Fachhochschule Nordwestschweiz
Hochschule für Technik

Fachbericht

Realtime Monitor für spektrale Leistungsdichte
Projekt 2 Team 1

Windisch, 06.03.2021

Hochschule	Hochschule für Technik - FHNW
Studiengang	Elektro- und Informationstechnik
Autor	Bhend Matthias, Glutz Fabian,
	Jauslin Timeo, Moser Daniel,
	Rohner Melvin, Schwarz Roman
Betreuer	Anita Gertiser & Pascal Buchschacher
Auftraggeber	Manuel Di Cerbo & Stefan Gorenflo

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Ein Realtime Monitor ist heute bereits weit verbreitet und hat ein breites Anwendespektrum. Er detektiert Störungen via Vibration, Schall oder Magnetfelder und wertet die Daten entsprechend aus. Solch ein System wird häufig verwendet, um die Umwelt oder wichtige Industrieanlagen zu überwachen und um sie damit zu schützen. Das heisst, dass diese Methode der Datenakquise bereits weit verbreitet ist. Jedoch ist eine vollumfängliche App, wie diejenige welche entwickelt werden sollte, noch nicht vorhanden.

Der Hauptfokus der Arbeit wurde vorgegeben und soll auf der Signalverarbeitung, dem Logging und dem Auslösen von Alarmen liegen, denn die komplette Entwicklung eines Realtime Monitors sprengt deutlich den Rahmen des Projektes.
Zentral in der Entwicklung der Software ist, dass diese auch ohne GUI, nämlich auf einem Embedded System, laufen kann. Damit kann das System auf einem Raspberry Pi zum Beispiel auf der Baustelle, fest installiert werden und es kann darauf verzichtet werden immer einen Computer zu benützen. Weiter wurden folgende Ziele vorgeschrieben: Die Daten sollen laufend in ein LogFile gespeichert werden. Das Spektrum kann beliebig in Frequenzbänder eingeteilt und darin weiter individuell eingestellt werden. Die Samplingrate, der Window Overlap, die Telefonnummer, Emailadresse und die Warnmeldung sollen nach Belieben gewählt werden können.

Wie bereits erwähnt ist die Signalverarbeitung ein wichtiger Aspekt in der Software. Nachfolgend wird das in der Software benutzte Verfahren in der Abbildung 1.1 dargestellt.

Abbildung 1.1: Signalflussdiagramm

Dank der Teilung des Signals und der Software in diverse kleinere Pakete, kann jedes Paket als eigenes Teilproblem separat gelöst und am Schluss zu einem funktionierenden System zusammengesetzt werden. Ein grosser Vorteil bei einem solch modularen Aufbau ist die bessere Testbarkeit.

Das Ergebnis wird zeigen, dass in der Applikation verschiedene Sensoren hinzugefügt werden können und pro Sensor diverse Einstellungen gemacht werden können. Doch dazu später noch mehr. Alle relevanten Daten, wie zum Beispiel der maximale und minimale Wert, werden periodisch dem User in ein Logfile geschrieben. Falls eine Überschreitung der eingestellten Grenze beim jeweiligen Signal auftritt, bekommt der User umgehend eine Nachricht auf seinem gewünschten Weg.

Der nachfolgende Bericht ist wie folgt unterteilt: Zu Beginn kommen die theoretischen Grundlagen, welche vorausgesetzt werden, um ein solches Projekt überhaupt realisieren zu können. Hier besonders nennenswert sind die Fourier-Transformationen (FTA & DFT), welche die Basis für die Signalverarbeitung liefern.
Als nächstes kommt die Realisierung der Signalverarbeitung, gefolgt vom Aufbau der Software. Hier wird erläutert, wie die Software konzeptioniert und schliesslich als GUI realisiert wurde, was im nächsten Abschnitt folgt. Zum Schluss werden die Ergebnisse der Tests und des Projekts im Ganzen erklärt.

2 Theoretische Grundlagen

Abbildung 2.1: Signalflussdiagramm markiert

Zu Beginn wird ein analoges, zeitkontinuierliches Signal ausgelesen. Das heisst es wird zu bestimmten Zeitpunkten abgetastet und in ein digitales, zeitdiskretes Signal verwandelt. Somit folgt, dass das Signal $x(t)$ zu $x(kT_s) \rightarrow x[k]$ umgewandelt wird. Wobei auf das Abtastkriterium (Nyquist-Shannon-Abtasttheorem) geachtet werden muss, was zur Folge hat, dass die minimale Abtastfrequenz grösser sein muss, als zweimal die maximale Frequenz des zu betrachtenden Signals. Ansonsten kann es zu Aliasing-Effekten kommen.

$$F_{Abtast} > 2 * F_{signal} \tag{2.1}$$

2.1 Leck-Effekt und Fensterung

Abbildung 2.2: Signalflussdiagram markiert

Da ungefilterte Signale nicht rein periodisch sind, gelten sie als quasiperiodische Signale. Weil diese Signale «verunreinigt» sind, entsteht die Problematik, dass eine periodische Darstellung nicht ohne Sprungstelle ist. Die DFT kann jedoch nur periodische Signale genau darstellen. Bei quasiperiodischen Signalen entstehen nach der Transformation bei den Unstetigkeitsstellen neue Frequenzen, welche im ursprünglichen Signal gar nicht vorhanden waren. Dies nennt sich: der Leck-Effekt. Da solche Sprungstellen beim Welch-Verfahren und auch bei der DFT entstehen, beide verfahren werden weiter unten noch erklärt, werden die Signale noch zusätzlich gefenstert. Dieses Fensterungs-verfahren kann bei allen möglichen Signalen mit Sprungstellen angewendet werden.

Das Verfahren funktioniert so, dass das Signal mit der Fensterungskurve gefaltet wird. Dadurch entsteht eine Verringerung der Sprungstellen. Und es wird so einem Überschwingen vorgebeugt und der Leck Effekt beseitigt.

Es gibt verschiedene Fensterungstypen2.3, welche je nach Signal und erwarteten Störungen ausgewählt werden können. Jeder Fensterungstyp birgt seine Vor- und Nachteile. [2, s.187ff]

2.2 Fourier-Transformation für Abtastsignale (FTA)

Wie aus der allgemeinen Fourier Theorie bekannt ist, wird die Fouriertransformation angewendet, um Signale vom Zeitbereich in den Frequenzbereich zu transformieren. Da in diesem Projekt nicht mit periodischen Signalen gearbeitet wird, kann die normale Fourier Transformation nicht angewendet werden. Da aber dieses Projekt nicht das erste ist mit diesem Problem wurde hierfür schon lang eine Lösung gefunden. Diese Lösung nennt sich die Fouriertransformation für Abtastsignale (FTA). Hierbei wird die Ausblendeigenschaft von Diracstössen verwendet, was uns zu dieser Formel führt.

$$X(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x[n] \cdot e^{-jn\Omega}; \qquad \Omega = \omega \cdot T = \frac{\omega}{f_A} \tag{2.2}$$

