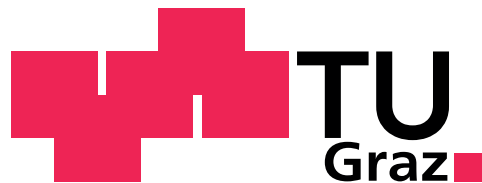


Master's Thesis

SigViewer: An Improved Version with New Features

Christoph Eibel



Graz University of Technology

Institute for Knowledge Discovery

Krenngasse 37

A-8010 Graz

Supervisor

Dipl.-Ing. Dr.techn. Clemens Brunner

Graz, September 2010

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 28. 9. 2010



Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, 28th September 2010



Abstract

SigViewer is a software application to view, annotate and process multi-channel signals. The main field of application is BCI (brain-computer interface) research, where SigViewer is used to display and process biosignals like electroencephalogram (EEG). However, SigViewer 0.2.6 (the latest version before the beginning of this thesis) lacks some important features such as undo/redo functionality and basic signal processing tools. Furthermore, some parts of the source code depend on the deprecated Qt 3 library.

Therefore, the aim of this master's thesis was to further develop SigViewer. This comprises: refactoring the source code; improving the usability by revising all dialogs and context menus, and by introducing a new zooming approach; and adding undo/redo functionality, signal processing tools, and a Debian package.

The latest version of SigViewer, which has been released within the time frame of this master's thesis, provides a more user-friendly interface with many new features. Additionally, the improved code structure allows adapting SigViewer to the user needs and adding new features more easily in the future. Upcoming work could comprise adding more signal processing tools, further improving the graphical user interface, and continuing source code refactorings.

Kurzfassung

SigViewer ist eine Software zum Darstellen, Annotieren und Verarbeiten von Signalen. Das primäre Anwendungsgebiet ist die BCI-Forschung (brain-computer interface), in der SigViewer benutzt wird, um Biosignale wie das Elektroenzephalogramm (EEG) zu betrachten und zu verarbeiten. Der Version 0.2.6 (die letzte vor Beginn dieser Masterarbeit) fehlen jedoch einige wichtige Funktionen, wie etwa die Rückgängig/Wiederherstellen-Funktion und grundlegende Signalverarbeitungs-Werkzeuge. Weiters hängen ein paar Teile des Quelltextes von der veralteten (und nicht weiter gewarteten) Qt 3 Bibliothek ab.

Aufgrunddessen war das Ziel dieser Masterarbeit die Weiterentwicklung von SigViewer. Diese umfasste: Refaktorisierung des Quelltextes; Verbesserungen der Benutzerfreundlichkeit durch Überarbeitung aller Dialoge und Kontextmenüs, sowie Einführung eines neuen Zoom-Ansatzes; Hinzufügen einer Rückgängig/Wiederherstellen-Funktion und Hilfswerkzeugen zur Signalverarbeitung und Erstellen eines Debian-Pakets.

Die letzte Version von SigViewer, die im Zeitrahmen dieser Masterarbeit veröffentlicht wurde, bietet eine benutzerfreundlichere Bedienoberfläche mit vielen neuen Funktionen. Weiters erlaubt die verbesserte Quelltext-Struktur SigViewer in Zukunft leichter an neue Bedürfnisse der Benutzer anzupassen und neue Funktionen einzubauen. Künftige Arbeiten könnten sein: Hinzufügen von weiteren Signalverarbeitungswerkzeugen, weitere Verbesserung der Bedienoberfläche und stetige Quelltext-Refaktorisierungen.

Acknowledgements

First of all, I would like to thank my supervisor Clemens Brunner.

Furthermore, I would like to thank my colleagues at the BCI Lab (Institute for Knowledge Discovery) at Graz University of Technology for their feedback and efforts for testing SigViewer.

Finally, very special thanks go to Gabi, Alois and Elfriede.

Contents

Abstract	iii
Kurzfassung	iv
Acknowledgements	v
1 Introduction	1
1.1 Motivation	1
1.2 Aim and Overview	2
1.3 Approach	3
2 Framework	4
2.1 Biosignals and Events	4
2.2 Fourier Transform	4
2.3 Used Software	5
2.3.1 SigViewer 0.2.6	5
2.3.2 Qt 4	7
2.3.3 BioSig	7
2.3.4 FFTW	8
3 Extension of Functionality	9
3.1 Undo and Redo	9
3.1.1 Implementation	10
3.2 Signal Processing	13
3.2.1 Mean and Standard Deviation	14
3.2.2 Power Spectrum	14
3.2.3 Other	15
3.3 Event Features	17
3.3.1 Event Browsing	17

3.3.2	Event Inserting	17
3.3.3	Fit View to Selected Event	17
3.3.4	Hide Events of Other Type	18
3.3.5	Show All Events	18
3.4	Other New Features	19
3.4.1	File Opening and File Dropping	19
3.4.2	Read User-Defined Event Types	19
3.4.3	Channel Colors	19
3.4.4	Export to PNG	19
4	Graphical User Interface Improvements	21
4.1	Zooming	21
4.2	Context Menus	23
4.3	Mode-specific Widgets	24
4.4	Enhanced Dialogs	26
4.5	Small Improvements	29
4.5.1	Animations	29
4.5.2	Tool- and Statusbar	29
4.5.3	X-Position Highlighting	29
5	Restructuring and Refactoring	30
5.1	Porting to Qt 4	31
5.2	New Module Structure	32
5.3	MainWindowModel Refactoring	35
5.4	Automated Tests	36
5.5	Qt Designer	38
6	Deployment	40
6.1	Building SigViewer from Source	40
6.2	Installation Packages	41
6.2.1	Debian Package	41
6.2.2	Windows	43
6.2.3	Mac OS X	43
7	Outlook	44
7.1	Further Features	44
7.1.1	Undo View Setting	44

7.1.2	ERD/ERS Maps	44
7.1.3	EOG Artifact Correction	44
7.2	Further Improvement of GUI	45
7.2.1	Info Widget	45
7.2.2	Event Table	45
7.2.3	Animations	45
7.3	Further Refactoring	45
7.3.1	Improved File Support	45
7.4	Deployment	46
7.4.1	User Guide	46
7.4.2	Further Linux Packages	46
8	Concluding Remarks	47
	Bibliography	48

List of Figures

2.1	Screenshot SigViewer 0.2.6	6
3.1	Undo/Redo Class Diagram	11
3.2	Event Time Channel Dialog	13
3.3	Approach for Calculating Mean	14
3.4	Screenshot of SigViewer displaying the Mean of an Event Type	16
3.5	Screenshot of SigViewer displaying a Power Spectrum	16
3.6	Fit View to Event	18
3.7	Channel Color Dialog	20
4.1	Screenshot new Zooming	22
4.2	Class Diagram of New Zooming Implementation	22
4.3	Event Context Menus	23
4.4	Edit Event Widget	25
4.5	View Options Widget	25
4.6	Channel Selection Dialogs compared	26
4.7	Select-Event-Types-Dialog Comparing	27
4.8	Event Table Dialog	28
5.1	New Module Structure	32
5.2	MainWindowModel Refactoring	36
5.3	Tests Dialog	37
5.4	Screenshot of Test Data	38
5.5	Screenshot of Qt Designer	39
6.1	Debian Package Directory Structure	42

List of Tables

6.1	Debian package <i>control</i> file entries.	42
6.2	Desktop file entries.	42

1 Introduction

1.1 Motivation

SigViewer, introduced in [1] and [2], is a software application to view multi-channel biosignals and corresponding annotations. It can read various file formats (such as GDF [3], EDF [4], EDF+ [5], and BCI2000 [6]), provides an intuitive user interface, and is available for different operating systems such as Windows (XP, Vista, 7), Linux or Mac OS X.

One application area of SigViewer is the analysis of electroencephalographic signals (EEG) and other biosignals, which is particularly important for the development of brain-computer interfaces (BCIs). BCIs are systems which allow a person to control a computer only by thinking of, for example, moving a hand or foot, without relying on activity of peripheral nerves or muscles [7]. These mental activities can be measured with EEG electrodes which are applied on the scalp; they register excitatory and inhibitory post-synaptic potentials of neurons in the cortex. The acquired EEG signals are forwarded to a computer where a software analyzes and interprets the signal and triggers actions depending on special events in the signal.

In BCI research and development, easy-to-use software applications like SigViewer are needed to view recorded and annotated EEG data and other biosignals, and to edit or add annotations. Annotations mark special regions in the signals, for example, events of a BCI experiment. Therefore, it is possible to see what and when something like stimuli, motor imagery, etc. have occurred during such an experiment.

The annotations can also be used to mark noise artifacts, or to calculate evoked potentials.

One of the main advantages of SigViewer compared to similar applications is that it is a stand-alone program. No mathematical software package like Matlab [8] or GNU Octave [9] is needed to run SigViewer. Furthermore, its graphical user interface is specifically

designed to view biosignals and edit annotations. It allows fast navigation through the data, scaling, and zooming. In addition, SigViewer is open source software and licenced under the GPL [10], which guarantees that it will stay free.

1.2 Aim and Overview

The aim of this master's thesis is the further development of SigViewer. It comprises the following core aspects:

1. Extension of SigViewer with new functionality
2. Improvement and revision of the user interface
3. Restructuring and refactoring of the source code
4. Deployment of SigViewer

Each aspect is discussed in a separate chapter in this thesis. Chapter 2 gives an introduction of the theoretical background and the framework of SigViewer.

The extension of functionality is discussed in Chapter 3, which mainly comprises adding undo and redo functions, event-related signal processing tools, and several functions to make the annotation process easier.

Improvement of the user interface (Chapter 4) comprises the introduction of a new animated zooming approach, revision of all dialogs and context menus, new mode-specific widgets, and some further small adaptations.

Refactorings of the source code have been done continuously in small iterations. However, some big restructurings had to be made to guarantee the expandability and testability of the source code. These restructurings are discussed in Chapter 5.

Chapter 6 deals with the deployment of SigViewer. It rounds up this work and describes how binary packages are built with the aim to provide easy-to-install routines for different operating systems.

Finally, ideas for further work regarding each of the core aspects are discussed in Chapter 7.

1.3 Approach

The starting point of this master's thesis is SigViewer version 0.2.6. SigViewer is located as a project on SourceForge.net and is available at <http://sigviewer.sf.net>. SourceForge.net is a platform for the development of open source software. It provides free webspace, servers for version control, a bug tracking and feature request system, and many more features.

As programming environment, Qt Creator was chosen. This integrated development environment (IDE) is part of the Qt software development kit (SDK). This SDK also contains Qt Designer (a tool to layout dialogs, widgets, etc.), build tools (qmake), a comprehensive documentation of the SDK, and the Qt libraries. Chapter 2.3 describes the used libraries in more detail.

During the development, some interim releases have been made to show results and get user feedback. The interim releases comprise SigViewer version 0.3.0 and 0.4.0. The latest version released within the time scope of this master's thesis is SigViewer 0.4.1.

2 Framework

2.1 Biosignals and Events

The term *biosignal* generally comprises any measurable signal from biological origin. In this work, the term is used in a more specific sense namely for electrical signals which can be measured from human beings (ExG signals). That mainly comprises the electroencephalogram (EEG), the electrocardiogram (ECG), the electromyogram (EMG), and the electrooculogram (EOG).

During BCI experiments, special events may occur, for example when a cue is displayed on a screen. The file format GDF [3], which can be used to store biosignals, supports storing these events in the same file as the recorded data. Therefore, SigViewer is able to read events and integrate them into the visualization of the signal data.

As the type of an event marker is adjustable in SigViewer, these markers can be used to tag artifacts as well. Artifacts are regions in the EEG where, for example, another signal significantly contaminates the EEG signal. One source of such noise is the electrooculographic (EOG) artifact [11].

Of course, the annotations can be used to mark any area of interest for user-specific purposes.

In SigViewer, and consequently in this thesis, “annotations” are generally called “events”.

2.2 Fourier Transform

The Fourier transform is a method to transform a signal from the *time domain* into the *frequency domain*. Any signal can be seen as the sum of sine waves in different frequencies, phases and amplitudes. The Fourier transform allows to disclose these frequencies of a given signal. Furthermore, the inverse Fourier transform allows to recreate the original time signal or a bandpass-filtered version of it.

2 Framework

The elements of a discrete Fourier transform X of a discrete signal x can be calculated with formula 2.1 as described in [12].

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j(\frac{2\pi}{N}) \cdot kn} \quad (2.1)$$

x is a sequence of N real numbers $x[0], \dots, x[N-1]$ which are transformed into the sequence of N complex numbers $X[0], \dots, X[N-1]$ by the discrete Fourier transform. N is the number of elements in the analyzed period of signal x . n is the index of the current element of signal x , k is the index of the current element of the result X and j is the imaginary unit.

