TU Graz

Kurzfassung

Das manuelle Absuchen eines zeitabhängigen Signals nach Artefakten, kann sehr zeitaufwendig und umständlich sein. Diese Arbeit versucht diesen Prozess einfacher zu gestalten. Dieses Ziel soll durch eine Gruppe von kollaborierenden Benutzern erreicht werden. Außerdem soll es möglich sein aus dem Wissen anderer Mitarbeiter zu profitieren und daraus zu lernen.
Dazu wurde ein experimentelles Web-Service namens “SignalCloud” implementiert, welches das gleichzeitige Arbeiten an einer Signal-Datei erlaubt. Zusätzlich werden den Benutzern verschiedenste Werkzeuge und Visualisierungen zur Verfügung gestellt um den Prozess einfacher zu gestalten.
Derzeit gibt es zwei Einsatzgebiete für das Service, aufgrund der unterstützten Dateiformate: aufgezeichnete Biosignale und Sensor-Daten aus der Automobilindustrie

Schlüsselwörter: Zusammenarbeit, Visualisierung, Sensor-Daten, manuelles Erfassen, Signal, Web-Service, SignalCloud, GDF, EEG, MoTeC

Abstract

The manual detection of artifacts in time-dependent data is a very timeconsuming and tedious task. This work tries to simplify this process, by using a collaborative approach. Additionally, it should be possible to benefit from the knowledge of other collaborators and learn from them.
For this reason, an experimental web-service called “SignalCloud” has been implemented, which allows simultaneous working on a single signal-file. This web-service provides various tools and visualizations to facilitate the process. Currently there are two usage-domains of this service due to its supported file formats: recorded biosignals and sensor-data from the automobile industry

Keywords: Collaboration, Visualization, sensor-data, manual detection, signal, web-service, SignalCloud, GDF, EEG, MoTeC

Acknowledgment

Firstly, I would like to thank my thesis advisors Eduardo E. Veas, Dr.techn. MSc and Cecilia Sciascio, MSc for their continuous support. The meetings on a weekly basis allowed this project to constantly face into the right direction.

Additionally, I want to thank my sister Maria Legat-Rath and my brother in law Klaus Legat for reading this paper and providing me with tips and phrases for writing an appropriate scientific thesis.

Last but not least, I would like to express my sincere gratitude to my girlfriend Anna Christian and my family for their ongoing encouragement and supporting me up during bad times.

Graz, February 21, 2017 Lukas Rath

Contents

List of Figures

“*Coming together is a beginning, staying together is progress, and working together is success.*”

— **Henry Ford**

Chapter 1

Introduction

Analyzing time-dependent data is a ubiquitous task in science. Sometimes algorithms are capable of doing the hard work, but in many cases manual analysis is required. For example, when the existing algorithm is not accurate enough, or during the development of an algorithm.. Anyway, manual detection can be a very time-consuming and tedious task. [Lawhern et al., 2012]
There are many scenarios that require analyzing time-dependent data. The focus of this work lies on manual artifact detection, done by multiple collaborators.
To make one of these artifacts available for further processing or a more precise analysis, it has to be located first at least coarsely and labeled. Another common term for the labeling process is classification.

1.1 Terminology

Before we enter the subject, we first have to discuss some important terms to understand all aspects.
The term electroencephalogram or its abbreviation EEG will be mentioned several times in the further course of the document. It is defined as “the graphic record of electrical disturbances arising the brain.” [Gibbs et al., 1968, p. 59]. This record is similar to the electrocardiogram or ECG, which is a recording of the electrical activity associated with the contraction of the heart.
Additionally, there is a variety of terms for annotations of certain artifacts in various signals in use. In most cases they are referred by the term “events” due to the terminology of the GDF file format[1]. A specific instance of a certain event-type (e.g. an eye blink) is referred to as an “occurrence”.

[1] a file format for storing biosignals

An artifact within an EEG-signal is defined as “a structure or substance not normally present, but produced by an external agent or action [. . .]” [American Heritage, 2008, p. 45]. Some graphic examples of artifacts in recorded EEG-signals can be seen in Figure 1.1.

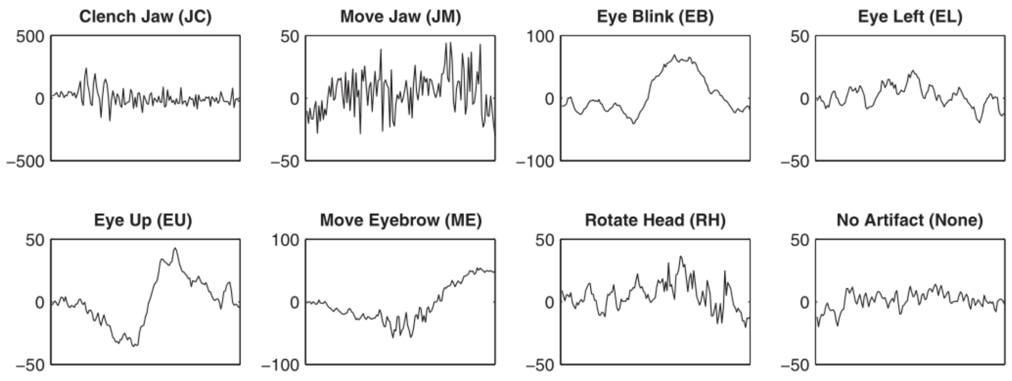


Figure 1.1: Examples for EEG-artifacts, recorded at channel Cz, from [Lawhern et al., 2012, p. 183]

1.2 Issues

As mentioned above, manual detection of artifacts in data is a tedious work. However there are more issues, which emerge from this task.

- **Reliability**
  Manual Detection is done by individuals, which brings the “human factor” into the working process. Naturally, this can be an advantage and a disadvantage at the same time. In any case the quality of the output will be different with each individual and maybe even with consecutive attempts of the same individual. An algorithm will always produce an output with more or less the same quality in contrast to the manual detection.

- **Knowledge**
  One key factor for a reliable output is the knowledge and experience of the individual. As one can imagine, artifact detection has to be learned before it can be carried out by a person. Not only is it necessary to know about artifact detection in general, one must have specific knowledge about the data, the subject and the wanted artifacts.

- **Time-Consumption**
  Even if the required knowledge is present and the person is an expert,

the task remains a complex one. The contestant has to browse through the whole signal, probably multiple times, to produce a meaningful result. Furthermore, if the objective is to provide samples for an algorithm, it is not enough to analyze only one signal file, but rather many of the same kind.

These are just some examples to illustrate the complexity of the given task.

1.3 Motivation

The main goal of this work is to tackle the above mentioned issues and improve the quality of manually labeled data through a collaborative approach. By means of a "groupware" new possibilities arise to make this task a less tedious one. [Gutwin and Greenberg, 2002]
Let us discuss how issues can be solved, by using a collaborative approach.

- **Reliability**
  The quality of separate outputs of all individuals might be as high as in the non-collaborative approach, but there are multiple opinions on which participants can agree upon. This work incorporates consensus features for groups of experts to discuss and agree upon their input, and thus create a more reliable output.

- **Knowledge**
  Assuming the presence of at least one expert or individual with a deep knowledge of the process, other persons can watch this process and learn by example. This work supports collaborative workflows whereby newbies can observe the activity of an expert, they can pose questions and discuss.

- **Time-Consumption**
  Through a collaborative approach this task can be divided in smaller subtasks. For example, collaborators could divide the signal file evenly to speed up labeling and add a checking round where they cross-check their choices. There is a trade-off between faster turn around time and reliability. The latter implies that each expert needs to check and label the complete signal and the latter that each expert only observes a portion of it. The present work takes this trade-off into account and offers methods choose the desired strategy. However, this argument is in conflict with the reliability problem, in which case a final round of assessment might be necessary.

These are just a few examples, a collaborative approach could bring more benefits that we have not thought about so far.

1.4 Aspects of Collaboration

To understand, what collaboration is all about, we have to clarify the two important aspects or sub-types of collaboration.
On the one hand, there is synchronous collaboration where everything happens simultaneously. On the other hand, we have asynchronous collaboration, which happens indirectly via tools or the environment itself. [Weyns and Holvoet, 2003] [Vernon, 2014]

1.4.1 Synchronous Collaboration

Synchronous collaboration resembles a conversation in realtime, a good example for this kind of collaboration is a chat. The obvious benefit is the immediate tackling of problems as they appear. So in many cases synchronous collaboration is much more effective than asynchronous collaboration. However, there are some obvious disadvantages as well, for example other collaborators may not have enough time to conceive their actions which could lead to misinterpretations. [Vernon, 2014]

1.4.2 Asynchronous Collaboration

If collaborators work in an asynchronous manner, everyone has as much time as desired to handle a problem or task. Therefore an ubiquitous example is reading and writing emails. One could say it is happening outside of realtime. Obviously this might not be the best type of collaboration for urgent issues. [Vernon, 2014]

1.5 Summary

In theory a collaborative approach could bring many improvements to to the task of signal labeling and analysis. However, collaboration also brings challenges previously unexplored in this domain. For example, how should different opinions from two or more collaborators on the duration of a particular event be represented. How should conflicts be addressed? Furthermore, implementation of a collaborative tool brings a number of technical challenges, from the synchronization of events and activities, to the strategy for data exchange. This thesis develops a collaborative tool to identify articats in signals and deals with said challenges.

Chapter 2

Related Work

This work builds upon existing solutions for signal analysis to create a collaborative tool. We look first at existing state-of-the-art signal visualization / analysis tools. Furthermore, we consider the field of visualization, in particular with respect to time-dependent data. Finally, we review the theories of collaboration upon which we base the design of our tool.

2.1 State-of-the-art Tools

Many tools which offer a variety of features for adding labels to artifacts of time-dependent signals. We want to examine two of them more closely, because of their high significance for this project. From a performance point of view, those tools are far more advanced than the one developed in this work. Nevertheless, they lack any type of realtime-synchronization or other collaborative features.

2.1.1 SigViewer

SigViewer is a visualization- and annotation-tool for biosignals which was developed at the Technical University of Graz. It supports various file formats, whereas the most important one for this application is the GDF-format. Moreover, the tool is an open-source program licensed under GPL[1]. It uses the cross-platform Qt 4 toolkit for its GUI. [Graz University of Technology, n.d.]
Many features of the tool developed in this work are inspired by SigViewer. For example the different cursor tools or modes which enable the user to draw events, select them, move the signal area by dragging or setting it up (although the last one was implemented differently in this work).

[1] a popular open-source license which gives much freedom to the public domain, more info at `https://www.gnu.org/licenses/gpl.html`

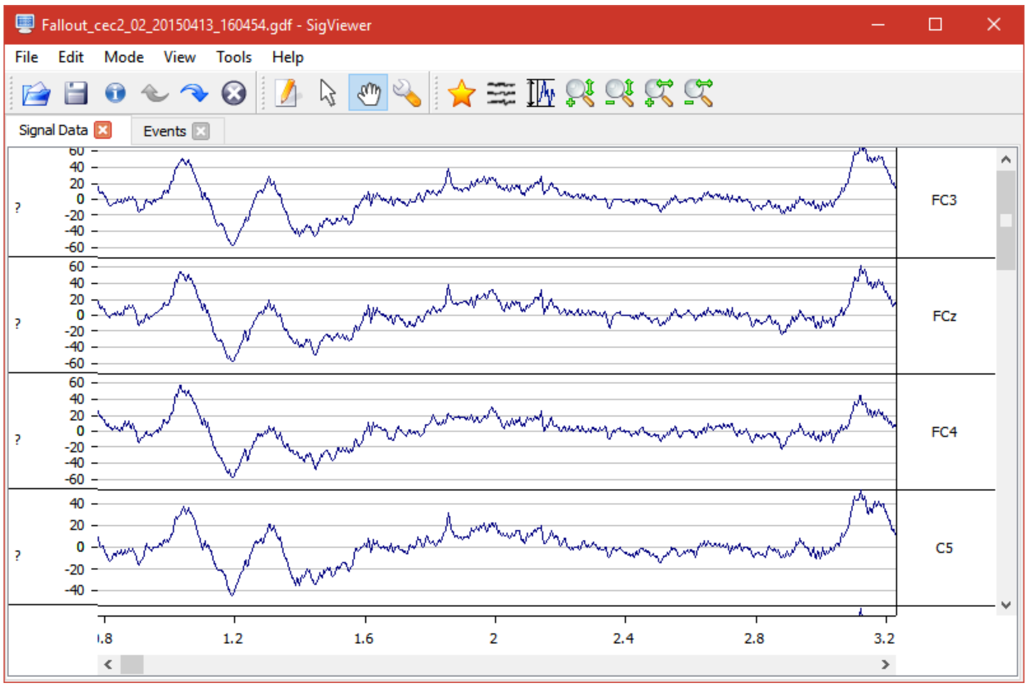


Figure 2.1: Screenshot of SigViewer - a visualization- and annotation-tool for biosignals

Biosignals have in most cases more than one channel. Figure 2.1 shows the interface of SigViewer after loading a recorded EEG stored in a GDF-file. All annotations of a file can be seen in the “Events”-tab of the SigViewer. They can be sorted by position, duration, channel or type. Moreover, the tool provides a lot of configuration commands in the “Edit Events”-mode, like bringing an event to all channels, changing its beginning, duration, type and channel as well as copying it to other channels.
In addition to the annotation-features it is possible to calculate the mean value or the power spectrum of a specific event-type for selected channels. The visualization appear as new tabs at the right side of the “Events”-tab.

2.1.2 MoTeC i2

MoTeC i2 is the affiliated data analysis software to the many different logging devices of the MoTeCcompany. It is already many years in development and is extremely specialized on its usage domain, the automobile industry. [MoTeC, n.d.]
The tool supports only the proprietary and binary MoTeC-log data format, with the file-ending “.ld”.

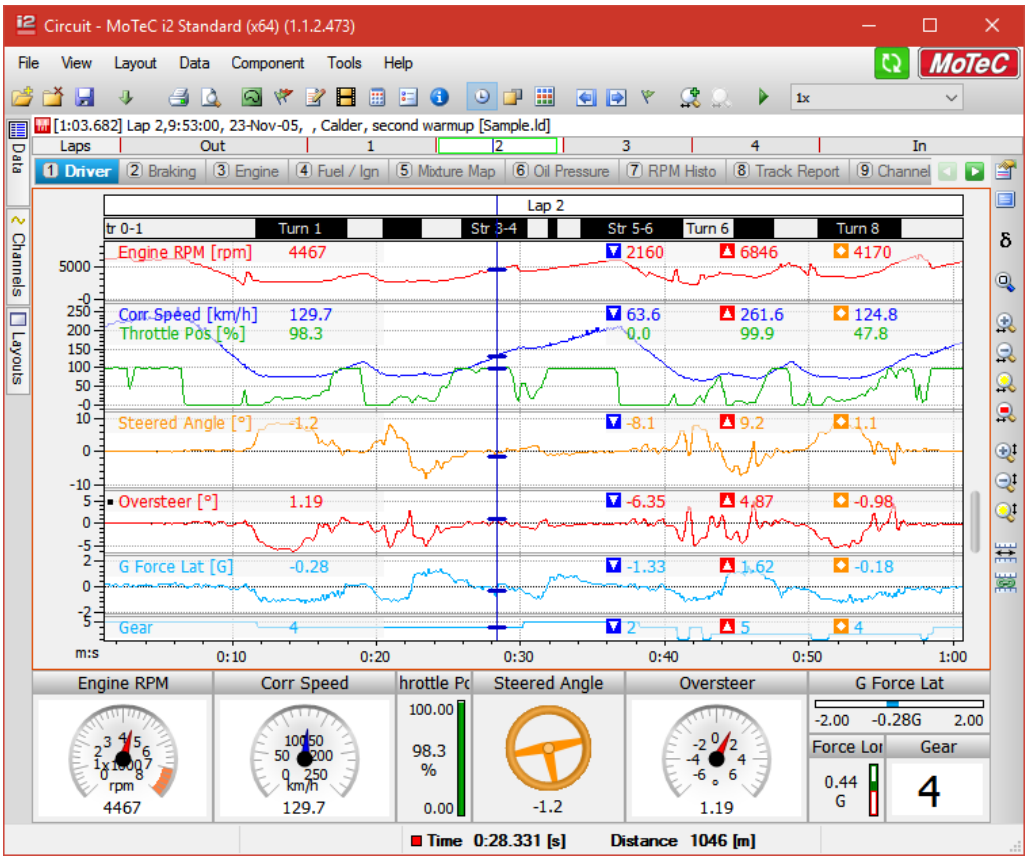


Figure 2.2: Screenshot of MoTeC i2 is a data analysis tool for car-racing applications

At first, a user has to select a workspace, which loads some predefined visualizations. The chosen preset in Figure 2.2 is the “Circuit” workspace. Furthermore, other visualizations of the same workspace can be shown by changing the tab at the top. This makes it for instance possible, to see the whole racing track, with color-encoded G-force plots or brake-pressure.
The most important feature of MoTeC i2, which is actually needed for this work, is the “export to CSV” functionality. These exports include either all recorded or only selected channels.

2.2 Client-generated Web-based Visualizations

Visualization methods based on client-side drawing often struggle with performance. Choosing the right approach as well as the correct tools is a difficult challenge.

Linked open data in sensor data mashups [Le-Phuoc and Hauswirth, 2009]

This paper describes the joined output of various sensor data sources in a web-based user interface or an API[2]. The data originates from different sensors for example weather stations, traffic sensors, webcams, etc. The data can be accessed through SPARQL or RESTful web services under the formats JSON, XML and RDF, after the mashup.
Each sensor has an URL pattern for streaming real-time data and historical data. This makes it possible to link them into a virtual RDF graph.
The interesting part for this thesis is that it is completely based on a web-service similar to this project.

Cytoscape Web: an interactive web-based network browser [Lopes et al., 2010]

This tool is a web-based network visualization tool. It is capable of showing complex networks and making them interactive. The tool is open source and is actively developed[3]. As it is web-based, it is easy to integrate its visualizations into web pages. Furthermore, it is possible to import data from different file formats.
The old version used Flex/Actionscript and provided a JavaScript-API, but the newer version is written in pure JavaScript.

2.3 Visualization of time-dependent data

This thesis is all about visualization of data in the temporal domain. Therefore, we evaluated some relevant visualization techniques in this domain. Many of these papers originate in the financial sector, as it is of great significance to learn from old data in this domain.

Importance-Driven Visualization Layouts for Large Time Series Data [Hao et al., 2005]

The generation of appropriate visualization layouts for large time series appears to be a hard task in many cases. By using the "degree of interestingness" (DOI) the authors of this paper try to tackle this problem. They attempt to perceive a partially or totally intrinsic importance- or interestingness-relation among the time series. The most important display properties for supporting

[2] a well defined interface for the communication between software-components
[3] more info at http://js.cytoscape.org/

the perception of importance-relations are according to the authors position and size.

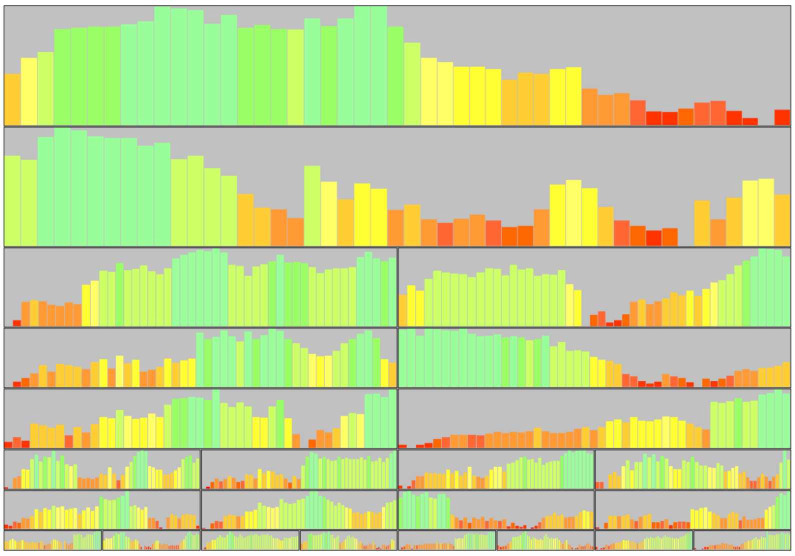


Figure 2.3: Importance driven layout of 24 stock price time series with favorable aspect ratios, from [Hao et al., 2005, p. 204]

Used visualization methods for time series are usually line- or bar-charts, which are typically accommodated by overlaying them in one common chart or using tabular (equal-sized) layouts. The authors propose an overlap-free and space-filling approach. A vital criterion of this approach is regularity, which consists of aspect ratio and the alignment of partitions of the chart. According to some experiments the authors argue that a low number of horizontal scales might be more important than a low number of unique vertical scales.
A conflict with their importance-driven layout could arise if hierarchical relationships are present among the time series. There are five constraints which are considered when generating a layout for unstructured sets of time series: size proportionality, space-filling and non-overlapping, weighted aspect ratio error, ordering and aspect ratio regularity. In a structured case, two additional constraints arise: rectangular containment and hierarchical ordering. It is not possible to generate an optimal layout that satisfies all constraints simultaneously.
The choice of a proper importance measure is vital for the generation of a layout. A set of basic measures are already implemented in the proposed software.
The “display masks” are schemes for partitioning subsets of the time series into a certain number of sub rectangles. These masks are then used by the

algorithm for the generation of the layout. If the used data set is hierarchically organized, a rooted tree is used for the ordering.
An example applications of the described algorithm is sale analysis. Dominant sales patterns can be compared by region or stock analysis, where the algorithm is used to provide an effective overview over sets of categorized stocks.

Multi-Resolution Techniques for Visual Exploration of Large Time-Series Data [Hao et al., 2007]

The current state of the art technology leads to enormous amounts of real-time data in many important application domains. An appropriate visualization for these large time series is needed but rather hard to achieve.
Basically, there are two options to address this problem. The first one is reducing the data size by sampling or aggregation which may lead to a loss in information. The second option is the improvement of the drawing methods to be more space efficient, which may lead to visualization metaphors not immediately familiar to the user.
In the cited work, the notion of the degree of interest (DOI) is used to generate adequate layouts of long time series. The DOI defines the distribution of interest over a dataset by distinguishing between areas of focus (high interest) and context (low interest). However, according to the authors, it is important to limit the number of distinct resolution levels, otherwise the perception of the amount of data density and the relative interest of the data partitions in context with the whole series could get lost.

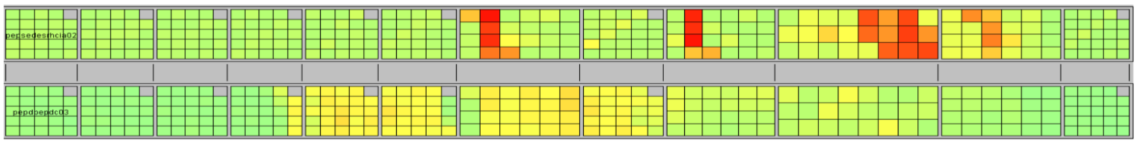


Figure 2.4: Example of the DOI visualization: CPU Monitoring, from [Hao et al., 2005, p. 32]

In their first example, the DOI is directly proportional to the age of data. An application of this concept is network monitoring where a less current observation is a less important one. To visualize this kind of monitoring, the color-coded matrix technique is used. The data density is incremented by putting increasingly more data into the same amount of display space. An example for this visualization can be seen in Figure 2.4.
The authors note that there exists an open problem of the animation of the visualization, due to the positional discontinuities.