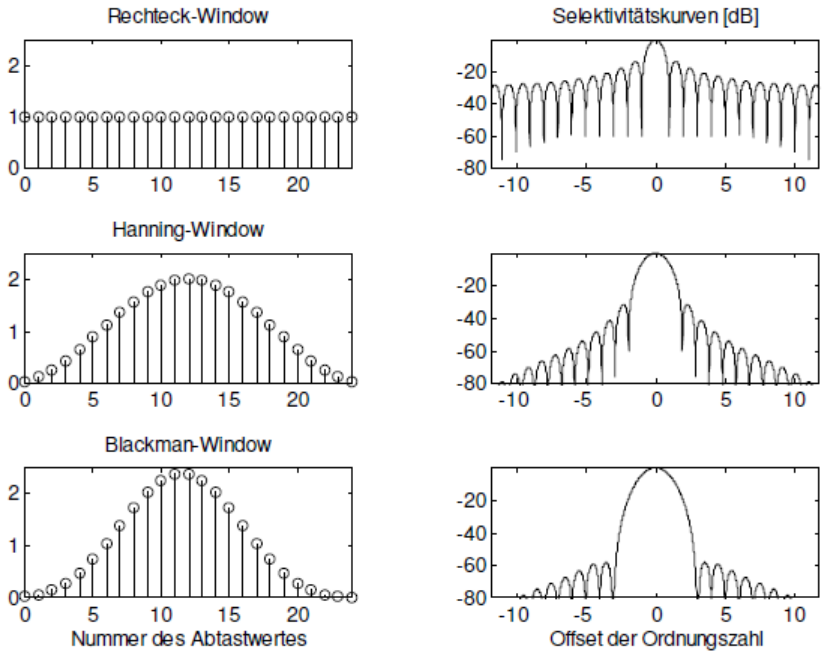

Abbildung 2.3: Verschiedene FFT-Windows: links: Zeitbereich, rechts: Frequenzbereich [2, s.189]

Abbildung 2.4: Signalflussdiagramm markiert

sowie die Inverse der FTA:

$$x[n] = \frac{T}{2\pi} \int_{-\pi/T}^{\pi/T} X(e^{j\Omega}) \cdot e^{jn\Omega} d\omega \tag{2.3}$$

Hieraus kann man eine Parallele zur normalen Fouriertransformation schliessen und man erkennt, dass diskrete Signale ein periodisches Spektrum haben und periodische Signale ein diskretes Spektrum. [2, s.152 ff]

2.3 Diskrete Fourier Transformation (DFT)

Es besteht nun aber immer noch das Problem, dass das Signal von $-\infty$ bis ∞ zusammen summieren werden muss. Da dies in der Praxis nicht sehr gut machbar ist, wurde die Diskrete Fourier Transformation (DFT) entwickelt. Sie stellt eine abgetastete Näherung an die FTA dar. DFT:

$$X[m] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi \frac{mn}{N}} \tag{2.4}$$

Sowie die Inverse der DFT:

$$x[n] = \frac{1}{N} \sum_{m=0}^{N-1} X[m] \cdot e^{j2\pi \frac{mn}{N}} \tag{2.5}$$

wobei:

x[n]	Folge der Abtastwerte
X[m]	Folge der komplexen Amplituden (Spektralwerte)
N	Anzahl der Abtastwerte im Zeitfenster = Blocklänge
n = 0,1,...,N-1	Nummer der Abtastwerte
m = 0,1,...,N-1	Nummer der Spektrallinien, Ordnungszahl

Als Merksatz hierzu gilt:

Das DFT-Spektrum ist die abgetastete Version des FTA-Spektrums. Es ist diskret und periodisch.

Die DFT ist einfach zu programmieren. Die Abtastwerte werden in einem Array(Vektor) abgelegt mit n als indice. Die komplexen Amplituden werden auch in einem Array zurück gegeben, nur ist hier m der indice.[2, s.164 ff]

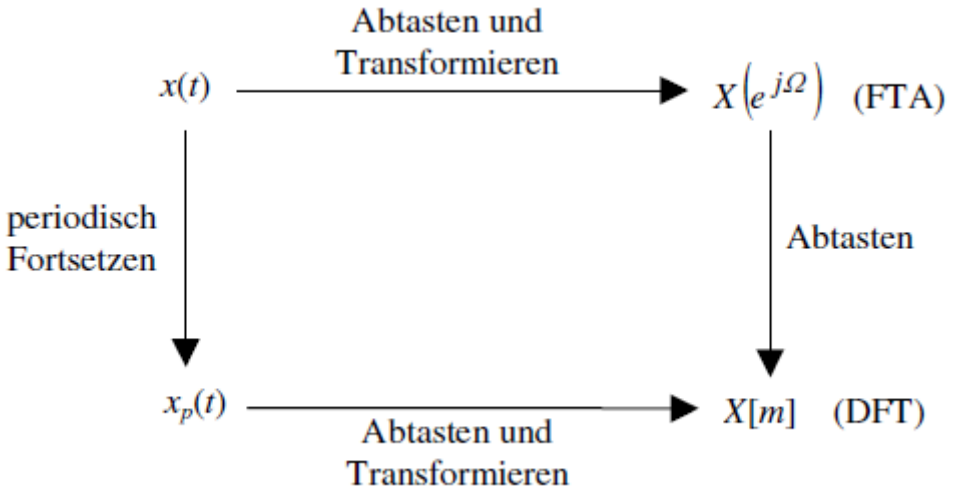

Abbildung 2.5: Lineare Abbildung der DFT [2, s.168]

2.3.1 Schnelle Fourier Transformation (FFT)

Um die DFT für eine spezifische Frequenz mit einem Computer zu berechnen benötigt man N komplexe Multipliaktionen. Für das ganze Spektrum sind es schon N^2 komplexe Multiplikationen. Bei reellen Zeitsequenzen genügt es bereits $\frac{N}{2}$ Spektralwerte zu berechnen, was $\frac{N^2}{2}$ komplexe Multiplikationen braucht. Man erkennt jedoch, dass der Rechenaufwand schnell sehr hoch wird. Da aufgrund der Periodizität von Sin/Cos sehr viele Werte identisch sind wurde 1965 der Fast-Fourier-Transform-Algorithmus (FFT) entwickelt. Die Idee des FFT-Algorithmus ist es, eine lange Zeitsequenz in zwei kurze aufzuteilen. Damit werden zwei DFT's mit halber Blocklänge berechnet, was wegen dem quadratischen Ansteigens des Rechenaufwandes eine enorme Einsparung bringt. Die beiden Blöcke können natürlich weiter unterteilt werden. Im Prinzip ist jede Unterteilung von N in ganzzahlige Blöcke anwendbar. Folglich werden immer möglichst kleine Blöcke gebildet. Am einfachsten sind natürlich Blöcke, welche eine Zweierpotenz sind.

$$X[m] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}mn} = \sum_{n=0}^{N-1} x[n] \cdot (e^{-j\frac{2\pi}{N}})^{mn} = \sum_{n=0}^{N-1} x[n] \cdot W_N^{mn} \tag{2.6}$$

wobei: $W_N = e^{-j\frac{2\pi}{N}}$ ein komplexer Faktor ist, der nur von der Länge der Blöcke abhängig ist.

2.4 Mittelung des Signals

Abbildung 2.6: Signalflussdiagram markiert

Da das Signal nun immer noch ein Rauschen aufweist, wird es mit Hilfe von stochastischen Eigenschaften wie Mittelwert, Varianz und Korrelationsfunktionen beschrieben. Folglich geht man von der Fourier Transformation zu einem Leistungsdichtespektrum über, dieses kann auch messtechnisch ermittelt werden. Es ist daher naheliegend, mit der DFT das Leistungsdichtespektrum zu schätzen.

$$S_n = \frac{|DFT|^2}{N} = \frac{|X_n|^2}{N} \tag{2.7}$$

Diskret: mittlere Leistung

$$\frac{1}{N}\sum_{k=0}^{N-1} x[k]^2 = \frac{1}{N}\sum_{n=0}^{N-1} \frac{|X_n|^2}{N} \tag{2.8}$$

sowie die unnormierte Darstellung:

$$\frac{1}{N}\sum_{k=0}^{N-1} x[k]^2 = \frac{f_s}{N}\sum_{n=0}^{N-1} \frac{|X_n|^2}{N} \tag{2.9}$$

[1] Diese Methoden führen aber weiterhin zu einem sehr grossen Varianzanteil in der Darstellung des Leistungsdichtespektrum. Um dem entgegenzuwirken gibt es das Welch-Verfahren.