Formula 2.2 [12] shows the inverse discrete Fourier transform to recreate the signal.

$$x[n] = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X[k] \cdot e^{j(\frac{2\pi}{N}) \cdot kn} \quad (2.2)$$

The complexity of the direct approach is $\mathcal{O}(N^2)$ [13]. However, more efficient algorithms exist which reduce the complexity to $\mathcal{O}(N \cdot \log(N))$ by utilizing the *divide and conquer* paradigm [14]. These algorithms are called *Fast Fourier Transform* [15].

The visualization of the frequency domain is an important part of signal analysis. For example, the analysis of event-related synchronisation (ERS) and event-related desynchronisation (ERD) requires the investigation of diverse frequency bands [16].

Within this work the creation of an event-related power spectrum has been implemented as described in Chapter 3.2.2. The Fast Fourier Transform was not newly implemented because stable and freely (licenced under the GPL) available implementations already exist. The library *fftw3* [17], used by SigViewer, is introduced in Section 2.3.4.

2.3 Used Software

This section describes the source code basis and the used libraries for SigViewer.

2.3.1 SigViewer 0.2.6

The source code of SigViewer version 0.2.6 can be seen as the starting point of this work as a whole. It uses the *biosig4c++* library for reading signal data and reading and

2 Framework

writing event data. Therefore many different data formats are supported.

The main features of SigViewer 0.2.6 (as described in [2]) are:

- Load multi-channel signals
- Display them in various scales and show events (annotations)
- Graphical editing of events
- View basic information about the opened file
- The table-based widget “event table” to view and delete events in addition to graphical editing

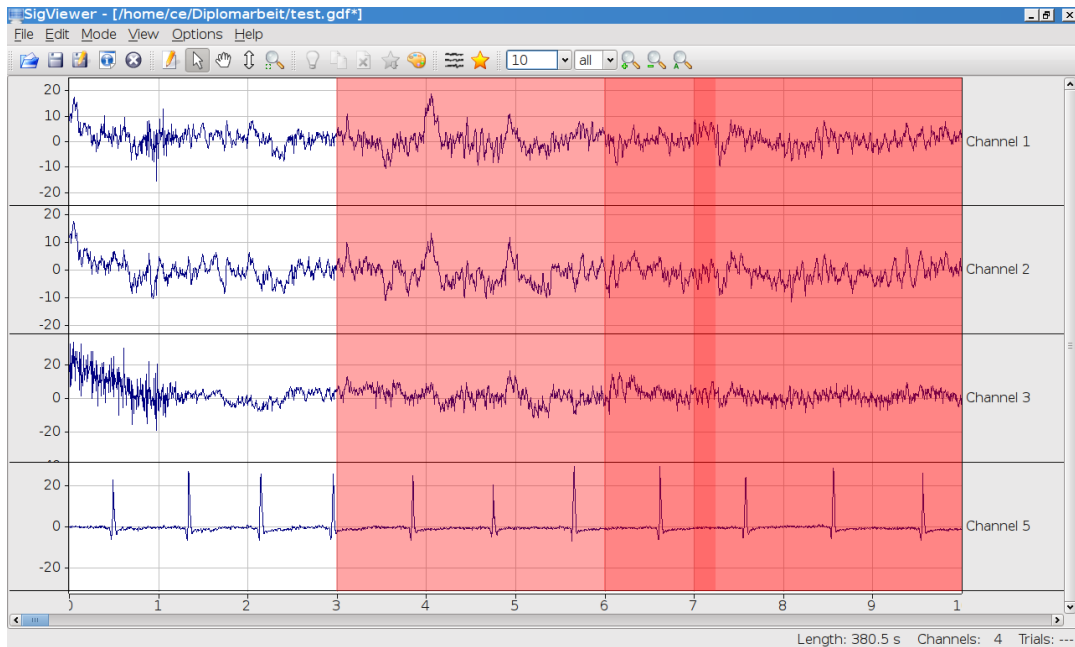


Figure 2.1: Screenshot of SigViewer 0.2.6 on Kubuntu Linux.

As mentioned in the introduction, the aim of this master’s thesis is to fix bugs, to add new features (Chapter 3 and 4) and to improve the structure of the source code (Chapter 5) while keeping the existing features of SigViewer 0.2.6.

2.3.2 Qt 4

Qt 4 is a GUI development framework mainly for the C++ programming language. Currently it is developed and hosted by Nokia Corporation at <http://qt.nokia.com>. The libraries are available under different licences. For the development of SigViewer, the open source licence has been chosen to provide SigViewer as open source software.

Qt is cross-platform on the source code level. This means that the same source code can be compiled on different platforms such as Windows, Linux or Mac OS X. The look and feel of the resulting application is adapted to the target platform. Therefore, binary packages of SigViewer for different platforms have been created (more details in Chapter 6).

The main arguments for using the Qt framework (besides being cross-platform) are that the Qt libraries ship with a comprehensive documentation and an easy-to-use framework and of course that SigViewer 0.2.6 is based on Qt.

Signals, Slots and QActions

Qt provides a powerful signals and slots concept. Any object of a class which is derived from QObject may emit signals. Other objects, which have methods that are marked as slots, are able to connect to such signals. Consequently, these slots are executed any time the signal is emitted. This approach allows the decoupling of classes, because sender and receiver do not have to know from each other.

QAction is a central class within Qt GUI software applications. Objects of this class can be put into main menus, context menus or toolbars. There they appear as an entry or button which can be triggered by the user. The “trigger” signals can be connected to slots of other implemented classes where the processing of the triggered user action is done.

More details are described in the documentation of Qt at <http://doc.qt.nokia.com>.

2.3.3 BioSig

BioSig is a project which provides several tools for reading, writing and processing biosignals like EEG. It is located on SourceForge.net at <http://biosig.sf.net>.

In [2] BioSig is introduced as followed:

“The open source software project BioSig was founded with the aim to provide a software library for biomedical signal processing.”

BioSig comprises several subprojects like “BioSig for Octave and Matlab” or “BioSig for C/C++”. The latter, which is also known as “biosig4c++”, is of main interest for SigViewer. This library provides functions to read many different biosignal file formats like GDF, EDF or BCI2000 [3, 4, 6] and to write event data into GDF files. Additionally, it should be possible to convert any supported file format that can be read into GDF.

Biosig4c++ is statically linked to SigViewer. This means the library is only required during the building process.

2.3.4 FFTW

FFTW is a free (GPL) library for calculating the Fast Fourier Transform of a discrete signal [18]. It is written in the C programming language. FFTW is the abbreviation for *Fastest Fourier Transform in the West*. This library is used to calculate the power spectrum (see Chapter 3.2.2). It can be easily installed on Linux because binary packages already exist. For Windows and Mac OS X, precompiled libraries are available for download at the SigViewer project website <http://sigviewer.sf.net>.

In SigViewer, a small C++ wrapper for FFTW is used which is called FFTW++ [19]. It is also licenced under the GPL. This wrapper is well documented and it avoids increasing complexity of the SigViewer source code.

3 Extension of Functionality

In this chapter, the features of SigViewer 0.4.1 are described which have been implemented in the scope of this master's thesis.

3.1 Undo and Redo

Implementing *undo* and *redo* has been one of the first contributions of this work as SigViewer 0.2.6 does not provide this functionality. Undo and redo are very important features of any software application with a focus on user interaction.

In SigViewer, the following *editing actions* can change an open file. Therefore, they have been made undoable:

- Creating a new event
- Removing an existing event
- Changing the type of an event
- Changing the channel of an event
- Changing the position or duration of an event

These actions may be triggered by the user via the context menu of an event, the editing mode widget (Chapter 4.3), the event table (Chapter 4.4) or by graphical editing of events. Of course, undoing an editing action is independent from where it has been triggered. The number of actions that could be undone is not limited.

Undo and redo buttons are located in the toolbar and in the “Edit” menu. Furthermore, they are assigned to platform-dependent standard keyboard shortcuts. For example, undo is assigned to `Ctrl+Z` on Windows.

Undo and redo is supported since the interim release SigViewer 0.3.0. However, only editing actions are undoable. To further increase the usability, view setting changes could also be undoable as described in Chapter 7.1.1.

3.1.1 Implementation

Qt provides the *Undo Framework* [20] to implement undo commands. The approach follows the idea of the *Command* design pattern [21]. Each editing action has to be implemented in a class which is derived from `QUndoCommand` and override the virtual methods *redo* and *undo*. An object of the class `QUndoStack` manages execution, redo, and undo calls of these commands.

In SigViewer the classes which implement undoable actions are located in the *editing commands* module (more about the modules in Chapter 5.2). Furthermore, the abstract base class *CommandExecuter* provides an interface to handle execution of these commands. The *TabContext* implements the `CommandExecuter` interface. Figure 3.1 outlines the approach in a class diagram.

Any object that needs to do something that should be undoable (e.g. an object of *EventEditingGuiCommand* which handles the triggers to change the type of an event) needs a reference to an `EventManager` and to an `Command-Executer`. Then it has to create an instance of the desired `UndoCommand` and pass it to the `CommandExecuter`. The implementation of the `CommandExecuter` will internally put the `UndoCommand` on the `QUndoStack` which will call the *redo* method of the command and store it for possible future undo calls.

NewEventUndoCommand

This class is responsible for adding new events to the `EventManager`. The constructor requests an object of the class `SignalEvent` which contains all parameters needed for creating an event and a reference (in this case a *QSharedPointer*) to an `EventManager`. Calling *undo* on this command removes the event from the `EventManager`.

ChangeTypeUndoCommand

This class is responsible for changing the type of an existing event. Its constructor requires the ID of the event, a *QSharedPointer* to an `EventManager` and the new type. An *undo* call sets the type of the event back to the old type.

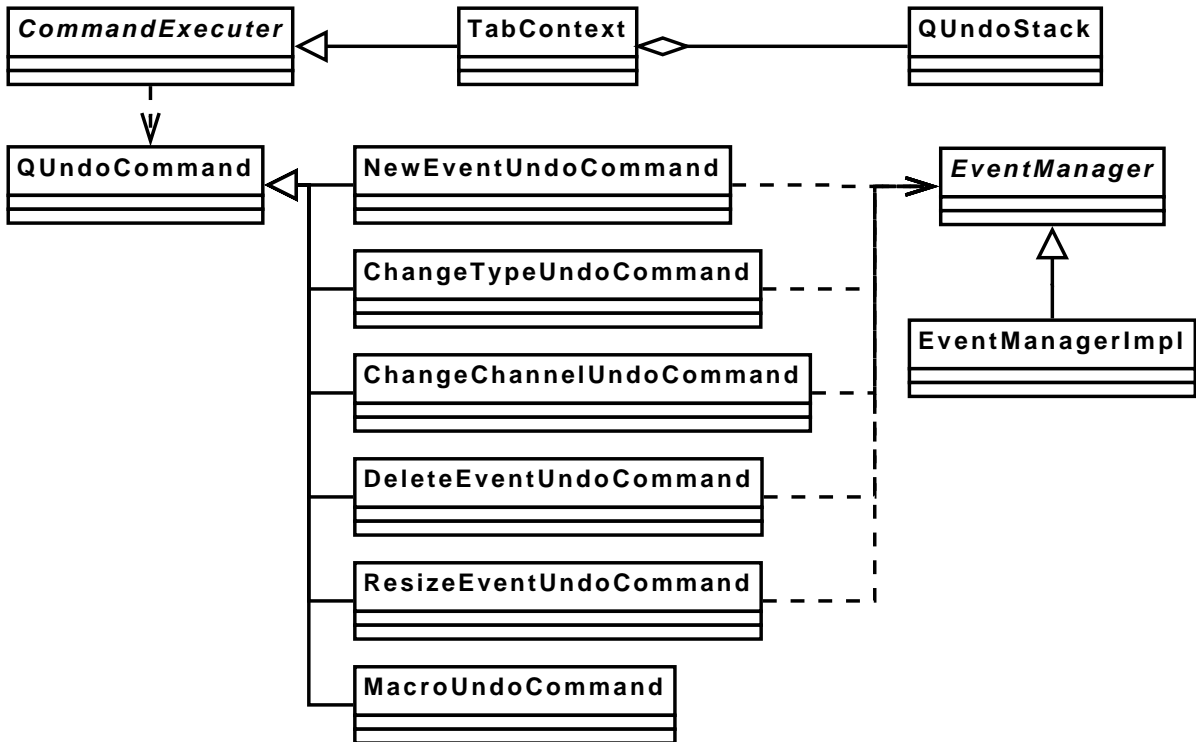


Figure 3.1: Class diagram of the undo/redo implementation. Each event editing action is implemented in its own class which is derived from *QUndoCommand*.