Instead of deriving the DOI-function from time, it can also be derived from the data itself. Therefore the time series is partitioned into a number of N equal-width data bins, which reflect the importance for a certain time interval (presuming a given importance function). Optionally, the bins can be reduced by merging adjacent bins if they have similar interestingness scores. The resulting DOI-function can then be used to generate a multi-resolution layout, e.g. by using the color-coded matrix technique again.
There are two different instances of the described algorithm listed in the paper. In the first one, a certain atomic time slice is assumed which divides the layouts into intervals (data chunks). The other modes allow variable time slices which leads to a more compact representation of the interestingness profile but is more difficult to process visually.

Visualizing Frequent Patterns in Large Multivariate Time Series [Hao et al., 2012]

Sequences of frequent patterns, called motifs, are used to reveal trends, anomalies, etc. in large time series. They can assist users in hypothesis evaluation and knowledge discovery. Motifs can for example be visualized by rectangles with a specific length. They have a starting and ending time. Moreover, shorter motifs can be nested within longer motifs.

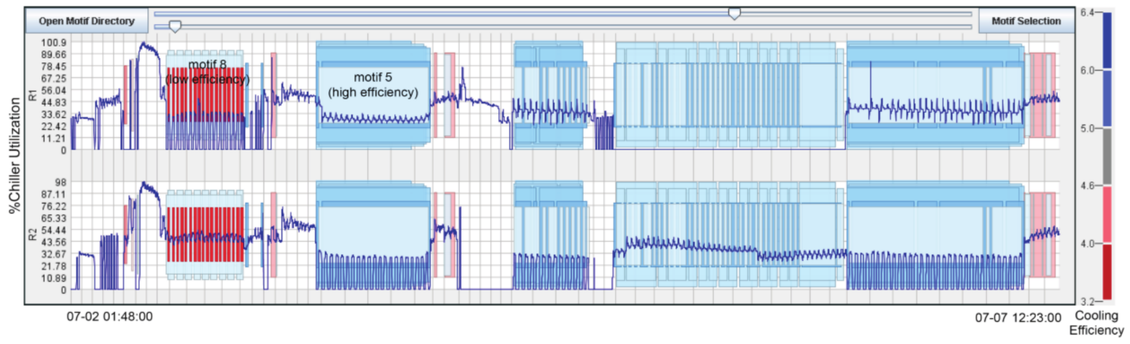


Figure 2.5: Frequent patterns (motifs) discovered in data chiller time series, from [Hao et al., 2012, p. 71]

The analysis and visualization of motifs involves a few challenges:

- The display of many potentially overlapping motifs associated with multivariate time series.
- The retrieval of the most efficient motifs and the analysis of their context and the motif itself for its root-cause.

The cited paper deals with the discovery, distortion and the merging of motifs. Each of the motifs can be linked to its performance coefficient (COP), which helps to identify the most efficient motifs.
The pattern finding can be decomposed into the following stages:

1. **Event Coding:** A k-means-clustering is performed considering time points as vectors and using the cluster labels as symbols to encode the time series. After this step, the multivariate series is now encoded as a single symbol sequence. The resulting sequence is analyzed to detect change points which transduces the stream into a sequence of events (a transition in the cluster labels).

2. **Motif discovery and mining:** A similar procedure to the previous algorithm is performed, which is a candidate generation followed by counting. The frequency is measured by non-overlapped counting (two motifs share no portion of the time series).

3. **Efficiency characterization:** It is difficult to determine the efficiency of two motifs by inspecting them visually. Hence the authors quantify the efficiency of all motifs.

For the visualization of the motif, a layout algorithm is used. It draws rectangles representing the occurrence of motifs. The color of the rectangle stands for the numerical importance. The height is used to show the statistical rank of the average duration.
The visualization task is very complex because of the large number, the nesting and the overlapping of motifs.
An interesting feature implemented by the authors is the motif distortion slider. By using it, the user is able to enlarge either areas with motifs or areas without motifs.
The motif merging slider is used to merge multiple occurrences of motifs into a single rectangle. This reduces data and the visual clutter.
There are many applications of the described systems, for example anomaly detection, prediction and clustering. The authors state that users get at least 87% time saving by applying motif layout, distortion and merging techniques, after carrying out an informal user study with 12 users.

Trajectory-Based Visual Analysis of Large Financial Time Series Data [Schreck et al., 2007]

This paper presents an unsupervised clustering algorithm combined with an appropriate data-visualization technique to extract useful information and knowledge from large data volumes. A fundamental task in financial analysis is the

discovery of market trends and the prediction of asset prices. The authors focused in the cited paper specifically on a system for visual analysis of 2-D time-dependent financial data sets.
These 2-D trajectories are input to an automatic clustering process, which is used to find prominent patterns in the data.
Two of the most important financial indicators are asset return and asset risk. These parameters vary usually over time and can therefore be observed by displaying trajectories in 2-dimensional indicator space.
For further analysis, the large amounts of data have to be reduced to smaller numbers of salient patterns which can be achieved by cluster analysis. This requires an appropriate similarity concept. More specifically, the task of the clustering algorithm is the reduction of a possible large set of weekly risk-return sequences to a smaller set of prototype trajectories. Each trajectory segment is obtained by the concatenation of a sequence of normalized coordinates. The normalization process implies an invariance with respect to position and scale.
The clustering process relies on the Self-Organizing Map (SOM) algorithm.
An efficient way to visualize a prototype trajectory alongside a few associated samples is the trajectory bundle visualization. It indicates the spatial distribution of many sample trajectories by changing the color density around the prototype trajectory.

The system consists of three main views:

- **Market View:** This view represents the distribution of chart movements of all assets over time. It is constructed by the visualizations of the trajectory bundles at the location of the prototype trajectories in the SOM grid.

- **Asset View:** This view is retrieved by simply restricting the set of sample trajectories to a selected asset. Based on this view a semi-automatic alert system could be established to notify analysts about an atypical pattern in near real time.

- **Sequence View:** It allows the comparison of assets over time and uses a row-by-column scheme, where rows refer to specific assets and columns refer to given weeks. The cells contain the prototype representation of the chart moment sample. For an easy spotting of prominent patterns a highlighting scheme is introduced. It assigns color saturation according to the similarity of the sequence pattern and the prominent pattern.

Visual Market Sector Analysis for Financial Time Series Data [Ziegler et al., 2010]

The financial market in all its complexity often lacks a certain transparency and hence does not allow a risk-free analysis. The authors of this paper try to tackle this problem by introducing two analysis techniques. Like in many cases, a good key to better understand the future is learning from the past. However, it is hard to process the sheer amounts of data updates, which are acquired just in seconds.
The most frequently used visualization is the line graph, but the main disadvantage is the low capacity of simultaneous time series, which can be observed.
A visualization technique which features already a small advantage in contrast to line graphs is a color coded bar. It displays the relative percentage change and the volatility coded by color. This visualization needs much less display-height and features the same information as a line graph. Therefore larger amounts of time series can be compared at the same time.

The second developed tool described by this paper is a clustering algorithm for similar time series trajectories and for analyzing their distribution among different market sectors.
To do so, it is necessary to normalize the time series data for comparison. After that a reduction of the data is unavoidable. The authors try to reduce the data to their main characteristics, which they call “Perceptually Important Points” (PIPs). To compute these points the Douglas-Peucker algorithm is used.

For the actual clustering, the k-means algorithm has been chosen. A few adjustments had to be made by the authors to apply the algorithm on the time series data based on PIPs.
The authors used Java for the implementation. A simple graphical user interface allows the user to easily select the desired market sectors for comparison and to adjust some other parameters, like for example the amount of clusters.
The last technique, described by the paper is the hierarchical low-resolution clustering which enables the avoidance of an automatic combination of two similar clusters. This is done by displaying a tree view, which can be pruned by the user at any height.

Visual Methods for Analyzing Time-Oriented Data [Aigner et al., 2008]

This paper describes time as a more special and unique dimension. There is already a wide repertoire for visualizing data with temporal dependencies. This is due to the fact that the parameter time is ubiquitous in many applications (business, medicine, planning, ...). The characteristics of the dataset and how it is connected to the time-dimension is important. For visualizations a few terminologies are of importance:

- **Linear time versus cyclic time:** Whereas linear time assumes a specific starting- and end-point , many natural processes have a cyclic aspect (e.g. the cycle of seasons).
  An example of a cyclic time visualization is a spiral-graph. However if the parameters are not set accordingly, periodical patterns may not be recognized by viewers as requested. This can be avoided by using analytical methods to detect patterns or let the user choose the parameter settings.
- **Time points versus time intervals:** A time point has no duration, but in case of an interval limited by two points in time, data elements are defined for a certain duration.
  A good visualization technique for showing time points is the "time-wheel". However it can not represent intervals accordingly. For showing them another visualization called "Planning Lines" can be used.
- **Ordered time versus branching time versus time with multiple perspectives:** If things happen one after another the situation may be considered ordered. If there is for example some kind of decision-making it will lead to multiple possible time-branches. However, a dataset can contain more than one perspective on the same time-branche.

The theme river is a visualization to show the number of occurrences for particular news topics on print media. Therefore it can only be used to show datasets, without time branches. The authors give a short introduction in a method called temporal data abstraction. This technique is used to analyze huge volumes of continuously assessed data. This is achieved by deriving qualitative values or patterns from the original dataset. They distinguish between basic temporal abstraction (e.g. state, gradient or rate) and more complex temporal abstraction. There are three important steps of data abstraction: eliminating data errors, clarifying the curve and qualifying the curve.
To reduce the number of variables and detect structure in multivariate data sets, the authors present the Principal Component Analysis (PCA). The PCA method is another approach to data abstraction which results in a transformation of the original data into the principal component space. It is preferable to exclude the variable "time" from the PCA, because otherwise the temporal context could get lost. As an example of the PCA approach they show climate research data. PCA is used to detect deviant hot or cold summers over a time

period. By means of the PCA users can get an abstracted view on the data very easily.
A widely used technique for data aggregation is clustering. Unlike in the PCA, time is typically included in the process of clustering. The Cluster Calendar View is used as an example of data aggregation by clustering. The cluster affiliation is hereby represented by color coding. It facilitates the comparison of cluster representatives (overview), exploration of single cluster representatives (abstract detail), and exploration of interest (specific details). The last major point of discussion in the paper is the user-centered analysis via events. However, visualization parameters are not easy to set in many cases. Therefore, event-based visualization is introduced. The workflow of this type of visualizations is the following:
At first the user has to specify interests as event types. This can be done by either directly typing the event-formula, setting the parameter “threshold” and “variable” or choosing from predefined event types.
Secondly, the interests have to be detected in the data (event instances). The last step of the workflow is a consideration of the detected event instances in the visualizations (event representation).

A Visual Analytics System for Financial Time-Series Data [Lei and Zhang, 2010]

The analysis of time-series data plays an important role for making lucrative investments. This paper discusses the drawbacks of current visual charts and gives an overview of new concepts. The main goal of technical analysis is the identification of market trends. There are many different factors which influence the stock prices. They can fluctuate for example because of economic crises, changes in interest rates, seasonal financial reports and many other unknown effects. The introduced system of this paper is designed from three different perspectives - knowledge discovery, trading patterns and information visualization.
One approach for the visualization of stock prices is clustering. The authors use a scatter plot for displaying stock prices and their change in percentage. Time is coded within the color shading of each dot.

An important factor used by investors for stock profiling is the volatility measure of a stock. In technical analysis it is measured by the Average True Range. It can show the activity level of a stock. To visualize it, another scatter plot with color shading was used.
In order to show trading patterns, the authors use a candlestick visualization. Another visualization technique described in this paper is a ring-map showing market information and stock performances. The ringmap is divided in an in-

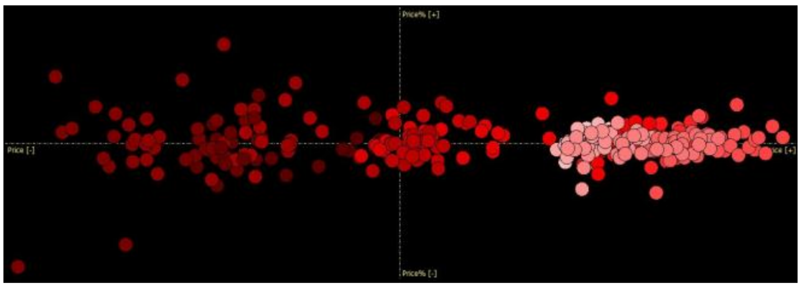


Figure 2.6: A stock price variation chart based on a scatter plot for displaying stock price and its change in percentage, from [Lei and Zhang, 2010, p. 72]

dex ring and an asset ring. The asset rings show stocks of similar performance, whereas the index ring shows the performance of the market index. Based on the above described visualization techniques, the authors implemented a visual analytics system with a modular architecture.
In the market view, the performance of all stocks is shown. The chosen visualization approach is a ringmap. The software enables the user to filter the stock data (e.g. date range, stock code range, ...) by setting parameters.
The underperforming stock view of the software shows stocks with lagged performances. For examining an individual stock, the asset view is used. It displays historical price volatility and possible resistance/support. In this view, price clusters of the selected stock are shown by a scatter plot with color shading.
The pattern view uses a combined line plot for showing specific occurrences of financial patterns. It enables the user to search for predefined patterns and fine tune the definition of each pattern.

2.4 Specialized Visualization-techniques

Another part of the theoretical focus lies on projects with new or original ways of visualizing data.

Parallel Tag Clouds [Collins et al., 2009]

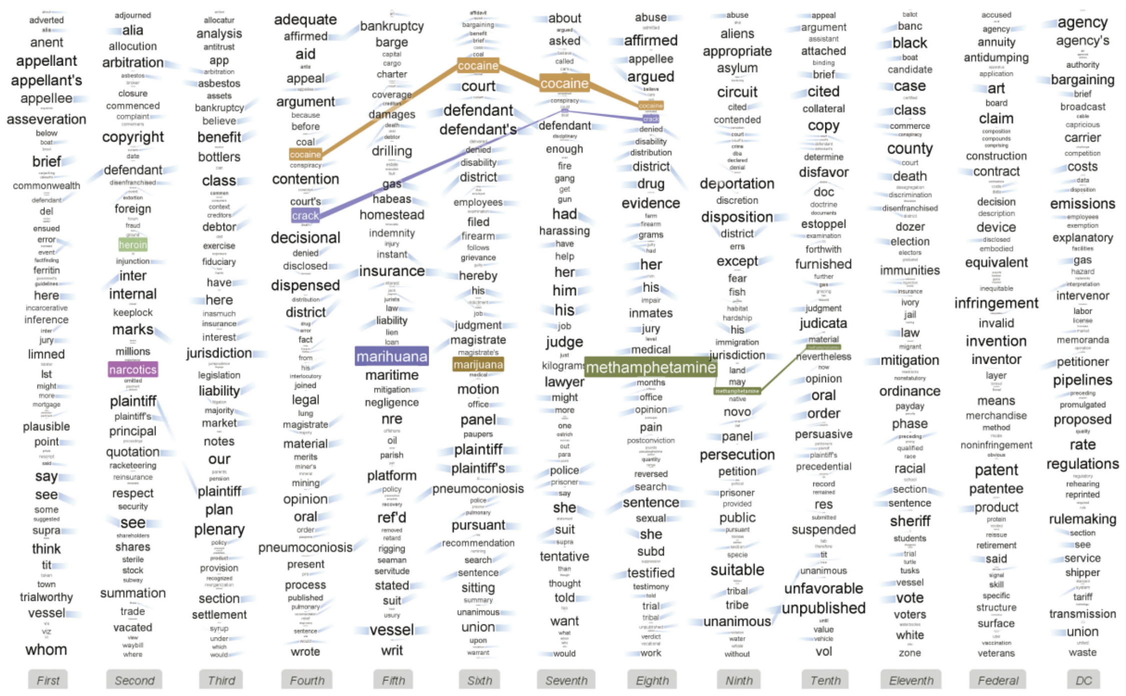


Figure 2.7: Parallel tag cloud revealing the differences in drug prevalence amongst circuits [Collins et al., 2009, p. 92]

The purpose of the first paper of this category is the research of a more efficient way to analyze massive amounts of text corpora. Therefore a visualization technique called "parallel tag clouds" is used to explore texts of court decisions in the USA. They have a spanning period of 50 years. A simple word count is already a powerful analysis tool, to show a significant presence or absence of keywords. This simple analysis leads e.g. to the fact that the Federal Circuit court is the most different from all other courts.
Other important information can be obtained from the time-domain. Using it enables the analysis tool to display significant data changes from one time period to another in color.

Visual cluster analysis of trajectory data with interactive Kohonen maps [Schreck et al., 2009]

The Kohonen Feature Map or Self-Organizing Map (SOM) algorithm is one of the most popular visual clustering techniques. The problem of most implementations of this algorithm is, that they do not allow the user to monitor and steer the clustering process. The cited paper describes a system which extends the automatic (unsupervised) SOM algorithm by means of a visual-interactive control and analysis framework.
The SOM algorithm uses a neural network to obtain a set of prototype vectors (clusters) and a low-dimensional arrangement (sorting) of the prototypes. However, it requires first a suitable vector representation of the trajectory items (feature vector). These feature vectors could for example consist of position, orientation and direction, curvature, etc. In the proposed algorithm, the vector representation is obtained by scaling each trajectory into the unit square. After the scaling is done, uniformly spaced coordinates spanning from the start to the end of the trajectory are sampled.
Through this procedure some information, like the absolute position and scale in space, is lost. However, the advantage is, that it can be used as a direct geometric interpretation and as the basis for the visualization.
The grid of cluster prototypes needs to be initialized before the training process. Therefore, an interactive trajectory editor is provided. It allows the user to draw example trajectories into the grid. Unassigned grid nodes are interpolated automatically.
The authors build in features, to pursue the concept of controllability even during the training of the SOM network. Therefore a continuously updated display of prototype trajectories is provided. Other useful information like the current quantization error is shown as well.
A user observing the training process is always capable of changing the parameters during the process.
The framework supports additional post-processing actions of trajectory maps, after the training. These actions are: merging, expansion, editing, creation, deletion or swapping of trajectory prototypes.

2.5 Collaborative Awareness

SignalCloud is a so-called groupware which means there are several collaborators working on the same project. Which factors are important and how to support awareness among all users, is described in the following paper.

A Descriptive Framework of Workspace Awareness for Real-Time Groupware [Gutwin and Greenberg, 2002]

Staying aware of others is something we take for self-evident in the everyday world, but maintaining this awareness is difficult in real-time distributed systems. Groupware designers have only little principled information about how to support awareness in new systems. They have to reinvent awareness from their own experience in the task at hand.
The goal of the cited article is a descriptive theory of awareness for the purpose of aiding groupware design. The focus of the research group is rather the analysis of collaboration-activities like communication, coordination and assistance, rather than how well the system supports the domain task.
The mentioned descriptive framework consists of three parts.

The first part deals with the question of what information makes up workspace awareness. There is a basic set of short questions which should be answered to increase awareness among collaborators. These question are:

- When do we work with others?
- Who are they?
- What are they doing?
- Where are they working?
- When will various events happen?
- How do those events occur?

Awareness involves knowledge about the past as well, but this requires several considerations. The authors stated some additional question for supporting awareness of the past. These questions can be seen in the following table:

Category	Element	Specific questions
How	Action history	How did that operation happen?
	Artifact history	How did this artifact come to be in this state?
When	Event history	When did that event happen?
Who	Presence history	Who was here, and when?
Where	Location history	Where has a person been?
What	Action history	What has a person been doing?

It is not necessary for groupwares to provide answers to all of these questions, but rather combinations of them.

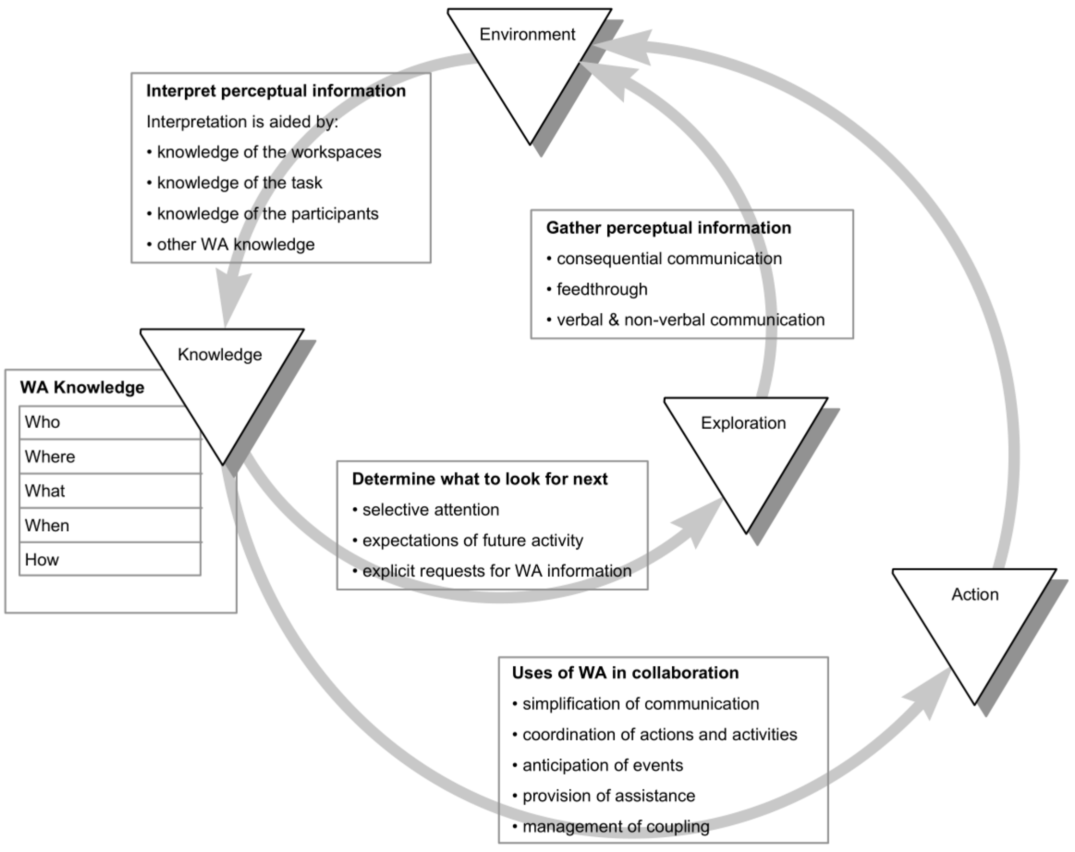

Environment
Interpret perceptual information
Interpretation is aided by:
• knowledge of the workspaces
• knowledge of the task
• knowledge of the participants
• other WA knowledge
Gather perceptual information
• consequential communication
• feedthrough
• verbal & non-verbal communication
Knowledge
WA Knowledge
Who
Where
What
When
How
Exploration
Determine what to look for next
• selective attention
• expectations of future activity
• explicit requests for WA information
Action
Uses of WA in collaboration
• simplification of communication
• coordination of actions and activities
• anticipation of events
• provision of assistance
• management of coupling

Figure 2.8: Visual representation of the awareness framework [Gutwin and Greenberg, 2002, p. 36]

The second part of the framework is a discussion about how to get the information mentioned in the first part. The authors describe a mechanism called consequential communication. It is defined as the information transfer, which is generated as a result of a person's activity within the workspace. Nonetheless, this information is never created intentionally by an user. Another mechanism, called feedthrough, is used to create the needed information. It is produced by the manipulation of artifacts (or elements) belonging to the workspace. The feedback of these actions can be used to inform others about the manipulation.
The last and probably most ubiquitous information-source is verbal- and nonverbal communication. In this case, collaborators might explicitly talk about the domain and pass on awareness information by doing so. Another possibility is that other persons gather information by listening to a conversation, but not actively take part in it. Or at last by picking up "verbal shadowing", the

comments produced by collaborators, but not spoken to a person in particular.