2.4.1 Welch-Verfahren

Für eine zusätzliche Qualitätsverbesserung der Varianz gibt es das Welch-Verfahren. Wird mit dem Welch-Verfahren der Datensatz in kleine Segmente, die anschliessend gemittelt werden. Hierbei kommt es aber immer zu einem Overlap der jeweiligen Teilabschnitte. Weshalb dann über die einzelnen Teilabschnitten gemittelt wird. [1]

2.5 Signalverarbeitung Realisierung

Das Signal $x(t)$ kommt vom Sensor in das System rein und wird zuerst im A/D Wandler $x[k]$ abgetastet. Dies wurde in der Software durch einem Signalgenerator simuliert, welcher danach mit den Tastwertenabgetastet wird. Danach wird das Signal gefenstert S_n über die Fensterungsfunktionen, welche ausgewählt werden können. Weitere Fensterungsfunktionen können noch implementiert werden. Das Gefensterte Signal wird nun der FFTfunktion übergeben und dann in den Frequenzbereich überführt X_n. Danach wir das Signal noch quadriert und dann gemittelt. Am Ende wird das Einseitige Leistungsdichtespektrum$X[m]$ geplottet.

Abbildung 2.7: Signalflussdiagramm

3 Software Implementierung

3.1 GUI Funktionalität

In diesem Kapitel wird der Aufbau und der Funktionsumfang des GUI beschrieben. Das Herzstück des GUI ist der "Realtime-Monitor". Auf dem "Realtime-Monitor" werden alle konfigurierten Regeln zur Überwachung der Sensorkanäle dargestellt. Damit der "Realtime-Monitor" übersichtlich ist, wurde ein Grossteil der Einstellungsmöglichkeiten in Untermenues ausgelagert. Diese Untermenues sind direkt vom "Realtime-Monitor" erreichbar.

3.1.1 Realtime-Monitor

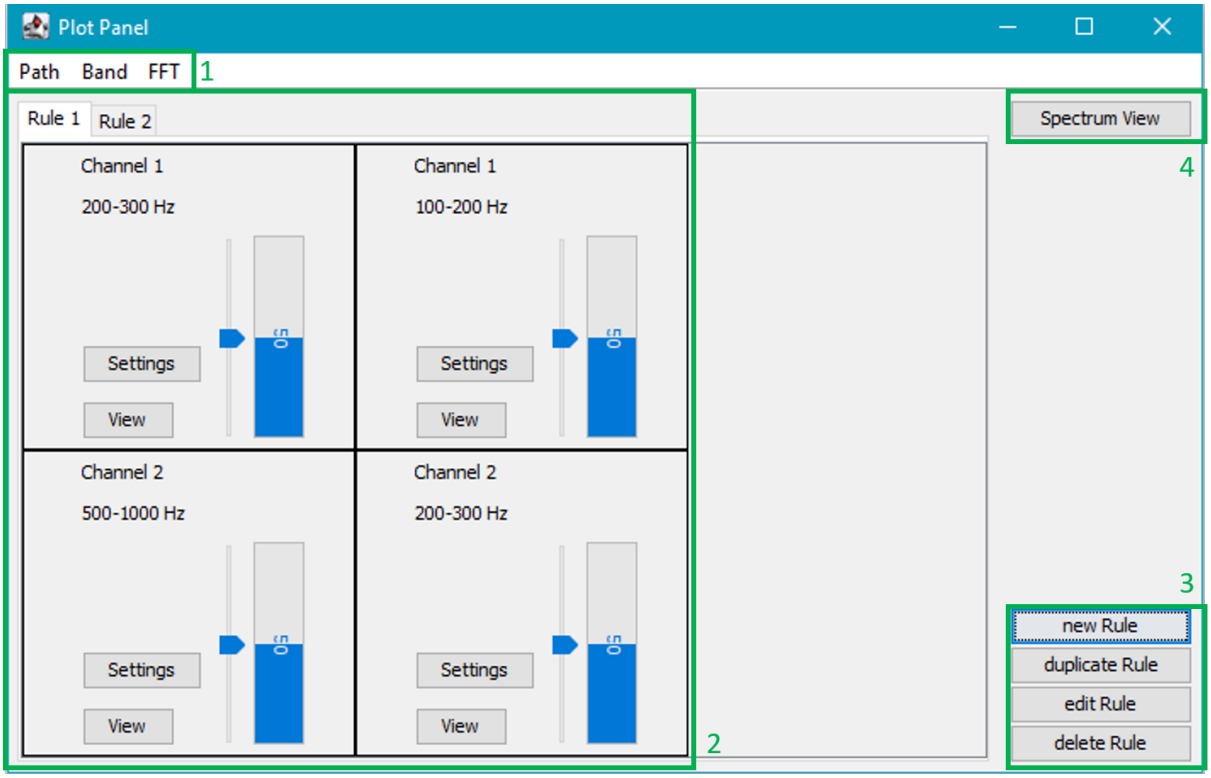

Abbildung 3.1: Realtime-Monitor (Herzstück des GUI)

Der "Realtime-Monitor" (Abb. 3.1) ist in drei Bereiche aufgeteilt: Menüleiste (1), Rules (2) und Rule-Einstellungen (3 und 4). Über die Menueleiste (1) navigiert man zu den Untermenues "FFT-Settings", "Band-Settings" und "Path-Settings". Im Bereich 2 werden die konfigurierten Regeln dargestellt. Jede Regel wird in einem separaten Tab dargestellt. Die überwachten Frequenzbänder werden nach Channel sortiert dargestellt. Über den Button "Settings" kann der Grenzwert des Signalpegels und die Anzahl der Pegelüberschreitungen pro Zeiteinheit, welche für das Auslösen eines Events nötig sind, konfiguriert werden (Abb. 3.2). Der Grenzwert für den Signalpegel kann ausserdem über den Schieberegler des Bandes eingestellt werden. Mit dem Button "View" wird eine Visualisierung der Leistungspegel der Frequenzen innerhalb des Bandes aufgerufen (Abb. 3.3).

