TnT - a statistical part-of-speech tagger. In: Proceedings of the ANLP-2000, pp. 224–231.

Daelemans, W., Gillis, S., Durieux, G., 1994. The acquisition of stress: A data-oriented approach. Comput. Ling. 20 (3), 421–451.

Dohmas, U., Plag, I., Carroll, R., 2014. Word stress assignment in German, English and Dutch: Quantitysensitivity and extrametricality revisited. J. Comp. Ger. Linguist. 17 (1), 59–96.

Ellbogen, T., Steffen, A., Schiel, F., 2004. The BITS speech synthesis corpus for German. In: Proceedings of the IV International Conference on Language Resources and Evaluation, pp. 2091–2094.

Gorman, K., Howell, J., Wagner, M., 2011. Prosodylab-Aligner: a tool for forced alignment of laboratory speech. Can. Acoust. 39 (3), 192–193.

Hinrichs, E., Krauwer, S., 2014. The CLARIN research infrastructure: resources and tools for E-humanities scholars. In: Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), pp. 1525–1531.

Hinrichs, E., Hinrichs, M., Zastrow, T., 2010. WebLicht: web-based LRT services for German. In: Proceedings of the ACL 2010 System Demonstrations, pp. 25–29.

Jarifi, S., Pastor, D., Rosec, O., 2008. A fusion approach for automatic speech segmentation of large corpora with application to speech synthesis. Speech Commun. 50 (2008), 67–80.

Jiahong, Y., Liberman, M., 2008. Speaker identification on the SCOTUS corpus. In: Proceedings of Acoustics ‘08.

Kipp, A., Wesenick, B., Schiel, F., 1997. Pronunciation modeling applied to automatic segmentation of spontaneous speech. In: Proceedings of the EUROSPEECH 1997, Rhodos, Greece, pp. 1023–1026.

Kisler, T., Schiel, F., Sloetjes, H, 2012. Signal processing via web services: the use case webmaus. In: Proceedings Digital Humanities 2012, Hamburg, pp. 30–34.

Kondrak, G., 2002. Algorithms for Language Reconstruction, (Ph.D. dissertation). University of Toronto.

Levenshtein, V.I, 1966. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 (8), 707–710 Appeared in Russian as: Влзди́*мир* И. Левенштейн (1965). Дв*о*ичные *к*оды *с* исп*р*звлением выпздений, в*с*тзв*ок* и ззмещений *с*имв*о*л*о*в [Binary codes capable of correcting deletions, insertions, and reversals]. Д*ок*лзды *Ак*здемий Нз*ук* СССР.

Liberman, M., Prince, A., 1997. On stress and linguistic rhythm. Linguist. Inq. 8, pp. 249–336.

Lubbers, M., Torreira, F. (2016) Praatalign: an Interactive Praat Plug-in for Performing Phonetic Forced Alignment. Retrieved from https://github.com/dopefishh/praatalign.

McAuliffe, M., Socolof, M., Mihuc, S., Wagner, M. (2016) Montreal Forced Aligner [Computer program]. Version 0.5, retrieved 13 July 2016 from http://montrealcorpustools.github.io/Montreal-Forced-Aligner/.

Pearson, S., Kuhn, R., Fincke, S., Kibre, N., 2000. Automatic methods for lexical stress assignment and syllabification. In: Proceedings of the Interspeech, Pittsburgh, Pennsylvania, pp. 423–426.

Pörner, N., 2016. Development of an automatic chunk segmentation tool for long transcribed recordings (Master thesis). Ludwig-Maximilian-Universität, München.

Quinlan, J.R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo.

Räsänen, O.J., Laine, U.K., Altosaar, T., 2009. An improved speech segmentation quality measure: the R-value. In: Proceedings of the INTERSPEECH 2009, Brighton, UK, pp. 1851–1854.

Reichel, U.D., Bucar Shigemori, L.S., 2008. Automatic correction of part-of-speech corpora. In: Demenko, G., Jassem, K., Szpakowicz, S. (Eds.), Speech and Language Technology, 11, pp. 167–174.

Reichel, U.D., Kisler, T., 2014. Language-independent grapheme-phoneme conversion and word stress assignment as a web service. In: Hoffmann, R. (Ed.), Elektronische Sprachverarbeitung. Studientexte zur Sprachkommunikation. 71, TUDpress, Dresden, pp. 42–49.

Reichel, U.D., Schiel, F., 2005. Using Morphology and phoneme history to improve grapheme-to-phoneme conversion. In: Proceedings of the Eurospeech, Lisbon, pp. 1937–1940.