The third and last part of the framework is about the usage of gathered information. They line out five kinds of collaborative concepts, which are possible, because of a high workspace awareness.

- **Management of coupling**
  People tend to shift from individual work to collaborative activity very often. It is beneficial for others to be aware of those transitions, however without disturbing other persons).

- **Simplification of communication**
  This involves using four kinds of communication:

  - *Deictic References:* Can be used to reference objects, persons or anything else without explicitly using a name or the correct word (e.g. that, here, this, etc). This requires always an embedding context.
  - *Demonstrations:* Gestures can be used in workspaces to demonstrate various things. This could be done for instance by tracing a path in the workspace.
  - *Manifesting actions:* This concept can be utilized to replace verbal communication in some cases. Instead of saying something, one uses a manifesting action to communicate a message. An everyday example from the authors is the placing of groceries on the counter, which tells the cashier, that I want to buy them.
  - *Visual evidence:* By giving feedback using visual actions, it is possible to assure other people their utterance has been acknowledged.

- **Coordination of actions**
  Awareness information about what objects are currently in use has to be available. With this information it is possible to plan and process collaborative tasks, otherwise users might disturb others, while working together.

- **Anticipation**
  There are many ways users anticipate others in a workspace. For example by providing materials or tools before they need it as well as avoiding conflicts. This concept needs of course some kind of prediction, but people are usually very good at recognizing patterns.

- **Assistance**
  By being aware of the current state of other collaborators, one is able to recognize the need of assistance without a prior request.

A visual explanation of the here described framework can be seen in Figure 2.8.

2.6 Summary

To sum it up, time-dependent data are everywhere and relevant for all fields of science. There are plenty of visualizations to help analyzing those data. Tools provide presets of those visualizations to simplify a setup of the analysis process. However, there are not much collaborative approaches in this area, so this project might bring some new methods to the field.

Chapter 3

Approach

3.1 A collaborative tool

There is a large selection of different collaborative tools on the Web. If a task is not defined well enough, it is hard to decide which collaboration tool to use. This brings us to the next step: Defining the task we want to accomplish.
The authors of "Choosing the right tools for your virtual team"[Brown et al., 2007] state some initial questions and the answers will help to choose the correct collaborative tool for a specific project. These questions are:

1. "What do you need to accomplish?"
2. "What are your current capabilities?"
3. "Which tool [or technology in this case] is appropriate for each task?"
4. "Who is on your team?"

These questions are actually designed for choosing the correct existing tool. However, we are going to use them for defining how the tool of this project needs to work and what functionalities are required.

3.1.1 Requirements

The first question is probably the most difficult to answer. A rather simple answer would be to tag all artifacts of a signal with a label. Although this is completely true, it is too inaccurate for our further considerations. There are multiple tasks which have to be examined.

All participants have to get access to the same signal visualization.
Before anyone is able to start tagging, they need to see where potential artifacts are. This sounds rather trivial, but it is crucial for deciding which technology to use. Moreover, it has to be possible for people to see only parts of the visualization which are not necessarily the same parts that other people see at the same time. In collaborative systems this is called relaxed WYSIWIS (what you see is what I see).[Stefik et al., 1987].

Collaborators annotate a signal.
Any detected artifact has to be labeled. This means that the artifacts have to be marked in the time domain and a specific type of artifact has to be assigned. To sum it up, we have to store at least the starting-position, duration (or ending-position), and type of an annotation to make it available for others or further processing. If the artifact is located in a certain channel of the signal, this information has to be stored too.

If annotations occupy a similar position and duration, a consensus has to be chosen.
SignalCloud is a collaborative tool, which inevitably means, that there will be more than one opinion about an artifact. In most occasions, such situations can be detected automatically as one will see later on in this work. However, an annotation could be located in different channels or belong to different types, despite of taking up a similar space in the time domain. This results in an opinion conflict of two or more collaborators. A consensus of these overlapping annotations has to be reached. Further on, we will refer to these solved conflicts as “resolved overlaps”.

Keeping track of all ongoing activities.
Especially when two or more individuals agree on dividing the work in parts, it is necessary to keep track of what is already done and what still needs to be done. Being able to keep track of all present and past activity of other collaborators is called workspace-awareness [Gutwin and Greenberg, 2002, p. 10]. We will investigate further below why this functionality is so important for collaborative systems.

Administrate and advice a group of collaborators.
Like in any big project or system, there needs to be some kind of controlling entity to support others with their work or to fulfill administrative tasks. This person needs of course extended permissions over the workspace of files or specialized tools to reach this goal. Furthermore, it could be desirable for this person to get some overview of the current annotation status (distributions,

summaries, etc.).

The tool supports existing file formats.
The SignalCloud-Server will support two file formats. On the one hand there is the binary GDF-file format, which is used to store biomedical signals and on the other hand, CSV-files in the MoTeC-format. The CSV-files have to be exported with the tool MoTeCi2 Standard.
We will have a closer look on the file formats in the next section.

Get a summary of the current state of the signal.
Another planned concept for this tool are “Summary Modules”.
These modules should be accessible by switching the active “view-mode” on a signal.
One can see these modules as different representation types of the signal data points, events, overlaps etc. Currently, there are only three modules planned, but there is an easy way to add new modules to the tool.

Unbiased Opinion
Expert user or not, sometimes it might be interesting to retrieve the unbiased opinion of collaborators. This is not possible if each new annotation is synchronized instantly among all users. A feature has to be added to deactivate the synchronization functionality, at least in one direction. So that a user will not receive new annotations from other people anymore.

3.1.2 Current Tools and Inspiration

Currently SigViewer is one of the tools used to annotate EEG-files at the Technical University Graz. SigViewer has a very high drawing performance and is able to plot millions of data points instantly. For a single user, it is definitely a good choice. The problem emerges if more than one person wants to use the same signal file at the same time. Of course it is possible to copy the file, transfer it to another work-station and start annotating the file. Obviously, this results in multiple versions of the same file, which have to be merged again. This is the point, where it usually gets tricky. If it is not necessary, that collaboration happens simultaneously, it would be advisable to work on the same file in a serial manner.
Regarding the synchronization of signal files, we compared some popular cloud-services on the web with our set of requirements. The two usual services Dropbox and Google Drive provide some interesting inspirations for this tool. It was only logical to use especially Google Drive as a source for some of the concepts, because it was already in use for documentation-parts of this

project.
Especially the online-interfaces of these services are interesting for this project. It is a fact that online-services like Google Docs provide many advantages, in contrast to offline-versions of various office-tools. [Slone and Mitchell, 2014]
A concrete and interesting concept or interface that we mimic in this work is the files overview of Google Drive. It shows not only the current state of an item; additionally, one is able to see the whole history of an item by selecting it. This is a key feature for asynchronous collaboration as well. Another interesting functionality of Google Docs is the indicator that shows a cursor with the current position of other active users and their names. This significantly improves the awareness of other users in the same workspace.

3.1.3 Technologies

After reviewing existing tools which already support some of the required functionalities, it is time to decide which technologies and frameworks need to be used for implementing this tool.
The possibilities and frameworks are numerous, but the only rational choice is a web-service considering all the stated requirements. This decision makes even more sense considering the current developments in technology and the trend towards web-oriented systems.

Server

There are many server-side frameworks available. Some examples with an additional explanation can be found at [Mills and Willee, n.d.]. Many of the frameworks are good and feature a performance which is safely sufficient for this prototype. In this case the decision has fallen on Node.js.
The first reason for that is its growing popularity. Additionally, Node.js makes it easy to reuse the same pieces of code on server and client-side which can be a huge advantage in some cases.

Synchronization

Using Node.js for SignalCloud restricts the selection of available synchronization-frameworks to a smaller set. In former Web-based-implementation this subtask probably would have been realized using AJAX and REST methods. Fortunately, we can use the HTML5-websocket to achieve this goal.
Using a websocket enables a two-way communication between server and client. It is standardized by the W3C and is supported by most of the major browsers. After an upgrade-request (HTTP-Code: 101 Switching Protocols)

a full-duplex connection gets established by building upon the TCP-protocol. [Ian Fette, 2011]

Drawing Framework

After determining the server-side technology, it was time to choose an appropriate JavaScript drawing-library. For choosing a library, some performance and usability tests were carried out.

- Cubism.js[1]
  It features nice functionality with great performance. It is a plugin for D3.js, but it is using canvas[2] to draw its line-charts. The main advantage of this framework is the automatic update functionality, which is not needed for this project. Additionally, its main purpose is rendering horizon-charts or charts for comparison.

- dygraph[3]
  It comes already with many features to use out-of-the-box. For example brushing, labels, etc.. However, most of the functionalities are unnecessary for the given task and the needed features have to be customized at a level of detail that is not supported by the framework and so it was easier to implement the needed features manually.

- D3.js[4]
  It is the most customizable framework listed here which makes it a very complex one as well. Therefore it can be used for various purposes, not only for time-series visualization. However, the drawing part is svg-based which makes it rather easy to implement user interactions. The disadvantage of a svg-based drawing library is that the svg tags are part of the DOM. This results in a poor performance if there are millions of data points, like in our case.

- D3.js plus canvas
  The best performance was achieved by using D3.js for some of the calculations as well as drawing the axes. The drawing part itself is made using a simple canvas. Unfortunately, the maximum width of a canvas is 32,767 pixels in Chrome and Firefox. The Internet Explorer even allows much fewer width (8,192 pixels). Therefore, the canvas has to be split in parts. The chosen width of one canvas part depends on the current

[1] more info at `https://square.github.io/cubism/`
[2] the canvas element provides objects and properties for rendering 2D- or 3D-objects
[3] more info at `http://dygraphs.com/`
[4] more info at `https://d3js.org/`

screen width. The necessity of splitting the canvas is turned into an advantage by using a loading-on-demand strategy. So if the user wants to view a given signal file only the first canvas part is drawn. If the user scrolls in the vicinity of an undrawn canvas part, only then it is drawn. This technique is described in more detail below.

3.1.4 Target Audience and User Constellation

Considering the area of expertise in which this tool will operate, it is clear that the audience will consist primarily of expert users. Additionally, one can assume that the users are already familiar with such kind of tools.
In some cases, mixed constellations with untrained users and experts might take advantage of the collaborative features. This constellation makes even more sense for training purposes.

3.2 File formats

After specifying the tool in some more detail, let us take a look on supported file formats and how they are constructed.

3.2.1 GDF

The General Data Format for biomedical signals unifies the needs of many specialized file formats for various bio-signals like ECG research, EEG analysis, sleep research, etc. Furthermore, it is a binary file format and is released under the general public license (GPL).
A typical GDF-file consists of five sections. [Schlögl, 2006]

1. **Fixed Header**
   Holds general information about the file like patient info or number of channels. The fields belong to the file itself.

2. **Variable Header**
   Holds information about channels, these fields belong to each corresponding channel. They contain information about how to interpret the data records as well.

3. **Data Records**
   This section contains the records (data points) itself. The number of data points per record is defined by the sample rate of a channel.

4. **Event Header**
   Defines the mode of the event table, the number of entries and the sample rate associated with the event-records.

5. **Event Records**
   Contains the actual event entries. Depending on the mode of the table, a record consists of either channel, position duration and type or position and type only. If this is the case, the end of an event is coded into another record and the event is always present in all channels. In addition, it is possible to save a time stamp, if the corresponding mode is set.

There are already plenty of libraries for various programming languages which are capable of loading a GDF files. However, Node.js is unfortunately not one of them.

3.2.2 MoTeC CSV

Another file format which we agreed upon supporting with the SignalCloud-prototype is CSV in a MoTeC format. The tool MoTeC i2 is capable of exporting the data of its default file format with the extension `.ld` into CSV. Because the format is not binary and has no real annotations, it is much easier to implement an importing-function. A typical CSV file could look like the listing below.

```
"Format","MoTeC CSV File",,,"Workbook","Default"
"Venue","WSID",,,"Worksheet","General"
"Vehicle","v219",,,"Vehicle Desc","Car 01"
"Driver","Driver 1",,,"Engine ID","x428i"
"Sample Rate","102.400","Hz",,"End Time","196.289","s"


"Time","Engine RPM","Air Temp Inlet","Eng Oil Pres"
"s","rpm","F","psi"


"0.000","3084","85.1","153.99"
"0.010","3084","85.1","153.97"
"0.020","3078","85.1","153.95"
...
```

The file consists of three sections. First of all, some information about the file itself is shown. Secondly, the label and unit of all channels is described. At last, the actual data points of the file are visible.
Each section is separated by two newline-characters.

3.3 Workflows

In this section, we will discuss the basic workflow of the tool. Note that the following scenarios could be considered as sample use cases of the tool. Figure 3.1 illustrates a visual overview of all the following scenarios. The workflow is color-coded, but note that one color does not strictly match one scenario. Colors are used to illustrate a new branch in the flow which introduces for example additional steps in a collaborative setup.

3.3.1 Single User

This is the simplest scenario. There is one user only, who wants to label a single file. The goal is to label artifacts of the file correctly and download it again. The scenario involves the following steps:

1. If not already done, the user signs up on the server.
2. Now it is possible to upload a recorded signal file to the server, by using the “Upload file”-button on the `/files`-page.
3. By double clicking the file or pressing the corresponding button, the file will open in a new screen.
4. The user is now capable of choosing the “Add Event Tool” and start tagging artifacts in the file.
5. After one has tagged some events, it may be interesting to review them in the summary view. It can be activated by clicking on the corresponding button in the menu-panel.
6. After editing, the user is able to download the file by clicking on the “Download as GDF”-button in the menu panel. If the signal file was holding events at the time of the upload, it is now possible to include them or not within the downloaded file.

3.3.2 Collaborative Session

In this case, more than one user contributes to one signal file. The goal of the scenario is reaching a consensus. Afterwards the file should be downloaded again.

1. If not already done, each user signs up on the server.

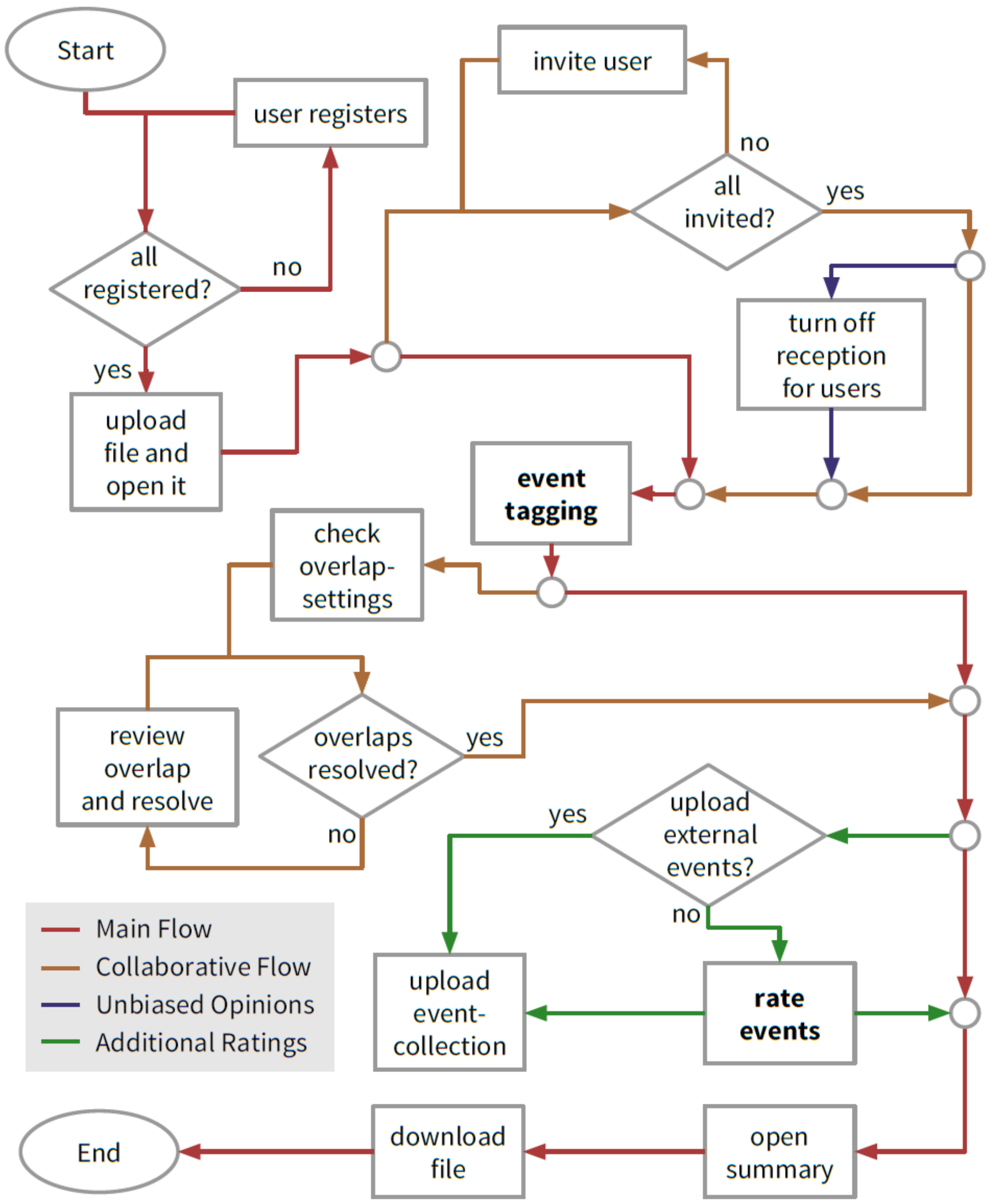


Figure 3.1: state chart diagram of the workflow - summarizes all of the following scenarios

2. The recorded signal-file is uploaded by its owner. This happens via the `/files`-page.
3. The owner of the file is now able to open the uploaded file, by double-clicking it.
4. To start a collaborative session the owner has to invite users to the uploaded file first. In order to do so, she or he has to click on “Invite User” in the left menu panel. The owner has to repeat this operation for each user she or he wants to invite. By selecting the role of the user, one is able to distribute certain rights. To be able to tag events, a user has to be “admin” or “editor”.
5. Users with the above mentioned roles are now capable of tagging events in the signal file. Therefore they have to select the “Add Event Tool” (flag symbol) in the toolbar. Afterwards, they can select the required occurrence by clicking on the line chart and dragging a rectangle in the corresponding channel.
   Additionally, the user is able to choose a preset for new Events. This regards either the event type or the channel, if activated. Otherwise it can be done using a confirmation-dialog-window as well.
6. A user is able to signal the owner or any admin of the file that she or he is finished by writing a comment or simply logging off. The owner is able to see which individuals are currently online.
7. Depending on the given task the owner or any admin is able to configure parameters of the overlap detection. This is done by opening the settings dialog-window by clicking on “Settings” in the menu panel.
8. If there are any overlapping events due to the editing of multiple users, they will be detected as new overlaps and become visible under the tab “Overlaps”.
9. The owner or any admin creates a consensus for each overlap by clicking on the pencil-icon to open it and dragging the left or right end of the transparent box to the desired position.
10. Depending on the configuration of the overlap detection, someone has to review the channel or the event type of the overlap as well.
11. If an overlaps is reviewed completely, it is possible to resolve it. This will add the overlap to the section “Resolved Overlaps” in the same tab
12. Resolved overlaps will appear as new events under the tab “Events”.

13. Users are able to rate the resolved overlaps now, by using the stars at the bottom of each overlap-event.

14. Finally, all users are able to download the file and to select which event-collection they want to integrate in the file (e.g. only "Resolved Overlaps").

3.3.3 Unbiased Collaborators

There is one supervisor and many users. The goal is the same as in the collaborative scenario. However, the supervisor is not contributing to the file himself in this scenario. Another difference is that the users are supposed to give their unbiased opinion of where they think significant events are happening in the signal file.

1. If not already done, each user signs up on the server.

2. The recorded signal-file gets uploaded by its owner. This happens via the `/files`-page.

3. The owner of the file is now able to open the uploaded file, by double-clicking on it or by using the open-button.

4. To start a non-collaborative session, the owner has to invite users to the uploaded file as well. In order to do so, she or he has to click on "Invite User" in the left menu panel. The owner has to repeat this operation for each user she or he wants to invite. By selecting the role of the user, one is able to distribute certain rights. To be able to tag events, a user has to be "admin" or "editor".

5. To get their unbiased opinion, the supervisor has to disable the event-reception of all the users. Therefore he has to use the context-menu on all of the users and select the option "Disallow Event-Reception".

6. Users are now able to start tagging events in the signal-file. In this case events are synchronized with the database and the supervisor, but users will not be able to see events from other users.

7. During the tagging process, the supervisor might enable the summary view to better review the overall process.

8. A user is able to signal the owner or any admin of the file that he is finished by writing a comment.

9. Depending on the given task, the owner or any admin is able to configure parameters of the overlap-detection.
10. If there are any overlapping events, due to the editing of multiple users, they will be detected as new overlaps and become visible under the tab “Overlaps”.
11. The owner or any admin creates a consensus for each overlap.
12. Depending on the configuration of the overlap-detection, someone has to review the channel or the event-type of the overlap as well.
13. Once an overlap is reviewed completely, it is possible to resolve it. This will add the overlap to the section “Resolved Overlaps” in the same tab.
14. Resolved overlaps will appear as new events under the tab “Events”.
15. Users are able to rate the resolved overlaps now by using the stars at the bottom of each overlap-event.
16. The supervisor re-enables the event-reception for all users again, so that they are able to see the events of all other users.
17. Finally, all users are able to download the file and to select which event-collection they want to integrate in the file (e.g. only “Resolved Overlaps”).

3.3.4 Rating Performance

This scenario is different from the others, mainly because of its goal. This is namely rating the performance of an algorithm. There are simply multiple users in this scenario, no one sticks out in particular.