ChangeChannelUndoCommand

This command changes the channel of an event. The constructor of this command requires the ID of the event, a `QSharedPointer` to an `EventManager` and the ID of the new channel.

DeleteEventUndoCommand

This command deletes the event with the given ID from the given `EventManager`. Calling undo recreates the event with the same ID. Therefore the IDs stay in a consistent state independent from creation and deletion.

ResizeEventUndoCommand

This command comprises resizing and repositioning. The constructor requests the new position (in samples) and new duration (in samples) beside the event ID and a `QSharedPointer` of an `EventManager`.

MacroUndoCommand

The macro command makes it possible to execute multiple commands. For example, it is used if several events are selected in the event table and deleted at once. Triggering undo without using MacroUndoCommand would restore only the last event (e.g. if 100 events were deleted at once 100 undo calls would be necessary to restore all of them). However, by using MacroUndoCommand, all events are restored at once and therefore a more intuitive behavior is provided.

The constructor just requires a QList with QSharedPointers of QUndoCommands.

3.2 Signal Processing

Another main goal of this work was to provide SigViewer with event-based signal processing functionality. “Event-based” means that only parts of a signal are processed which are covered by a predefined type of event marker instead of the whole data of a signal.

The operations to calculate the *Mean and Standard Deviation* and to create a *Power Spectrum* have been implemented. In SigViewer 0.4, they are available via the “Tools” main menu.

After triggering these actions and before the actual processing starts, a dialog is shown which allows the user to select the channels and the event type of interest. The duration is calculated automatically from the events, but for including data before and after the events it can be adapted manually too. The dialog is shown in Figure 3.2.

For displaying the result of the processed data, a new tab is added to the main window of SigViewer. The same widgets as for browsing the original signal data are used. Therefore, the same operations like zooming, hiding channels, shifting, etc. are possible. Figure 3.4 shows a screenshot of SigViewer displaying the mean of an event type in a separate tab, and Figure 3.5 shows a power spectrum.

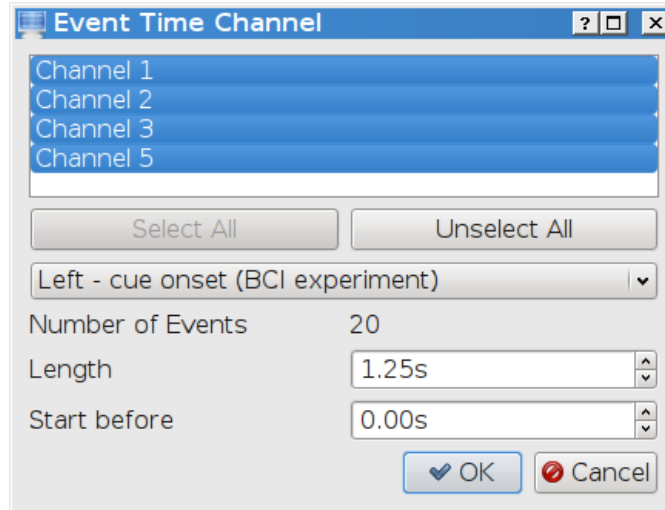


Figure 3.2: Event Time Channel Dialog. This dialog is used to select channels and events for different kinds of signal processing. Only channels and event types are selectable which are not hidden in the signal data view. This approach avoids a bloated interface.

3.2.1 Mean and Standard Deviation

Mean and standard deviation of an event is calculated in the following way: The first sample of the first event in one channel is summed up with the first sample of the other events of the same type in the same channel. Afterwards, the sum is divided by the number of events of that type. Doing this for every sample which is covered by the event, a new signal is generated, which has the same length as one event of that type. The approach is illustrated in Figure 3.3.

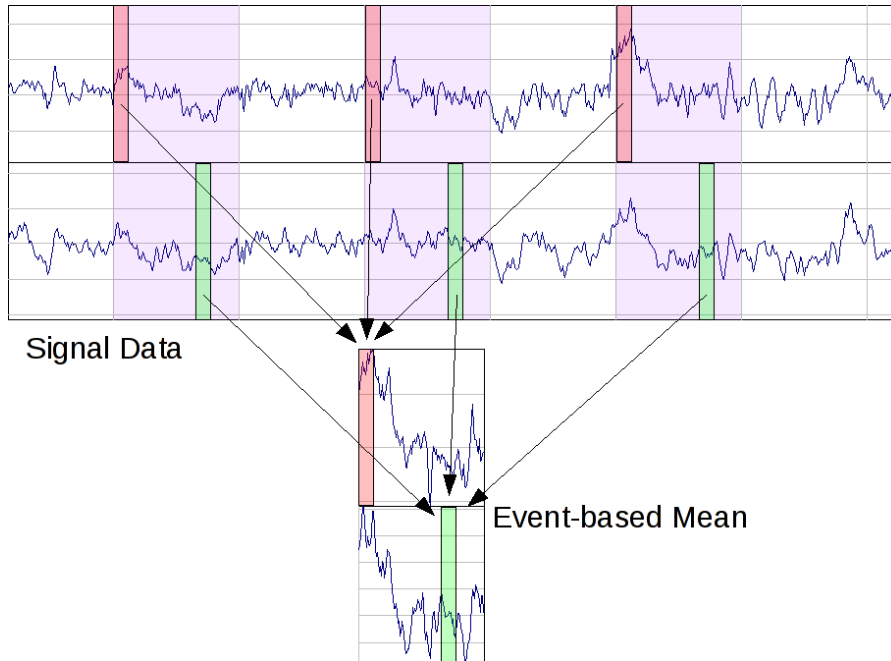


Figure 3.3: Approach for calculating the mean of an event. The events are highlighted in blue. The first samples of the events of same type and channel are used to calculate the first samples of the mean. The mean is calculated for each channel. In the figure, two regions of the events are highlighted (red and green) to show which region of the signal data is used for the result.

The calculation of the mean of events can be used, for example, to detect event-related potentials in the ongoing EEG [22].

3.2.2 Power Spectrum

To get an overview of the occurring frequencies in the signal during an event, the event-based power spectrum can be generated. It displays the result of the Fourier transform.

The power spectrum is the mean squared value of the Fourier transform of the part of the signal which is covered by the event.

The event-based power spectrum is computed through calculating the power spectrum of each event and channel, and then calculating the mean within each channel. The calculation is similar to that of the mean described in Section 3.2.1, with the difference that the power spectrum of an event is averaged and not the signal data in the time domain. Figure 3.5 shows a screenshot of SigViewer displaying the result of a power spectrum calculation.

Implementation

For generating the power spectrum, a C++ wrapper of the FFTW library (described in Section 2.3.4) is used to calculate the Fourier transform. By using this wrapper, only a few lines of code are needed to generate the power spectrum of a DataBlock.

3.2.3 Other

Adding further event-based signal processing functionality to SigViewer such as the creation of ERD/ERS maps as described in [16] has been requested too. However, due to the limited time frame of this master's thesis, this feature has not been implemented yet.

3 Extension of Functionality

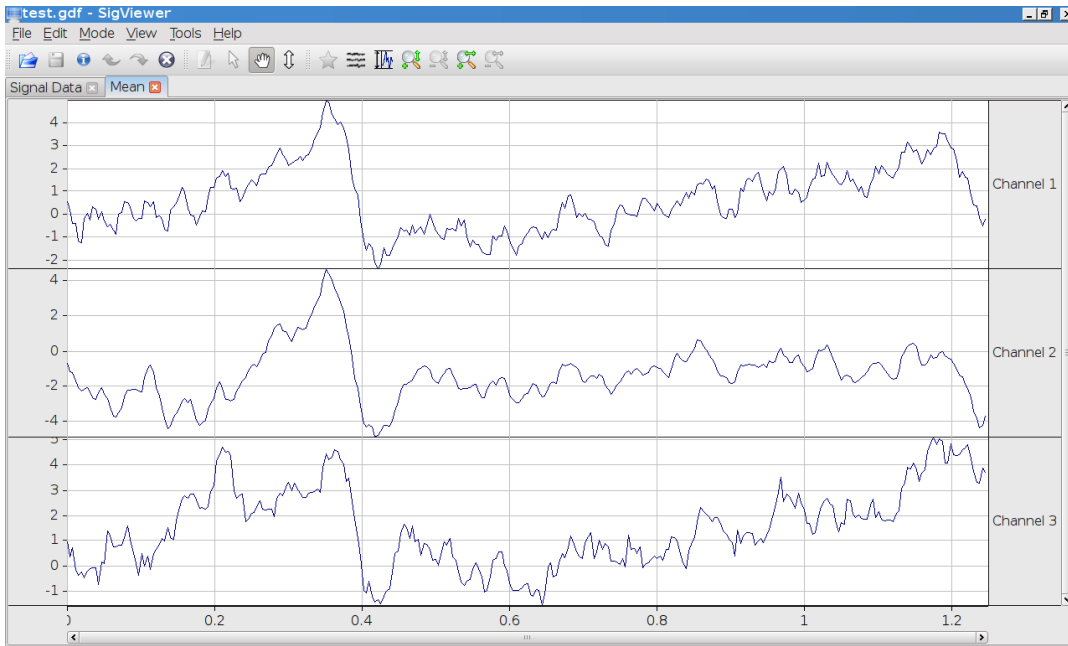


Figure 3.4: Screenshot of SigViewer displaying the mean of an event type.



Figure 3.5: Screenshot of SigViewer displaying a Power Spectrum. The unit of the x-axis is Hz. The unit of the y-axis is $\frac{\mu V^2}{Hz}$.

3.3 Event Features

3.3.1 Event Browsing

This feature has been introduced to support fast navigation between events of the same type. After selecting an event, the shortcut `Ctrl+Left` (`Cmd+Left` on Mac OS X) sets the viewing position to the previous event of the same type and the shortcut `Ctrl+Right` (`Cmd+Right` on Mac OS X) to the next event of the same type. After the viewing position has been changed, the newly visible event is selected automatically and editing is possible subsequently. Besides the keyboard shortcuts, these action may also be triggered via the event context menu, or the “View” main menu.

Implementation

The implementation of the `EventManager` stores the events in two different maps. The first uses the ID of the events as key. The other one uses the position of the events as key and the ID as value. The position map allows fast look up of next and previous events. The interface of the `EventManager` provides methods to get the next/previous ID of the event, which is of the same type as of the event with the given ID. The code which triggers and interprets the result of these methods is located in the class `AdaptEventViewGuiCommand`.

3.3.2 Event Inserting

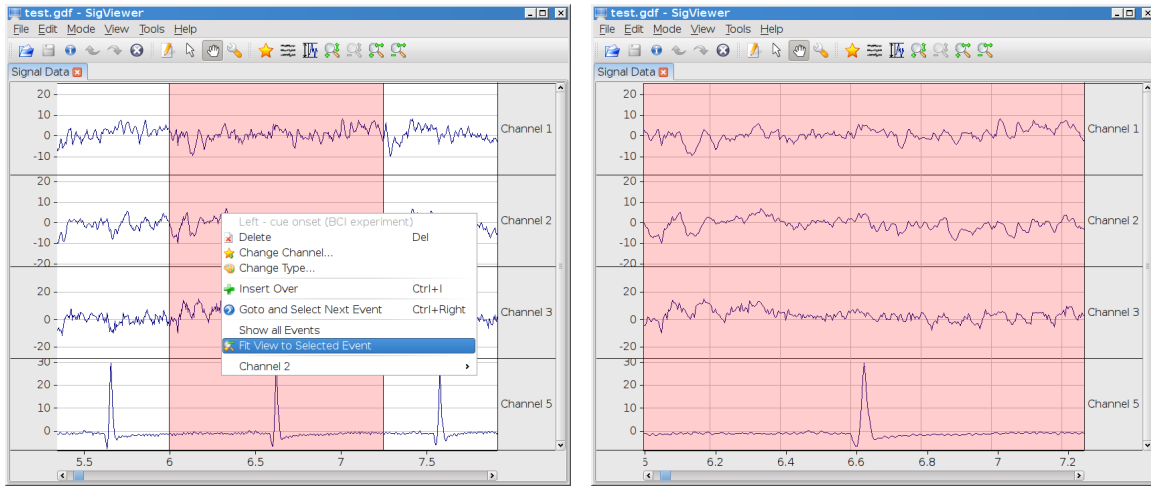
This feature allows fast creation of events which have the same position and duration as the currently selected event. It is called “Insert Over” and can be triggered via the context menu of an event, via the “Edit” main menu, and by the keyboard shortcut `Ctrl+I`. The type of the newly inserted event is the same as that for graphically creating new events in the “New Event” mode and can be set in that mode.

