Available online at www.sciencedirect.com

ScienceDirect

Computer Speech & Language 45 (2017) 392–410

www.elsevier.com/locate/csl

EMU-SDMS: Advanced speech database management and analysis in R

Raphael Winkelmann*, Jonathan Harrington, Klaus Jänsch

Institute of Phonetics and Speech Processing, Ludwig-Maximilians University of Munich, Munich, Germany

Received 7 September 2016; Accepted 24 January 2017

Abstract

The amount and complexity of the often very specialized tools necessary for working with spoken language databases has continually evolved and grown over the years. The speech and spoken language research community is expected to be well versed in multiple software tools and have the ability to switch seamlessly between the various tools, sometimes even having to script ad-hoc solutions to solve interoperability issues. In this paper, we present a set of tools that strive to provide an all-in-one solution for generating, manipulating, querying, analyzing and managing speech databases. The tools presented here are centered around the R language and environment for statistical computing and graphics (R Core Team, 2016), which benefits users by significantly reducing the number of tools the researchers have to familiarize themselves with. This paper introduces the next iteration of the EMU system that, although based on the core concepts of the legacy system, is a newly designed and almost entirely rewritten set of modern spoken language database management tools.

Keywords: EMU-SDMS; emuR; wrassp; EMU-webApp; Speech databases; Speech annotation; Speech database management

1. Introduction

In the speech sciences and spoken language research, researchers often initially attempt to either find or generate a set of digital speech recordings which contain enough instances of the subject matter they are interested in. To find the relevant items of spoken speech, the data is annotated by temporally aligning symbolic information with the raw digital data, which consist of sampled values over time. As the object of spoken language research is the speech signal itself as well as other accompanying supplementary signals such as electromagnetic articulography (EMA) or electropalatography (EPG) data, researchers will then attempt to process the relevant speech items to extract information (e.g., fundamental frequencies, formants, RMS-energy) that can be used to support or falsify their hypotheses. The retrieved values are then used to visually and statistically inspect and compare the conditions at hand.

Although the above schematic process sounds simple, everyone who has been involved in the process knows how extremely tedious and error-prone the process from annotating to signal processing and statistical analysis can be.

* Corresponding author.
E-mail address: raphael@phonetik.uni-muenchen.de (R. Winkelmann), jmh@phonetik.uni-muenchen.de (J. Harrington), klausj@phonetik.uni-muenchen.de (K. Jänsch).

http://dx.doi.org/10.1016/j.csl.2017.01.002
0885-2308/ 2017 Elsevier Ltd. All rights reserved.

Usually there are multiple steps, often with multiple software tools necessary at each step. Before data can be analyzed, researchers are required to work with a plethora of tools that may not have been designed to work together as a system. Over the years, hundreds of very specialized software tools have been developed that provide new and innovative methods to spoken language researchers. However, these tools usually lack a common interface to allow and perpetuate interoperability. Only recently, tools have become available that have a well-defined concept of a speech and spoken language database and help users manage these databases. Three noteworthy systems are the LaBB-CAT system (Fromont and Hay, 2012), the Speech Corpus Tools (SCT) (McAuliffe and Sonderegger, 2016) as well as Phon (Rose et al., 2006). The LaBB-CAT system (formerly known as ONZE Miner) is a browser-based linguistics research tool that relies on a cross-platform Java server back-end implementation and a MySQL data store. Once the server is set up, which requires (semi-)advanced administration knowledge, the user can access local or remote databases via a web browser. LaBB-CAT offers features such as storage and maintenance of media and transcripts, automatic annotation (e.g., syllabification, part-of-speech tagging and forced alignment), search functions and various import and export routines which are also used for manual corrections of various transcripts. The Phon software, which was developed as part of the efforts of PhonBank, TalkBank and CHILDES projects (Rose and MacWhinney, 2014; MacWhinney, 2007; 2000), is provided as an easy to install standalone cross-platform Java application. Similar to the LaBB-CAT system, it provides the user with multiple database management, automatic annotation and analysis features. Phon stores its data using the Apache Derby DB (The Apache DB Project, 2016) and has references to external files (e.g., TextGrids and media files) that supplement the internal representations. The SCT are built on a client-server architecture that uses a graph database back-end using the graph database Neo4J (2016). Although the above tools offer sophisticated speech database interaction and management routines, they still require that the users export their data to then perform their analysis and evaluation with a different software tool. In this paper, we will present a new set of tools that are fully integrated into a free and widely used statistical processing ecosystem to try to unify and simplify common workflows and address interoperability issues.

The EMU system, which has been developed continuously over a number of years (e.g., Harrington et al., 1993; Cassidy and Harrington, 1996; 2001; Bombien et al., 2006; Harrington, 2010; John, 2012), sets out to be as close to an all-in-one solution for generating, manipulating, querying, analyzing and managing speech databases as possible. Unfortunately, due to advances in compiler standards, cross-platform installability issues, modern developer language preferences and several other maintainability issues, the system has become dated. For example, the scripting and user interface language of the EMU system was Tcl and Tk (Ousterhout and Jones, 2009). Although still used by some speech tools (e.g., Wavesurfer by Sjölander and Beskow, 2000) modern developer and speech science programming languages preferences have shifted towards other languages such as Python (e.g., the NLTK by Bird, 2006) as a general purpose language and R for graphical and statistical analysis (R Core Team, 2016).

This paper introduces the components that comprise the completely rewritten and newly designed next incarnation of the EMU system, which we will refer to as the EMU Speech Database Management System (EMU-SDMS). The EMU-SDMS keeps most of the core concepts of the previous system, which we will refer to as the legacy system, in place while improving on things like usability, maintainability, scalability, stability, speed and more. We feel the redesign elevates the system into a modern set of speech and language tools that enables a workflow adapted to the challenges confronting speech scientists and the ever growing size of speech databases. The redesign has enabled us to implement several components of the new EMU-SDMS so that they can be used independently of the EMU-SDMS for tasks such as web-based collaborative annotation efforts and performing speech signal processing in a statistical programming environment. Nevertheless, the main goal of the redesign and reimplementation was to provide a modern set of tools that reduces the complexity of the tool chain needed to answer spoken language research questions down to a few inter-operable tools. The EMU-SDMS aims to answer questions such as, *Do the phonetic segments annotated /s/, /z/, /S/ or /Z/ (sibilants) in a specific spoken language database differ with respect to their first spectral moment?* The tools EMU-SDMS provides are designed to streamline the process of obtaining usable data, all from within an environment that can also be used to analyze, visualize and statistically evaluate the data.

2. EMU-SDMS: system architecture

Rather than starting completely from scratch, it seemed more appropriate to reuse partially the concepts of the legacy system in order to achieve our goals. A major observation at the time was that the R language and

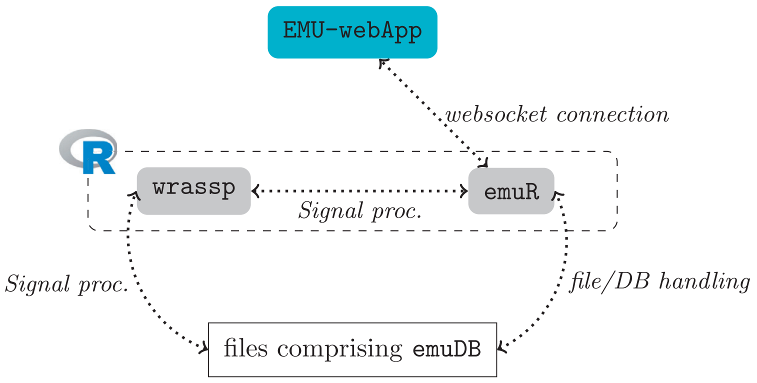


Fig. 1. Schematic architecture of the EMU-SDMS.

environment for statistical computing and graphics (R Core Team, 2016) was gaining more and more traction for statistical and data visualization purposes in the speech and spoken language research community and many other disciplines. However, R was mostly only used towards the end of the data analysis chain where data usually was pre-converted into a comma separated values or equivalent file format by the user using other tools to calculate, extract and pre-process the data. While designing the new EMU-SDMS, we brought R to the front of the tool chain to the point just beyond data acquisition. This allows the entire data annotation, data extraction and analysis process to be completed in R, while keeping the key user requirements in mind. Due to the experience gained with the legacy system, we learned that the key user requirements were data and database portability, a simple installation process, a pleasant user experience and cross-platform availability. Supplying all of EMU-SDMS's core functionality in the form of R packages that do not rely on external software at runtime seemed to meet all of those requirements.

As the early incarnations of the legacy EMU system and its predecessors were conceived either at a time that predated the R system or during the infancy of R's package ecosystem, the legacy system was implemented as a modular yet composite standalone program with a communication and data exchange interface to the R/Splus systems (see Cassidy and Harrington, 2001, Section 3 for details). Recent developments in the package ecosystem of R, such as the availability of the `DBI` package (R Special Interest Group on Databases (R-SIG-DB) et al., 2016) and the related packages `RSQLite` and `RPostgreSQL` (Wickham et al., 2014; Conway et al., 2016), the `jsonlite` package (Ooms, 2014) and the `httpuv` package (RStudio and Inc., 2015), have made R an attractive sole target platform for the EMU-SDMS. These and other packages provide additional functional power that enabled EMU-SDMS's core functionality to be implemented in the form of R packages. The availability of certain R packages had a large impact on the architectural design decisions that we made for the new system.