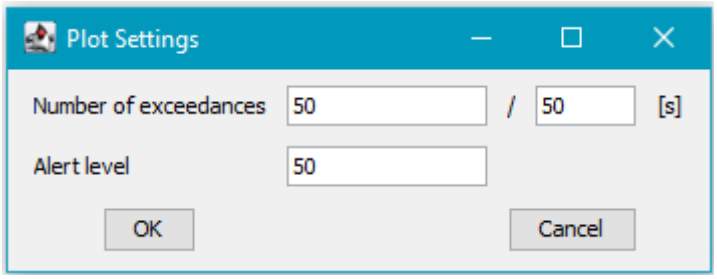

Abbildung 3.2: Settings eines Bandes

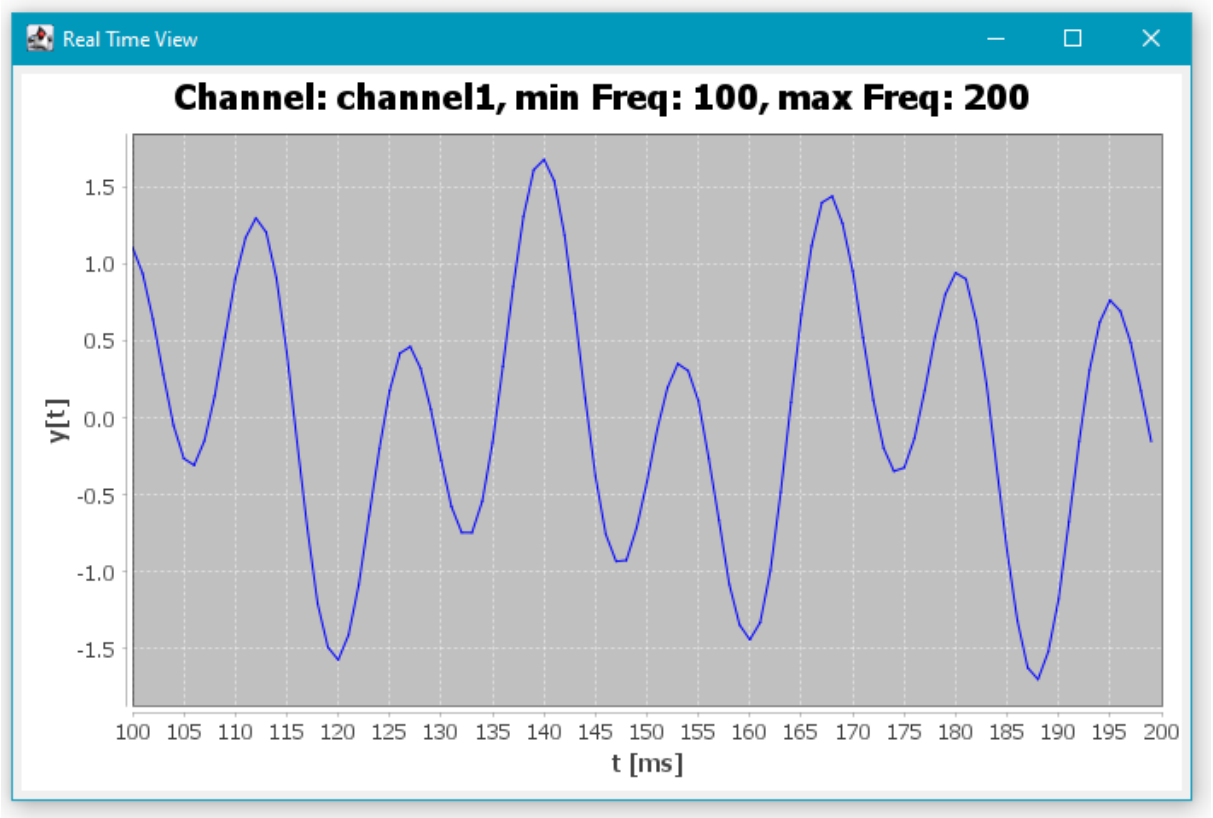

Abbildung 3.3: Visualisierung der Leistungspegel der Frequenzen in einem Band

3.1.2 FFT-Settings

In den ”FFT-Settings” (Abb. 3.4) werden die Channels für die Sensoren erstellt und verwaltet. Für alle Sensoren die überwacht werden, muss ein separater Channel angelegt werden. Im Dropdownmenue(1) werden alle bestehenden Channels aufgelistet. In den ”FFT-Settings” können für jeden Channel die nötigen Einstellungen für die Fast-Fourier-Transforamtion (FFT) vorgenommen werden. Die Zeitdauer einer einzelnen FFT wird durch die Sampel Frequency und den Overlap oder die Anzahl Samples bestimmt. Es kann jeweils nur der Overlap oder die Anzahl Samples konfiguriert werden. Über das Dropdownmenü Windowtype, kann die gewünschte Fensterung eingestellt werden. In der aktuellen Version ist das Hanning-Fenster und das Rectangular-Fenster implementiert.

Über den Button ”Add Channel” kann ein neuer Channel hinzugefügt werden. Damit der User

weiss, welcher Sensor von welchem Channel überwacht wird, kann jedem Sensor ein Name, ein Standort und eine Beschreibung zugewiesen werden(Abb. 3.5).

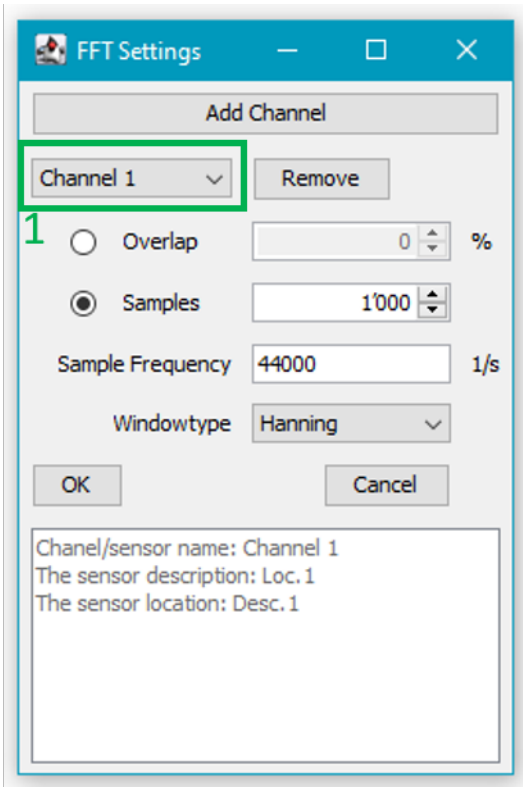

Abbildung 3.4: FFT Settings

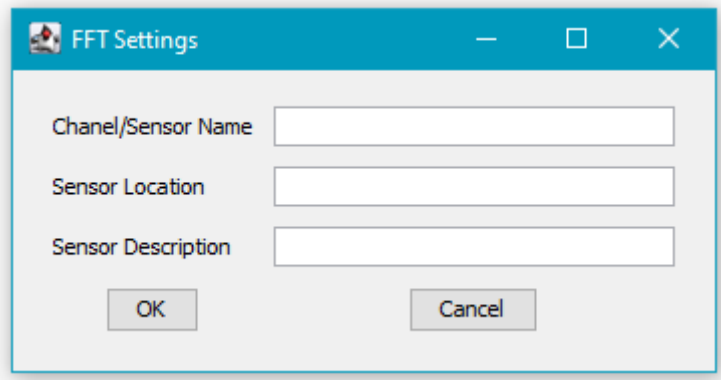

Abbildung 3.5: Erstellen eines Channels

3.1.3 Band-Settings

In der Realität wird normalerweise nicht das gesamte Frequenzspektrum eines Sensorsignales auf Pegelüberschreitungen überwacht. Damit die Überwachung auf einen bestimmten Frequenzbereich eingeschränkt werden kann, müssen in den ”Band Settings” (Kapitel 3.1.3) für jeden Channel Bänder definiert werden. Die Bänder gehören immer zu einem Channel und nicht zu einer Rule. Jedes Band kann in beliebig vielen Rules überwacht werden. Der maximal zulässige Pegel wird hingegen in der Rule definiert. Dadurch kann einem Band in verschiedenen Rules ein unterschiedlicher Maximalpegel zugewiesen werden.

Über das Dropdownmenue(1) kann ein Channel ausgewählt werden. In der Bänderliste (3), werden alle auf diesem Channel existierenden Bänder angezeigt. Die Grenzen eines Bandes werden durch eine minimale und eine maximale Frequenz festgelegt. Diese können über die Textfelder (2) definiert werden. Es können nur positiv-ganzzahlige Frequenzen eingegeben werden. Mit dem Button "Ad" kann dem ausgewählten Channel(1) ein neues Band mit den definierten Frequenzen hinzugefügt werden. Beim Versuch ein Band mit unzulässigen Werten hinzuzufügen, wird dem User eine Fehlermeldung angezeigt. Mit einem Mausklick kann in der Bänderliste ein bestehendes Band markiert werden. Mit dem Button "Remove" kann das markierte Band gelöscht werden.