1. A file with multiple users exists already.
2. All of them are editors (the owner is absent).
3. One of them uploads an external event-collection by using the intended upload-form.
4. Others start rating the uploaded event occurrences by using a score from one to five.
5. All the users are able to review the aggregated score by looking on the colored number attached to each event occurrence.

3.4 Summary

To sum it up, the practical part of this work will be a collaborative web-service, which features visualizations, mainly line-charts, annotations and realtime-synchronization among users. There are a few sample workflows but the tool has probably more potential use-cases than the listed ones. We will have a more detailed look on the finished version of this tool in the next two chapters.

Chapter 4

Implementation

4.1 Concept Realization

At first, let us take a more detailed look on concrete realizations of certain previously explained concepts.

4.1.1 Drawing Mechanics

Technically, the drawing process is a non-trivial issue. Although, most of the plotting is implemented without any external libraries, some functions of D3.js are used for calculation purposes. By doing so, the native scroll-functionality of browsers could be used instead of some manual implementation. Consequently, it is impossible to use SVG for drawing the graph, because of the many data points which would appear in the DOM as well. Thereby, the only option left is using HTML5-canvas.
Nevertheless, the performance is not sufficient to draw the whole signal as soon as it is loaded. There are more reasons why this naive approach is not the best one. A better and probably obvious attempt is to plot only the visible parts of the signal. As a result, the plotting performance is superior to the previous concept. Admittedly, some browsers had problems while scrolling through the whole plot. After some research about infinite-scroll techniques [Heleine, n.d.], a way was found to improve the scroll-performance in most modern browsers. Since the plots are already cut into sections drawn on demand, it was an easy task to set the visibility[1] of sections, located outside the scroll-window, to hidden. A better visual explanation can be seen in Figure 4.1.

[1] visibility is a common CSS-attribute

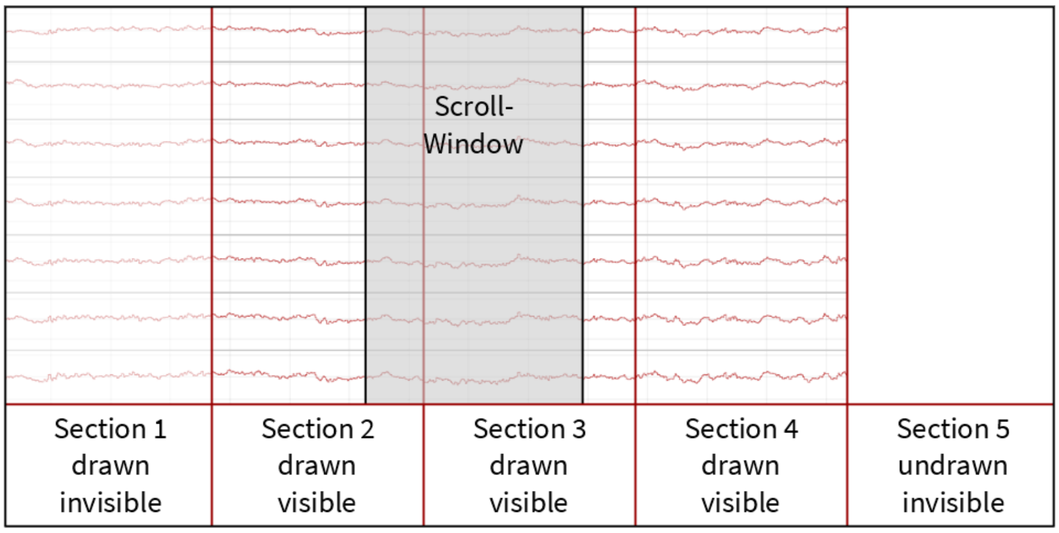


Figure 4.1: Drawing mechanics

Note that the hypothetical user previously visited “Section 1” in the figure, hence it is already drawn.
The width of these sections depends on the screen width of the viewing device. In fact, it has to be bigger than the screen width, so that an update of one section is enough to support the illusion that the whole signal is updated. This update could be triggered for example by a variation of the zoom-level or the addition of a new event.
Consequently, there is more than one version of an update. Technically, there are four different update variants which can be triggered trough various reasons.

- “Full”-Update
  Reloads everything, even the data points are split again into parts. Each data part is reserved to be drawn by a plot section. This is needed, e.g., if a new data slice was received by the client through the asynchronous loading of the signal. Additionally, the scaling factors which define the transformation between pixel and seconds are updated as well. In most of the cases this update is needed, because of the recalculation of the scales.

- “Draw”-Update
  Very similar to the “Full”-Update. However there is neither a re-splitting of the signal nor a scale-recalculation

- “Event”-Update
  Smaller update which is executed after changing or adding events of a

section. It only redraws the section currently visible. Other sections are marked “undrawn”.

- “Scale”-Update
  Smallest update possible. It is executed to redraw the visible scale for the Y-axises of the signal. Like the previous update, it only affects the current section and invalidates others.

4.1.2 User-Management

Each file can be shared with other registered users among the server. Therefore it could be unattractive to the owner of a file to give all permissions to any user. For solving this issue, some kind of permission-distribution had to be considered. The preferred solution to this problem is a permission system using roles. Each user role is assigned a certain amount of privileges. Note that a user role applies only to one file. The roles are organized hierarchically which means a higher role has all the permissions from its predecessor and more. There are four or actually five user roles:

- **Owner** (Highest)
  Has all permissions possible. This user role is not “distributed”, it is automatically determined through the upload of a file. It is the only role in a file which has the opportunity to delete it. Furthermore a user with this role cannot be un-invited.

- **Admin**
  Has nearly all privileges but is not allowed to delete the file. Supposed to help the owner with any administrative work, for example inviting more users or resolving overlaps.

- **Editor**
  The most common role, supposed to do the “hard work”, i.e. labeling all artifacts in the signal. Any editor is not able to invite or un-invite other users.

- **Reader**
  Lowest distributable role. Only allowed to view the file or download it. Furthermore this role is authorized to create comments and rate events.

- **Uninvited** (Lowest)
  Actually the lowest reachable role. After an admin or the owner decides to un-invite one user, she or he will get the role “uninvited”. The reason why the individual gets not deleted from the user list instantly is functional on the one hand and technical on the other hand. The functional

reason is that an uninvited user is able to download the file one last time before it becomes inaccessible for her or him. However, this only works if the user has currently opened the file.
The technical reason behind the uninvited-role is that the data of the user has to be kept. For example, if the user has created events, the data is needed to show others the connections of the events and the uninvited user.

4.1.3 Overlaps

Let us have a more detailed look on the overlap handling. It can be divided into two subtasks: Detection and Management.

Detection

The matching of a new event against an existing overlap is straightforward. There are a few sizes which have to be explained first to understand the matching-process. At first, there are the outer borders. These are defined by earliest start t_{es} and the latest end t_{le} of all occurrences on the x axis. Moreover, there are the inner border which is defined by the latest start t_{ls} and the earliest end t_{ee}. At last, the detection margin t_m is defined by the width between the outer border multiplied by the detection-tolerance factor d.
By combining these sizes, one can define the detection region R by following formulas:

$$\begin{aligned}
t_m &= (t_{le} - t_{es}) \cdot d \\
R_{s1} &= t_{es} - t_m \\
R_{e1} &= t_{ls} + t_m \\
R_{s2} &= t_{ee} - t_m \\
R_{e2} &= t_{le} + t_m \\
M_o(R, t_o) &= \begin{cases} \text{true} & \textbf{if } R_{e1} < t_{os} < R_{s1} \land R_{s2} < t_{oe} < R_{e2} \\ \text{false} & \textbf{else} \end{cases}
\end{aligned}$$

where t_{os} is the start and t_{oe} is the end of the tested occurrence, respectively; finally, M_o is the boolean result of the matching process and gives insight if the occurrence fits the overlap or not. A visual explanation of these sizes can be found in Figure 4.2.

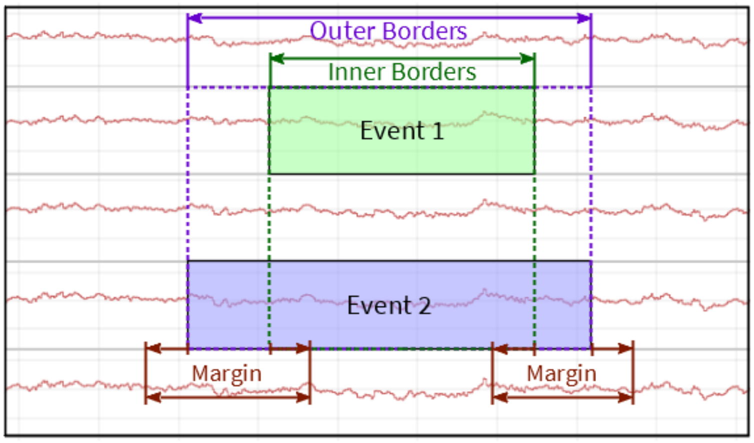


Figure 4.2: Explanation of relevant sizes

The explanation from above assumes a hypothetical existing overlap. However, there are no overlaps at the beginning which means there has to be an initialization first. This is done by choosing occurrences which do not fit with any overlap. So this means that each occurrence is attached to an overlap in the beginning, because if there is no match, the occurrence initializes a new overlap.
Three sample-outcomes of a matching-process can be seen in Figure 4.3.

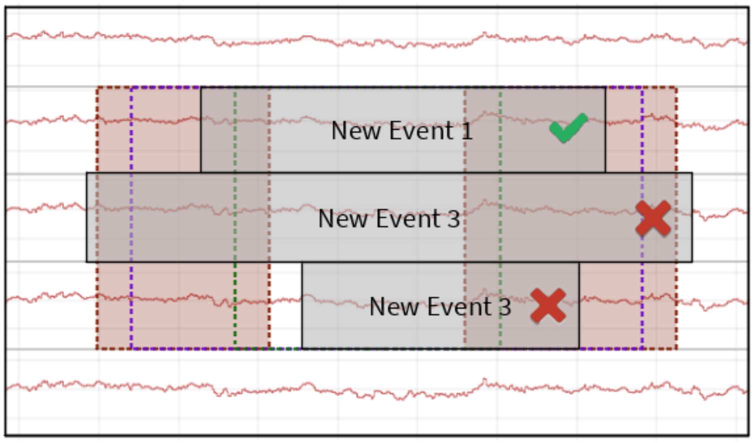


Figure 4.3: Three new events matched against an existing overlap

Note that two of the possible user-settings are not considered until now. The detection can take place either only within channels, among event-types or both. Nevertheless, this does not change the algorithm itself. This means if one of the settings is enabled, the detection is executed multiple times on alternating scopes. One scope could be a certain event-type, channel or both. For example, if there are three different event-types and three channels and additionally both options are enabled, this means there are nine different scopes, assuming that each type exists in each channel at least once.

Management

The technical management of overlaps itself is a non-trivial task from an implementation point of view because of the volatile nature of each overlap.
The first step of each detection cycle has to be a sorting of event-occurrences and already existing overlaps by their starting position in the time-domain. This ensures a higher performance of the algorithm. Because of the sorting, it is unnecessary to match later occurrences with overlaps from the start.
As mentioned before, these overlaps have a volatile nature. Therefore it is necessary to have an additional working cache per file in the database. It is mainly used to store information about resolved overlaps. By reading and comparing this cache with the actual situation in a detection-cycle, it is possible to carry two vital tasks out.
At first, it reveals an “unresolve”-action effectuated by an user. This means the overlap is available for changes again.
Secondly, it allows to detect deleted resolved occurrences[2], so their references can be removed as well.
Both of those actions are necessary to guarantee data-consistency.
A visual representation of the algorithm behind the overlap-detection can be seen in Figure 4.4

[2] occurrences belonging to a resolved overlap

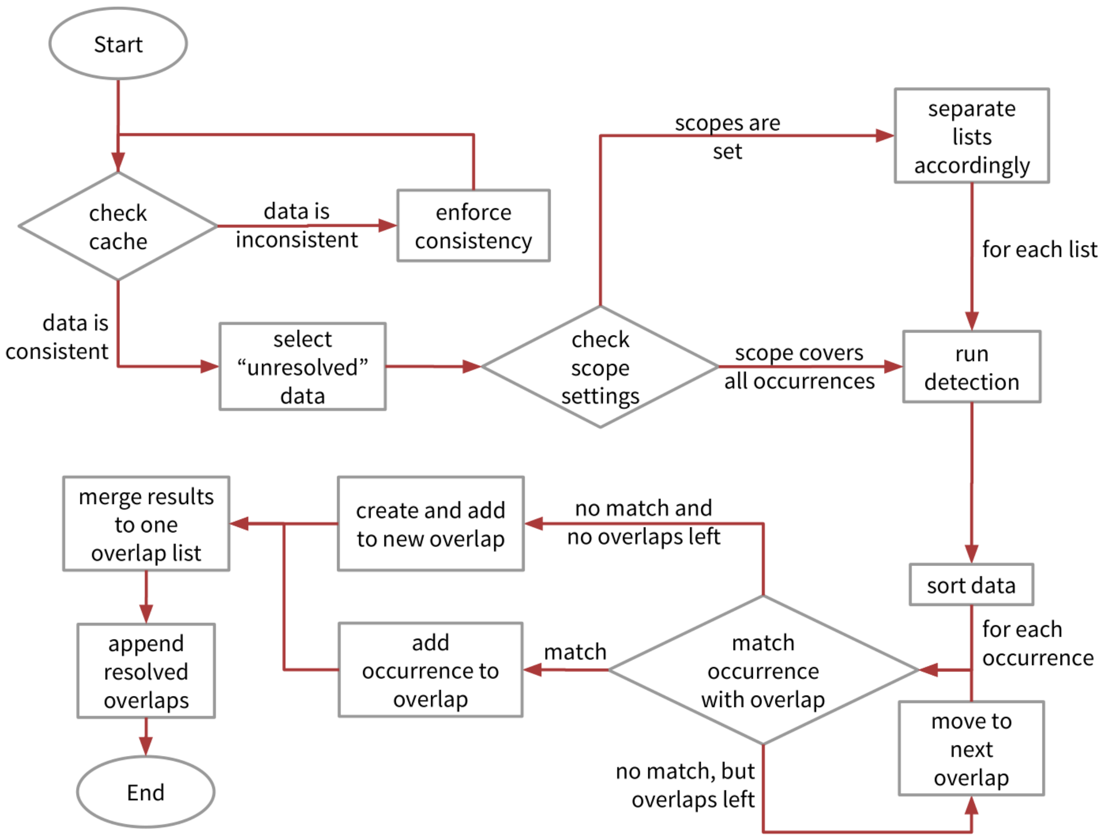


Figure 4.4: State chart of the detection algorithm

4.2 Frameworks

SignalCloud uses a variety of frameworks for communication, storing, drawing, building, etc. To understand how the assembled code-structure behind this tool works, we have to a closer look on the most important external frameworks and libraries.

4.2.1 Express

Express is a popular http-server-implementation for Node.js, which features many module like extensions. Setting up express with the pug(or jade)-template-engine is really easy. It can be used to send HTML, JSON or any other answer on a REST-request.

4.2.2 Socket.IO

Socket.IOis the most popular framework on Node.js, which simplify the usage of web-sockets. There is a server-side and a client-side library.

4.2.3 Passport

It integrates seamlessly into express and provides methods and objects, which enable a simple user- and session-management. It even integrates in Socket.IO too, which makes this library extremely useful.

4.2.4 Webpack

The main purpose of webpack are all types of bundling tasks. It brings the module-based system, with the “require”-keyword, from Node.js to client-side JavaScript. This allows a more structured code and a few new features as well. For example it is possible to load templates simply by requiring them, if the corresponding webpack-loader has been installed.
If webpack is configured correctly, it is possible to create bundles for each specified route. Moreover, webpack allows bundling external libraries or other common code among all scripts into bundles. An additional loader is even capable of loading babel, which allows it to convert es6 code into es5-code.

4.2.5 Babel and es6

Babel is used to transform a preset version of ECMAScript into es5 (5th version of ECMAScript). It can even be used to compile `.jsx`-files meant to be used with React[3].
In this tool babel is used to transform es6- into es5-code to support all available browsers. Although, most of the current browsers are already capable of interpreting es6-code.

4.2.6 AngularJS

AngularJSenhances static HTML with dynamic components. This allows a two-way binding of JavaScript-code-components and HTML. It completely changes the way of coding web-apps. Developer have to define AngularJS-components, which have a specific structure. These components are for example controllers, directives, services, etc. AngularJSis even more powerful, when it

[3] a JavaScript-library for building interactive user-interfaces, similar to AngularJS

comes to smaller client-side-only projects without additional building tools, because it is capable of loading only the necessary code.

4.2.7 Pug

Pug is the newer version of jade. It is a template-format, which can be compiled into HTML-code, with a syntax similar to HAML[4]. If a pug-code gets compiled, a data-object, the “locals”, is needed. This data-object is available inside the pug-template, which allows if-functions, for-loops or even literal JavaScript-code.
Moreover, pug features code-reduction functionalities like inheritance, includes or mixins, which make it possible to reuse various parts of code in many cases.

4.2.8 SASS/SCSS

SASS is a pre-compiler for CSS-files. It adds many features to the CSS-syntax, which results in big code-reductions and more readability. There are two variants: SASS and SCSS. They are more or less the same, but SCSS uses more brackets and is therefore more related to traditional CSS. This project uses the SCSS-syntax.

4.2.9 CouchDB and nano

CouchDBis a popular no-SQL database, which stores data in the JSON-format. It is all about versioning. Each update-request has to provide a revision-ID, to prove that the client is aware of the current version of a certain item. This allows CouchDBto provide data-access to all clients, the whole time. Older versions of items are deleted, if a database is compressed. This is a separate task, which is executed on demand. Instead of SQL-like `JOIN`-commands, CouchDB features so-called views. These are JavaScript-functions, which are needed to execute a map-reduce-task on a given database.
Nano is a small Node.js-library, which simplifies the access to a running CouchDB-instance.

4.2.10 Gulp

Gulp is a task-runner tool, which runs on Node.js. It can be used for almost everything, like minifying code, running babel, compiling sass to css, running webpack, copying stuff from one folder to another, etc. It is a extremely powerful tool, which simplifies everyday tasks when building web-apps.

[4] another pre-compiler for HTML, where nesting is done by whitespace-characters

4.3 Prototypes

Many prototypes built different frameworks and technologies were made, after learning various lessons about web-development and JavaScript. Let us take a look on the road to SignalCloud.

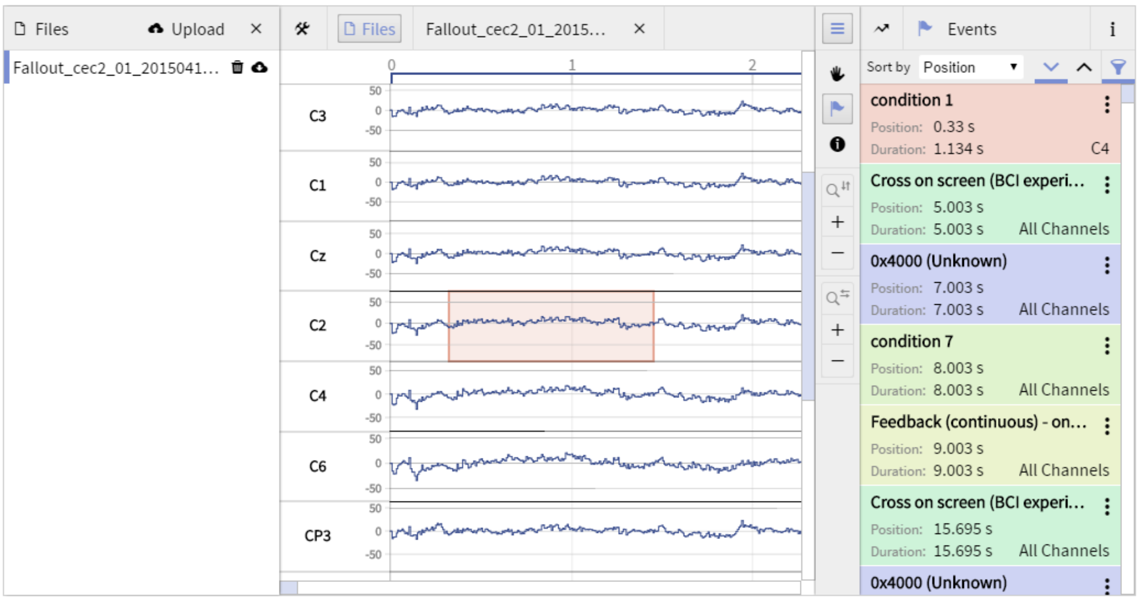
4.3.1 Web-based SigViewer

The first goal was to make a simple web-based version of SigViewer. This was already a hard task, because of the absence of an importing framework for the GDF-fileformat. So the actual first task was the implementation of a function, capable of importing a GDF-file. This was all done within JavaScript, more specifically client-side JavaScript. Fortunately, the newer File API[5] made it possible to upload a file locally and process it further. After solving some issues with the different samplerates of channels, the firs prototype was able to draw a short part of the signal.

4.3.2 Client-only with Sql-lite Database

The second iteration of the tool featured already a small SQL-lite database. Altough, it was still client-side only and had no access to an actual server. JavaScript-objects, which got created after uploading a new GDF-file were simple converted into a JSON-string and uploaded to a SQL-lite-file using AJAX. Despite of having some real performance issues and there was no synchronization among collaborators, this prototype had already some interesting features.

[5] an API specified by the W3C for accessing file objects via JavaScript, additional subfeatures allow an easy handling of binary files as well, more info on `https://w3c.github.io/FileAPI/`



condition 1
Position: 0.33 s
Duration: 1.134 s C4

Cross on screen (BCI experi...
Position: 5.003 s
Duration: 5.003 s All Channels

0x4000 (Unknown)
Position: 7.003 s
Duration: 7.003 s All Channels

condition 7
Position: 8.003 s
Duration: 8.003 s All Channels

Feedback (continuous) - on...
Position: 9.003 s
Duration: 9.003 s All Channels

Cross on screen (BCI experi...
Position: 15.695 s
Duration: 15.695 s All Channels

0x4000 (Unknown)

Figure 4.5: Prototype with small SQL-lite-server

The drawing technique of this prototype was only slightly changed in newer versions. Additionally, the user-interface was divided in the same way as in the final version.

4.3.3 Node.js Server

After having a running client, which was already capable of drawing a GDF-file and allowing someone to make annotations on the signal visualization, the necessity of an actual server grew. A framework called “express” looked promising and so it was used as http-server, alongside “jade” as its template engine. Of course, all the html-code had to be converted to jade-files now, in order to support all the advantages of jade.
The main purpose of this new established server was actually the realtime synchronization of all actions among users. Therefore, the popular framework “Socket.IO” was added as well.