Reichel, U.D., 2012a. PermA and Balloon: Tools for string alignment and text processing. In: Proceedings of the Interspeech. Portland, Oregon, Paper 346.

Reichel, U.D., 2012b. Probabilistic induction of metrical trees for word stress assignment. In: Wolff, M. (Ed.), Elektronische Sprachverarbeitung. Studientexte zur Sprachkommunikation. 64, TUDpress, Dresden, pp. 137–144.

Romary, L., Chambers, S. (2014) DARIAH: Advancing a digital revolution in the arts and humanities across Europe. Data Archiving and Networked Services (DANS).

Schmidt, R., 2002. Automatic speech alignment: a method for handling disturbances and simultaneous utterances. In: Proceedings of the KONVENS 2002, Saarbrücken, Germany, pp. 171–175.

Schiel, F., Burger, S., Geumann, A., Weilhammer, K., 1998. The partitur format at BAS. In: Proceedings of the 1st International Conference on Language Resources and Evaluation, Granada, Spain, pp. 1295–1301.

Schiel, F., 1999. Automatic phonetic transcription of non-prompted speech. In: Proceedings of the ICPhS 1999, San Francisco, pp. 607–610. August 1999.

Schiel, F., 2004. MAUS Goes Iterative. In: Proceedings of the IV International Conference on Language Resources and Evaluation, Lisbon, Portugal, pp. 1015–1018.

Schiel, F., 2015. A statistical model for predicting pronunciation. In: Proceedings of the ICPhS. paper no. 195.

Schroeder, M., Charfuelan, M., Pammi, S., Steiner, I., 2011. Open source voice creation toolkit for the MARY TTS Platform. In: Proceedings of the Conference of the International Speech Communication Association - Interspeech 2011, Aug 2011. Florence, Italy. ISCA, pp. 3253–3256.

Sproat, R., Black, A.W., Chen, S., Kumar, S., Ostendorf, M., Richards, C., 2001. Normalization of non-standard words. Comput. Speech Lang. 15, 287–333.

van den Bosch, A., Daelemans, W. (1993) Data-Oriented Methods For Grapheme-To-Phoneme Conversion, ITK Research Report, Tilburg, Tech. Rep. 42.

Vennemann, T., 1988. Preference Laws For Syllable Structure and the Explanation of Sound Change. Mouton de Gruyter, Berlin.

Wagner, R.A., Fischer, M.J., 1974. The string-to-string correction problem. J. ACM 21 (1), 168–173.

Wagner, M., Tran, D., Togneri, R., Rose, P., Powers, D., Onslow, M., Loakes, D., Lewis, T., Kuratate, T., Kinoshita, Y., Kemp, N., Ishihara, S., Ingram, J., Hajek, J., Grayden, D.B., Göcke, R., Fletcher, J., Estival, D., Epps, J., Dale, R., Cutler, A., Cox, F., Chetty, G., Cassidy, S., Butcher, A., Burnham, D., Bird, S., Best, C., Bennamoun, M., Arciuli, J., Ambikairajah, E., 2010. The big Australian speech corpus (the Big Asc). In: Tabain, M., Fletcher, J., Grayden, D., Hajek, J., Butcher, A. (Eds.), Proceedings of the 13th Australasian International Conference on Speech Science and Technology. ASSTA, Melbourne, pp. 166–170.

Wang, L., Zhao, Y., Chu, M., Zhao, J., Cao, Z., 2004. Refining segmental boundaries for TTS database using fine contextual-dependent boundary models. In: Proceedings of the ICASSP 2004, 1, pp. 641–644.

Wester, M., Kessens, J., Strik, H., 1998. Improving the performance of a Dutch CSR by modeling pronunciation variation. In: Proceedings of the Workshop on Modeling Pronunciation Variation Rolduc, Netherlands, pp. 145–150.

Windhouwer, M., Broeder, D., van Uytvanck, D., 2012. ACMD core model for CLARIN web services. In: Proceedings of LREC 2012: 8th International Conference on Language Resources and Evaluation, pp. 41–48.

Winkelmann, R., 2015. Managing Speech Databases with emuR and the EMU-webApp. In: Proceedings of the Sixteenth Annual Conference of the International Speech Communication Association (ISCA).

Wittenburg, P., Brugman, H., Russel, A., Klassmann, A., Sloetjes, H., 2006. Elan: a professional framework for multimodality research. In: Proceedings of Language Resources and Evaluation Conference (LREC).

Young, S.J., Odell, J., Ollason, D., Woodland, P., 1996. The HTK Book. Entropic Cambridge Research Laboratory.