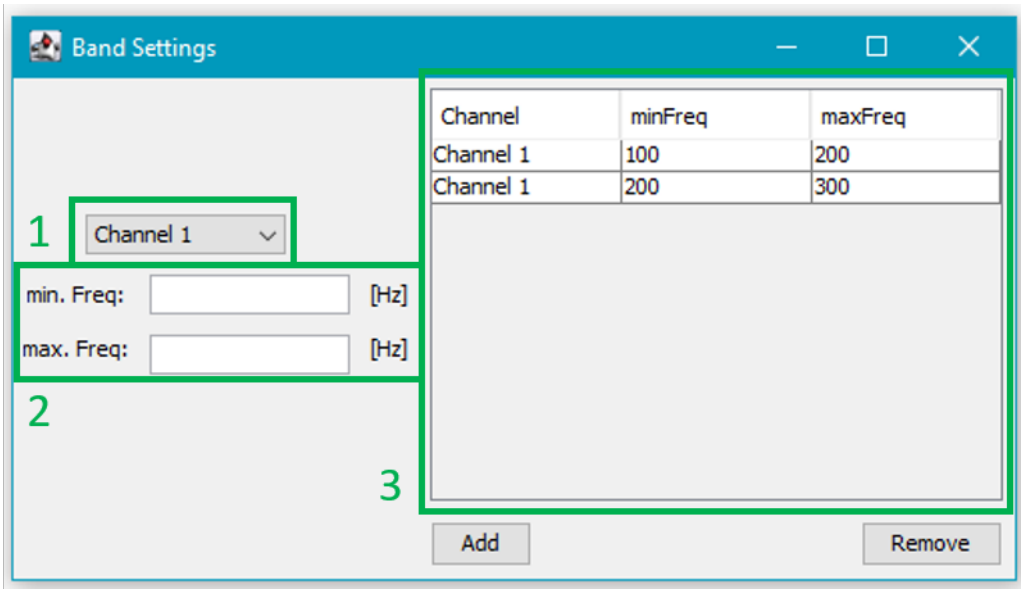

Abbildung 3.6: Band Settings

3.1.4 Path-Settings

In den "Path-Settings"(Abb. 3.7) kann der User die Einstellungen für den Log-Pfad, den Pfad für die Konfigurationsdatei, den Log-Intervall und den Eventexport vornehmen.
Standardmässig ist in den Textboxen, die den Pfad anzeigen, das Home-Verzeichnis des Users hinterlegt. Über den Knopf neben den Feldern kann der User einen Pfad zu einem Ordner auswählen, in dem das Log-file bzw. das Config-file abgelegt werden soll. Wird der Knopf gedrückt öffnet sich ein JFileChooser(Abb. 3.8) um den gewünschten Pfad auswählen zu können. Wird die Auswahl mit dem OK-Button bestätigt wird der Pfad in das entsprechende Textfeld geschrieben.
Unterhalb der Pfadauswahl kann der Log-Intervall und der Eventexport über einen JSpinner eingestellt werden. Beim ersten Starten der Applikation, ist jeweils der Wert: 60 Sekunden bei den JSpinner eingestellt. Mit der Einstellung Logintervall gibt man an, nach wie vielen Sekunden jeweils eine Logausgabe in das Logfile geschrieben werden soll. Mit Hilfe der JSpinner für den Eventexport kann der User parametrisieren, wie viele Rohdaten eines Channels vor und nach einem Regelbruch exportiert werden sollen. Die genauere Beschreibung zum Logging und Eventexport ist im Abschnitt (Kapitel 3.4) zu finden.

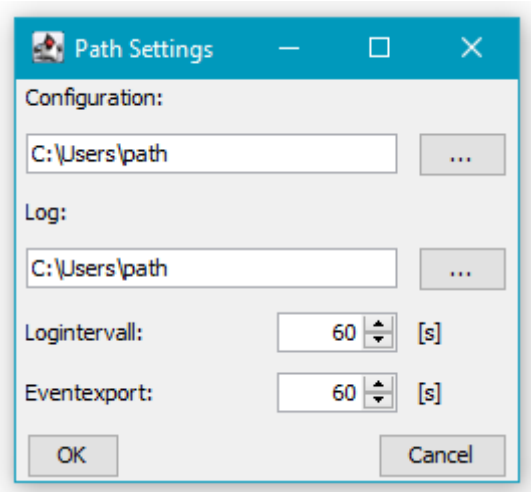

Abbildung 3.7: Path Settings

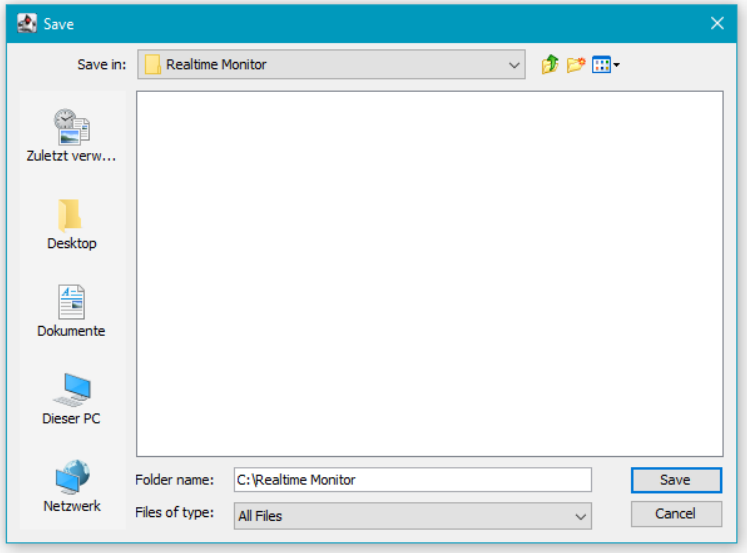

Abbildung 3.8: JFileChooser

3.1.5 Erstellen und bearbeiten einer Rule

Die Hauptaufgaben der Software sind die Überwachung von Sensoren und das Auslösen von Events. Mit einer Rule wird definiert, wann ein Event ausgelöst werden soll. Der Bereich Rule-Einstellungen(3) auf dem ”Realtime-Monitor”(Abb. 3.1) beinhaltet die 4 Buttons ”new Rule”,”duplicate Rule” , ”edit Rule” und ”delete Rule”. Mit ”new Rule” und ”edit Rule” wird der ”Rule Configurator” geöffnet. Über den ”Rule Configurator” können folgende Einstellungen konfiguriert werden:

- Name der Rule
- Überwachte Bänder
- Alert-Typ (INFO, WARNING, DANGER)
- Benachrichtigungsmethode (Telefon, SMS, E-Mail)
- Empfänger der Nachricht
- Inhalt der Nachricht

Wenn eine neue Rule erstellt oder eine bestehende Rule bearbeitet wird, öffnet sich der ”Rule Creator” im Tab ”Rule Configuration”(Abb. 3.9). Im oberen Bereich(1) kann der Rule ein Name (Textfeld) und der Alert-Type (Dropdownmenue) zugewiesen werden. Die Liste ”Available Bands”(2) enthält alle Bänder die existieren, aber der Regel nicht zugewiesen sind. Die Liste ”Selected Bands” enthält alle Bänder, die durch die Regel überwacht werden. Wenn eine neue Rule erstellt wird, werden standardmässig alle Bänder überwacht. Falls ein Band nicht überwacht werden soll, muss dieses aktiv abgewählt werden. Mit einem Mausklick kann ein Band markiert werden. Mit den beiden Pfeil-Buttons in der Mitte, kann ein Band von einer Liste in die andere verschoben werden.

Im zweiten Tab ”Message”(Abb. 3.10) wird die Nachricht, welche im Falle einer Pegelüberschreitung versendet wird, konfiguriert. Es stehen die drei Benachrichtigungsmethoden Telefon, SMS und E-Mail zur Verfügung. Mit einer Checkbox kann die jeweilige Benachrichtigungsmethode aktiviert oder deaktiviert werden. Die Telefonnummern und die Mailadressen werden über das dazugehörende Textfeld konfiguriert. Im Textfeld ”Alertmessage” kann die Nachricht, welche bei einem Event übermittelt wird, definiert werden.