The new EMU-SDMS is made up of four main components. The components are the `emuDB` format; the R packages `wrassp` and `emuR`; and the `EMU-webApp`, which is EMU-SDMS's new graphical user interface (GUI) component[1]. An overview of the EMU-SDMS's architecture and the components' relationships within the system is shown in Fig. 1. In Fig. 1, the central role of the `emuR` package becomes apparent as it is the only component that interacts with all of the other components of the EMU-SDMS. It performs file/DB handling on the files that comprise an `emuDB` (see Section 3); it uses the `wrassp` package for signal processing purposes (see Section 4); and it can serve `emuDB`s to the `EMU-webApp` (see Section 5).

3. Annotation structure modeling and database format

One of the most common approaches for creating time-aligned annotations has been to differentiate between

[1] The source code of the components of the EMU-SDMS is available from our institution GitHub account (https://github.com/IPS-LMU), the respective R package components have been released on the Comprehensive R Archive Network, CRAN (https://cran.r-project.org/package=emuR and https://cran.r-project.org/package=wrassp) and a system overview page is also available (http://ips-lmu.github.io/EMU.html). Both R packages contain introductory long form documentation in the form of vignettes (see `wrassp_intro` and `emuR_intro` vignettes in the respective R packages) and the EMU-webApp provides its own documentation via its modal help window.

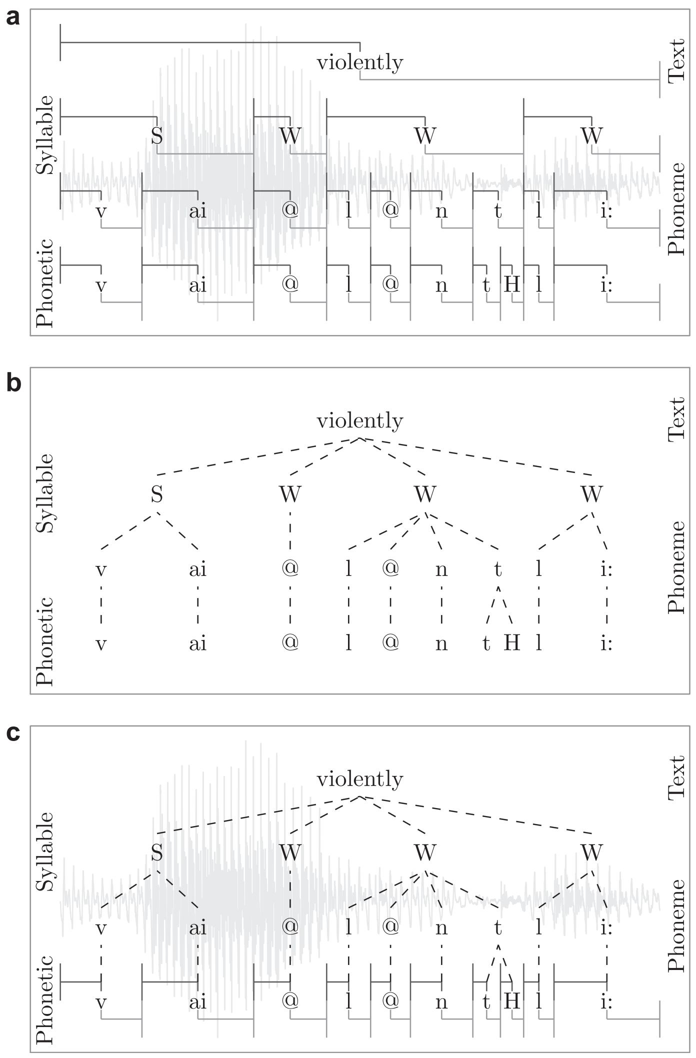


Fig. 2. **a:** purely time-aligned annotation; **b:** purely timeless, symbolic annotation; **c:** time-aligned hierarchical annotation.

events that occur at a specific point in time but have no duration and segments that start at a point in time and have a duration. These annotation items are then grouped into time-ordered sets that are often referred to as tiers. As certain research questions benefit from different granularities of annotation, the timeline is often used to relate implicitly items from multiple tiers to each other as is shown in Fig. 2a. While sufficient for single or unrelated tier annotations, we feel this type of representation is not suitable for more complex annotation structures, as it results in unnecessary, redundant data and data sets that are often difficult to analyze. This is because there are no explicit relationships between annotation items and it is often necessary to introduce error tolerance values to analyze slightly misaligned time values to find relationships iteratively over multiple levels. The main reason for the prevalence of this sub-optimal strategy is largely because the available software tools (e.g., Boersma and Weenink, 2016) do not permit any other forms of annotations. These widely used annotation tools often only permit the creation and manipulation of segment and event tiers which in turn has forced users to model their annotation structures on these building blocks alone.

Linguists who deal with speech and language on a purely symbolic level tend to be more familiar with a different type of annotation structure modeling. They often model their structures in the form of a vertically oriented, directed acyclic graph that, but for a few exceptions that are needed for things like elision modeling (e.g., the /I/ elision that may occur between the canonical representation of the word ‘family’ /fæmIli/ and its phonetic representation [fæmli]), loosely adheres to the formal definition of a tree in the graph-theoretical sense (Knuth, 1968) as depicted in Fig. 2b. While this form of modeling explicitly defines relationships between annotational items (represented by dashed lines in Fig. 2b), it lacks the ability to map these items to the timeline and therefore the respective speech signal.

To our knowledge, the legacy EMU system (Cassidy and Harrington, 2001) and its predecessors (e.g., Harrington et al., 1993) were the first to fuse pragmatically purely time-aligned and symbolic tree-like annotations. This was achieved by providing software tools that allowed for these types of annotation structures to be generated, queried and evaluated. In practice, each annotational item had its own unique identifier within the annotation. These unique IDs could then be used to reference each individual item and link them together using dominance relations to form the hierarchical annotation structure. On the one hand, this dominance relation implies the temporal inclusion of the linked sub-level items and was partially predicated on the *no-crossing constraint* as described in Coleman and Local (1991). This constraint does not permit the crossing of dominance relationships with respect to their sequential ordering (see also Section 4.2 of Cassidy and Harrington, 2001). Since the dominance relations imply temporal inclusion, events can only be children in a parent-child relationship. To allow for timeless annotational items, a further timeless level type was used to complement the segment and event type levels used for time-aligned annotations. Each level of annotational items was stored as an ordered set to ensure the sequential integrity of both the time-aligned and timeless item levels. The legacy system also reduced data redundancy by allowing parallel annotations to be defined for any given level (e.g., a segment level bearing SAMPA annotations as well as IPA UTF-8 annotations).

The new EMU-SDMS has adopted some concepts of the legacy system in that levels of type `SEGMENT` and `EVENT` contain annotational units with labels and time information, similar to the tiers known from other software tools such as Praat, while levels of type `ITEM` are timeless and contain annotational units with labels only. `SEGMENT` and `EVENT` levels differ in that units at the `SEGMENTs` level have a start time and a duration, while units at the `EVENT` level contain a single time point only. Additionally, every annotational unit is able to contain multiple labels and has a unique identifier which is used to link items across levels. These building blocks provide the user with a general purpose annotation modeling tool that allows complex annotation structures to be modeled that best represent the data. An example of a time-aligned hierarchical annotation is depicted in Fig. 2c, which essentially combines the annotation of Fig. 2b with the most granular time-bearing level (i.e. the “Phonetic” level) of Fig. 2a.

In accordance with other approaches (see among others Bird and Liberman, 2001; Zipser and Romary, 2010; Ide and Romary, 2004), the EMU-SDMS annotation structure can be viewed as a graph that consists of three types of nodes (`EVENTs`, `SEGMENTs`, `ITEMs`) and two types of relations (`dominance` and `sequence`) which are directed, transitive and indicate the dominance and sequential relationships between nodes of the graph. As was shown in a pseudo-code example that converted an Annotation Graph (Bird and Liberman, 2001) into the legacy EMU annotation format in Cassidy and Harrington (2001), these formats can be viewed as conceptually equivalent sub- or super-set representations of each other. This has also been shown by developments of meta models with independent data representation such as `Salt` (Zipser and Romary, 2010), which enable abstract internal representations to be derived that can be exported to equal-set or super-set formats without the loss of information. We therefore believe that the decision as to which data format serializations are used by a given application should be guided by the choice of technology and the target audience or research field. This is consistent with the views of the committee for the Linguistic Annotation Framework who explicitly state in the ISO CD 24612 (LAF) document (ISO, 2012):

> Although the LAF pivot format may be used in any context, it is assumed that users will represent annotations using their own formats, which can then be transduced to the LAF pivot format for the purposes of exchange, merging and comparison.

The transduction of an EMU annotation into a format such as the LAF pivot format is a simple process, as they share many of the same concepts and are well defined formats.

3.1. Per database annotation structure definition

Unlike other systems, the EMU-SDMS requires the user to define the annotation structure formally for all annotations within a database. Much like document type definitions (DTD) or XML schema definitions (XSD) describe the syntactically valid elements in an XML document, the database configuration file of an `emuDB` defines the valid annotation levels and therefore the type of items that are allowed to be present in a database. Unlike DTDs or XSDs, the configuration file can also define semantic relationships between annotation levels which fall out of the scope of traditional, syntactically oriented schema definitions and validation. This global definition of annotation structure has numerous benefits for the data integrity of the database, as the EMU-SDMS can perform consistency checks and prevent malformed as well as semantically void annotation structures. Because of these formal definitions, the EMU system generally distinguishes between the actual representations of a structural element which are contained within the database and their formal definitions. An example of an actual representation i.e. a subset of the actual annotation would be a level contained in an annotation file that contains `SEGMENT`s that annotate a recording. The corresponding formal definition would be a level definition entry in the database's configuration file, which specifies and validates the level's existence within the database.