This feature can be used to additionally mark the section of an event, for example to mark it with an artifact event type.

3.3.3 Fit View to Selected Event

The “Fit View to Selected Event” action automatically scales and scrolls the viewport along the x-axis so that the currently selected event is fully shown. The action can be

triggered via the event context menu or the “View” main menu. Figure 3.6 illustrates this feature.



(a) Standard View

(b) Fitted View to Event

Figure 3.6: Calling “Fit View to Selected Event” in the event context menu automatically adapts the viewport to the event.

3.3.4 Hide Events of Other Type

As the name suggests, this feature allows to hide all events which are of different type as the currently selected one.

The command is located in the event context menu and the “View” main menu.

3.3.5 Show All Events

As a complement to “Hide Events of Other Type”, the feature “Show All Events” has been introduced. It allows to set the visibility of all event types to true with one click. It is located in the “View” main menu and in the event context menu. It only appears in the context menu if “Hide Events of Other Type” has been triggered just before.

3.4 Other New Features

This section summarizes some features of smaller extent or complexity.

3.4.1 File Opening and File Dropping

Since version 0.3, SigViewer supports opening a file via command line parameter. This furthermore implies that SigViewer can be associated with special file types. For example, it is possible that “.gdf” files are opened automatically with SigViewer when double-clicking such files within Windows Explorer (or any other file manager).

Additionally, SigViewer opens supported files if they are dragged and dropped into the SigViewer window.

3.4.2 Read User-Defined Event Types

The GDF file format supports many predefined different types of events. Additionally, the first 255 event types are reserved for user-specific purposes. In SigViewer 0.2.6, these types are displayed with the names “condition 1”, “condition 2”, etc. However, the current biosig4c++ library can read user-defined names of these reserved types from a GDF file.

SigViewer 0.4 supports reading and displaying these names.

3.4.3 Channel Colors

SigViewer 0.3 draws each channel with the same color. With SigViewer 0.4, color settings for signal channels have been introduced. Comparable to setting the colors of event types, the dialog for choosing the visible channels can now be enhanced with a column for setting colors. Figure 3.7 shows a screenshot of the dialog. Additionally, it is possible to set the color of a channel via the context menu and to set the default color.

3.4.4 Export to PNG

This is an extension to demonstrate how exporting the current viewport to a graphic file format can look like. For this demonstration, the lossless graphic file format PNG (portable network graphic) has been chosen, which is often used for storing screenshots.

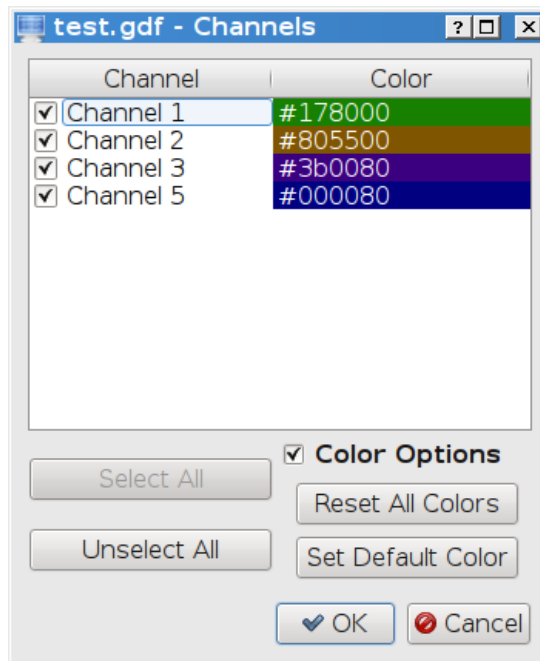


Figure 3.7: Channel Color Dialog. The color for a single channel can be set by clicking on the colored field. Then the operating system dependent color selection dialog is opened. Furthermore, all colors can be reset to the default color by clicking “Reset All Colors”. This default color can be set by clicking “Set Default Color”.

4 Graphical User Interface Improvements

This chapter describes the improvements of the graphical user interface of SigViewer.

4.1 Zooming

In SigViewer 0.2.6, the *Navigation* toolbar provides five different widgets, namely two editable comboboxes to set *seconds per page* and *channels per page* (a *page* in this sense is the viewport) and three tool buttons which display magnifiers to increase, decrease and automatically adapt the scaling of all channels along the y-axis.

With SigViewer 0.4, a new approach for zooming has been introduced. The *Navigation* and the *Options* toolbars have been integrated to a new *View Options* toolbar. The comboboxes for setting *seconds per page* and *channels per page* are not available any more. Instead, the magnifiers provide a new look and a new behavior to allow more intuitive types of zooming:

- **Zoom In/Out Vertically** changes the height of a channel (*zoom in* decreases the amount of channels per page, *zoom out* increases the amount of channels per page), shown in Figure 4.1(a).
- **Zoom In/Out Horizontally** changes the scaling along the x-axis, shown in Figure 4.1(c).

Furthermore, the actions are also located in the “View” main menu and operating system dependent standard keyboard shortcuts are provided for vertical zooming. On most operating systems, these shortcuts are **Ctrl++** to zoom in and **Ctrl+-** to zoom out.

However, it is still possible to explicitly set the *channels per page* via the context menu of the y-axis and to set the *seconds per page* via the context menu of the x-axis.

4 Graphical User Interface Improvements

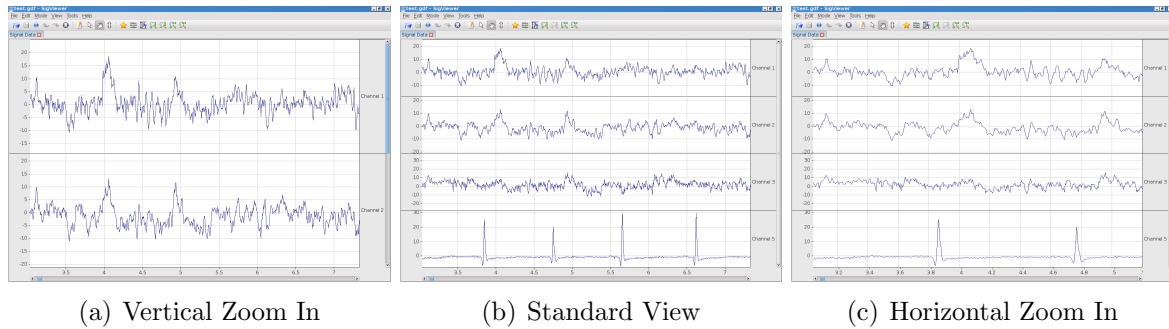


Figure 4.1: Screenshot of new zooming approach.

Implementation

The *SignalVisualisationModel* interface provides methods to set the height of one channel to influence the vertical size and methods to set the pixels used per sample to influence the horizontal zooming. These methods are called by the *ZoomGuiCommand* if the corresponding private slots of this command are triggered.

Figure 4.2 shows a diagram of the involved classes.

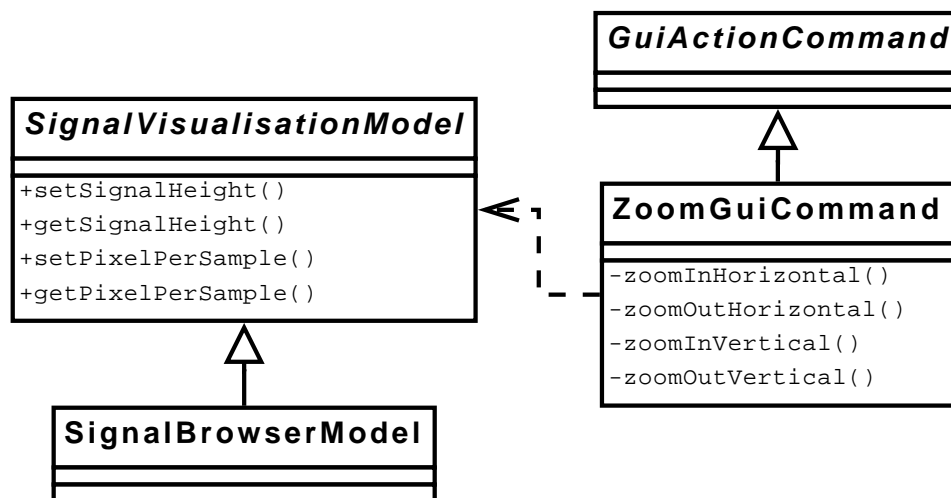


Figure 4.2: Class diagram of the new zooming implementation. The architecture of *GuiActionCommands* is described in more detail in Chapter 5.3

4.2 Context Menus

The event context menu has been completely revised. In SigViewer 0.2.6, this menu allows editing of the already selected event only, even if several events are overlapping each other. In SigViewer 0.3, the new context menu approach was introduced. Right clicking on overlapping events opens a context menu which lists all events and provides a submenu for each event. No preceding selection is necessary. Furthermore, any actions that are not triggerable are hidden. For example the action “To All Channels” is not shown if the event is already assigned to all channels. Figure 4.3 compares the context menus of SigViewer 0.2.6 and 0.4.1.

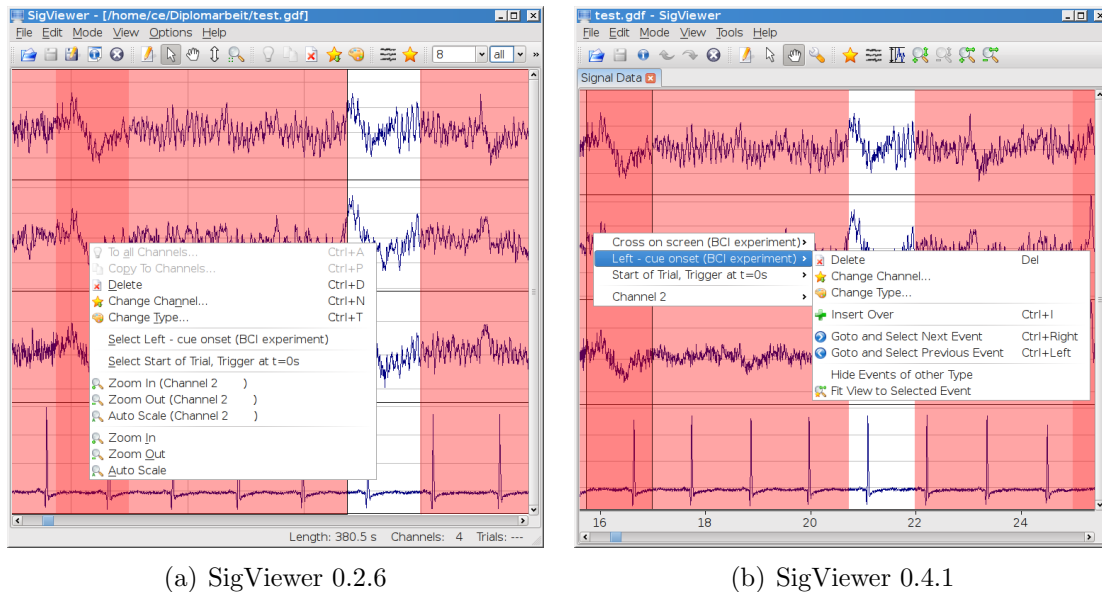


Figure 4.3: Event context menus. SigViewer 0.4.1 instantly allows editing of all events beneath the mouse cursor.

Further context menus have been implemented for

- **Channels:** to hide the channel, to set the color of the channel and to scale it
- **Y-Axis and Labels:** to set the vertical zooming (number of channels per page) and to hide these widgets.
- **X-Axis** to set the horizontal zooming (seconds per page) and to hide the x-axis widget.

4.3 Mode-specific Widgets

SigViewer 0.2.6 supports five different mouse modes. Each mode leads to different behavior of the mouse in the signal view widget (e.g. scrolling the viewport, editing an event, etc.)

In SigViewer 0.3, the *Event Toolbar* has been introduced to show information about the currently selected event. In SigViewer 0.4, the idea of this special widget has been expanded. Depending on the current mode, a special widget is shown.

Edit Event Widget

The *Edit Event Widget* is shown if the *Edit Event* mode is active. The mode allows graphical resizing of events. The widget shows navigation tool buttons and information about the currently selected event.

The following information about the events is displayed in editable widgets (as shown in Figure 4.4):

- The type of the event is displayed in a combobox (a drop-down list) which allows to change the type.
- The starting position is shown in a spinbox (an editable widget to enter numbers, the number of decimal places is automatically adapted to the samplerate).
- The duration is also shown in a spinbox (the number of decimal places is also automatically adapted).

The shown tool buttons comprise:

- Go to and select previous event
- Go to and select next event
- Fit view to selected event

The functionality of these buttons is described in the chapters 3.3.1 and 3.3.3.