4.3.4 SignalCloud

Subsequently, it was time to leave all the prototyping behind and start to built up the final framework. This included some client-side compilers and tools as well as the famous css-precompiler “SASS”. “webpack” was used to generate bundles of JavaScript-code for different the server-routes, which were added

with the new server. These routes include a route for registering new users, logging in, getting a list of all the (personal) files on the server and of course a route for viewing a signal. Later on an additional site was added, which could be used to configure some account settings.
Moreover, after some failed experiments with MongoDB, CouchDB was already used to store all the signal-files. In addition, the library “nano” was installed for an easier communication with CouchDB.

4.3.5 SignalCloud with AngularJS

The last and final version of this tool is using additionally to all frameworks AngularJS. This makes updating parts of the user-interface much easier. Many parts of the logic could be moved into template-files, which results in less code. Furthermore, AngularJS enforces a certain code-structure, which increases the readability of all code-files.
Because jade was deprecated at this time, the decision to switch to its newer version called “Pug” was made.
Moreover, I wanted to use some features of ECMAScript v6[6] (es6 or es2015). However, not all browsers support all features of es6 currently. Therefore, the framework Babel has been set up, which converts es6-code into an earlier version of ECMAScript.

4.4 Code Structure

There is a certain structure behind each part of the software. Each substructure consists of frameworks, building-tools, configuration-files and design-pattern.
The root-folder structure can be reviewed in 4.6. This level holds basically the server-structure, of the tool. All the routing is done in `./controllers`. The pug-templates are located inside the `./views` directory. Moreover, all necessary client-side JavaScript-code is located in `./js-client-libs`. The uncompiled source-scss files can be found in `./scss`. In addition to these folders, there are some important files as well. The file named `app.js` is start-file for the whole application. It loads everything and glues the code basically together. The other files are more or less configuration-files for npm, gulp and webpack.

[6] ECMAScript is a scripting language based on JavaScript and is the current language used by all major browsers and Node.js

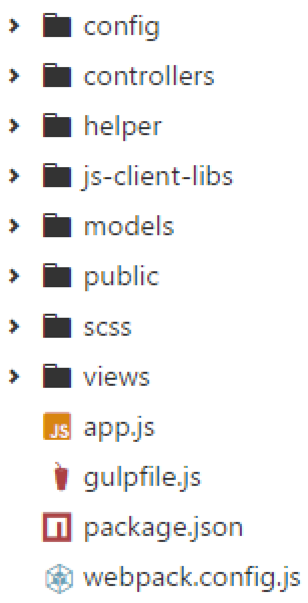

```
> config
> controllers
> helper
> js-client-libs
> models
> public
> scss
> views
  app.js
  gulpfile.js
  package.json
  webpack.config.js
```

Figure 4.6: Root folder of the project

4.4.1 Server

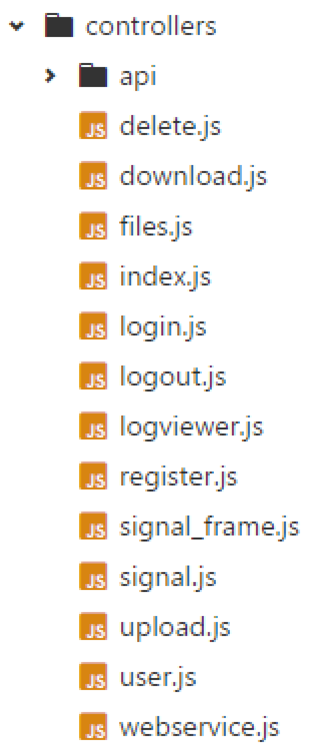
The server-side JavaScript-structure is much inspired by some Model-View-Controller[7]-examples for express and Node.js. [Tsonev, 2013] [Roberts, 2013]

Controllers

Basically, the components of the design-patterns are stored in the corresponding folders (Figure 4.6). The `./controllers`-folder has an additional subfolder, which holds all the logic for handling API-requests.
There is a file or folder for each accessible route in the `./controllers`-folder. For example, if someone wants to access the url “`DOMAIN:PORT`[8] `/api/signal/metadata`”, the responding controller-file will be located in `./controllers/api/signal/metadata.js`. The folder-structure of the controllers can be reviewed in 4.7.
A little bit different is the file called “`webservice.js`”. This file is responsible for all the websocket-communication, which is powered by Socket.IO. It holds all necessary code for answering websocket-calls and broadcasts.

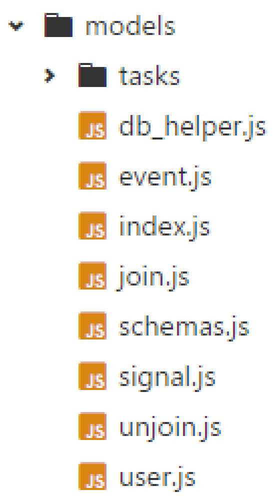
[7] popular software-design-pattern, which consist of three components: **view** - controlls GUI and forwards interactions to controller, **model** - stores, loads and alters data, **controller** - holds all of the necessary controlling logic

[8] DOMAIN:PORT is for example replaced by “localhost:3333”, if the server is running locally



```
controllers
  api
  delete.js
  download.js
  files.js
  index.js
  login.js
  logout.js
  logviewer.js
  register.js
  signal_frame.js
  signal.js
  upload.js
  user.js
  webservice.js
```

Figure 4.7: Folder-structure of the controller



```
models
  tasks
  db_helper.js
  event.js
  index.js
  join.js
  schemas.js
  signal.js
  unjoin.js
  user.js
```

Figure 4.8: Folder-structure of the model

Models

All code-modules for accessing the database can be found in the folder `./models` (Figure 4.8). The file called `db_helper.js` provides some extremely useful methods for an easier access to the database. It handles updates, deletion and creation of items as well as the construction of views and databases on a running CouchDB-instance.
There is another “specialized” file in this folder called “`join.js`”. This module fetches data by some database-requests and merges the data in a specific way. For example, there is a method for loading the metadata-entry for a file and adding the userdata of each user to the userlist. In addition, there a file, which performs the exact opposite and stores the separated data in the database again.
Furthermore, the file called “`schemas.js`” holds a JSON-structure, which defines the allowed format of the data-entries in the database. Objects will be sanitized using an external library with this schema’s, before they are saved in the database. This means additional properties of the object will be deleted and properties with the wrong data-format will be converted, if possible.
At last, there are the generic files to access data-entries in CouchDB, these

are `event.js`, `signal.js` and `user.js`.
The folder `./models/tasks` contains modules for altering data in the database. All the necessary code for the overlap-detection is located in this folder. Moreover, there is a module for generating events with joined datapoints.

4.4.2 Client

The client-side code is organized in bundles. These bundles are `default`, `files`, `user` and the most important `signal`. Each of them is located in the corresponding folder (Figure 4.9).
Every time a site gets compiled using one of the main views, a script-bundle-name has to be provided.
Other files in the `./js-client-libs`-folder could be considered as shared or global, because they provide some common code, which is needed by all script-bundles.

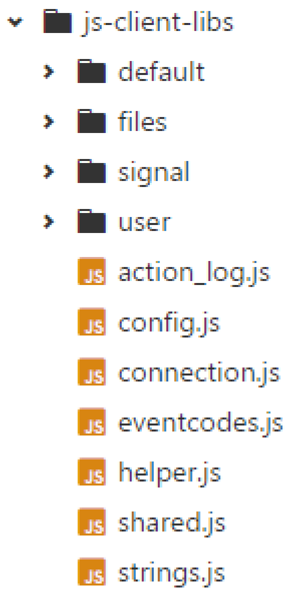

```
js-client-libs
  default
  files
  signal
  user
  action_log.js
  config.js
  connection.js
  eventcodes.js
  helper.js
  shared.js
  strings.js
```

Figure 4.9: Folder-structure of the client-side code

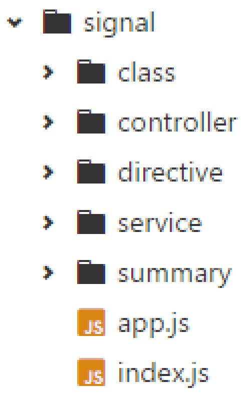

```
signal
  class
  controller
  directive
  service
  summary
  app.js
  index.js
```

Figure 4.10: Folder-structure of the signal-bundle

Signal Bundle

As it is the most important code-collection, let us take a closer look on the signal-bundle.
It is the one, which is loaded, if any signal-file is opened.

Everything starts in the file called `model.js` located in `./js-client-libs-/signal/service`. The initialization method of this module gets called as soon as the corresponding AngularJS-controller has loaded this service. The initialization is divided into phases. At first the relevant data-entries for the opened file are fetched by using the websocket. These fetch-requests are of course executed asynchronously, which caused some race-condition problems at first. To prevent them from happening, an external library called "async"[9] ensures the correct execution of all callbacks. Additionally, other services and modules wait until the member-variable "`ready`" is set to `true`, so that nothing is done before all data is ready.

After everything necessary is loaded via the websocket, the so-called "analyzer"-module located in the same folder starts initializing. This module is more or less an extension of the model. It is responsible for loading the actual datapoints via AJAX-calls. Moreover, it provides some useful information about the signal, like length, min- and max-values per channel, etc.
The last step of the initialization is the loading of all the lists located in the detail-panel. Most of these lists are accessible, via the model, in its hierarchical format, which gets built up in this process (or is already present), or on the other hand in its "flat" format. The advantage of these additional lists is an easier access of data in some cases.
If the analyzer-module sets his `ready`-flag to `true` as well, the draw-signal-directive located in `./js-client-libs/signal/directive` is starting to draw the channels. Due to the fact, that the signal-datapoints is divided into parts, a redrawing is necessary after each part-update. Furthermore, there is a small timeout, between each redraw. If a new part gets loaded, before this timeout ends, the update is postponed. This means, if the connection is fast enough, the signal is loaded entirely without any drawing-update. However, many user-interactions demand a redraw of the signal area, for example any type of zooming, changing the order, color or scaling of channels, etc.
A file, nearly as important as the model, called "`view.js`"is located in `./js-client-libs/signal/controller`. This file is the main AngularJS-controller of the whole signal-bundle. It provides an angular-scope[10], with functions and variables for many purposes and could be seen as a first responder for many mechanics.
All the needed code for showing any kind of context-menu is located in the

[9] async is a library, which features many methods for handling the correct execution order of asynchronous JavaScript-functions, more info at `http://caolan.github.io/async/`

[10] the JavaScript-angular-scope is accessible from HTML by using certain attributes, elements, or classes (= directives), or by using the double bracket-notation (e.g. {{var-name}}), more info at `https://docs.angularjs.org/guide/scope`

service-file called “`context_menu.js`”. It is meant to be used by an external library[11], which is a plugin for AngularJS.
Finally, the last of the most important file is called “`dialogs.js`” and it is located in the same folder. This file is responsible for displaying and controlling any type of dialog.

Summary Modules

Part of the signal-bundle are all summary-modules, but they behave a little bit different than the rest of the code. The reason for this, is its simplified extendability. It is very easy to add a new summary-module to the existing ones.
Each modules consist of an initial configuration, which can be seen further below, and an AngularJS-directive. This directive has to be named like stated in the configuration.

```
let config = {

  label:            'Event Type Plots per Channel',
  component:        'linechartEventtypes',
  main_description: `
    Shows plots for each Event-Type per channel.
    Depending on your draw options, the average line will be drawn too.<br />
    X: Time (Scale varies between graphs)<br />
    Y: Channel-Value
  `,
  detail_content:   `
    <b>X</b>: Time (Scale varies)<br />
    <b>Y</b>: Channel-Value<br />
  `,
  parameter: {

    rescale: {

      type:  'switch',
      def:   true,
      label: 'Rescale Lines'
    },
    draw: {

      type:    'radio',
      def:     'both',
      label:   'Draw Line',
      options: {

```

[11] ui.bootstrap.contextMenu a library for showing web-based context-menus, more info at https://github.com/Templarian/ui.bootstrap.contextMenu

```

          // Label: Value
          Average:  'average',
          Channels: 'channels',
          Both:     'both'
        }
      }
    }
};
```

Listing 4.1: Example configuration of a summary-module

The parameter section of the configuration defines the user-controllable variables of the module. In the above example are two parameters made controllable. One of them is a radio-button, which means there is a predefined set of options, the other one is a simple on-off-toggle.
Legit types for these parameter are: switch, checkbox, slider, text, radio.
These parameter are available by using the method `getSummaryParameter()`, which is provided by the model. The method needs one parameter, which is the name of the summary-module.

Building Process

The main tool of the client-side building process is webpack. It is executing babel, loading pug files using a specialized loader, minifying the code etc. To work the way it does, webpack needs a configuration-file, which is located in the root-directory. This file defines all the bundles for each route. Note that most of the less significant routes (e.g. error-, login-, register-sites) use the default JavaScript-bundle, which provides a simple AngularJS-controller.
Another bundle called `vendor.js` is generated, which holds code from external libraries like jQuery, AngularJS, etc.
The structure of the output-folder can be seen in Figure 4.11. This folder is the only one, which is directly accessible by the web-browser. Therefore, it holds compiled style-sheets, images and fonts too.

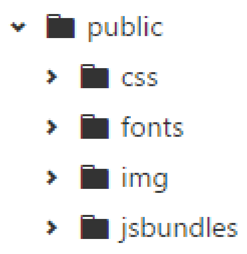
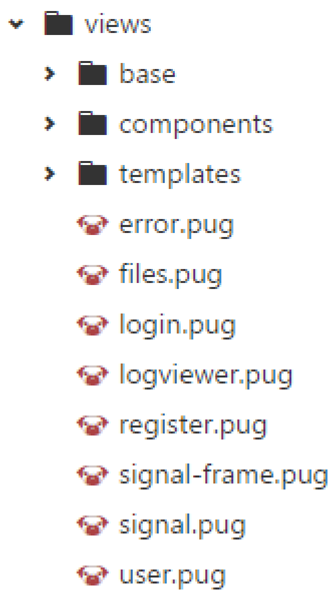


Figure 4.11: Folder-structure of the output, after the building process

4.4.3 Templates

All the pug-templates are located in the folder `./views`. These are accessible by the server and the client. This is possible due to the pug-loader-plugin for webpack. The files in this directory could by considered as the main views. These views include or extend other “minor” pug-files from the three other folders in this directory.
The `./views/base`-folder contains some very basic pug-templates, like the outer html-tag as well as the head-tag.
All the list-related files (e.g. Events, Channels, Comments, etc.) are located in the `./views/components`-directory.
Other stuff like dialog-templates, buttons, directive-templates, etc. is kept in the `./views/templates`-folder.



```
views
  base
  components
  templates
  error.pug
  files.pug
  login.pug
  logviewer.pug
  register.pug
  signal-frame.pug
  signal.pug
  user.pug
```

Figure 4.12: Folder-structure of the views-directory

4.4.4 SCSS

As mentioned above, the CSS-files are pre-compiled by SCSS. This adds many features to the syntax.
The design is based on bootstrap, all bootstrap-overwriting is done in the file `_bs_overwrite.scss`. Style-overwrites of other external style-sheets are located in the file `_overwrite.scss`. The file `_general.scss` holds some simple and small reusable classes, which are the foundation for an atomic design. Bigger style concepts and more complex classes and components are located in the files `_layout.scss` and `_site_specific.scss`.

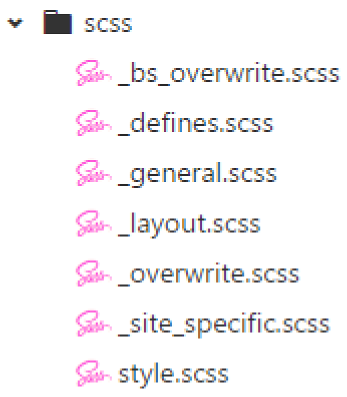


Figure 4.13: Folder-structure of the model

4.5 Datamodel

The data-concept was changed very often, they final model can be seen in Figure 4.14. Note that this figure does not strictly show the model used in the database, it rather shows the data-relations used in the client. Furthermore, there is actually an additional relation between activity and almost all other data-entries. The reason is due the fact, that the activity stores an ID to the related entry.

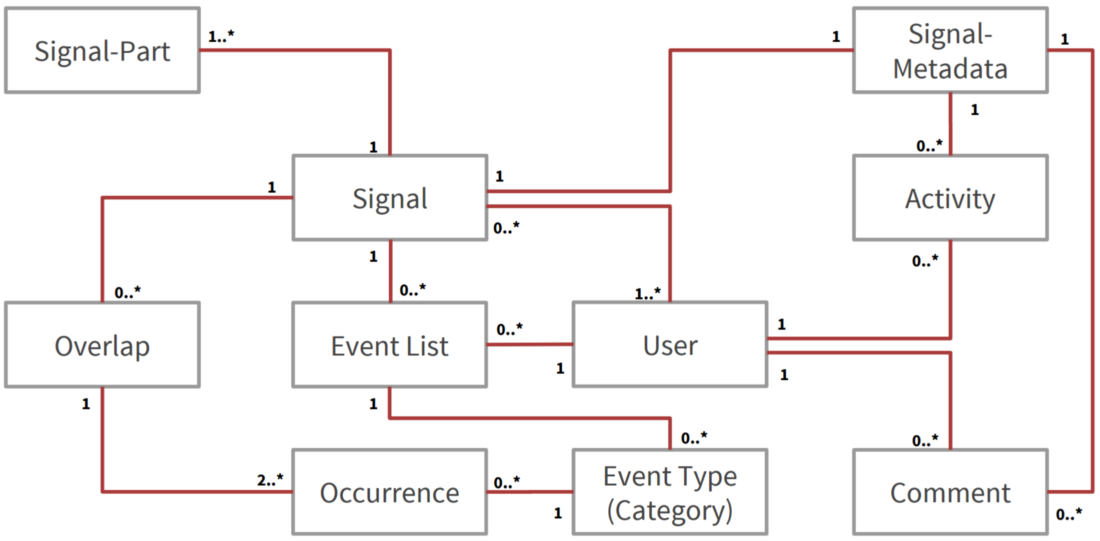


Figure 4.14: Datamodel with multiplicities

Because of some functional reasons, the database stores data slightly differently. There are eight CouchDB-databases, which are used to store the above datamodel.

`signal`

The entries of this database represent the actual signal-files. Originally the datapoints were stored in these entries. Because of the bad performance at loading the signal, these datapoints got chopped into pieces and stored in the database `signal_parts`.
These entries are usually not altered after their creation. They could be considered the static part of a signal-file. An entry in this database has the following format.

`signal_metadata`

Formerly, this was the only separated part of the signal entries, due the fact, that this data is altered very often and could be considered the dynamic part of a signal-file.

`signal_parts`

As already mentioned the datapoints of the signal are choped into pieces. Entries of this database represent one part of those The length of these pieces is exactly 10 seconds. However, this is a arbitrary number, which could be changed easily.

`events`

Each entry in this database either belongs to a user, is generated by the server or is imported from a GDF-file. Events are stored in hierarchical format. The highest container is refereed to as “Event-List” or “Event-Collection”. Each collection has none or many “Event-Types”. These types have in most cases an unique description, because it is enforced by this tools client. The last, member of the hierarchical chain are the occurrences. The most important properties of them are position, duration, channel and user-ratings.

`overlaps`

The entries of the `overlaps`-database are actually collections of overlaps. The reason for this lies in the overlap-detection. Each signal-file has exactly one overlap-collection, which holds a cache as well. This cache is necessary for the detection algorithm. It is used to surveillance overlaps and detect the deletion of occurrences as well as the changing of overlaps.

`users`

This database is simply needed to store user information. Each entry represents one registered user.

`joined_event_data`

By running the join-events-command using the REST-API, this database gets filled with entries. Each entry represents one occurrence with datapoints grabbed from the corresponding signal.

`sessions`

This database is needed for passport to store sessions of the current logged on users.

4.6 Design

This project is trying to take advantage of what is called “atomic design”. Although, the concept was not performed in every template.
Atomic design consists of four or actually five stages. At first, there are the atoms. These are HTML-elements, which cannot be broken down any further, examples are a button or an input-field. These atoms are used to build molecules, which form already useful groups of small UI-elements. This could be for example a search-field with its submit-button. The next stage are organisms, which are constructed using molecules. A good example from SignalCloud is the toolbar at the top of each main-sites, with its specific controls. The penultimate stage of atomic design are complete templates, which consist of all necessary UI-components. This could be for example the signal-site, without a signal loaded. The final stage is the complete rendered site with all its dynamic content. [Frost, 2013]

4.6.1 Design-Elements

The design itself is completely based on Bootstrap, apart from a few modifications. All the icons originate from the popular icon-font[12] "Font Awesome".
Most of the shadows simulate the same behavior like the ones from Google's material design. However, the elements cannot be considered real "materials", because they violate some of the constraints in the guidelines. [Google, n.d.]

4.7 User Interface

A certain structure is the foundation to all visitable pages. These are derived from two main structures.

4.7.1 Centered Panel

Instances of this design structure are the `./login`-page and the `./register`-page. It features already a message system to display simple messages, if for example the login-credentials are wrong.

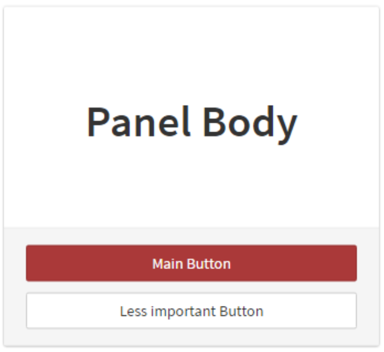


Figure 4.15: Centered panel structure

The same structures is used for server errors, like the "404 Not Found"-message, too.
Moreover, the visible buttons in Figure 4.15 originate directly from bootstrap and are unchanged.

[12] an icon-font is a complete icon-set, which gets loaded into a font-file. The icons can be used like normal characters, which brings many advantages

4.7.2 Main Area with Side-Panels

This user interface is structured in four containers. This structure can be reviewed in Figure 4.16.