As mentioned above, the actual annotation files of an `emuDB` contain the annotation items as well as their linking information. To be able to check the validity of a connection between two items, the user specifies which links are permitted for the entire database just as for the level definitions. The permitted hierarchical relationships in an `emuDB` are expressed through link definitions between level definitions as part of the database configuration. There are three types of valid links: `ONE_TO_MANY`, `MANY_TO_MANY` and `ONE_TO_ONE`. These links specify the permitted relationships between instances of annotation items of one level and those of another. The structure of the hierarchy that corresponds to the annotation depicted in Fig. 2c can be seen in Fig. 3a. The structure in Fig. 3a is a typical example of an EMU hierarchy where only the "Phonetic" level of type `SEGMENT` contains time information and the others are

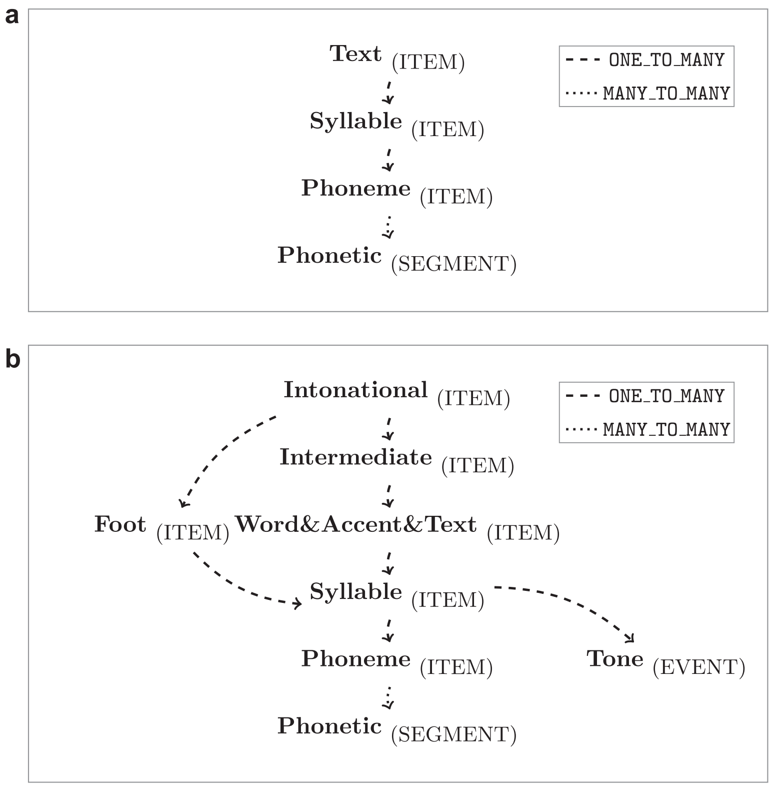


Fig. 3. **a:** schematic representation of the hierarchical structure of an `emuDB` that corresponds to the annotation depicted in 2 c; **b:** example of a more complex, intersecting hierarchical structure.

timeless as they are of the type `ITEM`. The top three levels, "Text", "Syllable" and "Phoneme", have a `ONE_TO_MANY` relationship specifying that a single item in the parent level may have a dominance relationship with multiple items in the child level. In this example, the relationship between "Phoneme" and "Phonetic" is `MANY_TO_MANY`: this type of relationship can be used to represent sonorant syllabification in which the final syllable of *"sudden"* might include *"d@n"* at the "Phoneme" but *"@n"* at the "Phonetic" level. Fig. 3b displays an example of a more complex, intersecting hierarchical structure definition where Abercrombian feet (Abercombie, 1967) are incorporated into the ToBI prosodic hierarchy by allowing an intonational phrase to be made up of one or more feet (for further details see Harrington, 2010, page 98).

Based on our experience, the explicit definition of the annotation structure for every database which was also integral to the legacy system addresses the excessively expressive nature of annotational modeling systems mentioned in Bird and Liberman (2001). Although, in theory, infinitely large hierarchies can be defined for a database, users of the legacy system typically chose to use only moderately complex annotation structures. The largest hierarchy definitions we have encountered spanned up to fifteen levels while the average amount of levels was between three and five levels. This self-restriction is largely due to the primary focus of speech and spoken language domain specific annotations, as the number of annotation levels between chunks of speech above the word level (intonational phrases/sentences/turns/etc.) and the lower levels (phonetic segmentation/EMA gestural landmark annotation/tone annotation/etc.) is a finite set.

3.2. `emuDB` *database format*

We decided on the Java Script Object Notation (JSON) file format as our primary data source for several reasons. It is simple, standardized, widely-used and text-based as well as machine and human readable. In addition, this portable text format allows expert users to (semi-) automatically process/generate annotations. Other tools such as the BAS Webservices (Kisler et al., 2012) and SpeechRecorder (Draxler and Jänsch, 2004) have already taken advantage of being able to produce such annotations. Using database back-end options such as relational or graph databases, whether of the SQL or NoSQL variety, as the primary data source for annotations would not directly permit other tools to produce annotations. Further, intermediary exchange file formats would have to be defined to permit this functionality with these back-ends. Our choice of the JSON format was also guided by the decision to incorporate web technologies for which the JSON format is the de facto standard as part of the EMU-SDMS (see Section 5).

In contrast to other systems, including the legacy EMU system, we chose to fully standardize the on-disk structure of speech databases with which the system is capable of working. This provides a standardized and structured way of storing speech databases while providing the necessary amount of freedom and separability to accommodate multiple types of data. Further, this standardization enables fast parsing and simplification of file-based error tracking and simplifies database subset and merging operations as well as database portability.

An `emuDB` consists of a set of files and folders that adhere to a certain structure and naming convention (see

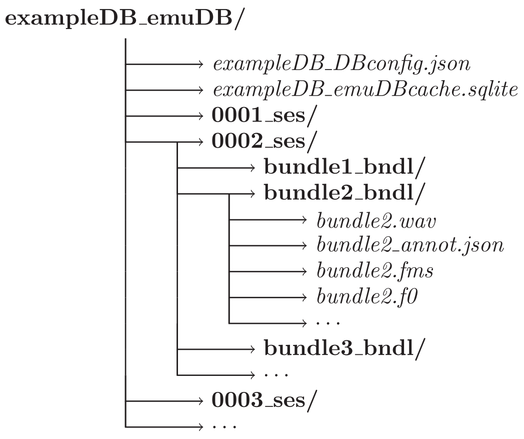


Fig. 4. Schematic `emuDB` structure.

Fig. 4). The database root directory must include a single `_DBconfig.json` file that contains the configuration options of the database such as its level definitions, how these levels are linked in the database hierarchy and how the data is to be displayed by the graphical user interface. The database root folder also contains arbitrarily named session folders (except for the obligatory `_ses` suffix). These session folders can be used to group the recordings of a database in a logical manner. Sessions could be used, for example, to group all recordings from speaker `AAA` into a session called `AAA_ses`.

Each session folder can contain any number of `_bndl` folders, e.g., `rec1_bndl` `rec2_bndl` ... `rec9_bndl`. All files belonging to a recording, i.e. all files describing the same timeline, are stored in the corresponding bundle folder. This includes the actual recording (`.wav`) and can contain optional derived or supplementary signal files in the simple signal file format (SSFF) (Cassidy, 2013) such as formants (`.fms`) or the fundamental frequency (`.f0`), both of which can be calculated using the `wrassp` package (see Section 4). Each bundle folder contains the annotation file (`_annot.json`) of that bundle i.e. the annotations and the hierarchical linking information. JSON schema files for all the JSON files types used have been developed to ensure the syntactic integrity of the database. The optional `_emuDBcache.sqlite` file in the root directory (see Fig. 4) contains the relational cache representation of the annotations of the `emuDB` (see Section 6.1 for further details).

4. `wrassp`: signal processing

There has been substantial growth in the contribution of field-specific R packages to CRAN across many different disciplines in recent years. This large set of inter-operable tools has made R a very mature and diverse ecosystem well beyond the realm of pure statistical computation and visualization. Unfortunately, however, when evaluating the platform as a target environment for the EMU-SDMS, we discovered that no speech-specific signal processing or speech signal file handling packages that filled our requirements were available. As these operations are an integral part of the EMU-SDMS, we needed to add this functionality to the R ecosystem.

In order to do this, the existing advanced speech signal processor library (`libassp`) (Scheffers and Bombien, 2012) that was integral to the legacy system was ported to R. This resulted in the wrapper for R for the libassp (`wrassp`) package. The `libassp` and therefore the `wrassp` package provide time- and frequency-based signal processing routines as well as functionality for handling a variety of common speech signal and other signal file formats.

As the `libassp` is written entirely in C, we used the C foreign language interface provided by R to write several wrapper functions to map the internal memory representations of the `libassp` to an appropriate R object representation. This resulted in `wrassp`'s central `AsspDataObject` object that can be viewed as a generic in-memory R object representation of any of the file formats that the `libassp` supports. Therefore, this object is also used as the single file, in-memory target output of all of `wrassp`'s signal processing functions. The file handling read and write functions provide the R ecosystem with a simple mechanism for accessing speech data files such as WAV audio files that can be read, evaluated or manipulated and written back to disk from within R. A schematic representation of how these functions interact with each other is illustrated in Fig. 5. Both the file handling functions (`read AsspDataObject`/`write AsspDataObject` in the light gray box) as well as the signal processing functions (see box heading) can be used to generate an `AsspDataObject` in R (indicated by white arrows). The black arrows indicate file interactions with the various on-disk file formats.