View Options Widget

The *View Options Widget* is shown in the *View Options* mode. It comprises checkboxes to switch the visibility of the labels widget, the x-axis and the y-axis and buttons to set the auto scale behavior (if the zero line of a channel is centered or fitted to the

minimum and maximum of the signal). In SigViewer 0.2.6, these options are settable in the “Preferences” dialog. Due to the multitab view of SigViewer 0.4, the view options (per tab) have been moved to this new *View Options* mode which has been introduced in SigViewer 0.4.1 and replaces the *Shift Signal* mode.

New Event Widget

The widget which is shown in the *New Event* mode enables the user to set the type for the new event. This setting is used for the “Insert Over” action (described in Chapter 3.3.2) too.



Figure 4.4: Edit Event Widget. This widget shows information about the currently selected event and enables the user to edit type, start position and duration.

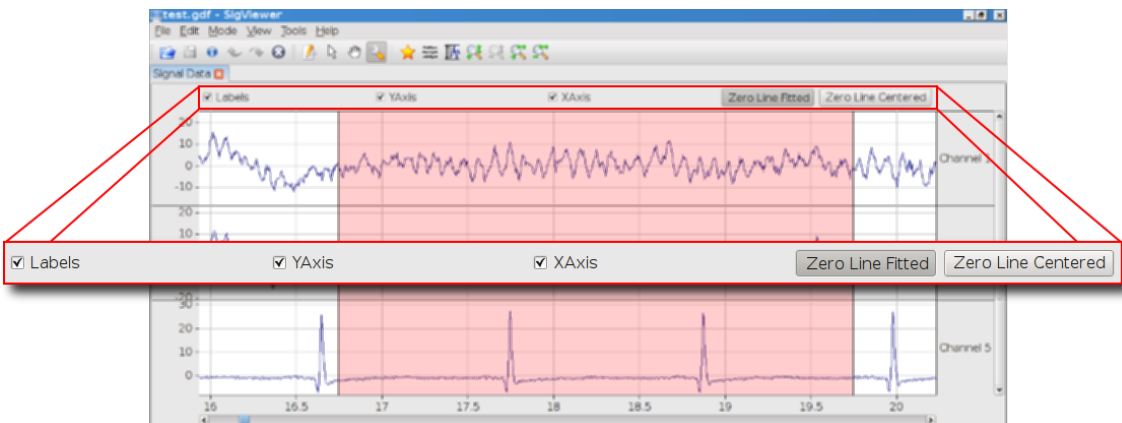


Figure 4.5: View Options Widget. This widget shows checkboxes to toggle the visibility of the labels widget, the x-axis and the y-axis.

4.4 Enhanced Dialogs

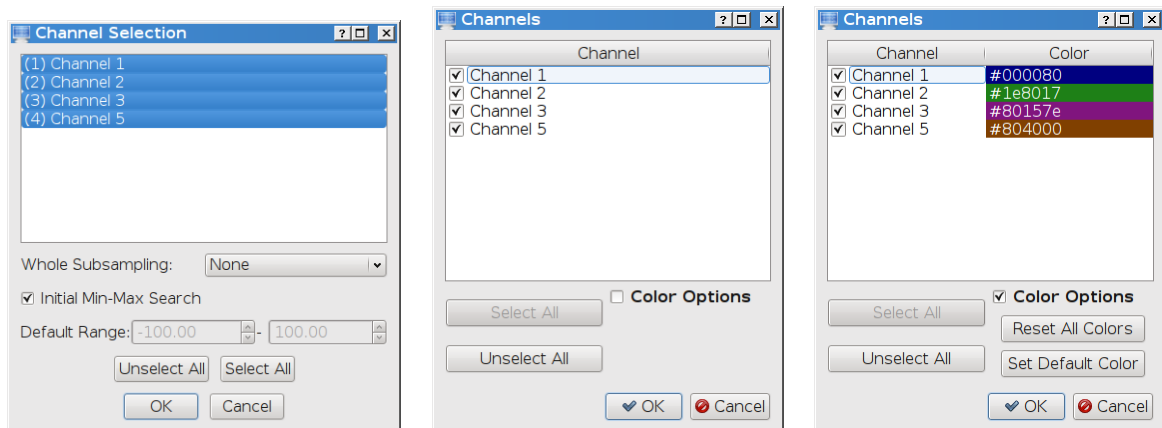
Many dialogs have been revised to provide a more compact interface at first and advanced options on demand. Furthermore, mechanisms have been implemented to avoid invalid inputs.

Channel Selection Dialog

The *Channel Selection Dialog* is shown during the process of opening a file. It allows the end user to select the channels that should be shown. The same dialog is used to set the shown channels of an already opened file. Figure 4.6 compares the dialog in SigViewer version 0.2.6 and 0.4.1.

The following features have been added and following bugs have been removed:

- Cancelling the dialog during opening a file does not open the file any more.
- “Initial Min-Max Search” is always done and therefore the checkbox and spinboxes have been removed.
- The “OK” and the “Unselect All” buttons are disabled if no channel is selected.
- The “Select All” button is disabled if all channels are selected already.
- The newly introduced channel color settings are switchable.



(a) SigViewer 0.2.6

(b) SigViewer 0.4.1

(c) SigViewer 0.4.1 including Color Options

Figure 4.6: Channel Selection Dialogs compared.

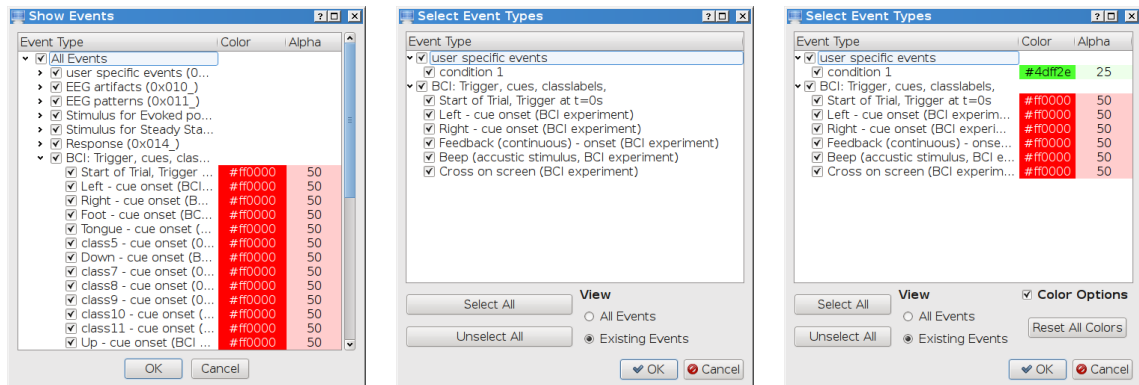
Event Types Selection

The dialog for selecting event types is used to set the visible event types and to select event types for importing/exporting events.

SigViewer 0.4.1 ships with the following improvements compared to SigViewer 0.2.6:

- The color settings are not shown if the event types are selected for importing/exporting.
- The list of event types can be shrunk to that types that are used in the currently opened file.
- The tree item “All Events” has been removed as the buttons “Select All” and “Unselect All” have been added.

Figure 4.7 shows screenshots of the dialog.



(a) SigViewer 0.2.6

(b) SigViewer 0.4.1

(c) SigViewer 0.4.1 including Color Options

Figure 4.7: Select Event Types Dialog. SigViewer 0.4 optionally provides a compact view which hides all event types that are not used in the currently opened file.

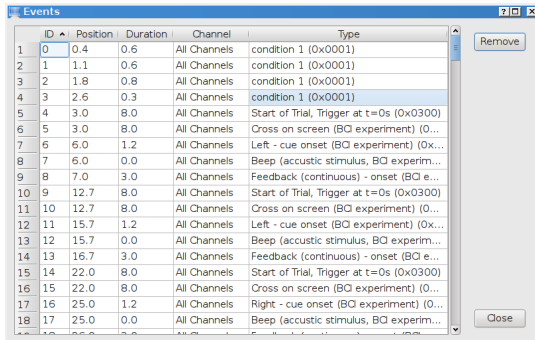
Event Table Dialog

This dialog provides an overview of all existing events arranged in a table. Currently, the dialog allows deletion of multiple events at once. Figure 4.8 shows screenshots of this dialog.

4 Graphical User Interface Improvements

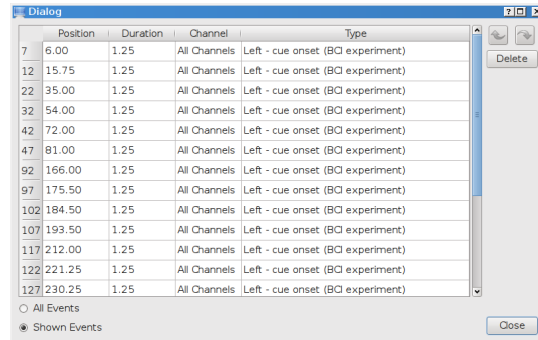
Compared to SigViewer 0.2.6, the following features have been added to this dialog:

- Show only visible events: All event types that are hidden are optionally hidden in the table too.
- Undo / redo buttons to restore the deleted events.
- The “ID” column is not visible any more as it contains no information for the end user.



ID	Position	Duration	Channel	Type	
1	0	0.4	0.6	All Channels	condition 1 (0x0001)
2	1	1.1	0.6	All Channels	condition 1 (0x0001)
3	2	1.8	0.8	All Channels	condition 1 (0x0001)
4	3	2.6	0.3	All Channels	condition 1 (0x0001)
5	4	3.0	8.0	All Channels	Start of Trial, Trigger at t=0s (0x0300)
6	5	3.0	8.0	All Channels	Cross on screen (BCI experiment) (0...
7	6	6.0	1.2	All Channels	Left - cue onset (BCI experiment) (0x...
8	7	6.0	0.0	All Channels	Beep (accustic stimulus, BCI experim...
9	8	7.0	3.0	All Channels	Feedback (continuous) - onset (BCI e...
10	9	12.7	8.0	All Channels	Start of Trial, Trigger at t=0s (0x0300)
11	10	12.7	8.0	All Channels	Cross on screen (BCI experiment) (0...
12	11	15.7	1.2	All Channels	Left - cue onset (BCI experiment) (0x...
13	12	15.7	0.0	All Channels	Beep (accustic stimulus, BCI experim...
14	13	16.7	3.0	All Channels	Feedback (continuous) - onset (BCI e...
15	14	22.0	8.0	All Channels	Start of Trial, Trigger at t=0s (0x0300)
16	15	22.0	8.0	All Channels	Cross on screen (BCI experiment) (0...
17	16	25.0	1.2	All Channels	Right - cue onset (BCI experiment) (0...
18	17	25.0	0.0	All Channels	Beep (accustic stimulus, BCI experim...

(a) SigViewer 0.2.6



Position	Duration	Channel	Type	
7	6.00	1.25	All Channels	Left - cue onset (BCI experiment)
12	15.75	1.25	All Channels	Left - cue onset (BCI experiment)
22	35.00	1.25	All Channels	Left - cue onset (BCI experiment)
32	54.00	1.25	All Channels	Left - cue onset (BCI experiment)
42	72.00	1.25	All Channels	Left - cue onset (BCI experiment)
47	81.00	1.25	All Channels	Left - cue onset (BCI experiment)
92	166.00	1.25	All Channels	Left - cue onset (BCI experiment)
97	175.50	1.25	All Channels	Left - cue onset (BCI experiment)
102	184.50	1.25	All Channels	Left - cue onset (BCI experiment)
107	193.50	1.25	All Channels	Left - cue onset (BCI experiment)
117	212.00	1.25	All Channels	Left - cue onset (BCI experiment)
122	221.25	1.25	All Channels	Left - cue onset (BCI experiment)
127	230.25	1.25	All Channels	Left - cue onset (BCI experiment)

All Events
 Shown Events

(b) SigViewer 0.4.1

Figure 4.8: Event Table Dialog

4.5 Small Improvements

Beyond the already presented improvements, some further (small) improvements are summarized in this section.

4.5.1 Animations

Some animations have been introduced to provide smooth transitions from one view setting to another. This ensures that the end user can track the transition of the view and does not have to reorient.

Currently, the zooming (Chapter 4.1) and the event browsing (Chapter 3.3.1) is animated. More ideas are described in Chapter 7.2.

The “View” main menu allows to set the duration of the animations and to deactivate animations at all.

Implementation

Qt 4.6 provides a special *Animation Framework* which allows smooth transitions of defined properties (member variables) by using the class *QPropertyAnimation*. After setting the property of interest, the start value, the end value and the desired duration of the animation, the framework automatically calls the according setter-methods.