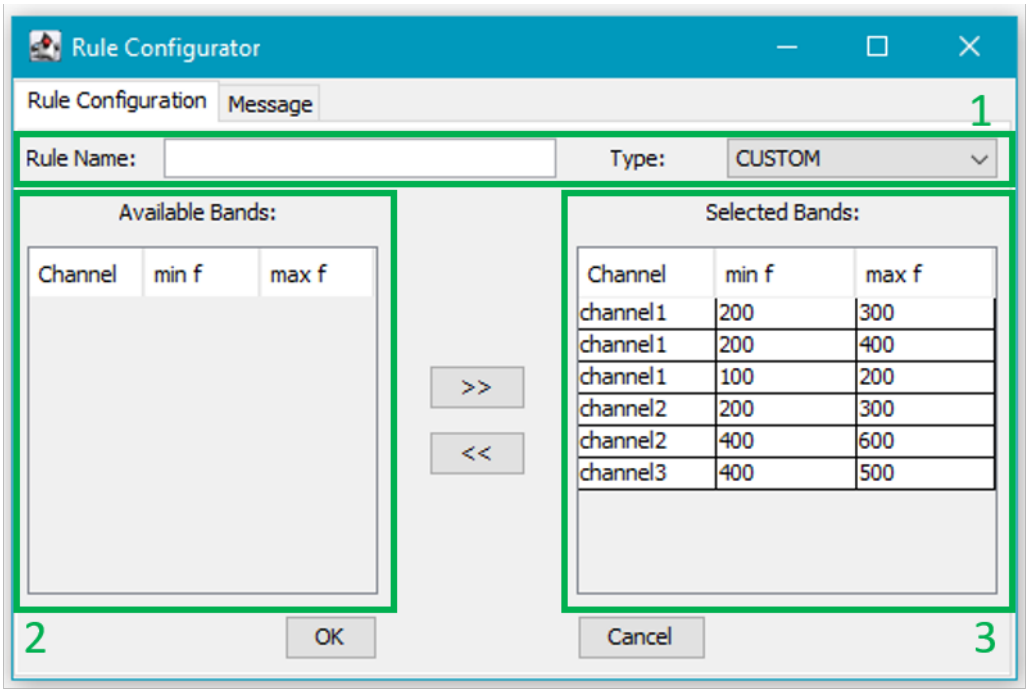

Abbildung 3.9: Rule-Settings Tab1: Auswählen der Bänder

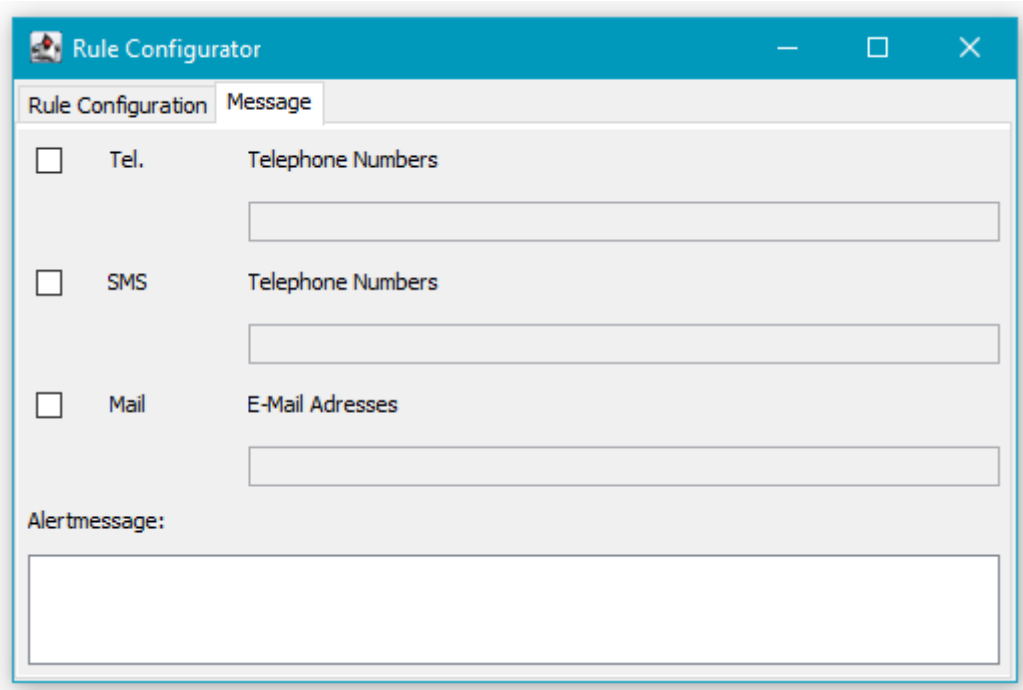

Abbildung 3.10: Rule-Settings Tab2: Konfigurieren der Nachricht

3.2 GUI Hierarchie

3.2.1 Model View Controller (MVC)

Alle Fenster des GUI's wurden nach MVC implementiert. Für jedes Fenster existiert eine Klasse View und eine Klasse Controller. Die Fenster haben keine eigenen Models. Die Fenster haben über die Klasse Ueberwachung Zugriff auf die Model der Channels, der Bänder und der Rules (Siehe Kapitel 3.3).

3.2.2 Handling der Fenster

Das GUI besteht aus mehreren Fenstern, die zur Programmlaufzeit immer wieder geöffnet und geschlossen werden. Jedes Mal, wenn ein Fenster geöffnet wird, wird ein neues Frame instanziiert und die View neu geladen. Immer wenn ein Fenster geschlossen wird, wird der Frame gelöscht. Damit wird sichergestellt, dass das GUI im Hintergrund nicht unnötig Ressourcen beansprucht. Für die Instanziierung der Fenster ist die Klasse FramGenerator zuständig. Die Klasse FrameGenerator stellt für jedes Fenster des GUI eine Methode zur Verfügung, welche ein Frame, die View und den Controller instanziiert. Über die Singelton Klasse "Controller" kann die Update-Methode von jedem Controller eines Fensters aufgerufen werden. Dadurch müssen die verschiedenen Controller nicht direkt miteinander kommunizieren und müssen sich daher nicht kennen.

3.3 Model (Channel, Band, Rule)

In diesem Kapitel wird der Fokus auf die Channels, die Bänder und die Rules gelegt. Es wird aufgezeigt, wie diese Objekte zusammenhängen und wo sie existieren.
Alle Channels, Bänder und Rules werden von der Klasse Ueberwachung verwaltet. Die Model müssen unabhängig vom GUI verfügbar sein. Die Klasse Ueberwachung ist als Singelton implementiert und kann dadurch als Schnittstelle zwischen dem GUI und den Models verwendet