The focal point of the `wrassp` package is its signal processing functions. These include procedures for the calculation of formants, fundamental frequency (e.g., an implementation of the Schäfer-Vincent, 1983, algorithm), root mean square, autocorrelation, a variety of spectral analyses, zero crossing rate and filtering. As these procedures are written entirely in C they can run at native speeds on every platform supported by R and have almost no additional interpreter overhead.

The `wrassp` package (Winkelmann et al., 2015) solved many of the problems that arose in using R as the primary platform. `wrassp` is, to our knowledge, the first CRAN package that specializes in speech signal processing procedures and handles common speech file formats. It therefore elevates the R ecosystem to a viable sole target platform on which to implement a speech database management system.

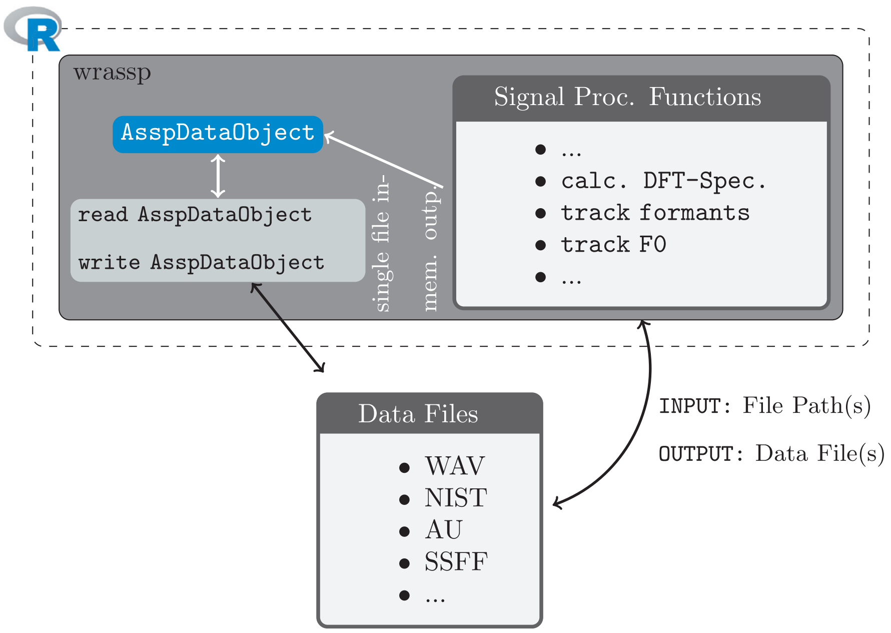


Fig. 5. Schematic `wrassp` architecture.

5. `EMU-webApp`: browser-based graphical user interface

Although R has very sophisticated visualization capabilities, it lacks a modern tool set to build complex interactive graphical user interfaces that are required for annotation and visualizing speech data. In addition, we needed a tool that was able to construct and edit annotations of the form described in Section 3.2. Existing tools for speech processing and annotation such as Praat (Boersma and Weenink, 2016) and ELAN (Wittenburg et al., 2006) cannot integrate external data such as EMA traces and also do not meet the EMU-SDMS's requirements for annotation structure modeling. Therefore, we decided to implement a new annotation tool that met these needs.

As mentioned in Section 3, two of the initial key requirements for the EMU-SDMS that also apply to its graphical user interface (GUI) were to write an interface that is cross-platform and as easy as possible for the user to install. The classic approach for building cross-platform GUIs is either to use a cross-platform widget toolkit or to implement multiple versions of the same GUI to accommodate the different platforms. Both methods can be very cumbersome. Fortunately, the GUI component for the new EMU system was devised at a time when new APIs that are part of the HTML5 specification made it possible to use the browser as a platform upon which to implement a full client-side speech annotation tool. Modern browsers offer a unified solution for fitting the write-once-run-everywhere paradigm that are better than most alternatives. Today most machines, including mobile devices such as smart-phones and tablets, already have the appropriate runtime for a web application installed. If such a browser is available to the user, the browser only needs to be pointed to a specific URL without the need for any installation.

The `EMU-webApp` (Winkelmann and Raess, 2015) is written entirely in HTML, Javascript and CSS. This allows the complete annotation functionality to be accessed online as well as offline in a browser. The offline functionality of the `EMU-webApp` is achieved by using the HTML5 application cache browser API (W3C, 2016). All the visual representations of the speech signal including the calculation and rendering of the spectrogram are calculated client-side without the need for a server.

Although the `EMU-webApp` does not perform any signal processing directly, except for the calculation of the spectrogram it renders, it can be used to display any data that is stored in the simple signal file format (SSFF), which matches the output of most of the signal processing functions provided by the `wrassp` package (see Section 4) as well as supplementary data such as EMA or EPG contours. As such, the `EMU-webApp` is able to integrate external

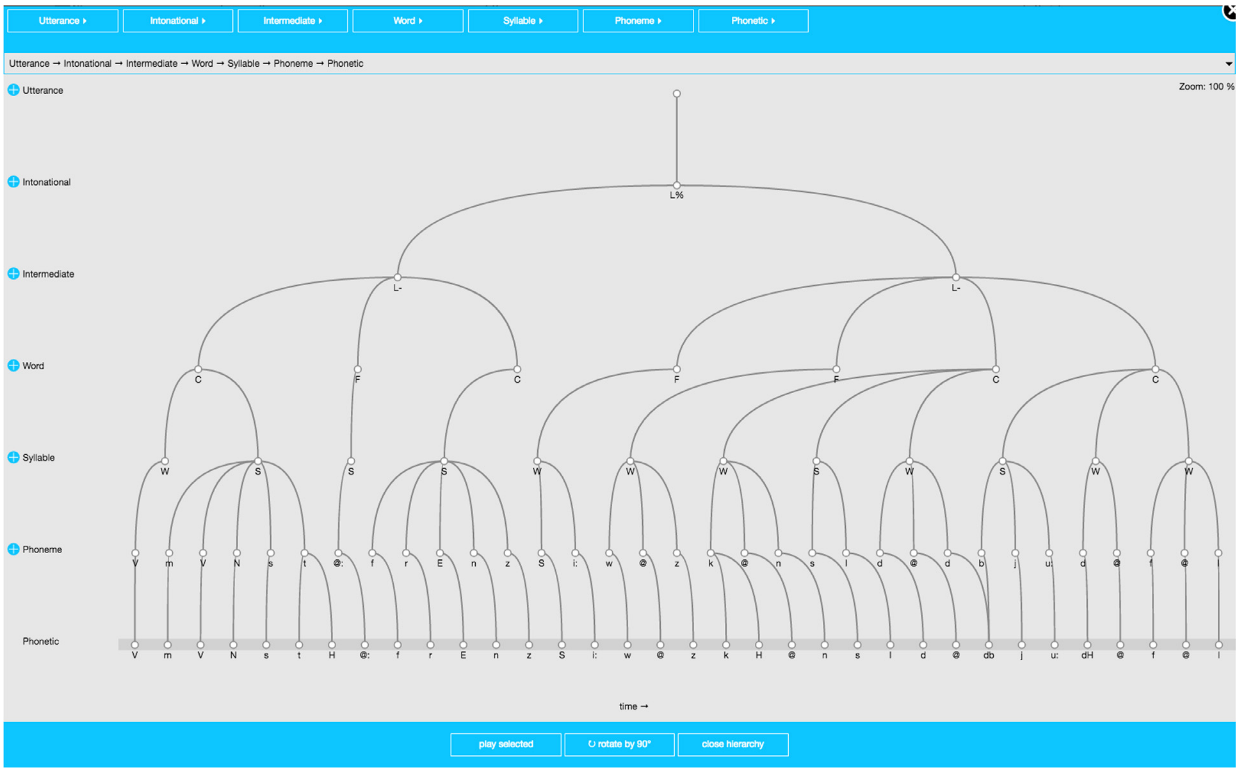


Fig. 6. Screenshot of `EMU-webApp` displaying hierarchical annotation.

supplementary and derived data sources. As its internal annotation structure is identical to an `_annot.json` file, it is also able to construct and edit hierarchical annotations (see Fig. 6) and in doing so fulfills the annotation structure editing needs of the EMU-SDMS. As audio files can also be parsed and displayed, the `EMU-webApp` is capable of displaying any valid data in an `emuDB`.

Two further noteworthy features of the `EMU-webApp` are its ability to allow the user to perform manual formant corrections and to define multiple views, called perspectives, for any `emuDB`. This means that the webApp could, for example, be configured to display and edit the "word" as well as the "tt" (tongue tip gestural landmark annotation) in one perspective while allowing for formants to be manually corrected with the help of the "phonetic" segmentation in another perspective. A screenshot of the `EMU-webApp` displaying EMA data is shown in Fig. 7.

5.1. Communication protocol

A large benefit gained by choosing the browser as the user interface is the ability to easily interact with a server using standard web protocols, such as http, https or websockets. In order to standardize the data exchange with a server, we have developed a simple request-response communication protocol on top of the websocket standard. This decision was strongly guided by the availability of the `httpuv` R package (RStudio and Inc., 2015). Our protocol defines a set of JSON objects for both the requests and responses. A large subset of the request-response actions, most of them triggered by the client after connection, are displayed in Table 1.