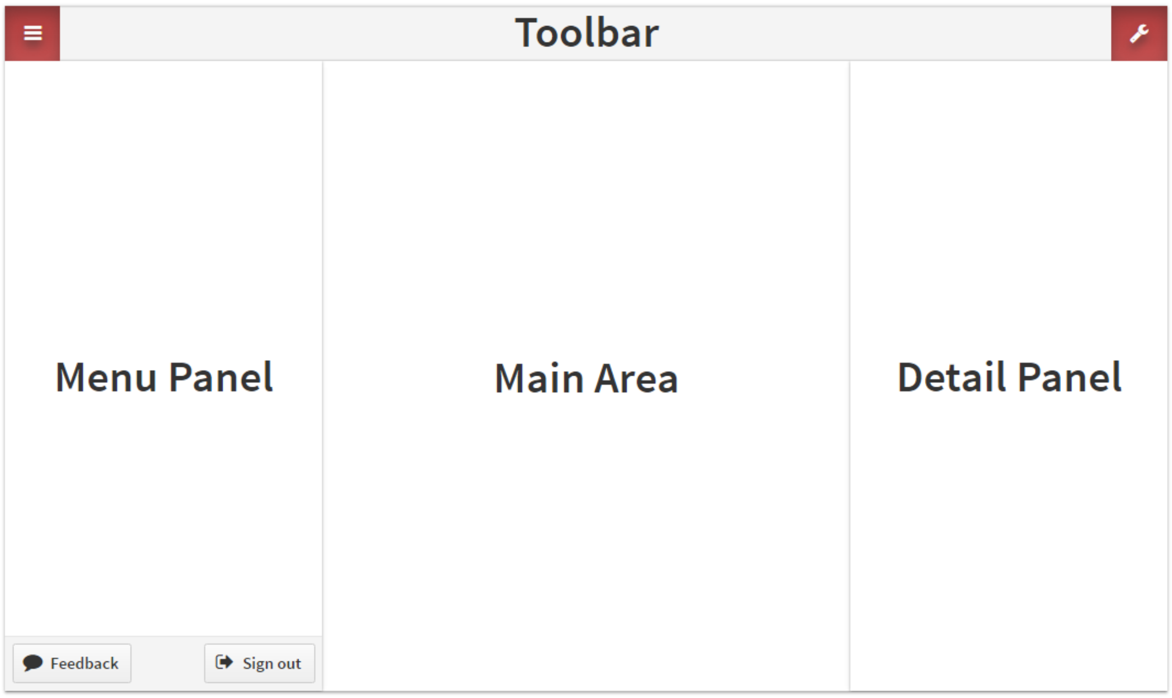


Figure 4.16: Main structure of the user interface

These structure has three sub-structures. One with the menu-panel only, the detail-panel or both of them.

Main Area

This section could be seen as the working space of the current opened context (e.g. signal). This area adopts to the available, if the side-panels are hidden.

Menu Panel

It shows the current user-identicon and its name all the time. Additionally, this panel provides always access to all kinds of navigation (e.g. switching between sites or changing the current view) as well as access to some site-specific actions like for instance inviting another user.

Detail Panel

This panel provides detailed information, hence the name, about either the current selected context. This could be a highlighted file in the files-overview or the opened signal-file itself.

Toolbar

The toolbar allows closing the side-panels as well as changing the active cursor-tool or making some ubiquitous settings.

Chapter 5

SignalCloud

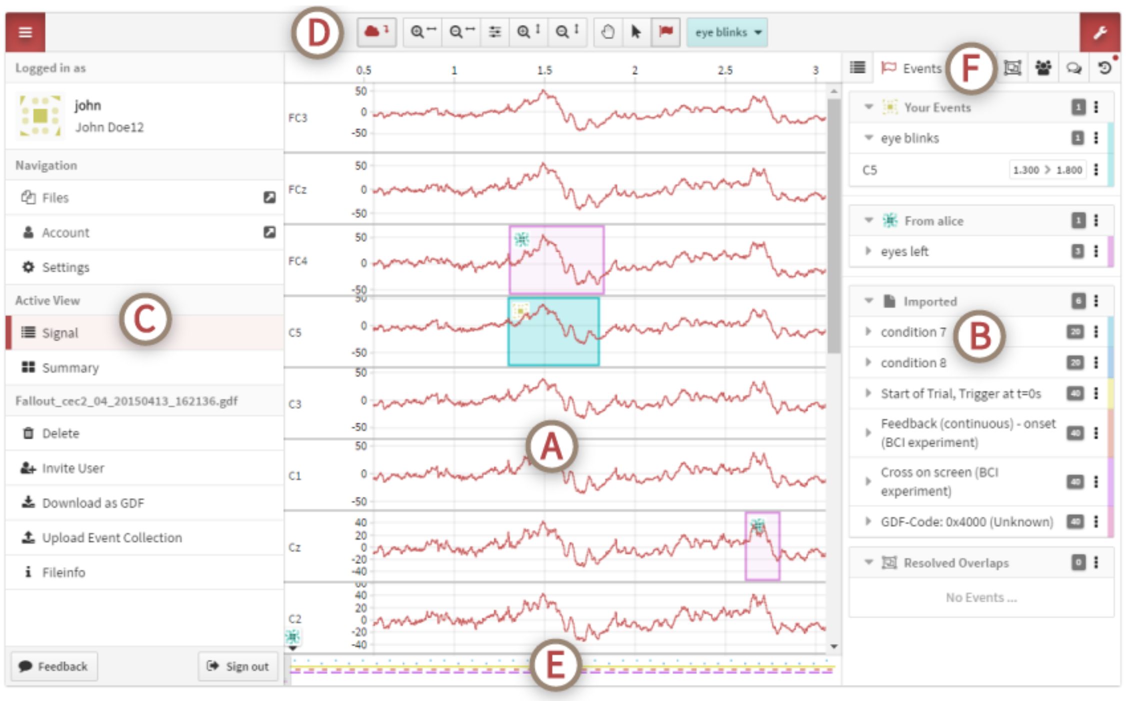


Figure 5.1: User-interface of the SignalCloud-client. Section **A** shows the main area, where annotations are added. Section **B**, the detail panel, represents basically all the metadata of the signal and allows editing them by switching tabs (see focus point **F**). Section **C**, the menu panel, enables the user to navigate around the server or to make some basic actions and preferences. The toolbar (focus point **D**) provides tools and buttons that are often required in the workflow. Section **E** shows the so-called “Enhanced Scrollbar”, which visualizes all event-occurrences within a file and allows an easier navigation.

This chapter illustrates all implemented features in the SignalCloud-service. We start with user and file management. Signal visualization used to display and interact with the signal is introduced thereafter. The last two parts are event tagging and management interfaces as well as collaborative features.

5.1 User and File Management

There are some functionalities in the SignalCloud-client, which could be considered ordinary on server-based tools.

5.1.1 Sign up

If the user is completely new to the system, he has to register first. This can be achieved by using the “Sign Up”-form which is available at the `./register`-route. It can be seen in Figure 5.2.

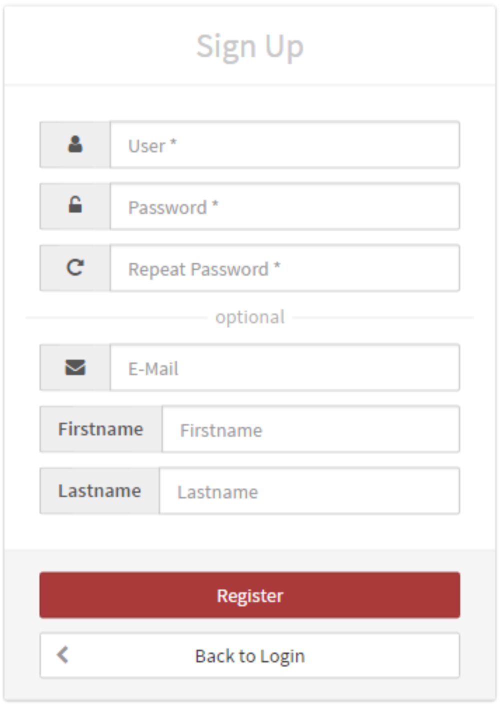


Figure 5.2: Sign up form

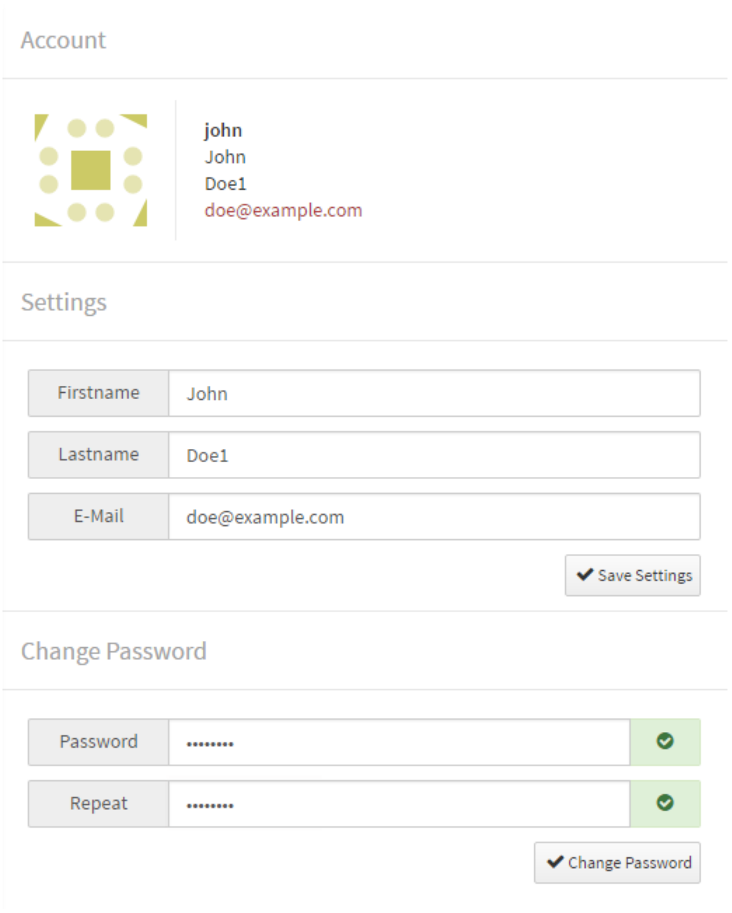
The fields username and password are mandatory. Optionally one is able to reveal the real first- and lastname as well as the email-address. These informations will be available in the “Users”-tab, if a signal has been opened and the user was invited by the owner.

5.1.2 Login/-out

After a user was registered, it is possible to login using the login form. It is accessible at the "./login"-route. After a successful login, one will be redirected to the files-overview.

5.1.3 Account Settings

Quite a default feature of online-based-services is a page for making changes to the user-account. That is why SignalCloudprovides this functionality as well. The site can be accessed by clicking on the button called "Account", which leads to the route `./user`.
Some user-related information like the first- and lastname or the email-address can be added subsequently by using the forms shown in Figure 5.3. Additionally it is possible to change the current password.


Account

john
John
Doe1
doe@example.com

Settings

Firstname	John
Lastname	Doe1
E-Mail	doe@example.com

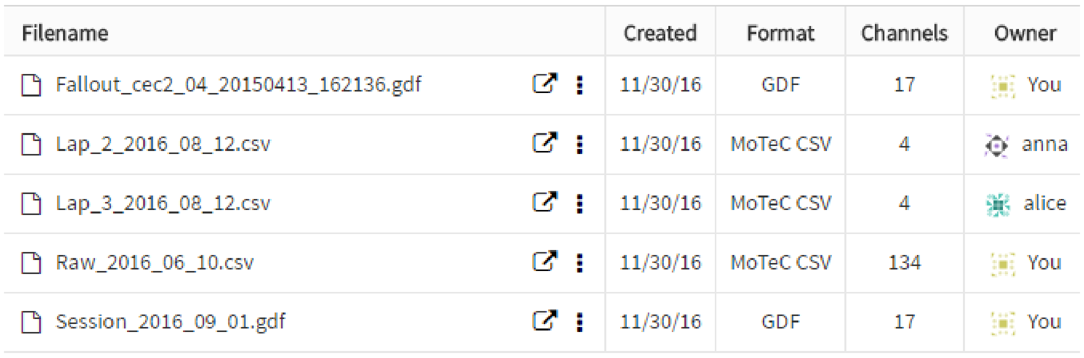
Change Password

Password	••••••••
Repeat	••••••••

Figure 5.3: User Settings

5.1.4 Files Overview

After a file has been uploaded, it will be automatically added to the files overview. This list is filterable through the input-field, located in the toolbar. Most of the information is already given by the rows itself (Figure 5.4). Furthermore, a single click on one of the rows reveals even more information in the detail-panel of the files overview. A double click or a click on the open-button, will redirect the user to the chosen file.
The overview is the first page, a user gets redirected after a successful login. It is available at the `./files`-route.



Filename	Created	Format	Channels	Owner
Fallout_cec2_04_20150413_162136.gdf	11/30/16	GDF	17	You
Lap_2_2016_08_12.csv	11/30/16	MoTeC CSV	4	anna
Lap_3_2016_08_12.csv	11/30/16	MoTeC CSV	4	alice
Raw_2016_06_10.csv	11/30/16	MoTeC CSV	134	You
Session_2016_09_01.gdf	11/30/16	GDF	17	You

Figure 5.4: Some example files

The `./files`-route is the only place for uploading new files to the database. This is done by clicking on the button “Upload File” in the menu panel.

5.1.5 File-Upload

This is nothing new for a web-based tool. However compared with the SigViewer this is a huge advantage. There is no need to transfer files with hard-drives, usb-sticks or other means. The file is simply uploaded to the database and is available for all persons, who need an access. There is no installation needed to view the file as well.

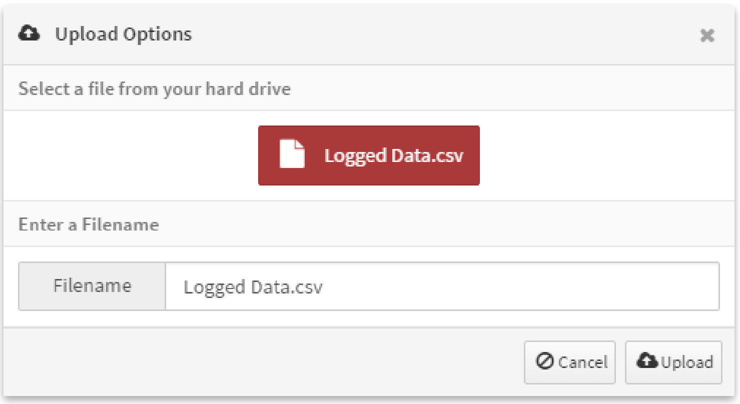


Figure 5.5: Upload Dialog

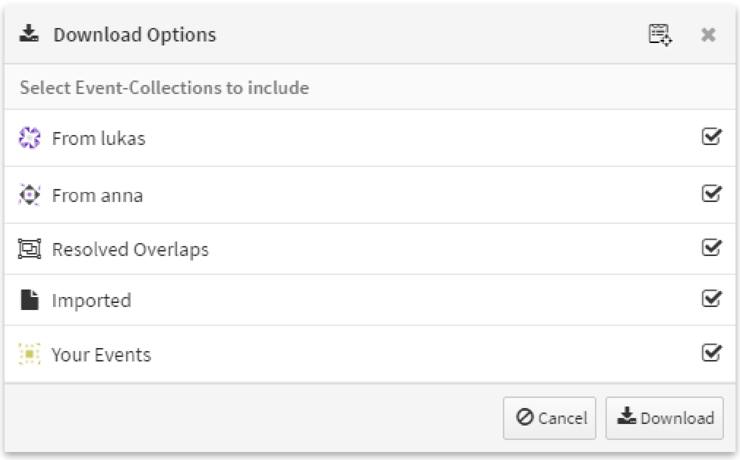
The used Node.js-framework express features already many functionalities to handle post- or get-requests. The additional plug-in “multer” makes it really easy to handle uploaded files. These files are stored in a temporary directory, which gets created at start of the server. It is located under `./uploads/temp/`. After the file is successfully uploaded, it is parsed by the corresponding parser, which in this prototype-state could be either the GDF-parser or the MoTeC-CSV-parser.

5.1.6 File-Removal

A file can always be deleted by its user after it was uploaded. This can be achieved either by using the context-menu in the files-overview on one of the rows or directly in the signal-view by clicking on the button called “Delete”. This will need a moment, because the server is searching for all database-entries, which belong to the selected file.
Note that this will not delete any joined-events belonging to the deleted file, which where created using the REST-API.

5.1.7 File-Download

What is uploaded also needs to be downloaded. Therefore a download-feature had to be implemented into the tool. The need of this functionality is drastically increased by the current state-of-the-art tagging-work-flow for eeg-files. Because of the many analysis, which are carried out after the labeling process, it is necessary to retrieve the file in its GDF-version. Thereby the labeled signal-file can be processed by post-processing tools or MATLAB.


Download Options

Select Event-Collections to include

From lukas	✓
From anna	✓
Resolved Overlaps	✓
Imported	✓
Your Events	✓

Figure 5.6: Upload Dialog

Technically is this feature basically the inverse operation of the upload-parser. The signal needs to be packed into one file again and sent to the user. By directly opening the server-route `/download?id=...&format=gdf`, the file can be downloaded as well. Even so it cannot be configured as well as by using a post-request started by the shown dialog in Figure 5.6.
Note that the http-headers have to be modified to force the browser downloading a file. This can be done by setting the header `'Content-Type'` to the value `'application/force-download'`. Another header can be used to set the filename of the downloaded file. In this case the filename, which was entered at the file-upload is taken.

5.2 Signal Visualization

There are multiple visualization-techniques used in the SignalCloud-client.

5.2.1 Signal Plot

Probably the most important feature. Therefore this functionality has to be well considered. The drawing process has to be fast enough, in order to not annoy the user. Since the drawing has to be done after each event-addition as well as on each change of the zoom-level.
The visual representation of the signal is based on the SigViewer. Each row holds one of the channels and the clipping of the signal is configurable by scrolling left or right. Moreover it is possible to navigate within the signal by using the “Navigation Tool” located in the toolbar. After its activation it is possible to move the section by dragging on the line-chart.

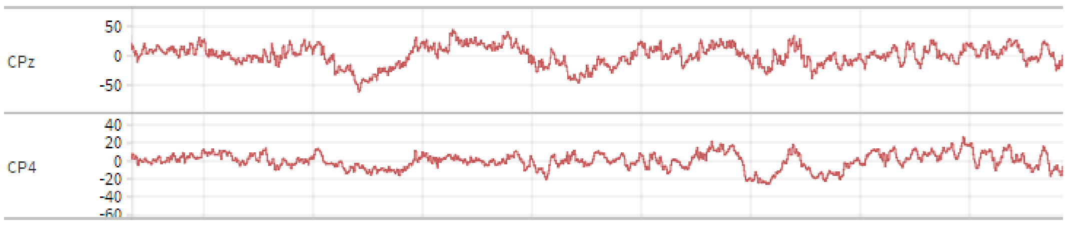


Figure 5.7: Example of two channels

The visualization displays the channel name on the left side. Additionally a scale for the y-axis is shown on this side as well. The corresponding scale for the x-axis is shown on top of all channels as it is valid for all of them.
By default the interpolation mode is "step", which means there is an additional rise or sink after each data-point. If this behavior is not desired, it can be turned off, switching the interpolation mode to linear, which can be achieved in the "Settings"-dialog.

5.2.2 Channel-Configuration

Each channel of a signal is completely customizable to everyones personal needs. There are various functionalities to support the different requirements of each user.

Zooming/Rescaling

Obviously needed to view the signal at varying scales. One has to differentiate between the horizontal and the vertical zoom. The first one changes the px per second ration, the other one increments or decrements the height of all channels.
The levels can be changed by clicking on the corresponding button (Figure 5.8), with the magnifier-icon, located in the toolbar. Or by clicking on the sliders-icon in the center of the zoom-button group, which shows a popup-panel holding sliders for reaching the desired zoom-level in a faster way (figure 5.9).

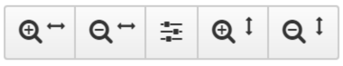


Figure 5.8: Buttons for controlling the zoom-level

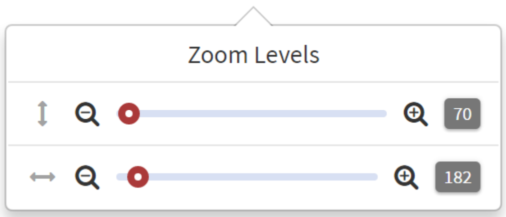


Figure 5.9: Sliders for a faster manipulation of the zoom-level

Reordering

If the original channel-order from the imported file does not fit the users needs, it can be changed by dragging the channels in the desired order. Therefore one has to open the tab “Channels” in the detail panel. After the three lines at the left site of each channel-item act as a grip or handle for the dragging process.

Merging

In some cases it might be useful to overlay two channels on top of each other. Therefore this functionality can be activated by using the context-menu on a channel-item in the detail-panel and selecting the option “Merge”. After hovering over it, another context-menu will become visible. It will display a list of all available un-merged channels. A merged channel will appear, after clicking on one of the options. Figure 5.10 shows a merge of three channels.

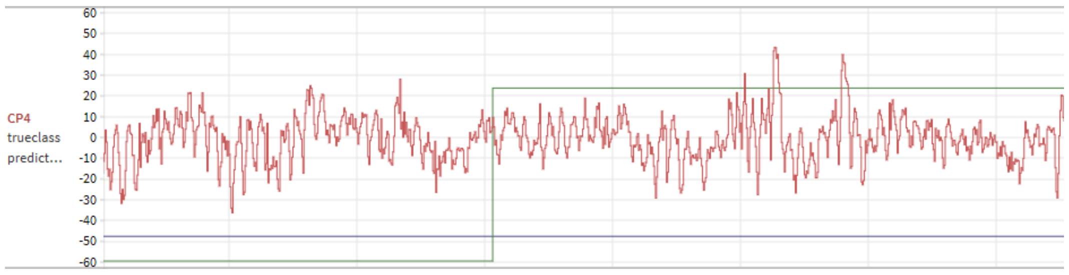


Figure 5.10: Output of three merged channels of an eeg-file

The list items, located in the detail panel merge as well, to represent the newly created line-chart. This can be seen in Figure 5.11.

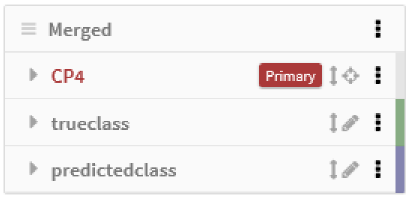


Figure 5.11: Corresponding channel-list-item after the merging-process

Visibility

Unnecessary channels can be hidden, by using the context menu to select the option “Hide”. After that the whole channel will vanish. Additionally the list-item located in the detail-panel will show an eye-icon to represent the hidden channel (Figure 5.12). If the user decided to show it again, this is possible by using the context-menu again and selecting the “Show” option.

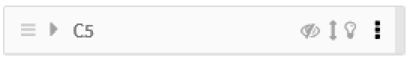


Figure 5.12: Example of a hidden channel-item

Color

If a user wants to highlight a specific channel, this can be done by using a different color for its line. To do so, the user has to open the context-menu of the desired channel and select the option “Color...”. This will open a dialog-window, where a color can be chosen using a HSL-based color-picker.