4.5.2 Tool- and Statusbar

The visibility of the toolbars and the statusbar can be toggled via the “View” menu. Hiding these bars enlarges the viewport for the signals. Furthermore, the statusbar has been cleaned up and the *Length* and *Channels* labels are not shown if no file is open. The *Trials* label has been removed on user request. The toolbar has been cleaned up too. The buttons for editing events have been removed because the functionality is already located in the “Edit” menu, the context menu of events and the new mode-specific widgets (Chapter 4.3).

4.5.3 X-Position Highlighting

The current x-position of the mouse is highlighted in the x-axis during some special actions (e.g. event resizing) and within some special modes.

5 Restructuring and Refactoring

The general aim of restructuring and refactoring is to improve the structure of the source code. Software quality criteria like testability, modularity, expandability, etc. play an essential role within these tasks. The implementation of new features moves into the background (in a strict sense, implementing new features is not desirable at all during refactoring sessions). [23] lists several indicators when the time has come to do refactorings. This comprises, for example, that refactoring should be done before new features are added or if the knowledge about the problem domain has increased and a better fitting structure of the source code can be introduced.

In SigViewer 0.2.6, many features have been added within small projects and therefore no *big* refactorings have been made so far. This has led to some so-called *bad smells in code* (as described in [24]) and other problems in the source code:

- Code duplications and big switch-case statements occur. The class `MainWindow-Model` is especially affected.
- Some classes are deeply coupled and cyclic dependencies exist, in particular around the class `SignalBrowserModel`.
- Parts of the source code depend on `Qt3Support`.
- No automatic tests exist.

In this chapter, some of the main aspects of the restructurings and refactorings are described, which will handle the listed issues. In Section 5.1, the porting of the source code from `Qt3Support` to `Qt 4` is described. Section 5.2 is about the new module structure, which focuses on reducing the coupling between the classes. The problems around the class `MainWindowModel` are handled in Section 5.3, and the new test mode is presented in Section 5.4.

5.1 Porting to Qt 4

With the transition from Qt 3 to Qt 4, some parts of the SigViewer source code could not be ported automatically. Therefore, SigViewer 0.2.6 contains some source code which uses the *Qt3Support* module of Qt 4.

An important task of this master's thesis has been to fully port SigViewer to Qt 4 and to remove all source code and runtime dependencies on Qt3Support. This has mainly included to reimplement the part of SigViewer which is responsible for signal drawing as the *Q3Canvas* framework had to be replaced by the new *Graphics View* framework. This new framework ships with more built-in functionality like event propagation, double precision (instead of integer precision), zooming, etc. [25]. Using these new features leads to less source code in SigViewer and therefore can contribute to reduce the complexity of the code.

The following classes of SigViewer were mainly affected by the porting:

- SignalBrowserModel and SignalBrowserView have been revised
- SignalCanvasItem has been converted to SignalGraphicsItem
- EventCanvasItem has been converted to EventGraphicsItem
- NavigationCanvasItem, ChannelSeparatorCanvasItem and XGridCanvasItem have been removed
- SmartCanvas, SmartCanvasRectangle and SmartCanvasView have been removed

An equivalent to the ChannelSeparatorCanvasItem seems not to be needed any more. Drawing the grid for the signals has moved into the new class SignalGraphicsItem. Therefore, no new implementation for XGridCanvasItem is needed. Additionally, the NavigationCanvasItem has been removed because the new framework of Qt provides the features for scrolling. The SmartCanvas classes have been removed as their functionality is already provided by the new Graphics View framework.

In the first step, the existing source code was adapted to the new framework to keep as many already implemented functionality as possible. However, due to other restructurings which are described in the following sections, many code parts were changed afterwards.

5.2 New Module Structure

As mentioned before, some classes in SigViewer 0.2.6 are deeply coupled. Before adding new features, it was necessary to group classes in different modules to avoid increasing coupling.

The classes of SigViewer 0.4 are therefore divided into several modules shown in Figure 5.1.

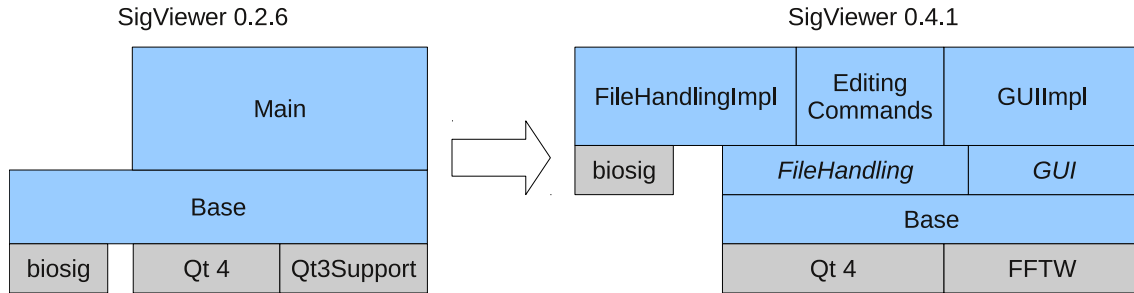


Figure 5.1: New module structure. The classes in the grey modules are external libraries. Modules only depend on subjacent modules. E.g. *FileHandling* does not depend on *GUIImpl*, but the *Editing Commands* module depends on *FileHandling*, *Base* and *Qt 4*.

Base

This module already existed in SigViewer 0.2.6. Due to the introduction of further modules, some parts of this module have been moved to other modules. Therefore, it now focuses on providing classes which represent basic data types like *DataBlock*, *SignalEvent* and user types. The module only depends on the *Qt 4* and the *FFTW* libraries.

The *DataBlock* class has been newly introduced to store and handle different kinds and sections of signal data. It provides index-based access and static methods to calculate the mean of a list of *DataBlocks*. Furthermore, it has a method to create a power spectrum of the data it contains by using the *FFTW* library (Chapter 2.3.4).

File Handling

In this module, classes are located which provide interfaces to the file handling. This comprises the abstract base classes *BasicHeader*, *FileSignalReader* and *FileSignalWriter*

for basic access to the data of a file and ChannelManager and EventManager which provide higher level access to that data.

Editing Commands

This module contains all classes which are responsible for editing events. It only depends on the *base* and the *file handling* module as shown in Figure 5.1. Each class is derived from QUndoCommand.

The implementation details about the undo-framework including these classes is described in Chapter 3.1.1.

File Handling Impl

The implementations of different FileSignalReaders and -Writers are located in this module. At the moment, it basically contains the BioSigReader and -Writer, which depend on the biosig4c++ library.

For testing purposes, a dummy implementation of FileSignalReader has been implemented which generates simple sine waves. More about this is described in Chapter 5.4.

GUI

Abstract interface classes for user interaction and signal drawing which are needed by other modules are located in this module. This comprises an interface for the MainWindow and SignalVisualisationModel, a progress bar dialog (which visualizes the progress of the file loading) and the GuiActionFactory. The latter is the central place to get QActions for nearly any kind of user action which may be directly put into toolbars or context menus. These QActions are initialized in the GUI Impl module in subclasses of GuiActionCommand.

GUI Impl

This module is a larger one as it contains the implementations of the *GUI module* which includes the drawing parts, user input and user dialog handling. Therefore, it is subdivided into further modules.

The Commands Submodule contains the implementations of the GuiActionCommand. Each GuiActionCommand creates several QActions, connects to their *trigger* signals and registers the QActions in the GuiActionFactory. In succession the factory

provides these QActions to other parts so that these QActions can be put into menus or toolbars.

In SigViewer 0.4.1, the following implementations of GuiActionCommand exist which provide the listed actions:

AdaptChannelViewGuiCommand: Setting displayed channels, setting the color of one channel, scaling channels, hide one channel, auto scale all channels, setting auto scale mode (zero line centered/fitted)

AdaptEventViewGuiCommand: Set shown event types, fit view to selected event, hide events of other type, show all events, goto and select next/previous event (Chapter 3.3)

CloseFileGuiCommand: Close file, exit application

EventEditingGuiCommand: Delete, change type, change channel, to all channels, copy to channels, show event table, insert over (Chapter 3.3)

HelpGuiCommand: Show “About SigViewer” dialog, run tests (Chapter 5.4)

MouseEventGuiCommand: Switch between the following modes: View options, scroll, edit event, new event

OpenFileGuiCommand: Open, import events, show file info

SaveGuiCommand: Save, save as, export events, export to GDF, export to PNG (Chapter 3.4)

SignalProcessingGuiCommand: Calculate mean, create power spectrum (Chapter 3.2)

UndoRedoGuiCommand: Undo and redo (Chapter 3.1)

ZoomGuiCommand: Zoom in/out vertical/horizontal (Chapter 4.1)

Figure 5.2 shows how the implementation moved from the class MainWindowModel in SigViewer 0.2.6 to the new classes in SigViewer 0.4.

The Dialogs Submodule comprises the more complex dialogs for user input. The dialogs are created with Qt Designer (Chapter 5.5). The dialogs are:

- ChannelDialog for setting the shown channels and channel colors
- EventTableDialog for providing a table-based overview of all events
- EventTimeSelectionDialog for setting the parameters for event-related signal processing (Chapter 3.2)
- EventTypeSelectionDialog for setting the shown events and event colors
- ScaleChannelDialog for setting the upper and lower boundary of the y-axis of one or all channels

The Signal Browser Submodule contains the implementations of SignalVisualisationModel and SignalVisualisationView. The latter is the central widget of SigViewer. It uses the *Graphics View* framework of Qt for drawing signals and events. Furthermore, the implementation of the mode-specific widgets, which are described in Chapter 4.3, are located in this submodule.

5.3 MainWindowModel Refactoring

In SigViewer 0.2.6, the MainWindowModel has become a very problematic class. The public interface of this class consists of 45 methods. Almost all user input handling and user dialog is implemented in these methods. Each new entry in the menu or the toolbar of the main window requires a new method. Therefore, a refactoring of this architecture was necessary.

Figure 5.2 shows a class diagram that illustrates the main aspect of the refactoring which follows the “Extract Class” approach in [24]. As shown in the figure, the implementation moved from public methods into private methods of subclasses of GuiActionCommand. In SigViewer 0.2.6, the class MainWindow constructs QActions which have been connected to public slots of MainWindowModel. In SigViewer 0.4, the subclasses of GuiActionCommand request their base class to automatically construct QActions and then connect these actions to their private slots. Therefore, much more compact public interfaces exist which counters the formation of a “Blob” [26] (a class that monopolizes the processing).

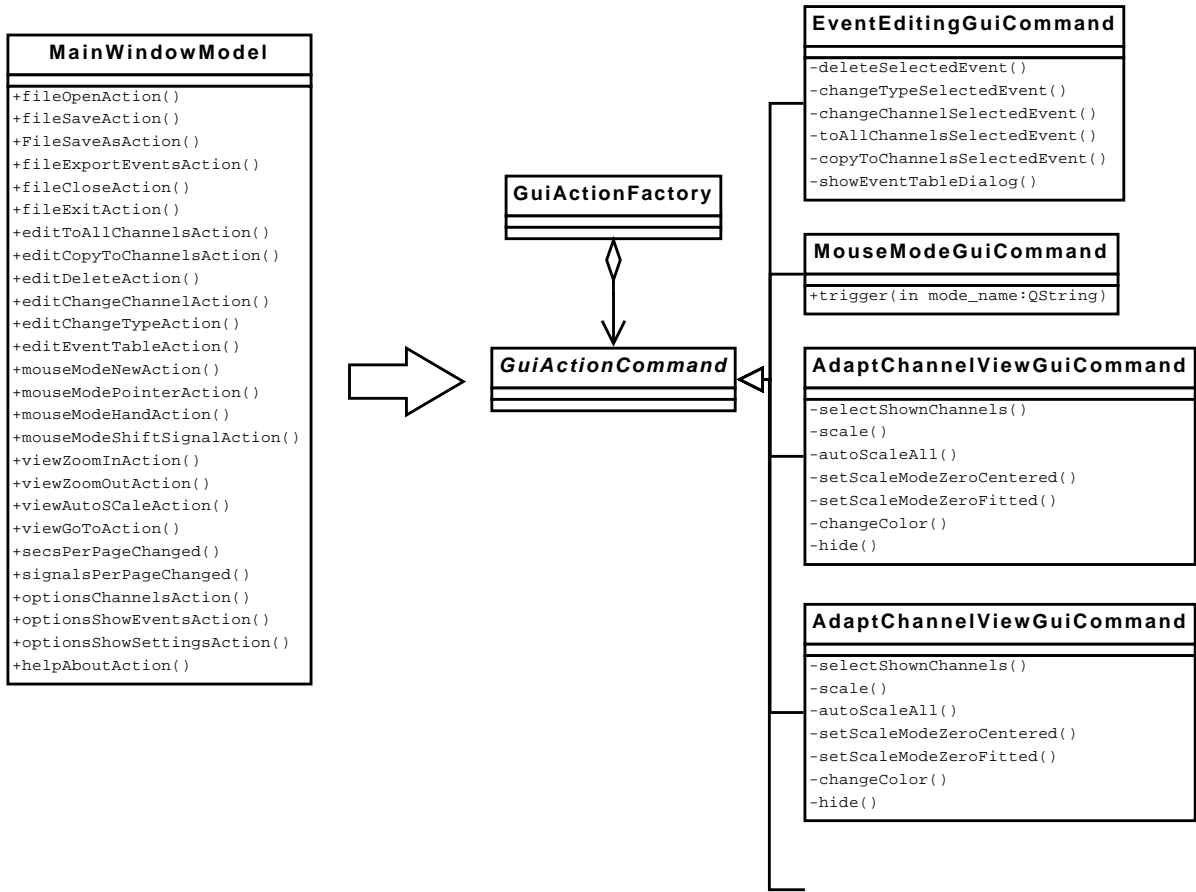


Figure 5.2: Aspects of MainWindowModel Refactoring: In SigViewer 0.2.6, the MainWindowModel implemented nearly all actions that could be triggered by the user. In SigViewer 0.4, the implementation moved into private methods of GuiActionCommand subclasses.