This protocol definition makes collaborative annotation efforts possible, as developers can easily implement servers for communicating with the `EMU-webApp`. Using this protocol allows a database to be hosted by a single server anywhere on the globe that then can be made available to a theoretically infinite number of users working on separate accounts logging individual annotations, time and date of changes and other activities such as comments added to problematic cases. Tasks can be allocated to and unlocked for each individual user by the project leader. As such, user management in collaborative projects is substantially simplified and trackable compared with other currently available software for annotation.

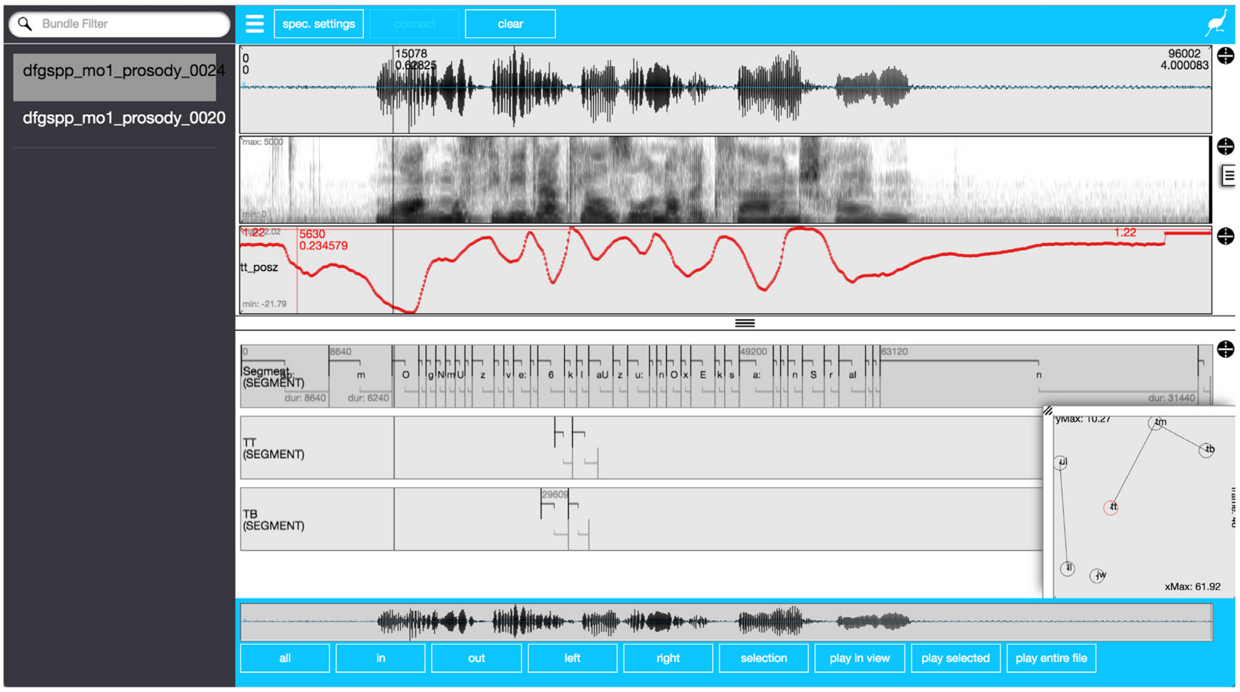


Fig. 7. Screenshot of `EMU-webApp` displaying EMA data as supplementary data in the form of contours (red track below spectrogram) and 2D display (bottom right-hand corner).

6. `emuR`: speech database management and analysis

As outlined in Section 1, the `emuR` package can be viewed as the main component of the EMU-SDMS as it is the only component that interacts with every other element of the system. This package provides the EMU-SDMS with database management, data extraction, data preparation and data visualization facilities. The database management routines provided by `emuR` allow the user to specify the permitted annotation levels, their linking relationship to other levels, the available signal files (e.g., pre-calculated formant values) as well as perform file handling duties. As the data preparation and data visualization routines of `emuR` are largely unaltered versions of the legacy system's functions, here we will focus on the new querying mechanics and underlying query engine. These allow complex structural queries of the symbolic information stored in the database's annotations while also de-referencing the respective time information. In addition, we describe how this symbolic time information can be used to extract the matching signal values (e.g., formant values).

Table 1
Main `EMU-webApp` protocol commands.

Protocol command	Comments
`GETPROTOCOL`	Check if the server implements the correct protocol
`GETDOUSERMANAGEMENT`	See if the server handles user management (if yes, then this prompts a login dialog → `LOGONUSER`)
`GETGLOBALDBCONFIG`	Request the configuration file for the current connection
`GETBUNDLELIST`	Request the list of available bundles for current connection
`GETBUNDLE`	Request data belonging to a specific bundle name
`SAVEBUNDLE`	Save data belonging to a specific bundle name

6.1. Query engine

It was essential that the newly developed `emuR` package had a query mechanism allowing users to query a database's annotations in a simple manner. The EMU Query Language (EQL) of the EMU-SDMS arose out of years of developing and improving upon the query language of the legacy system (Cassidy and Harrington, 2001; Harrington, 2010; John, 2012). As a result, today we have an expressive, powerful, yet simple, domain-specific query language. The EQL defines a user interface by allowing the user to formulate a formal language expression in the form of a query string. This query string results in a set of annotation items or, alternatively, a sequence of items of a single annotation level in the `emuDB` from which time information, if applicable, has been deduced from the time-bearing sub-level. An example of this would be a simple query that extracts all strong syllables from the "Syllable" level of annotations like those displayed in Fig. 9. The respective EQL query string `"Syllable == S"` results in a set of segments containing the label *"S"*. Due to the temporal inclusion constraint of the dominance relationship, the duration of the queried segments are derived from respective items of the "Phonetic" level, as this is the time-bearing sub-level.

The EQL user interface was retained from the legacy system because it has been shown to be sufficiently flexible and extensive to meet the query needs in most types of speech science research. The EQL parser implemented in `emuR` is based on the extended Backus-Nauer form (EBNF) (Garshol, 2003) formal language definition by John (2012) that defines the symbols and the relationship of those symbols on which this language is built. The slightly expanded Version 2 of the EQL, which was introduced with the `emuR` package, includes regular expression operands (= ~ and ! ~). These allow users to formulate regular expressions for more expressive and precise pattern matching of annotations. The `emuR` package comes with a formal description of the EQL including numerous hands-on examples in the form of an R package vignette. A minimal set of these examples displaying the new regular expression operands are shown in Table 2.

In accordance with other query languages, the EQL defines the user front-end interface and infers the query's results from its semantics. However, a query language does not define any data structures or specify how the query engine is to be implemented. As mentioned in Section 2, a major user requirement was database portability, simple package installation and a system that did not rely on external software at runtime. The only available back-end implementation that met those needs and was also available as an R package at the time was (R)SQLite (Hipp and Kennedy, 2007; Wickham et al., 2014). For this reason, `emuR`'s query system could not directly use the JSON files, i.e. the primary data sources of an `emuDB`, as described in Section 3.2. Instead, we implemented a syncing mechanism that maps the primary data sources to a relational form for querying purposes. This relational form is referred to as the emuDBcache in the context of an `emuDB`. The data sources are synchronized while an `emuDB` is being loaded and when changes are made to the annotation files. To address load time issues, we implemented a file check-sum mechanism which only reloads and synchronizes annotation files that have a changed MD5-sum (Rivest, 1992). Fig. 8 shows a schematic representation of how the various `emuDB` interaction procedures interact with either the file representation or the relational cache.

Despite the drawback of cache invalidation problems, there are several advantages to having an object relational mapping between the JSON based annotation structure of an `emuDB` and a relation table representation. One major advantage is that the user still has full access to the files within the folder structure of the `emuDB`. This means that external tools can be used to script, manipulate or simply interact with these files. This would not be the case if the files were stored in databases in a way that requires (semi-)advanced programming knowledge that might be beyond the capabilities of many users. Moreover, we can provide expert users with the possibility of using other relational database engines such as PostgreSQL, including all their performance tweaking abilities, as their relational cache.

Table 2
EQL V2: examples of simple and complex query strings using regular expression operands.