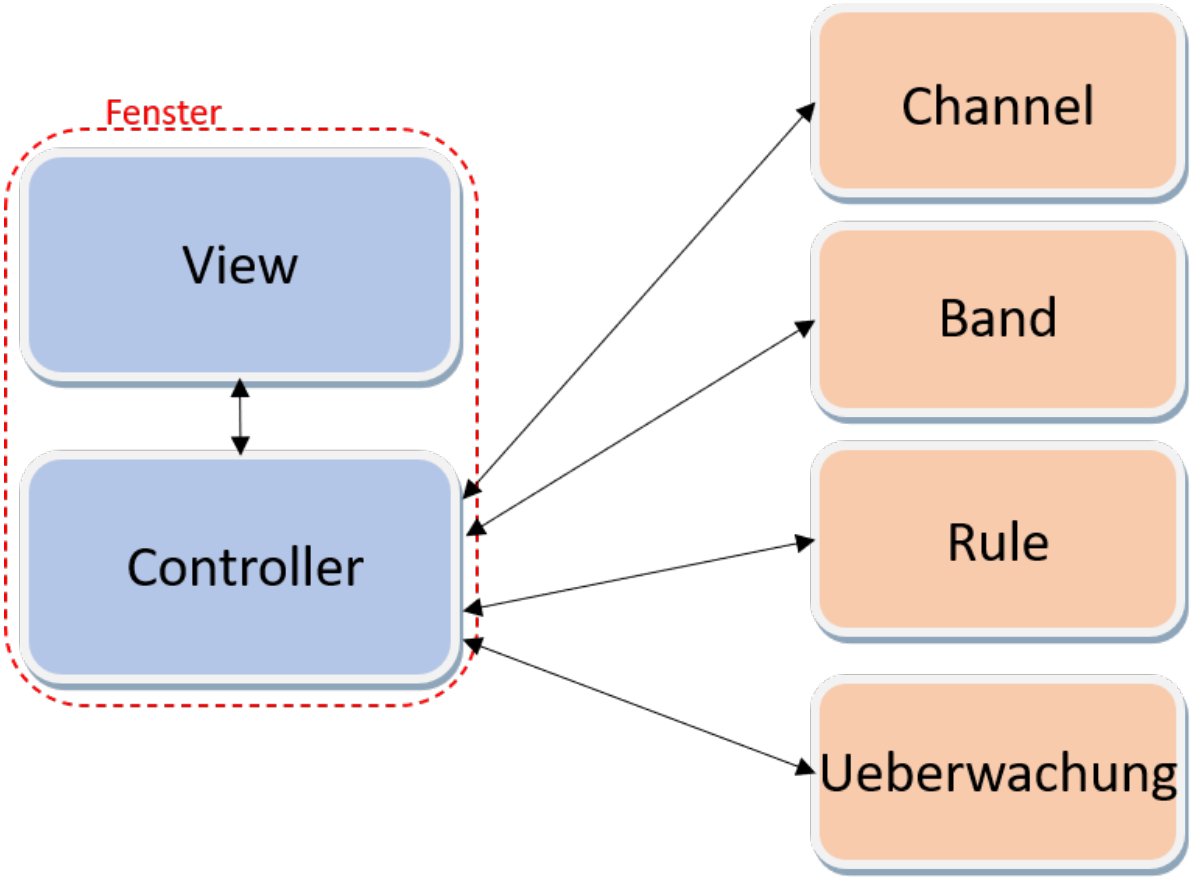

Abbildung 3.11: MVC Struktur eines Fensters

werden. Da sich die Anzahl der Objekte zur Programmlaufzeit stetig verändert, werden die Objekte in Listen und nicht in Arrays verwaltet. In der Abb. 3.12 sind die Zusammenhänge zwischen Ueberwachung, Channels, Bänder und Rules ersichtlich. Die Zusammenhänge werden in den nachfolgenden Abschnitten genauer erläutert.

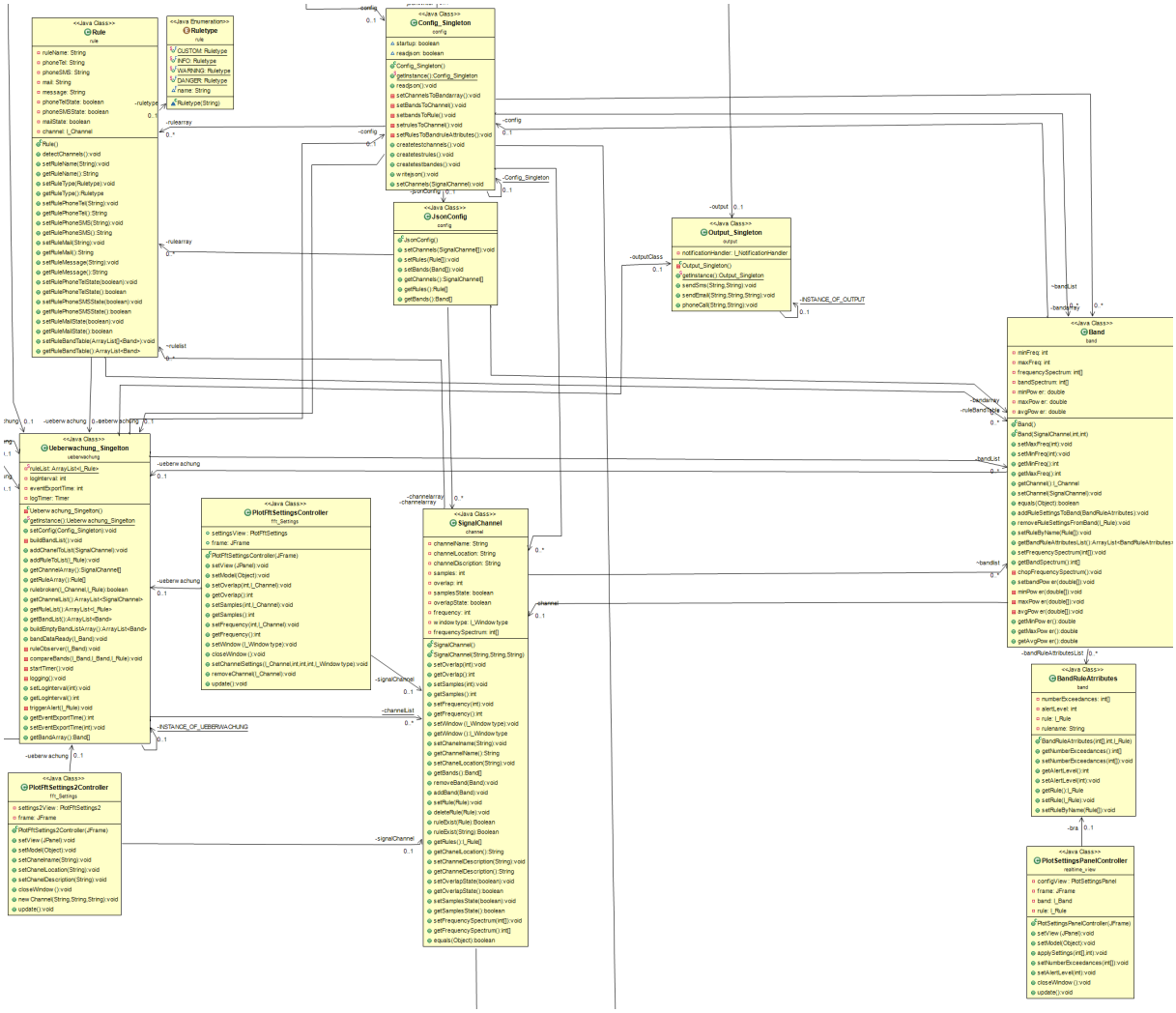

Abbildung 3.12: Verbindungen der Models im UML (Channels, Bands, Rules)

3.3.1 Channels

Jeder Channel ist ein Objekt der Klasse SignalChannel. Alle Channel-Objekte werden auf der Ueberwachung in der ArrayList ”channelList” instanziiert.

3.3.2 Bands

Ein Band gehört immer zu einem Channel. Damit ein neues Band erstellt werden kann, muss ein Channel ausgewählt sein. Das Band wird auf dem ausgewählten Channel instanziiert. Jeder Channel hat eine ArrayList ”bandList” welche alle Band-Objekte enthält, die auf dem Channel instanziiert wurden. Damit der Zugriff vom GUI her gewährleistet ist, werden die Listen der Bänder auf die Ueberwachung gespiegelt. Auf der Ueberwachung werden die Bandlisten aller Channels in einem Array abgelegt. Da sich die Anzahl der Channels zur Laufzeit der Software verändert, wird dieses Array immer wenn ein neuer Channel oder ein neues Band erstellt wird, neu erzeugt. Ein Band wird nie direkt auf der Ueberwachung instanziiert.