5.4 Automated Tests

Automated tests are very important for the software development process. However, writing automated tests for a GUI based application like SigViewer is quite difficult. Therefore, no tests existed for SigViewer 0.2.6.

However, due to the restructuring, which was already described in this chapter, the testability of the non-GUI source code of SigViewer has increased and also writing some automated GUI tests became possible.

A special *test mode* has been integrated to SigViewer. This mode is started with the commandline parameter `-test` (the parameter changed to `--test` in version 0.4.2).

Within this mode a dialog (Figure 5.3) is provided which runs tests and displays the results. The following tests are currently implemented:

- DataBlock tests
- EventManager tests
- Editing commands tests
- GUI commands tests

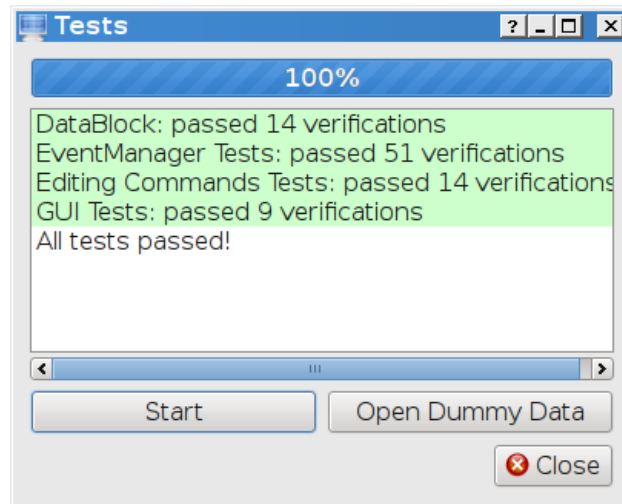


Figure 5.3: Tests Dialog: This dialog is shown if SigViewer is started in the *test mode* (commandline parameter “-test”). Clicking the “Start” button runs the tests and displays the results in the list. The “Open Dummy Data” button closes the dialog and generates simple sine waves which are shown in SigViewer (see Figure 5.4)

Additionally, the test data can be loaded into SigViewer. This data is generated during runtime by the SinusDummyReader which is an implementation of the FileSignalReader. However, instead of reading data from a file, it generates sine waves at different frequencies. Figure 5.4 shows a screenshot.

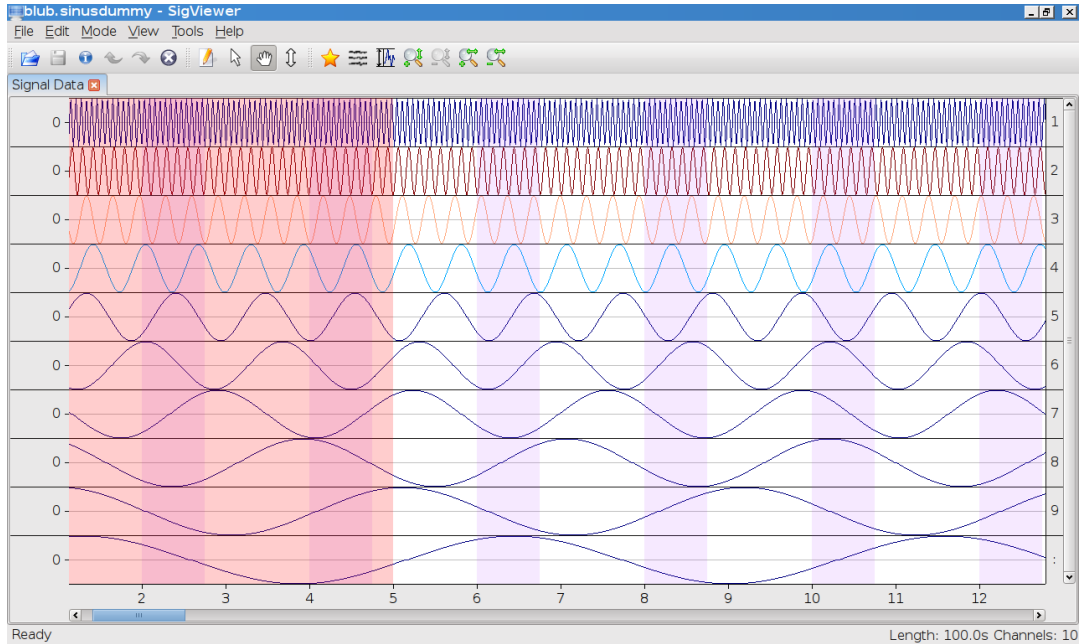


Figure 5.4: Screenshot of test data which is generated during runtime by the Sinus-DummyReader to test the functionality of SigViewer without the need to open a real file.

5.5 Qt Designer

In SigViewer 0.2.6, the layouts of all dialogs are defined in the source code, which leads to many lines of code that describe where a widget is positioned. However, Qt provides Qt Designer, a special tool to design the layout of dialogs and widgets graphically. Figure 5.5 shows a screenshot of Qt Designer. This tool is also integrated into Qt Creator. Using Qt Designer leads to several benefits:

- Creating and changing of dialogs is simplified due to the WYSIWYG (what you see is what you get) approach.
- Less source code is needed, the readability of the source code is improved and the complexity is reduced.

The data of the layout and the widgets is stored in a so-called *ui* file. This file has to be converted into a C++ class by the *Qt user interface compiler*, which is done automatically by *qmake* if the ui-file is listed in the project file (in the case of SigViewer that means in the file *src.pro* or in any *.pri* file in the subdirectories of the *src* directory). The documentation of Qt 4.6 [27] lists several different ways to use the *ui* files in the application. In SigViewer, the “Single Inheritance Approach” has been chosen. In

5 Restructuring and Refactoring

contrast to the other approaches the designed widget is only derived from `QWidget`. The layout of the widget is applied afterwards via an `Ui` member object, which is initialized in the constructor of the widget.

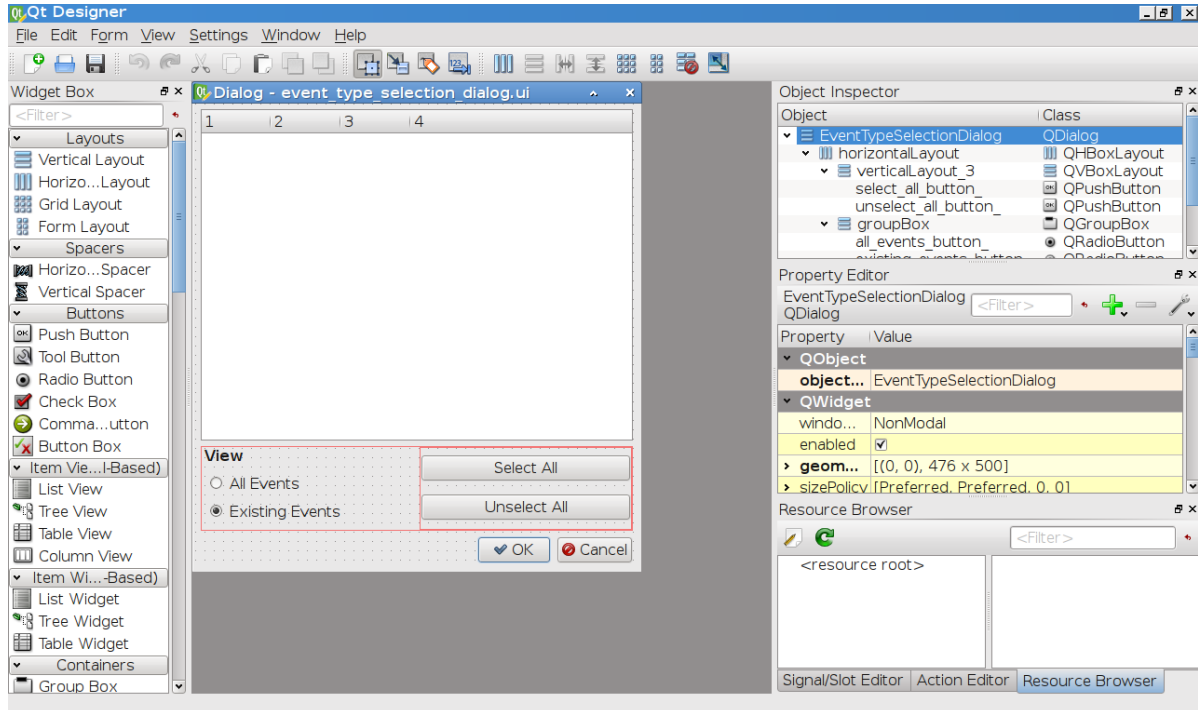


Figure 5.5: Screenshot of Qt Designer editing `EventTypeSelectionDialog`. The layout of the dialog can be changed graphically, which simplifies creating user-friendly dialogs a lot.

6 Deployment

Deployment is the way from the source code files of a software to the ready-to-use application for the end user. The following sections describe how to build SigViewer from the source code files and how to create packages for installing SigViewer on various platforms.

6.1 Building SigViewer from Source

This is a short description for how to build SigViewer from the source code files.

1. Get the source files of SigViewer from the project website <http://sigviewer.sf.net> and unzip them into a directory
2. Setup environment
 - a) Get and install the latest Qt 4 SDK including Qt Creator from <http://qt.nokia.com>
 - b) Get the precompiled biosig4c++ library from the SigViewer project website and unpack it into the *extern* directory of the SigViewer source directory
 - c) Windows and Mac only: Get the precompiled FFTW library from the SigViewer project website and unpack it into the *extern* directory of the SigViewer source directory.
 - d) Linux only: install the fftw3 development library (libfftw3-dev)
3. Start Qt Creator and open the file *sigviewer.pro*
4. Build the project *sigviewer* and run it

A more detailed description of how to build SigViewer on different platforms can be found at <http://sigviewer.sf.net/develop.html>.

6.2 Installation Packages

For users who are not involved in the development of SigViewer, it is important to provide easy installation routines of SigViewer without using build tools like a compiler. Therefore, binary packages to install the compiled SigViewer for multiple platforms are provided.

Part of this master's thesis has been to provide a Debian package [28] to simplify installation on Debian-based Linux distributions like Ubuntu.

All files are available on the SigViewer project website: <http://sigviewer.sf.net>.

6.2.1 Debian Package

The Debian package management system is used by many Linux distributions such as Debian, Ubuntu, Kubuntu, etc. Therefore, it has been of interest to provide a Debian package of SigViewer to support these platforms.

The Qt libraries are dynamically linked to SigViewer in Linux. Because Debian packages support dependencies, the Qt libraries are not included in the package. The package management systems automatically check the dependencies and will install missing libraries during the installation process of the SigViewer package.

For creating a Debian package, the *dpkg* tool is used. It requests a directory structure which contains the binary and a special *control* file which describes the package.

Directory Structure

The directory structure (shown in Figure 6.1) reflects the locations where the files from the package will be copied during the installation process. For example, the *sigviewer* binary will be copied into the */usr/bin* directory.

Debian Control File

The control file consists of several fields. The most important ones are listed in table 6.1. A complete description of the fields can be found in the manpages of *deb-control* (for example at [http://man.cx/deb-control\(5\)](http://man.cx/deb-control(5))).