Query	Comments
`"Phonetic =~'[AIOUEV]'"`	A disjunction of annotations using a RegEx character class
`"Word =~a.*"`	All words beginning with *"a"*
`"Word !~.*st"`	All words not ending in *"st"*
`"[Phonetic == n ^ #Syllable =~.*]"`	All syllables that dominate an *"n"* segment of the Phonetic level

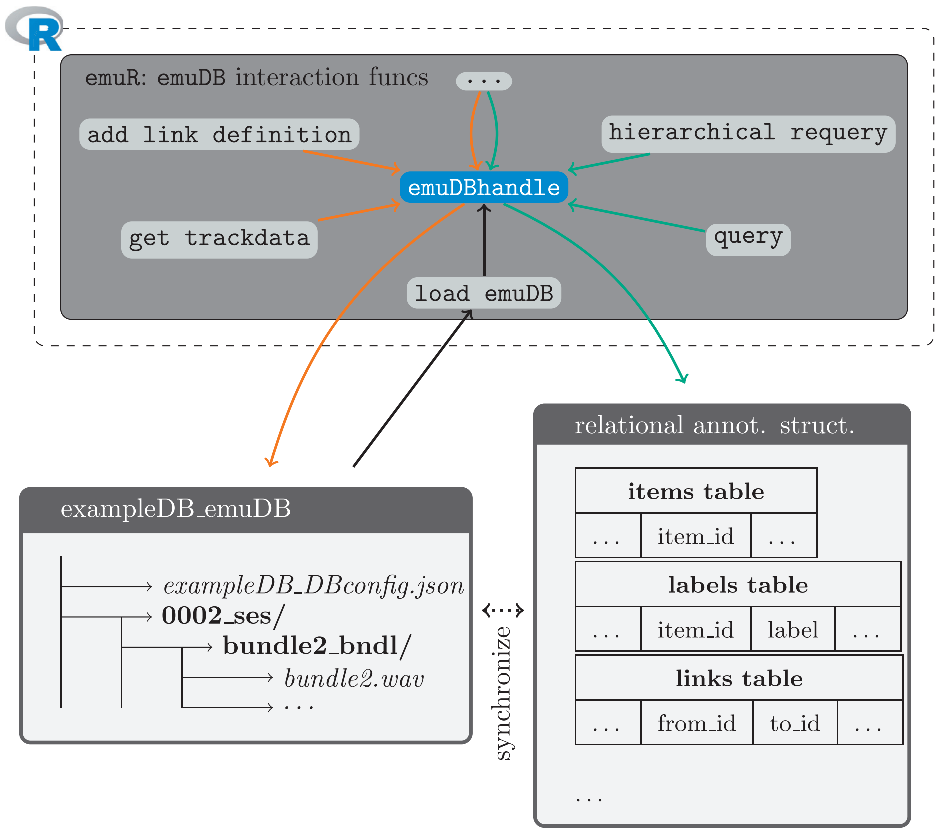
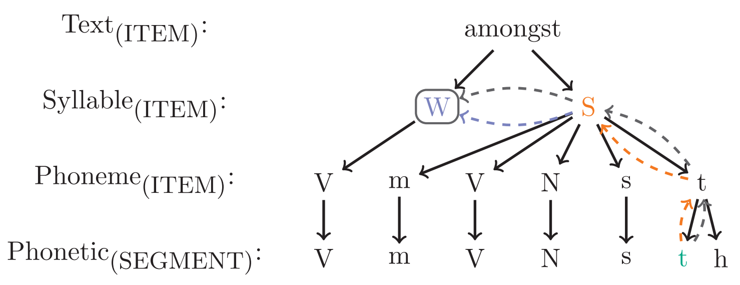


Fig. 8. Schematic architecture of `emuDB` interaction functions of the `emuR` package. *Orange* paths show procedures interacting with the files of the `emuDB`, while *green* paths show procedures accessing the relational annotation structure. Actions like saving a changed annotation using the `EMU-webApp` first save the `_annot.json` to disk then update the relational annotation structure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



```
sl1 = query emuDB with "Phonetic == t"
sl2 = hier. requery emuDB with sl1 and level = "Syllable"
sl3 = seq. requery with sl2 and offset = -1
vs.
sl3 = query emuDB with "[#Syllable =~ .* -> [Syllable =~
.* ^ Phonetic == t]]"
```

Fig. 9. Three step (query → **hierachical requery** → **sequential requery**) requery procedure, its single **query** counterpart and their color coded movements and intermediate results within the annotation hierarchy. The resulting “W” syllable item of both strategies is marked by a rounded-edged box.

This is especially valuable for handling large speech databases. It is worth noting that this separation into primary JSON data and a synchronized redundant database back-end also means that other types of database back-ends (e.g., back-ends of NoSQL variety) may easily be used in future, as changing the back-end does not affect the user interface/experience or the primary data source.

6.1.1. *Requery*

A popular feature of the legacy system was the ability to use the result of a query to perform an additional query, called a requery, starting from the resulting items of the first query. The requery functionality was used to move either sequentially (i.e. horizontally) or hierarchically (i.e. vertically) through the hierarchical annotation structure. Although this feature technically does not extend the querying functionality (it is possible to formulate EQL queries that yield the same results as a query followed by 1: *n* requeries), requeries benefit the user by breaking down the task of formulating long query terms into multiple simpler queries. Compared with the legacy system, this feature is implemented in the `emuR` package in a more robust way, as unique item IDs are present in the result of a query, eliminating the need for searching the starting segments based on their time information. Examples of queries and their results within a hierarchical annotation based on a hierarchical and sequential requery as well as their EQL equivalents are illustrated in Fig. 9.

6.2. *Signal data extraction*

After querying the symbolic annotation structure and de-referencing its time information, the result is a set of items with associated time-stamps. It was necessary that the `emuR` package contain a mechanism for extracting signal data corresponding to this set of items. As illustrated in Section 4, `wrassp` provides the R ecosystem with signal data

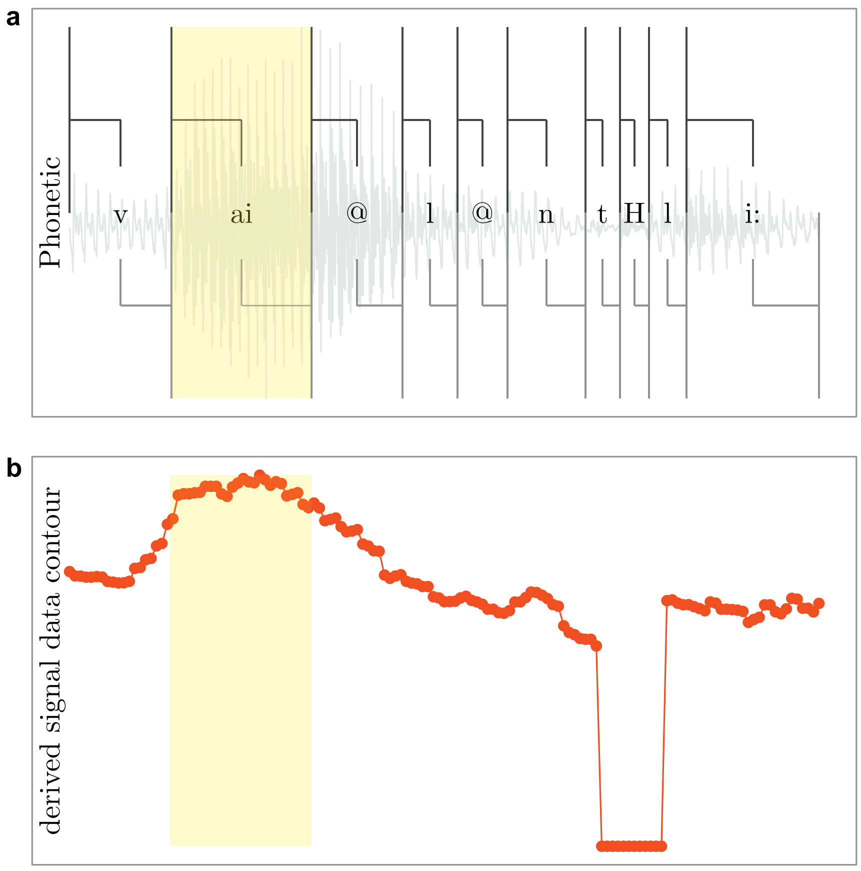


Fig. 10. Segment of speech with overlaid annotations and time parallel derived signal data contour.

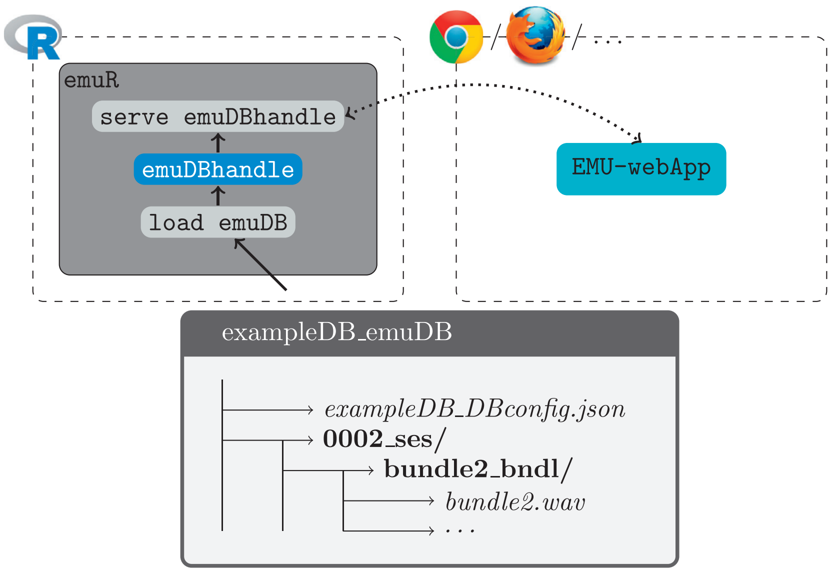


Fig. 11. Schematic of hosting a local emuDB to the EMU-webApp. Loading an emuDB produces an R object called emuDBhandle (marked in *blue*), which is used to reference the emuDB. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

file handling capabilities as well as numerous signal processing routines. emuR can use this functionality to either obtain pre-stored signal data or calculate derived signal data that corresponds to the result of a query. Fig. 10a shows a snippet of speech with overlaid annotations where the resulting SEGMENT of an example query (e.g., "Phonetic == ai")) is highlighted in yellow. Fig. 10B displays a time parallel derived signal data contour as would be returned by one of wrassp's file handling or signal processing routines. The yellow segment in Fig. 10b marks the corresponding samples that belong to the *"ai"* segment of Fig. 10a.