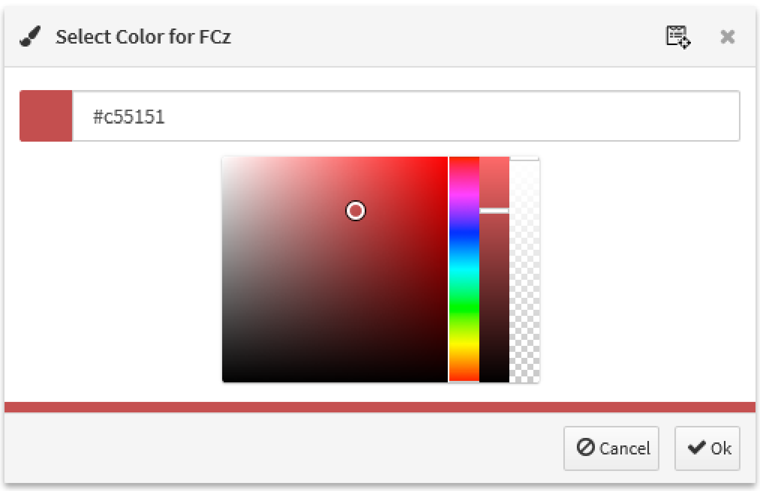


Figure 5.13: Dialog-window for selecting a channel-color

Scale

Especially needed for merged channels. This functionality lets the user decide, how the y-axis of the line should be adapted. At first the default option “Best fit” is selected for each channel. If it is chosen, the domain of the y-axis is defined by the minimum and maximum value of all datapoints. Moreover, there is another option called “Zero centered”, which uses the extreme values from before, chooses the one with the greater absolute value and mirrors it on the negative side of the axis. Trough theses operations the result will be a centered line-chart.
The last option is called “Manually...”, this opens a dialog-window with two input fields for a maximum and a minimum value.
The icon on the left side of a list-item shows, which scaling is currently active. The example item in Figure 5.12 holds the icon for the “Best fit”-option.

5.3 Event tagging interfaces

The tagging of artifacts is the main-purpose of the client. One is able to start tagging by clicking on the “Add Event Tool” in the toolbar. After its activation it is possible to add events by clicking and dragging on a channel in the main area. If the cursor gets near the end of the main area, it will start to scroll in this direction.
The color of the brush[1] depends on the last selected event-type. The color is derived from the description-string of the event-type. This means, that each type has exactly the same color on various signals or even formats. However, colors are limited, so some of the types might have a similar color.

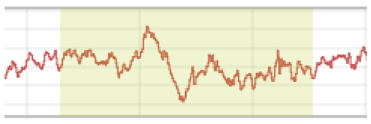
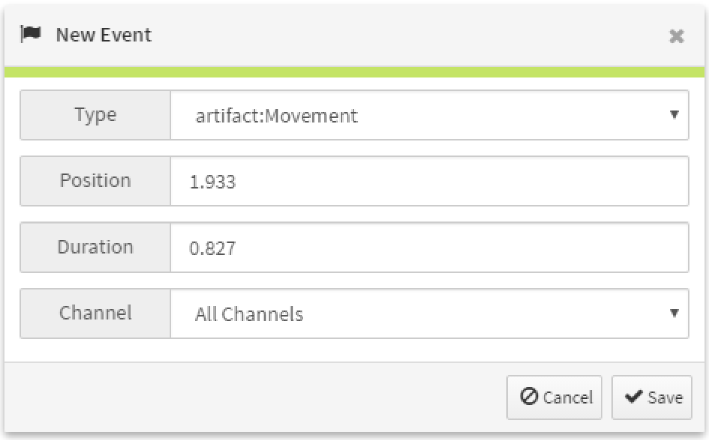


Figure 5.14: Preview of a new event

After the mouse button has been released, a dialog-window will pop up and show a summary of the current drawn event. The position or size of the event can be modified by changing the numbers in the input-boxes. It is possible to change the channel or event-type as well. An example of the dialog-window can be seen in Figure 5.15.

[1]rectangular box, which previews the size and position of the new event


New Event

Type	artifact:Movement
Position	1.933
Duration	0.827
Channel	All Channels

Figure 5.15: Dialog-window for selecting a channel-color

If there are many artifacts of the same type, it might be useful to define a preset for the "Add Event Tool". This can be done by clicking on the button right next to it, after activating the tool (Figure 5.16 and 5.17).

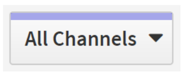
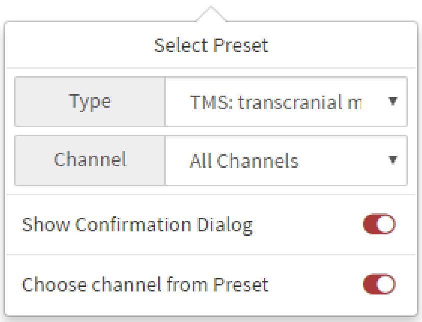


Figure 5.16: Preset-button (channel and type)

Figure 5.17: Preset-button (type-only)


Select Preset

Type	TMS: transcranial m
Channel	All Channels
Show Confirmation Dialog	
Choose channel from Preset	

Figure 5.18: Preset-panel with settings

A popup-panel will be shown (Figure 5.18) after a user clicks on the preset-button. This panel allows multiple settings. The "Show Confirmation Dialog"-switch will prevent the dialog-window from showing up, after each new event addition. Furthermore, it is possible to choose the channel from the preset-panel instead of the drawing position. This feature might be interesting, if someone wants to draw many "All Channel"-events. It could be useful as well, if the user wants to select an artifact in one channel, but wants the event to appear in another. Based on the activated switches, the preset button will change its appearance and show for example the current selected channel directly by its name and the type through a color bar at the top.

5.3.1 Visual Editing of Tags

After some events have been drawn, one might want to change them again. This can be done by simply clicking on a drawn event. Afterwards it will receive a shadow and the cursor will change to an arrow-cross. Moreover it is now possible to drag the event to a new position. This involves changing of the current channel as well, unless the event is an “All Channel”-event. Additionally the size of the event can be changed by dragging the borders on a horizontal axis.
As soon as the cursor leaves the main area, the event will go back in its non-editable mode.

5.3.2 Overlap-Detection

A very interesting feature is the automatic detection of overlapping events. This is implemented on server-side, therefore users will experience a short delay (a few seconds) between a new tag and the creation of a new overlap. It is added to the tab “Overlaps” located in the detail panel. Figure 5.19 shows an overlap thumbnail, it consists of combined information about the events that overlap in the same position in the signal file. The item shows a small thumbnail of the overlapping area. If there are more than one channels involved, the data-points of the first one are selected for drawing the line-chart-thumbnail. Additionally color-bars, which are stacked from the bottom, represent each occurrence of an event. Its color is defined by the type of the event. Moreover, the occurrences are represented as well by the user-identicons on the left side. It is possible to scroll directly to the event by clicking on one of them.

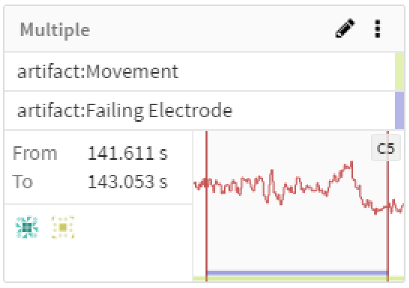


Figure 5.19: Overlap list-item

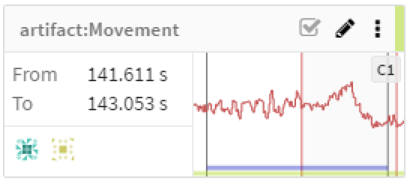


Figure 5.20: Overlap list-item after its resolution

An overlap can originate in a number of conflicts between events tagged by different users: type of event, channels, duration, beginning - end. Clicking on the pencil-icon, a user can edit the overlap (see Figure 5.21). Once the edit mode is entered, the user can change the type of the overlap. If there is a type conflict, the title will be "Multiple" otherwise the type is used for the header. If there are at least two channels in different tags that overlap, the channel can also be chosen. Finally, it is possible to change start position, end position and duration by moving the opaque box at the bottom of the dialog. This will set the position and duration of the consensus of all involved occurrences.

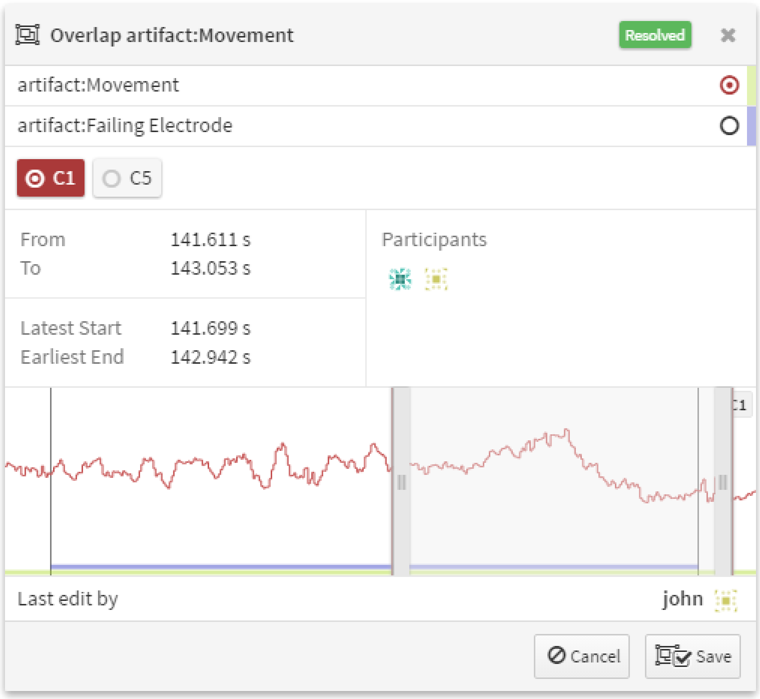


Figure 5.21: Dialog-window for editing an overlap

If there are neither “channel-conflicts” nor “type-conflicts”, the overlap will be resolvable, which enables a new button on the left side on the bottom of the dialog-window called “Save and Resolve”. This button allows the user to add this overlap to the “Resolved”-section of the list. After the resolution, a new event will be added to the collection “Resolved Overlaps” in the “Events”-tab. It can be seen in Figure 5.22. If the user wants to see only the one resolved occurrence, it is possible to hide all other occurrences by using the context-menu on the involved overlap and selecting the option “Occurrences → Hide all”.

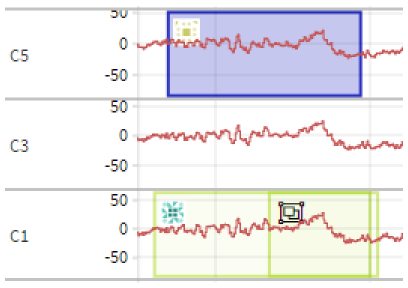


Figure 5.22: Corresponding plot-area to the overlap in Figure 5.19 and 5.20

5.3.3 Event-Collection-Upload

It might be useful in some cases to include the output of an algorithm to the online-version of a signal. The feature is accessed by pressing the button “Upload Event Collection” in the menu panel. This will open an upload-dialog-window, where it is possible to select a GDF-file. This feature is for GDF-files only, because the MoTeCCSV format includes nothing similar to tags or events. After a file was chosen, one has to type a name for the new collection. The action is completed by clicking on the button “Upload”. Note that the collection name has to contain more than two characters, otherwise the upload-button is invisible.
The site is reloaded if the button is pressed, because it was technically the easiest solution to get the events uploaded and show them to the user. If other users are currently viewing the same file, they will simply get an update.
There are certain issues connected to the upload of additional events. For example, some of them could belong to a channel, which does not exist in the current signal. If this is the case the event is simply transformed to an “All-Channel”-event. Furthermore an event could start outside the temporal domain of the signal. This problem still exists and is ignored in the current version of the SignalCloud-server.

5.3.4 Enhanced Scrollbar

It can be hard to navigate in a really big eeg-file. Therefore we came up with an idea to help users find certain hot-spots of the signal-file. The enhanced scrollbar allows users to see, where events of a specific type are concentrated. By looking either at the color or the row of an occurrence the type can be determined.
By looking at the opacity of the color, one can see the density of events as well, but only for one type.
To make it easier to perceive the information shown by the bar, it expands on mouse over. Each row in the bar stands for one type, this means the bar has to grow with each new type. However, there has to be a maximum for this height. Furthermore a minimum-height is needed too, because it replaces the default bottom scrollbar.

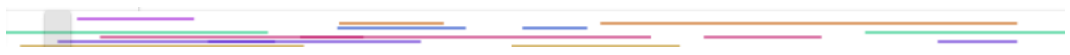


Figure 5.23: Enhanced Scrollbar with some event-occurrences

This interface-component was implemented without the use of any external drawing-libraries, except jQuery. It is a simple HTML5-canvas, which gets newly drawn on every event-update. Its expanded version has to be redrawn as well every-time the mouse enters. However, the canvas element is very performance-efficient, so that the many drawing-updates are no problem at all.

5.3.5 Summary View

There are two possible views of a signal. On one hand there is the “Signal”-view, which simply shows all channels of the signal in rows. On the other hand there is the so-called “Summary”-view, which is constructed by summary-modules. The user is always able to switch between both views by clicking on the corresponding button in the menu panel (Figure 5.24).

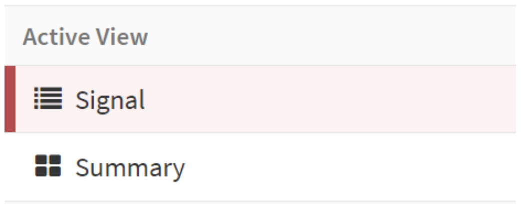


Figure 5.24: Buttons for switching the current view

Event-Type Heatmap

For some applications it could be useful to see where exactly the highest concentration of specific event types lies. The Event-Type Heatmap is a summary module that does exactly this.
The summary module draws a heatmap of all event-types. There is one row for each different event-type of a signal. The x-axis is separated into bins. Occurrences for each event-type are counted and assigned to each bin. The color intensity of a bin indicates, how many occurrences are concentrate in this specific time-unit. The bin-size is adjustable in the detail-panel. Additionally, it is possible to adjust the color of the event-types to red, for an easier comparison among types.

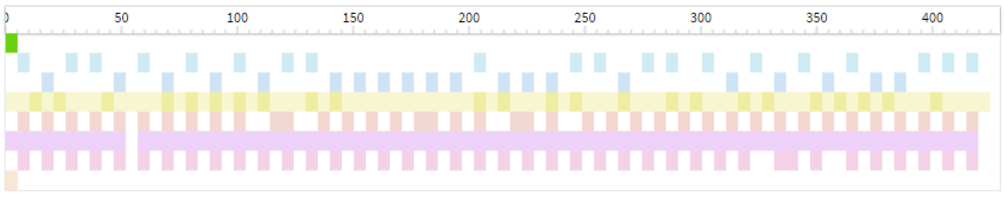


Figure 5.25: Heatmap with event-type-colors

Occurrences per Event-Type and Channel

An usecase scenario of this module could be the following: After an EEG file was successfully reviewed, other people might be interested on how an occurrence of a specific type looks like (the "ground truth"). This module is a rather trivial approach to show this "archetype" of an event. It plots the datapoints of all occurrences of a specific type per channel.
Optionally the plots can be rescaled to fill the whole width of the linechart. However, this means each line has a different scale and might be not comparable anymore.
An additional green line shows the average of all involved occurrences.
Moreover, the opacity, used to draw the lines, depends on the amount of occurrences. This can be seen by comparing Figure 5.26 and 5.27.

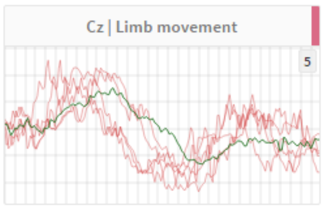


Figure 5.26: Five plotted occurrences for a specific event-type and channel

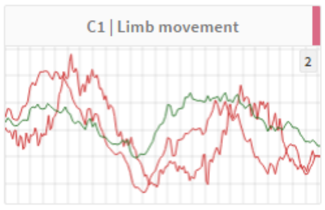


Figure 5.27: Two plotted occurrences for a specific event-type and channel

Resolved Overlaps

This is a very simple module, it shows linechart-plots of all resolved overlaps (the consensus).

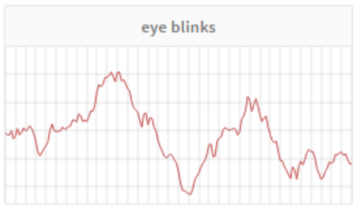


Figure 5.28: 2 plotted occurrences for a specific event-type and channel

5.4 Collaborative Features

This section concentrates on the features that make SignalCloud collaborative. These awareness features include: showing other users working in the file, the positions where users are working and a history of their activities.

5.4.1 Event-Reception

For some use cases it could be desirable to get the unbiased opinion of certain or even all users. To do so, one is able to use the reception-toggle. Each user has access to the reception toggle for themselves. By switching it off it is possible to hide all events from other users, even future ones and work undisturbed. This will hide their event-collections in the detail-panel as well. Additionally scroll-indicators at the bottom of the main-area will be hidden too.

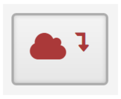


Figure 5.29: Event-reception-toggle (On)

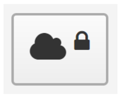
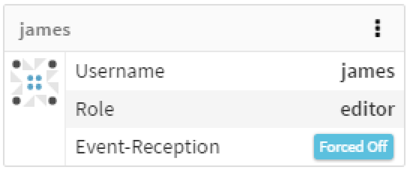


Figure 5.30: Event-reception-toggle (Off)

To ease the administrative work of the owner or any admin and to provide certain initial conditions, it is possible to remote control these buttons. This is done by using the context-menu on user-items in the “Users”-tab of the detail panel and selecting the option “Reception → Allow” or “Reception → Disallow”.



james

Username	james
Role	editor
Event-Reception	Forced Off

Figure 5.31: User without event-reception

After the reception of a specific user has been “disallowed” by an admin, the user her- or himself cannot turn it back on. Only the owner or admin is authorized to turn synchronization back on for that user.

5.4.2 User-Management

A new user can be invited either by clicking on the button "Invite User" located in the menu-panel or by using the context-menu in the "Users"-section of the detail-panel.

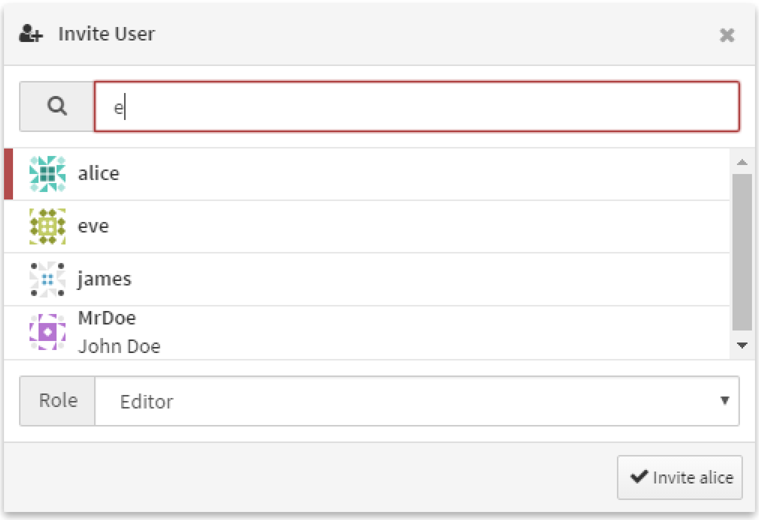
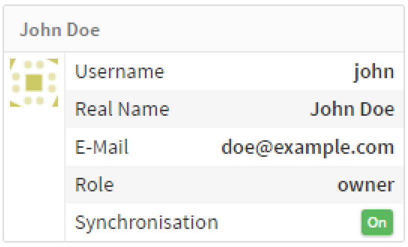


Figure 5.32: Dialog used for invitations

The dialog shows a filterable list of all registered users on the server. Moreover, it becomes only visible after typing at least one letter. The list is fetched from the server as soon as the dialog opens. By opening the select box at the bottom it is possible to change the role of the user. "Editor" is always the default option.

Now let us take a look on the technical site of the user-management. At first, we have to review the database-structure again. The user-list per each file is located in its metadata-entry in the database `signal_cloud_signal_metadata`. Furthermore its a JSON-object, which means there is a key-value storage. Each key is defined by the user-ID of the entry, but it is available through the field `uid` as well. In addition there are fields defining the role and the event-reception. Other user-related information is added through database-join-operations, if needed.
The metadata-entry of a signal is one of the first things, which are retrieved from the database via the websocket. Subsequently the user-list located in the detail-panel by activating the tab "Users". An example-entry can be seen in Figure 5.33.



John Doe

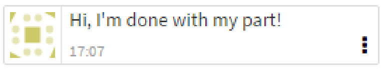
Username	john
Real Name	John Doe
E-Mail	doe@example.com
Role	owner
Synchronisation	On

Figure 5.33: Example for an user-entry

There are two entries in the structure, which can be changed, in fact these are the role and the event-reception. Although it is not directly possible to switch the role of an user. Instead the individual has to be uninvited and invited again using the desired user-role.

5.4.3 Comments/Chat

It is easier to collaborate, if all concerned persons are in the same room, for obvious reasons. This proves to be more difficult, if the individuals are separated and have to provide the same performance.
To lessen the difficulty of this collaboration, a chat-functionality was added to the tool. It can be used to broadcast messages to all collaborators of one file. The messages are stored in the database and assigned to only one signal-file. Furthermore it is possible to delete a message by using the context-menu. However this is only achievable by the author of the message.



Hi, I'm done with my part!

17:07

Figure 5.34: Example for a comment

The comments are located under the corresponding tab in the detail panel. New comments are added by using the input-field at the bottom of the tab, which can be seen in Figure 5.35.

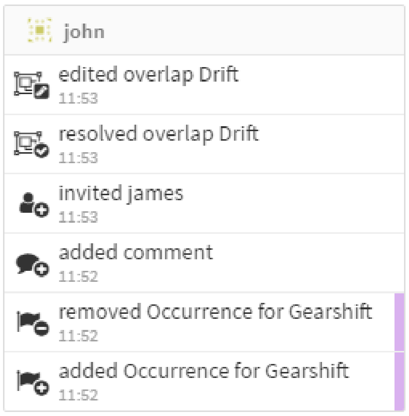
Figure 5.35: Input-field for new comments

Figure 5.34 shows already a good example of an use-case of the comment-functionality. At first I was thinking about some kind of "Done"-button to show other users that one has finished. However, a chat is anyway an important tool for making collaboration easier. Therefore users could simply use to chat to signal, that they have finished tagging. Thus the need of a "Done"-button was not given anymore.

By taking a look on the database-structure, one can see that the comments are stored in the database `signal_cloud_signal_metadata`. There is an javascript-object called `comments`, which is responsible for holding all the comments of a signal. Keys of this object are uuids, which only have to be unique around the comments of one file. The value consist of another object holding four records, which are a copy of the comment-ID, a timestamp of the creation of the comment, the user-ID of the author and finally the message itself.
The metadata-entry for a specific file gets loaded as soon as the user retrieves the corresponding signal-file. It is accessible for the client directly after the initialization. However, there are a couple of data-operations, which have to be done, before the messages are ready to be presented. At first the timestamp has to be converted into a javascript-date-object. After that it has to be sorted by the newly created date. Finally a grouping is done, to associate comments with days. The current day and the day are translated as "today" and "yesterday". Other days are shown by local defaults.