6 Deployment

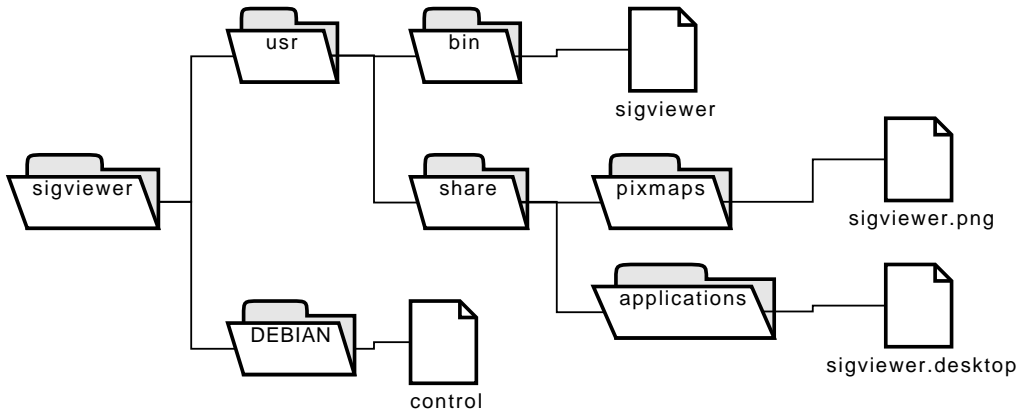


Figure 6.1: Directory structure of a Debian package.

Field	Description
Package	the name of the package
Version	the version number of the application
Depends	a list of other packages (which may contain libraries, applications, etc.) that are needed (e.g. the Qt runtime libraries)
Replaces	name and version number of the packages which are replaced (e.g. older versions of SigViewer)

Table 6.1: Debian package *control* file entries.

The Desktop File

The *sigviewer.desktop* file in the */usr/share/applications* directory contains information for the desktop entry [29] on many Linux desktops (for example GNOME or KDE).

Field	Description
Name	name of the application (SigViewer)
Exec	path to the executable (<i>/usr/bin/sigviewer</i>)
Icon	filename of the icon which should be located in <i>/usr/share/pixmaps</i> (<i>sigviewer.png</i>)
Categories	the categories in which the application will be listed in the desktop menu (e.g. Science, Education)

Table 6.2: Desktop file entries.

Build Script

To simplify the creation of a Debian package, a shell script has been created. This script automatically builds the directories and copies the necessary files to the right place. Furthermore, it automatically extracts the build architecture and the file size. Then it calls the *dpkg* tool for creating the package and afterwards deletes the directories.

6.2.2 Windows

The download statistics of SigViewer show that Windows seems to be one of the major platforms on which SigViewer is used.

For building the Windows installer, the *NSIS* tool is used (Nullsoft Scriptable Install System, <http://nsis.sf.net>). A configuration file for this tool already existed for SigViewer 0.2.6. The Windows installation package additionally contains the runtime libraries for Qt 4.

6.2.3 Mac OS X

SigViewer 0.4 has been also tested on Mac OS X 10.6. In contrast to SigViewer 0.2.6 and 0.3.0, a release has been possible, because compiler-related problems have been resolved. The installation package is built with the Qt deployment tool for Mac which is named *macdeployqt*. This tool bundles the executable into a so-called *dmg* package which also contains the needed runtime libraries of Qt.

7 Outlook

This chapter summarizes some ideas for new features, improvements of the user interface and refactorings of SigViewer.

7.1 Further Features

7.1.1 Undo View Setting

A core feature of SigViewer is to *display* biosignals and to allow the user to dynamically adapt the view. Many possibilities exist for the end-user to change the settings of the view like selecting the shown channels or fit the view to the selected event. However, only editing actions which alter the opened file can be undone currently. Providing menu entries and shortcuts to instantly restore the last view setting would increase the usability of SigViewer.

7.1.2 ERD/ERS Maps

Event-related desynchronization (ERD) and event-related synchronization (ERS) are characteristics of the brain, which are detectable in EEG [16].

The signal processing features of SigViewer could be expanded with the creation of ERD/ERS maps as mentioned in Chapter 3.2.

7.1.3 EOG Artifact Correction

One major noise source in EEG recordings is the so-called EOG artifact (which originates in electrical eye activity). However, automatic methods exist to remove these artifacts [11]. Such a method could be integrated in SigViewer to provide a filtered view on EEG data.

7.2 Further Improvement of GUI

7.2.1 Info Widget

Introduce an *Info* widget for the power spectrum and mean view. This widget can display information about the event that has been processed and may provide buttons to change the setting of the processing.

7.2.2 Event Table

The event table dialog could be integrated in the tab view. Beside the “Signal Data” tab, there could always be an “Events” tab. This would decrease the number of windows used by the application.

7.2.3 Animations

Smooth transitions are important for user-friendly interfaces. Some animations have been already implemented (Chapter 4.5.1). The following list presents some ideas for further work:

- Animate the “Fit View to Event” action.
- Fade out event markers that are deleted or hidden.
- Fade in event markers that are set to be visible.
- If a channel is hidden, shrink its height smoothly from its current value to zero.
- Roll out or shrink the x-axis, y-axis and label widget if their visibility changes.

7.3 Further Refactoring

7.3.1 Improved File Support

Currently, the library *libgdf* (<http://libgdf.sf.net>) is being developed at the Institute for Knowledge Discovery. This library contains optimized reader and writer for the GDF file format. SigViewer could use this library to improve its support for GDF.

Furthermore, converting other biosignal file formats into GDF could be realized.

7.4 Deployment

7.4.1 User Guide

One of the most important tasks of the future will be to provide a comprehensive user guide of all functions of SigViewer. This user guide should be written in a flexible format, so that it is possible to provide it as a printable document, as a website, and also as built-in help in the SigViewer program.

7.4.2 Further Linux Packages

SigViewer 0.4.1 has been deployed in binary Debian packages for Ubuntu 10.04 for 32 and 64bit platforms. Additionally, binary packages for other widely spread Linux distributions like Fedora (<http://fedoraproject.org>), openSUSE (<http://www.opensuse.org>), Debian (<http://www.debian.org>), etc. could be provided.

8 Concluding Remarks

Within the first two weeks after the release of SigViewer 0.4.0, it has been downloaded more than 225 times from <http://sigviewer.sf.net>. As no advertisement has been made, this shows some mentionable interest in this software application. Additionally, one positive rating has been published on the project website (by someone not affiliated with the project).

Compared to SigViewer 0.2.6, the latest version ships with lots of new features, bug fixes, improvements of the user interface, and improvements in the source code structure. Although cleaning up source code is an almost never-ending task, the restructurings (described in Chapter 5) allow faster implementation of new features in the future. The features which have been added in the scope of this master's thesis comprise basic functionalities like undo/redo, color settings and convenient methods for file opening. Furthermore, some new features are adjusted to the application domain of BCI research and comprise event-based signal processing, event browsing, etc. In addition, efforts have been spent to increase the usability of SigViewer. A graphical user interface that can be used easily is an important component for the success of a program.

However, considering all feature requests and wishes for SigViewer would have been beyond the scope of this master's thesis. Therefore, Chapter 7 gave an overview of some ideas for further improvements and extensions which may build the basis of future SigViewer releases.

Bibliography

- [1] A. Schlögl and C. Brunner. BioSig: A Free and Open Source Software Library for BCI Research. *IEEE Computer*, 41(10):44–50, 2008.
- [2] A. Schlögl, C. Vidaurre, and E. Hofer. BioSig - Standardization and Quality Control in Biomedical Signal Processing Using the BioSig Project. In *BIOSTEC 2008*, 2008.
- [3] A. Schlögl. GDF - A General Dataformat for Biosignals, 2009. Available online at <http://arxiv.org/abs/cs.DB/0608052> (last visited: 28th September 2010).
- [4] B. Kemp, A. Värri, A.C. Rosa, K.D. Nielsen, and J. Gade. A simple format for exchange of digitized polygraphic recordings. *Electroencephalography and Clinical Neurophysiology*, 82:391–393, 1992.
- [5] B. Kemp and J. Olivan. European data format 'plus' (EDF+), an EDF alike standard format for the exchange of physiological data. *Clinical Neurophysiology*, 114:1755–1761, 2003.
- [6] G. Schalk, D.J. McFarland, T. Hinterberger, N. Birbaumer, and J.R. Wolpaw. BCI2000: A General-Purpose Brain-Computer Interface (BCI) System. *IEEE Transactions on Biomedical Engineering*, 51(6):1034–1043, 2004.
- [7] J.R. Wolpaw, N. Birbaumer, D.J. McFarland, G. Pfurtscheller, and T.M. Vaughan. Brain-Computer Interfaces for Communication and Control. *Clinical Neurophysiology*, 113:767–791, 2002.
- [8] R. Schreiber. MATLAB. *Scholarpedia*, 2(7):2929, 2007.
- [9] J.W. Eaton. About Octave, 2006. Available online at <http://www.gnu.org/software/octave/about.html> (last visited: 28th September 2010).
- [10] Free Software Foundation. GNU General Public License, 2007. Available online at <http://www.gnu.org/licenses/gpl-3.0.html> (last visited: 28th September 2010).

Bibliography

- [11] A. Schlögl, C. Keinrath, D. Zimmermann, R. Scherer, R. Leeb, and G. Pfurtscheller. A Fully Automated Correction Method of EOG Artifacts in EEG Recordings. *Clinical Neurophysiology*, 118(1):98–104, 2007.
- [12] M. Werner. *Signale und Systeme. Lehr- und Arbeitsbuch mit MATLAB®-Übungen*. Vieweg+Teubner, Wiesbaden, 3rd edition, 2008.
- [13] P.E. Black. Big-O Notation. In P.E. Black, editor, *Dictionary of Algorithms and Data Structures [online]*. U.S. National Institute of Standards and Technology, 2008. Available online at <http://www.itl.nist.gov/div897/sqg/dads/HTML/bigOnotation.html> (last visited: 28th September 2010).
- [14] P.E. Black. Divide and Conquer. In P.E. Black, editor, *Dictionary of Algorithms and Data Structures [online]*. U.S. National Institute of Standards and Technology, 2010. Available online at <http://www.itl.nist.gov/div897/sqg/dads/HTML/divideAndConquer.html> (last visited: 28th September 2010).
- [15] P. Steffas. Fast Fourier Transform. In P.E. Black, editor, *Dictionary of Algorithms and Data Structures [online]*. U.S. National Institute of Standards and Technology, 2007. Available online at <http://www.itl.nist.gov/div897/sqg/dads/HTML/fastFourierTransform.html> (last visited: 28th September 2010).
- [16] B. Graimann, J.E. Huggins, S.P. Levine, and G. Pfurtscheller. Visualization of Significant ERD/ERS Patterns in Multichannel EEG and ECoG Data. *Clinical Neurophysiology*, 113(1):43–47, 2002.
- [17] M. Frigo and S.G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on “Program Generation, Optimization, and Platform Adaptation”.
- [18] M. Frigo and S.G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, volume 3, pages 1381–1384. IEEE, 1998.
- [19] J.C. Bowman and M. Roberts. FFTW++: A Fast Fourier Transform C++ Header Class for the FFTW3 library, 2010. Available online at <http://fftwpp.sourceforge.net> (last visited: 28th September 2010).
- [20] Nokia Corporation. Overview of Qt’s Undo Framework, 2010. Available online at <http://doc.qt.nokia.com/4.6/qundo.html> (last visited: 28th September 2010).

Bibliography

- [21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, 1995.
- [22] M.G.H. Coles and M.D. Rugg. Event-Related Brain Potentials: An Introduction. In M.G.H. Coles and M.D. Rugg, editors, *Electrophysiology of Mind: Event-Related Brain Potentials and Cognition*, pages 1–27. Oxford University Press, 1996.
- [23] A. Hunt and D. Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, Harlow, England, 1999.
- [24] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Reading, MA, USA, 1999.
- [25] Nokia Corporation. The Graphics View Framework, 2010. Available online at <http://doc.qt.nokia.com/4.6/graphicsview.html> (last visited: 28th September 2010).
- [26] W.J. Brown, R.C. Malveau, H. W. McCormick, and T. J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley, 1998.
- [27] Nokia Corporation. Qt Designer Manual, 2010. Available online at <http://doc.qt.nokia.com/4.6/designer-manual.html> (last visited: 28th September 2010).
- [28] The Debian GNU/Linux FAQ. Chapter 7 - Basics of the Debian package management system, 2008. Available online at http://www.debian.org/doc/FAQ/ch-pkg_basics (last visited: 28th September 2010).
- [29] P. Brown, J. Blandford, O. Taylor, V. Untz, and W. Bastian. Desktop Entry Specification. Version 1.0, 2008. Available online at <http://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-1.0.html> (last visited: 28th September 2010).