To access data that are stored in files, the user has to define tracks for a database that point to sequences of samples in files that match a user-specified file extension. The user-defined name of such a track can then be used to reference the track in the signal data extraction process. Internally, emuR uses wrassp to read the appropriate files from disk, extract the sample sequences that match the result of a query and return values to the user for further inspection and evaluation. Being able to access data that is stored in files is important for two main reasons. Firstly, it is possible to generate files using external programs such as VoiceSauce (Shue et al., 2011), which can export its calculated output to the general purpose SSFF file format. This file mechanism is also used to access data produced by EMA, EPG or any other form of signal data recordings. Secondly, it is possible to track, save and access manipulated data such as formant values that have been manually corrected.

With the wrassp package, we were able to implement a new form of signal data extraction which was not available in the legacy system. The user is now able to select one of the signal processing routines provided by wrassp and pass it on to the signal data extraction function. The signal data extraction function can then apply this wrassp function to each audio file as part of the signal data extraction process. This means that the user can manipulate quickly function parameters and evaluate the result without having to store the files that would usually be generated by the various parameter experiments to disk. In many cases this new functionality eliminates the need for defining a track definition for the entire database for temporary data analysis purposes.

6.3. *Serving* emuDBs *to the* EMU-webApp

To serve local emuDBs to the EMU-webApp, emuR supplies a websocket server that implements the EMU-webApp protocol described in Section 5.1. This server, which is available to the user as an R function, can be used to host loaded emuDBs from the user's R session to the web application. This enables the user to use the EMU-webApp for all of her/his annotation, visualization and correction needs. A schematic of how a local database is hosted to the EMU-

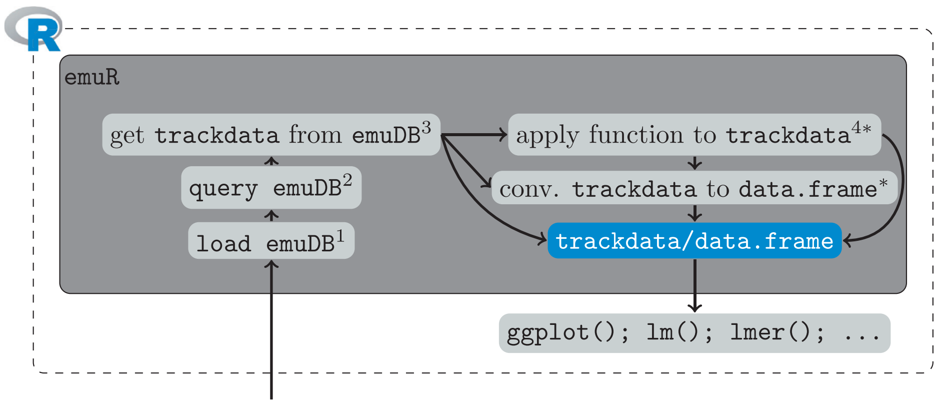

emuR
get trackdata from emuDB3
query emuDB2
load emuDB1
apply function to trackdata4*
conv. trackdata to data.frame*
trackdata/data.frame
ggplot(); lm(); lmer(); ...

Fig. 12. Schematic default workflow that outlines the initial steps in the analysis of an emuDB. Note that boxes marked with an asterisk (*) are considered optional steps.

webApp is depicted in Fig. 11. After loading a local emuDB, the user can simply invoke an R function in the current R session. This will automatically open the system's default browser with the URL of the EMU-webApp and automatically connect the client to the server.

7. Discussion

The emuDB format, EMU-webApp, wrassp and emuR together form the new EMU-SDMS. The combination of these components provides the user with an integrated system of tools with which to analyze spoken language research questions such as the question stated in Section 1: *Do the phonetic segments annotated /s/, /z/, /S/ or /Z/ (sibilants) in a specific spoken language database differ with respect to their first spectral moment?*. To answer these types of questions, the user initially creates a new, empty emuDB, loads it into the current R session (see superscript 1 in Fig. 12) and defines its annotation structure. After importing the audio files into the new emuDB, the files can be hosted to the EMU-webApp for annotation. Alternatively, conversion routines for existing annotation collections (e.g., WAV and TextGrid collections created in Praat) are available. Once the annotation process is completed, the database maintainer can query the relevant level for the target segments (e.g., the sibilant segments; see superscript 2 in Fig. 12). The result of a query can then be used to calculate and extract the relevant sample information (e.g., the DFT spectrum values; see superscript 3 in Fig. 12) and post-process the segments with helper functions provided by emuR (e.g., calculate the average first spectral moment trajectories for each sibilant; see superscript 4 in Fig. 12). The plethora of statistics and visualization packages that the R ecosystem provides can then be used for graphical and statistical analysis, as the resulting data can be provided in the common data.frame formant. The only prerequisite for such analyses is familiarity with the R platform which is typically the case in the speech sciences. Fig. 12 outlines the steps of the default workflow that are usually the initial steps for such an analysis, provided a annotated emuDB is available.

Although it is possible to carry out the above process using other tools that lack the explicit concept of a collection or database (e.g., Praat and ELAN), we feel it is a more cumbersome and error-prone process. With these tools, the user initially defines an abstract annotation structure for the data at hand. Due to limitations in modeling annotation structure, the user is often forced to choose multiple segment tiers to be able to relate implicitly items from multiple tiers to each other (e.g., "Words" containing "Syllables" containing "Phonemes"). During the annotation process, the database maintainer has to ensure that each annotation file adheres to the annotation structure, as these tools generally lack the (semi-)automatic checking routines necessary to ensure the integrity of an annotation structure (e.g., that the same number of correctly named tiers are present in all annotation files). Following annotation, to extract the relevant annotational items the researcher is faced with the challenge of scripting extraction routines to retrieve things such as: *all three-syllable words (including start and end times) that also precede the word "the" and which contain a schwa in the first syllable*. Solving these sometimes fuzzy (due to possibly misaligned boundaries) search problems in a multi-file and multi-tiered search space requires advanced programming knowledge that is beyond the capabilities of many users. Although some tools offer the ability to perform tier-based search operations (e.g., ELAN), they do not match the expressive power of the query mechanism that the EMU-SDMS provides. Once the relevant annotation items and their time-stamps have been retrieved, they are usually stored as comma separated value files. This is followed by a further procedure that extracts the relevant signal information (e.g., fundamental frequencies) which once again can require advanced programming as well as signal processing knowledge. Once analyzable data has been produced, the user usually switches into a statistical programming environment such as R for further analysis and statistical processing. The EMU-SDMS is an integrated system that is able to replace this cumbersome multi-tool and multi-environment process by moving it entirely into the R platform.

The EMU system overcomes the annotation structure modeling limitations of other systems by providing a general purpose modeling system which is built on items (i.e. nodes) containing annotation information which are sequenced and grouped together to form levels. These annotational items can then be linked to each other to form a complete hierarchical annotation. This graph-oriented type of annotation structure modeling is flexible enough to handle most modeling scenarios including, for example, the often complex modeling of dialog data. Given a database containing recordings of a two speaker dialog, for example, one could create two sets of partial hierarchies that annotate speaker A (e.g., “Phonetic-A”, “Word-A”) and speaker B (e.g., “Phonetic-B”, “Word-B”). Placing a fifth level above the two partial hierarchies (e.g., “Turn”), could connect the two partial hierarchies and thereby facilitate more powerful queries. An common solution is to add a further generic top-level (e.g., “Bundle”) containing only a single annotation item. This resolves a shortcoming of the `emuDB` format which is that it does not have an explicit method of storing speaker or other metadata. This single annotation item, which can be viewed as the root item of a bundle’s hierarchy, can be used to store the meta information of that bundle using EMU’s parallel annotations feature. It is also noteworthy that meta information stored in the form of coded bundle and session names, which is typical of existing annotation collections, may already be used at query time to filter by session or by bundle name. To ensure a cleaner method for storing meta information, in future we might add the possibility of placing such data into key-value type JSON files in either the bundle, the session or the root folder of a `emuDB`. This metadata can then be used at query time to filter sessions and bundles.

Although the query mechanic of the EMU-SDMS covers most linguistic annotation query needs (including coocurrence and dominance relationship child position queries), it has its limitations due to its domain specific nature, its desire to be as simple as possible and its predefined result type. Performing more general queries such as: *what is the average age of the male speakers in the database that are taller than 1.8 meters?* is not directly possible using the EQL. Even if the gender, height and age parameters are available as part of the databases annotations (e.g., using the bundle root item strategy described above) they would be encoded as strings, which does not permit direct calculations or numerical comparisons. However, it would be possible to answer these types of questions using a multi-step approach. One could, for example, extract all height items and convert the strings into numbers to filter the items containing a label that is greater than 1.8. These filtered items could then be used to perform two requeries to extract all male speakers and extract their age labels. These age labels could once again be converted intro numbers to calculate their average. Although not as elegant as other languages, we have found that most questions that arise as part of studies working with spoken language database can be answered using such a multi-step process including some data manipulation in R, provided the necessary information is encoded in the database. Additionally, we feel that from the viewpoint of a speech scientist, the intuitiveness of an EQL expression (e.g., a query to extract the sibilant items for the question asked in the introduction: `"Phonetic == s|z|S|Z"`) exceeds that of a comparable general purpose query language (e.g. a semantically similar SQL statement: `SELECT desired_columns FROM items AS i, labels AS l WHERE i.unique_bundle_item_id = l.uniq_bundle_item_id AND l.label = ’s’ OR l.label = ’z’ OR l.label = ’s’ OR l.label = ’S’ OR l.label = ’Z’`). This difference becomes even more apparent with more complex EQL statements, which can have very long, complicated and sometimes multi-expression SQL counterparts.