5.4.4 Activity/History

For an asynchronous collaboration it is necessary to see, what has happened during the time a user was "offline". Consequently, some sort of summary is mandatory to show this information. Therefore the "Activity"-tab was added to the detail-panel. It keeps track of all things, that happened to the file since its creation. Moreover it is sorted by a time-stamp and grouped by day. In addition consecutive actions of the same users are grouped as well. There are main-icons, which show the category of the action and sub-icons, which represent the action itself (e.g. add, remove, etc.).


john

edited overlap Drift
11:53

resolved overlap Drift
11:53

invited james
11:53

added comment
11:52

removed Occurrence for Gearshift
11:52

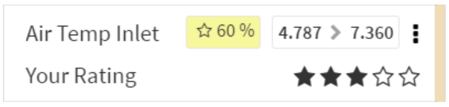
added Occurrence for Gearshift
11:52

Figure 5.36: Group of consecutive actions of a user

Clicking on one of the list-items allows the user to traverse to the involved item. This is obviously only possible, if the item was not removed. If this is the case, the involved tab will simply open without a transition to the element.
In addition to the “Activity”-tab, one can see immediately changes by looking at the small red dots on the right top of each tab-button. A dot will every time appear, if the number of list-items changes, because of an action of another user.

5.4.5 Rating of Events

In some cases, it might be useful to rate the output of either an algorithm or another user. This is done by using the star-buttons on the bottom of each occurrence-item in the “Events”-tab. The rating can be a number from one to five.


Air Temp Inlet ☆ 60 % 4.787 > 7.360

Your Rating ★★★☆☆

Figure 5.37: Event-occurrence with rating

If there is at least one vote, a score in percent will be displayed. The ratings of other users are hidden behind the average of all ratings, there is no visual way to view all individual ratings. However, it is possible to access them by the API.

5.4.6 Scroll-Indicators

For a collaborative software or tool it is necessary to be aware, what other users are doing currently. This is a non-trivial problem, because there is a fine line between informational and annoying. Not every mouse-move or keyboard-input has to be shown to others.
The user awareness in this tool is limited to a few, but meaningful, components. One of those is the scroll-indicator located at the top of the enhanced scrollbar (Figure 5.38).

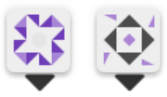


Figure 5.38: Indicators from two different users

These indicators are capable of showing the current x-axis position of another user. The reference to it is given by the enhanced scrollbar, which shows the whole signal. The indicators moves accordingly, if a user moves horizontally in the file. However, there is no indicator showing the vertical scroll-position of other users, because this information was considered unnecessary.

The technical basic system needed for this features is in general the websocket itself. After each scroll update a callback is executed, which broadcasts a message holding the current relative scroll-position of the user. Additionally to the position, this message holds the user-ID as well.
The broadcast itself is done by the server. Actually the word broadcast is misleading, because it is more comparable with a multicast, selecting only certain participants. Those are selected, who subscribed the same socket-io-channel, which is by default the signal-ID of the currently opened signal-file.
After another user receives the message, it is parsed by the client. If the user-ID is somehow not flagged as “online”, then it is taken care of now. The relative position (between zero and one) of the other user is scaled upon the width of the main-area.
There is a small timeout between each position update to reduce network traffic. Otherwise the server would be occupied too much with sending position updates.

5.5 Summary

This work has concentrated on enabling collaboration in signal analysis. We started with simple single user features. But, once collaboration was available, it became necessary to deal with two major challenges that arise when working in groups:

- Finding a common ground on certain aspect.
- Having possibilities to define the collaboration strategy.

This thesis offers methods to address these issues in collaborative analytics tools. The overlap detection and resolution interface helps administrators handle situations where users have different opinions on the events. The event voting features allows participants to express what they think of a particular label. With regards to the strategy, we have concentrated on enforcing different modes of collaboration.

Chapter 6

Conclusion

At the end, the resulted software worked as intended The drawing performance is fast enough to be not annoying and the planned features work as intended.
However, there is always much space for improvements on every corner.

6.1 Frameworks

Some of the frameworks make sense for a prototype, but there are better solutions which could improve the performance for an actual production version of this tool.
For example, the disadvantage with CouchDB is that all the join tasks have to be run by Node.js. This could be done very easily with a SQL-database and probably with a better performance as well.
Furthermore, using the native scroll overflow for the signal chart brings certain advantages. However, it could lead to a better performance to switch to chart.js or other external libraries, with an emphasis on drawing millions of datapoints.
Another, rather small, improvement might be possible by switching from AngularJSto react.JS, as it features a higher performance in many tests. [Harrington, 2015]

6.2 Future Work

Althoug, many features have been implemented in SignalCloud, some of them never left the concept phase.

6.2.1 Undo/Redo

An ubiquitous improvement example would be an undo-feature. This could be useful if someone deletes an event accidentally and wants to restore it again. Actually, the feature would not be that hard to implement because data is not changed by overwriting, but rather by making distinctive change or delete requests. If one would record these requests and basically “play then backwards” upon an undo-request, it could lead to the desired behavior.

6.2.2 Summary Modules

In general, there could be much more summary modules. These could for example show the frequency domain of the signal or show a better distribution of the event types.
It should be rather easy to add additional summary modules, using AngularJS-components.

6.2.3 Realtime Updates

A really interesting feature would be realtime updates of the signal itself. Applications for this functionality could be all kind of remote diagnosis. It could be implemented by extending the analyzer-module of the client, as this module loads the signal anyway asynchronously.

6.2.4 Linked IDs in Comments

The last of the proposed unimplemented features are links in comments. These links enable a transition to a certain item in the main area or the detail-panel (e.g. event-occurrence, overlap, user, etc.). The transition could be similar to the one which gets started by clicking on an activity item.

6.3 Summary

While a single user application may have fixed functionality to a task, the introduction of collaborative features makes it necessary to manage situations where participants have different opinions. Much of the added value from collaboration comes from the sharing of opinions and viewpoints. This thesis has developed a tool for collaborative signal analysis. The major challenges were on allowing users to choose the strategy how they wish to work together. Also sharing knowledge and finding a point of common ground in different opinions were topics of development. In general collaboration gains a lot of attention lately with tools that allow users to communicate, share and improve their working methods in the interaction with others.

List of Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
COP	Coefficient of Performance
CSS	Cascading Style Sheets
CSV	Comma-separated values
DB	Database
DOI	Degree of interestingness
DOM	Document Object Model
ECG	Electrocardiogram
ECMA	European Computer Manufacturers Association
EEG	Electroencephalogram
GDF	General Data Format for biomedical signals
GPL	General Public License
GUI	Graphical User Interface
HAML	HTML abstraction markup language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
NPM	Node Package Manager
PCA	Principal Component Analysis

PIP	Perceptually Important Points
RDF	Resource Description Framework
REST	Representational State Transfer
SASS	Syntactically Awesome Style Sheets
SCSS	Sassy Cascading Style Sheets
SOM	Self-Organizing Map
SPARQL	SPARQL Protocol And RDF Query Language
SQL	Structured Query Language
SVG	Scalable Vector Graphics
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
WYSIWIS	What You See Is What I See
XML	Extensible Markup Language

Bibliography

Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, and Christian Tominski. Visual methods for analyzing time-oriented data. *IEEE transactions on visualization and computer graphics*, 14(1):47–60, 2008.

American Heritage. *The American Heritage Medical Dictionary*. American Heritage Dictionary. Houghton Mifflin Company, 2008. ISBN 9780618947256. URL `https://books.google.at/books?id=kT7ykAhh3fsC`.

M. K. Brown, B. Huettner, and C. James-Tanny. Choosing the right tools for your virtual team: Evaluating wikis, blogs, and other collaborative tools. In *2007 IEEE International Professional Communication Conference*, pages 1–4, Oct 2007. doi: 10.1109/IPCC.2007.4464100.

Christopher Collins, Fernanda B. Viégas, and Martin Wattenberg. Parallel tag clouds to explore and analyze facted text corpora. In *Proc. of the IEEE Symp. on Visual Analytics Science and Technology (VAST)*, 2009. doi: 10.1109/VAST.2009.5333443.

Brad Frost. Atomic design, 2013. URL `http://bradfrost.com/blog/post/atomic-web-design/`. (Accessed on 12.12.2016).

F. A. Gibbs, H. Davis, and W. G. Lennox. The electro encephalogram in epilepsy and in conditions of impaired consciousness. *American Journal of EEG Technology*, 8(2):59–73, 1968. doi: 10.1080/00029238.1968.11080707. URL `http://www.tandfonline.com/doi/abs/10.1080/00029238.1968.11080707`.

Google. Material design guidelines, n.d. URL `https://material.google.com/`. (Accessed on 07.12.2016).

Graz University of Technology. About sigviewer. SourceForge, n.d. URL `http://sigviewer.sourceforge.net/`. (Accessed on 04.12.2016).

Carl Gutwin and Saul Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work (CSCW)*, 11(3):411–446, 2002. ISSN 1573-7551. doi: 10.1023/A:1021271517844. URL http://dx.doi.org/10.1023/A:1021271517844.

Ming C Hao, Umeshwar Dayal, Daniel A Keim, and Tobias Schreck. Importance-driven visualization layouts for large time series data. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 203–210. IEEE, 2005.

Ming C Hao, Umeshwar Dayal, Daniel A Keim, and Tobias Schreck. Multi-resolution techniques for visual exploration of large time-series data. In *EUROVIS 2007*, pages 27–34, 2007.

Ming C. Hao, Manish Marwah, Halldór Janetzko, Umeshwar Dayal, Daniel A. Keim, Debprakash Patnaik, Naren Ramakrishnan, and Ratnesh K. Sharma. Visual exploration of frequent patterns in multivariate time series. *Information Visualization*, 11(1):71–83, January 2012. ISSN 1473-8716. doi: 10.1177/1473871611430769. URL http://dx.doi.org/10.1177/1473871611430769.

Chris Harrington. React vs angularjs vs knockoutjs: a performance comparison. Codementor, 2015. URL https://www.codementor.io/reactjs/tutorial/reactjs-vs-angular-js-performance-comparison-knockout. (Accessed on 13.12.2016).

Jérémy Heleine. 6 jQuery Infinite Scrolling Demos. sitepoint, n.d. URL https://www.sitepoint.com/jquery-infinite-scrolling-demos/. (Accessed on 01.12.2016).

Alexey Melnikov Ian Fette. Rfc 6455 - the websocket protocol. RFC, 2011. URL https://tools.ietf.org/html/rfc6455. (Accessed on 09.12.2016).

Vernon Lawhern, W. David Hairston, Kaleb McDowell, Marissa Westerfield, and Kay Robbins. Detection and classification of subject-generated artifacts in {EEG} signals using autoregressive models. *Journal of Neuroscience Methods*, 208(2):181 – 189, 2012. ISSN 0165-0270. doi: http://dx.doi.org/10.1016/j.jneumeth.2012.05.017. URL http://www.sciencedirect.com/science/article/pii/S0165027012001860.

Danh Le-Phuoc and Manfred Hauswirth. Linked open data in sensor data mashups. In *Proceedings of the 2nd International Conference on Semantic Sensor Networks-Volume 522*, pages 1–16. CEUR-WS. org, 2009.

Su Te Lei and Kang Zhang. A visual analytics system for financial time-series data. In *Proceedings of the 3rd International Symposium on Visual Information Communication*, page 20. ACM, 2010.

Christian T Lopes, Max Franz, Farzana Kazi, Sylva L Donaldson, Quaid Morris, and Gary D Bader. Cytoscape web: an interactive web-based network browser. *BIOINFORMATICS*, 26(18):2347–2348, 2010.

Chris Mills and Hamish Willee. Server-side web frameworks - Learn web development | MDN. Mozilla Developer Network, n.d. URL https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks. (Accessed on 08.12.2016).

MoTeC. i2 data analysis - overview, n.d. URL http://www.motec.com/i2/i2overview/. (Accessed on 06.12.2016).

Tim Roberts. A simple mvc framework with node and express, 2013. URL http://timstermatic.github.io/blog/2013/08/17/a-simple-mvc-framework-with-node-and-express/. (Accessed on 12.12.2016).

Alois Schlögl. GDF - A general dataformat for BIOSIGNALS. *CoRR*, abs/cs/0608052, 2006. URL http://arxiv.org/abs/cs/0608052.

Tobias Schreck, Tatiana Tekušová, Jörn Kohlhammer, and Dieter Fellner. Trajectory-based visual analysis of large financial time series data. *ACM SIGKDD Explorations Newsletter*, 9(2):30–37, 2007.

Tobias Schreck, Jürgen Bernard, Tatiana Von Landesberger, and Jörn Kohlhammer. Visual cluster analysis of trajectory data with interactive kohonen maps. *Information Visualization*, 8(1):14–29, 2009.

Norah C Slone and Nathanel G Mitchell. Technology-based adaptation of think-pair-share utilizing google drive. *Journal of Teaching and Learning with Technology*, 3(1):102–104, 2014.

M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. Wysiwis revised: Early experiences with multiuser interfaces. *ACM Trans. Inf. Syst.*, 5(2): 147–167, April 1987. ISSN 1046-8188. doi: 10.1145/27636.28056. URL http://doi.acm.org/10.1145/27636.28056.

Krasimir Tsonev. Build a complete mvc website with expressjs. Envato Tuts+, 2013. URL https://code.tutsplus.com/tutorials/build-a-complete-mvc-website-with-expressjs--net-34168. (Accessed on 12.12.2016).

Todd Vernon. Not all collaboration types are equal. DevOps, 2014. URL `https://devops.com/collaboration-types/`. (Accessed on 09.12.2016).

Danny Weyns and Tom Holvoet. Synchronous versus asynchronous collaboration in situated multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 1156–1157. ACM, 2003.

Hartmut Ziegler, Marco Jenny, Tino Gruse, and Daniel A Keim. Visual market sector analysis for financial time series data. In *Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on*, pages 83–90. IEEE, 2010.

Appendix

Appendix A

API

There are two ways to retrieve data from the SignalCloud-server. On the one hand it is possible to use a websocket for all communication purposes. This enables the client-software to store and sync data as well.
On the other hand one could use the REST-API to retrieve JSON-data too. However there are no methods to alter data on the server via REST, because there was simply no necessity at this point. The REST-API exists, because of various reasons one is for example to provide all the tagged data in a convenient way, which can be used automatically by other clients.

A.1 Websocket Protocol

The SignalCloud-server uses the Node.js-framework “express”, which features a REST-API to transfer for example HTML-Pages or JSON-data. However the collaborative part needs realtime-synchronisation between clients. Therefore the websocket based framework called “Socket.IO” is used to support this functionality.
After the client page is loaded, it tries to connect to the Socket.IOwebservice. This connection is only accepted, if the user is logged in via the login site or the API.
All the messages consist of JSON-strings. The Socket.IO-framework allows to define event-handlers for specific message types. The supported message-types can be divided into categories.

Methods for data-retrieval

This methods behave like simple getters. A message is sent to the server, which triggers an answer holding the requested data. Note that the answer of the server is sent on the same message-channel as the request.

`get-signal-metadata`
Parameter: signal id / metadata id (these are the same)
Returns a message holding all the metadata for a specific signal. Note that the signal itself and the corresponding signal metadata have always the same uuid.

`get-signal`
Parameter: signal id
Returns a message holding the signal. This includes no datapoints.

`get-metadata-list`
Parameter: user id
Returns a message holding all signals associated with the provided user id.

`get-signallist`
Parameter: user id
Returns a message holding all signal-metadata-entries associated with the provided user id.

`get-userlist`
Parameter: none
Returns a message holding all the users registered on the Server.

`get-user`
Parameter: user id
Returns a message holding the data of a specific user.

`get-event-by-sid`
Parameter: signal id
Returns a message holding a list of event lists for each user, which has contributed to one signal file.

`get-my-user-id`
Parameter: none
Returns a message holding the id of the connected user.

`get-my-user-data`
Parameter: none
Returns a message holding the user-data of the connected user.

`get-signal-part`
Parameter: signal-part id
Returns a signal-part by id. These parts hold the actual signal points.

`get-joined-eventdata-by-sid`
Parameter: joined event id
Returns a message holding the requested joined event data. Note that this is a single occurrence of an event holding the data-points of the signal as well.

`get-overlaps`
Parameter: signal id / overlaps id (these are the same)
Returns a message holding all the overlaps of a signal.

Methods for overwriting data

`set-signal-metadata` Parameter: JSON-encoded metadata
Creates or updates the received signal metadata in the database.

`set-my-user-data`
Parameter: JSON-encoded user-data
Updates the user-data of the current user in the database.

`set-my-password`
Parameter: JSON-encoded id and password
Updates the password of the current user.

`set-joined-eventdata`
Parameter: JSON-encoded joined-event-data
Creates or updates the received joined-event-data in the database.

`set-event`
Parameter: JSON-encoded event
Creates or updates the received event-data in the database.

`set-overlaps`
Parameter: JSON-encoded overlap
Creates or updates the received overlap-data in the database.

Methods for merging data

To reduce data-inconsistency among users, while getting realtime-synchronisation, it is necessary to send only data, which has to be changed. This is done by using the following methods. Note that the JSON-encoded payload has to hold only the necessary data as it gets merged with an already existing (older) version of it.
One might notice that this inhibits the deletion of object-properties. Therefore there is a `delete`-method for each `assign`-method.

`assign-overlap`
Parameter: JSON-encoded part-dataset
Merges the overlap-part-dataset with the existing cache-version on the server as well as with each connected client.

`assign-event`
Parameter: JSON-encoded part-dataset
Merges the event-part-dataset with the existing cache-version on the server as well as with each connected client.

`assign-signal-metadata`
Parameter: JSON-encoded part-dataset
Merges the metadata-part-dataset with the existing cache-version on the server as well as with each connected client.

`delete-overlap`
Parameter: JSON-encoded array of keys, which lead to the property
Deletes a property of the current cache-version of the overlaps-data on the server as well as with each connected client.

`delete-event`
Parameter: JSON-encoded array of keys, which lead to the property
Deletes a property of the current cache-version of the event-data on the server as well as with each connected client.

`delete-metadata`
Parameter: JSON-encoded array of keys, which lead to the property
Deletes a property of the current cache-version of the metadata on the server as well as with each connected client.

Methods for starting Tasks

The server-side data-model provides two different tasks, which can be run on specific datasets in the database. One can be started by the websocket. Moreover, there are many timeouts to prevent multiple tasks, if there are many requests at the same time. `run-detect-overlaps`
Parameter: none
Starts a task for detecting overlapping events. This is for example started after an overlap has been edited or a new event has been added.

Listeners

`on-file-deleted`
Parameter: none
Invoked if a the subscribed file gets deleted. The client will redirect the user immediately to the files-overview to prevent data-inconsistency.

`on-user-invitation`
Parameter: none
Invoked if the user has been invited to a new file. Note that this message is received regardless of the subscribed channel.

`on-updated-overlaps`
Parameter: none
Invoked if there has been an update by the `set-overlaps`-method.

`on-updated-events`
Parameter: none
Invoked if there has been an update regarding all events of a certain file. For example if a new event-collection has been imported from a file.

`on-updated-event-single`
Parameter: none
Invoked if there has been an update by the `set-event`-method.

`on-updated-metadata`
Parameter: none
Invoked if there has been an update by the `set-signal-metadata`-method.

`on-assign-overlap`
Parameter: none
Invoked if there has been a merging-update by the `assign-overlaps`-method.

`on-assign-event`
Parameter: none
Invoked if there has been a merging-update by the `assign-event`-method.

`on-assign-signal-metadata`
Parameter: none
Invoked if there has been a merging-update by the `assign-signal-metadata`-method.

`on-delete-overlap`
Parameter: none
Invoked if there has been a deletion by the `delete-overlaps`-method.

`on-delete-event`
Parameter: none
Invoked if there has been a deletion by the `delete-event`-method.

`on-delete-signal-metadata`
Parameter: none
Invoked if there has been a deletion by the `delete-signal-metadata`-method.

Other methods

`subscribe-channel`
Parameter: name of the channel
This important message has to be sent directly after a new connection has been established. By doing so, the client will send and receive messages only within its subscribed channel-room. In the case of an open signal, the channel-name would be its id. This ensures that all online users communicate only with users viewing the same signal.

`user-invitation`
Parameter: JSON-object holding the user id (`uid`) of the invited user and file-name (`filename`) of the file
Sent if a new user is added to a signal. This triggers an emit of the `on-user-invitation`-message only to the user with the id from the parameter.

`broadcast`
Parameter: JSON-encoded message

Triggers a broadcast among all the connected users, which subscribed to the same channel.

A.2 REST

The API exists primarily for an automatic retrieval of data for further processing and analysis. However, it is used as well by the client to retrieve the data-parts of the signal, because it features a little performance advantage in contrast to the websocket.
The primary structure of an answer of the API can be seen in the following code-listing:

```
{
  "errors": [],   // array of errors
  "ok": true,     // true if the action was successfull
  "payload": {},  // if there is data
  "uid": "uuid"   // id of the current user (if logged in)
}
```

The requests can be categorized in GET-Requests and POST-Requests. Note that the domain itself (e.g.: `http://localhost:3333/...`) is not included in all URLs.

POST-Requests

URL: `./api/login`
Parameter: [`username` → registered username] **and** [`password` → related password]
Makes it possible to login via ajax. The answer will contain the `uid`, if the login was successfull or the error "Authentication Error", if the credentials are wrong.

URL: `./api/log`
Parameter: [`msg` → string to log] **and** [`Optional: level` → logging-level]
Appends a log message in the corresponding logfile on the server or creates it. The log-messages are located under `./logs`. The parameter logging-level has to have the format of a js-logger[1] context-level.

URL: `./api/logout`

[1]js-logger is a client-side JavaScript-library and features some logging tools (https://github.com/jonnyreeves/js-logger)

Parameter: none
Logs out the user by destroying the current session. This method can be executed by a GET-Request as well.

GET-Requests

URL: `./api/`
Parameter: none
Can be used for checking the current login-state. If the user is logged out, the message will contain an error called “Not allowed”.

URL: `./api/logout`
Parameter: none
Logs out the user by destroying the current session.

URL: `./api/log`
Parameter: none
Returns all logs, which got appended to the log-file of the current user by using the POST-Version of the `./api/log`-method.

URL: `./api/signal`
Parameter: `id` → signal id
Retrieves a signal from the database by using its id.

URL: `./api/signal/part`
Parameter: `id` → signal-part id
Retrieves one datapoint-part of a signal from the database by using its id.

URL: `./api/signal/metadata`
Parameter: `id` → signal id / metadata id
Retrieves a signal-metadata-entry from the database by using its id.

URL: `./api/event`
Parameter: [`id` → event id] **or** [`sid` → signal id]
Retrieves an event-collection from the database by using its id or to retrieve all event-collections of a specific signal. The result depends on the name of the parameter (`sid` or `id`).

URL: `./api/event/datapoints`
Parameter: [`id` → event id] **or** [`sid` → signal id]
Retrieves a certain joined-event by id or all joined-events by using a signal

id. Note that the joined-events may not be created for the signal yet, if the `./api/event/run_join` was not executed first.

URL: `./api/event/overlap`
Parameter: `id` $\rightarrow$ signal id
Retrieves all overlap-data of a signal from the database by using its id.

URL: `./api/event/run_join`
Parameter: `sid` $\rightarrow$ signal id
Runs a task which creates joined-events for all event-occurrences in one signal.