A problem which the EMU-SDMS does not explicitly address is the problem of cross-corpus searches. Multiple `emuDBs` will very likely have varying annotation structures with often varying semantics regarding the names or labels given to objects or annotation items in the databases. This means that it is very likely that a complex query formulated for a certain `emuDB` will fail when used to query other databases. If, however the user either finds a query that works on every `emuDB` or adapts the query to extract the items she/he is interested in, a cross-corpus comparison is simple. As the result of a query and the corresponding data extraction routine are the same, regardless of database they where extracted from, these results are easily comparable. However, it is worth noting that the EMU-SDMS is completely indifferent to the semantics of labels and level names, which means it is the user’s responsibility to check if a comparison between databases is feasible (e.g., *are all segments containing the label "@" of the level "Phonetic" in all* `emuDB`*s annotating the same type of phoneme?*).

Each speech database system mentioned in the introduction (LaBB-CAT, Phon and SCT), including the EMU-SDMS, strives to reduce the complexity of working with speech databases and each has its own and sometimes overlapping specialties, merits and target audiences. Of these systems, the SCT are especially interesting for the scalability of databases, as graph databases are known for their search performance benefits on large, linked data sets. The SCT offer powerful expressive query mechanisms comparable to those of the EMU-SDMS. In future, depending on user requirements regarding scalability, the EMU-SDMS might also consider implementing the emuDBcache representation in the form of a graph database or offering it as an alternative caching mechanism. One goal during the development of the EMU-SDMS was to try to avoid any installation complexity, as we have found this to be a major issue for the users of the legacy system, which relied on multiple components that had to be configured to work together. However, reducing the installation complexity meant solely focusing on the R user community as our target audience. Further, we decided not to use a database back-end as our primary data source, as we wanted to have scriptable text-based annotation files that met the annotation structure modeling needs of the EMU-SDMS (see Section 3) and could be produced and manipulated by other tools. We plan to introduce further export and import routines in order to move data sets between the various database-oriented tools. However, our primary focus is on improving EMU-SDMS as a self-contained system.

In this paper, we presented the new EMU-SDMS. This system offers an all-in-one speech database management solution that is centered around the R language and environment for statistical computing and graphics. It enriches the R platform by providing specialized speech signal processing, speech database management, data extraction and speech annotation capabilities. By doing so and not relying on any external software sources, except the web browser, the EMU-SDMS significantly reduces the number of tools and steps the speech and spoken language researcher has to deal with and helps to simplify answering research questions related to speech and spoken language.

Acknowledgments

We would like to thank the international team of past and present EMU developers and consultants for their suggestions and advice as well as the EMU users for their feedback, usability tests and feature requests. Without their critical input this project would not have been possible. Research supported by the European Research Council Grant (295573) to Jonathan Harrington.

References

Abercombie, D., 1967. Elements of General Phonetics. Aldine Publication Company.
Bird, S., 2006. NLTK: the natural language toolkit. In: Proceedings of the COLING/ACL on Interactive presentation sessions. Association for Computational Linguistics, pp. 69–72.
Bird, S., Liberman, M., 2001. A formal framework for linguistic annotation. Speech Commun. 33 (1), 23–60.
Boersma, P., Weenink, D., 2016. Praat: doing phonetics by computer (Version 6.0.19). http://www.fon.hum.uva.nl/praat/.
Bombien, L., Cassidy, S., Harrington, J., John, T., Palethorpe, S., 2006. Recent developments in the Emu speech database system. In: Proceedings of the 11th SST Conference. Auckland, pp. 313–316.
Cassidy, S., 2013. Chapter 9. Simple signal file format. In: The Emu Speech Database System Manualhttp://emu.sourceforge.net/manual/chap.ssff.html.
Cassidy, S., Harrington, J., 1996. Emu: an enhanced hierarchical speech data management system. In: Proceedings of the Sixth Australian International Conference on Speech Science and Technology, pp. 361–366.
Cassidy, S., Harrington, J., 2001. Multi-level annotation in the Emu speech database management system. Speech Commun. 33 (1), 61–77.
Coleman, J., Local, J., 1991. The "no crossing constraint" in autosegmental phonology. Linguist. Philos. 14 (3), 295–338.
Conway, J., Eddelbuettel, D., Nishiyama, T., Prayaga, S. K., Tiffin, N., 2016. RPostgreSQL: R interface to the PostgreSQL database system. https://CRAN.R-project.org/package=RPostgreSQL R package version 0.4−1.
The Apache DB Project, 2016. Apache Derby. https://db.apache.org/derby/.
Draxler, C., Jänsch, K., 2004. SpeechRecorder−a universal platform independent multi-channel audio recording software. In: Proceedings of the IV. International Conference on Language Resources and Evaluation. Lisbon, Portugal, pp. 559–562.
Fromont, R., Hay, J., 2012. LaBB-CAT: an annotation store. Australasian Language Technology Association Workshop 2012, vol. 113. Citeseer.
Garshol, L.M., 2003. BNF and EBNF: What are They and How do They Work, 16. acedida pela última vez em.
Harrington, J., 2010. Phonetic Analysis of Speech Corpora. John Wiley & Sons.

Harrington, J., Cassidy, S., Fletcher, J., Mc Veigh, A., 1993. The mu+ system for corpus based speech research. Comput. Speech Lang. 7 (4), 305–331.

Hipp, D. R., Kennedy, D., 2007. Sqlite. https://www.sqlite.org/.

Ide, N., Romary, L., 2004. International standard for a linguistic annotation framework. Nat. Lang. Eng. 10 (3−4), 211–225.

ISO, 2012. Language Resource Management — Linguistic Annotation Framework (LAF). ISO, 24612:2012. International Organization for Standardization, Geneva, Switzerland.

John, T., 2012. Emu Speech Database System. Ludwig Maximilian University of Munich Ph.D. thesis.

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

Knuth, D. E., 1968. The Art of Computer Programming vol. 1, Fundamental Algorithms. Addison-Wesley, Reading, MA 9, 364−369.

MacWhinney, B., 2000. The CHILDES Project: The Database, vol. 2. Psychology Press.

MacWhinney, B., 2007. The talkbank project. Creating and Digitizing Language Corpora. Springer, pp. 163–180.

McAuliffe, M., Sonderegger, M., 2016. Speech Corpus Tools (SCT). http://speech-corpus-tools.readthedocs.io/.

Neo4J, 2016. Neo4J. https://neo4j.com/.

Ooms, J., 2014. The jsonlite package: a practical and consistent mapping between JSON data and R objects. arXiv:1403.2805[stat.CO].

Ousterhout, J.K., Jones, K., 2009. Tcl and the Tk Toolkit. Pearson Education.

R Core Team, 2016. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

R Special Interest Group on Databases (R-SIG-DB), Wickham, H., Müller, K., 2016. DBI: R database interface. https://CRAN.R-project.org/package=DBI R package version 0.4.

Rivest, R., 1992. The md5 message-digest algorithm. https://tools.ietf.org/html/rfc1321.

Rose, Y., MacWhinney, B., 2014. The PhonBank project: data and software-assisted met. http://phonbank.talkbank.org/.

Rose, Y., MacWhinney, B., Byrne, R., Hedlund, G., Maddocks, K., O'Brien, P., Wareham, T., 2006. Introducing phon: a software solution for the study of phonological acquisition. In: Proceedings of the Annual Boston University Conference on Language Development. Boston University Conference on Language Development, vol.2006. NIH Public Access, p. 489.

RStudio, Inc., 2015. httpuv: HTTP and WebSocket server library. https://CRAN.R-project.org/package=httpuv R package version 1.3.3.

Schäfer-Vincent, K., 1983. Pitch period detection and chaining: method and evaluation. Phonetica 40 (3), 177–202.

Scheffers, M., Bombien, L., 2012. libassp... advanced speech signal processor. http://libassp.sourceforge.net/.

Shue, Y.-L., P., K., C., V., K., Y., 2011. VoiceSauce: a program for voice analysis. In: Proceedings of the ICPhS, vol. XVII, pp. 1846–1849.

Sjölander, K., Beskow, J., 2000. Wavesurfer−a open source speech tool. Interspeech, pp. 464–467.

W3C, 2016. Offline apps. http://www.w3.org/TR/2011/WD-html5-20110525/offline.html.

Wickham, H., James, D. A., Falcon, S., 2014. RSQLite: SQLite interface for R. https://CRAN.R-project.org/package=RSQLite R package version 1.0.0.

Winkelmann, R., Bombien, L., Scheffers, M., 2015. wrassp: an R wrapper to the ASSP library. http://cran.r-project.org/package=wrassp.

Winkelmann, R., Raess, G., 2015. EMU-webApp. http://ips-lmu.github.io/EMU-webApp/.

Wittenburg, P., Brugman, H., Russel, A., Klassmann, A., Sloetjes, H., 2006. Elan: a professional framework for multimodality research. In: Proceedings of LREC, vol. 2006, .

Zipser, F., Romary, L., 2010. A model oriented approach to the mapping of annotation formats using standards.. Workshop on Language Resource and Language Technology Standards, LREC 2010. La Valette, Malta.