3.3.3 Rules

Jede Rule ist ein Objekt der Klasse Rule. Alle Rule Objekte werden auf der Ueberwachung in der ArrayList ”ruleList” instanziiert. Jede Rule kennt die Channels und die Bänder die sie überwacht. Die Rules und die tangierten Bänder sind jeweils als ArrayLists auf der Rule abgelegt

3.4 Logging / Output

In diesem Kapitel wird das Loggen, sowie der Output der Applikation beschrieben. Das Loggen erfolgt in einem eingestellten Intervall in eine Textdatei. Dabei wird jeweils pro Tag ein neues File angelegt. Bei einem Regelbruch erfolgt eine Informieren der eingestellten Benutzer. Dazu wird der NotificationHandler mit den eingestellten Werten der Rule aufgerufen. In den folgenden Abschnitten wird darauf genauer eingegangen.

Log

An Hand des Fenster in Abbildung (Abb. 3.7) kann der User die Einstellungen für das Loggen sowie für den Eventexport treffen. Nach dem Start der Applikation beginnt die Anwendung automatisch mit dem Loggen, in den angegebenen Log-Pfad. Dabei läuft in der Klasse Überwachung ein Timer der jeweils nach dem eingestelltem Logintervall eine Logging Methode aufruft (siehe Abb. 3.13)

Dabei wird von jedem eingestelltem Band die minimale, die maximale und die durchschnittliche

```
private void startTimer() {
    TimerTask task = new TimerTask() {
        public void run() {
            logging();
        }
    };

    logTimer.schedule(task, (long) logInterval * 1000, (long) logInterval * 1000);
}

/*
 * Create log report for each band
 */
private void logging() {
    Logger_Singleton logger = Logger_Singleton.getInstance();
    // Go trough all bands and create log
    for (SignalChannel channel : channelList) {
        for (Band band : channel.getBands()) {
            logger.freereport(String.format("channel name|band area|min|max|avg: %s,%d-%d [Hz],%.2f,%.2f,%.2f",
                    channel.getChannelName(), band.getMinFreq(), band.getMaxFreq(), band.getMinPower(),
                    band.getMaxPower(), band.getAvgPower()));
        }
    }
}
```

Abbildung 3.13: Codeausschnitt Logimplementation

Leistung ausgegeben und in eine Textdatei geschrieben. Um die Daten in das Textfile zu schrieben werden die Informationen der Logger-Klasse übergeben. Diese formatiert die eingegeben Daten, fügt einen entsprechenden Zeitstempel hinzu und schreibt den String in das Textfile. Die Abbildung (Abb. 3.14) zeigt ein Beispiel, wie ein solcher Log in der Textdatei aussehen kann.

```
2021/06/07 21:22:16: Channel created ChannelName: Channel1 ChannelLocation: Gächliwil ChannelDiscription: Erdbeben
2021/06/07 21:23:09: channel name|band area|min|max|avg: test,100-200 [Hz],20.00,101.00,44.00
2021/06/07 21:23:09: channel name|band area|min|max|avg: test,200-300 [Hz],20.00,101.00,44.00
2021/06/07 21:25:18: rulename: Rule1, message: Test Rule
2021/06/07 21:25:18: Test message
2021/06/07 21:25:18: [DANGER],rulename: Rule1, message: Test Rule
```

Abbildung 3.14: Ausschnitt des Log-Files

Output eines Regelbruchs

In der Abbildung (Abb. 3.14) ist neben den periodischen Logeinträgen auch ein Regelbruch abgebildet. Wird in der Klasse Überwachung festgestellt, dass der Pegel eines Bandes, den durch den Benutzer eingestellte Maximalwert überschreitet, wird die Methode triggerAlert aufgerufen (siehe Abbildung Abb. 3.15). In dieser Methode werden die Informationen der entsprechenden Regel an die Output-Klasse weitergeben und ein Logeintrag im Logfile erstellt.

```
private void triggerAlert(I_Rule rule) {
    if (rule.getRulePhoneTelState() == true) {
        outputClass.phoneCall(rule.getRulePhoneTel(), rule.getRuleMessage());
    }
    if (rule.getRulePhoneSMSState() == true) {
        outputClass.sendSms(rule.getRulePhoneSMS(), rule.getRuleMessage());
    }
    if (rule.getRuleMailState() == true) {
        outputClass.sendEmail(rule.getRuleMail(), rule.getRuleType().toString(), rule.getRuleMessage());
    }
    logger.rulebroken(rule, rule.getRuleType());
}
```

Abbildung 3.15: Ausschnitt Regelbruch Ausgaben in Output

Bei der Output-Klasse handelt es sich um eine Singleton-Klasse, welche die Schnittstelle zwischen der Applikation und dem vorgegeben NotificationHandler aufweist. Sie verfügt über drei Methoden die jeweils ein SMS, eine E-Mail oder einen Telefonanruf auslösen.

```
@Override
public void sendSms(String recipients, String message) {
    notificationHandler.sendSms(recipients, message);
}

@Override
public void sendEmail(String recipients, String subject, String message) {
    notificationHandler.sendEmail(recipients, subject, message);
}

@Override
public void phoneCall(String recipients, String message) {
    notificationHandler.phoneCall(recipients, message);
}
```

Abbildung 3.16: Ausschnitt der Output-Klasse

3.5 JSON

In diesem Kapitel wird das JSON File beschrieben und wie es in der Software eingesetzt wird.

JSON File

JSON steht für JavaScript Object Notation. Dieses File speichert die Felder von Objekten in einem Textformat. JSON ist Programmiersprachen unabhängig, für die meisten modernen Sprachen gibt es Libraries welche das Arbeiten mit JSON Files erleichtern.

JSON schreiben

In der Software existiert eine Klasse Config-Singelton. Diese Klasse besitzt Arrays für Chan-

nels, Bänder und Regeln. Diese werden bei einen Schreibbefehl aus der Klasse Ueberwachung-Singelton eingelesen. Diese Klasse Config-Singelton wird als einzige Klasse Öbjektßerialisiert.

Die Klasse WriteJSON-Singelton benutzt die Library GSON (Version 2.8.6). GSON bietet eine einfache Möglichkeit Objekte in ein JSON File zu schreiben und diese auch wieder zu lesen.
Wenn eine Änderung im GUI vorgenommen wird welche die Daten im JSON betreffen, liest die Klasse Config-Singelton die benötigten Daten aus der Klasse Ueberwachung-Singelton ein und speichert diese in den Lokalen Feldern ab. Anschliessen wird die Methode writeConfigToJSON von der Klasse WriteJSON-Singelton aufgerufen und sich selbst als Parameter übergeben.

WriteJSON-Singelton erstellt ein JSON File falls noch keins existiert. Anschliessend wird das übergebene Objekt mit Methoden aus der Library von GSON in ein JSON File geschrieben. Die Funktionen von GSON können auf der GitHub Seite des Projekts nachgelesen werden. `https://github.com/google/gson`

JSON lesen

Beim aufstarten der Software wird das JSON File (falls vorhanden) gelesen.

```
Config_Singleton.getInstance().readjson();
```

In der Klasse WriteJSON-Singelton wird das File gelesen und die Objekte werden in eine Dataholder Klasse (JSONConfig) geladen, diese ist mit getter ausgestattet.

```
public JsonConfig getObject() {

        // GSON
        this.createFile();
        // Create Channels
        JsonConfig gsonconfig = gson.fromJson(this.getFileReader(),
        JsonConfig.class);
        return gsonconfig;
}
```

Anschliessend liest das Objekt der Klasse Config-Singelton die Daten über die getter ein und speichert diese in Lokale variablen.

Ein Problem welches GSON hat ist, dass es für jedes Objekt, das es einliest ein neues Objekt erstellt. Im Normalfall ist dies auch gewünscht, aber es gibt Ausnahmen. Eine Ausnahme zum Beispiel ist das jedes Objekt des Typen Band eine Referenz auf einen Channel hat. GSON erzeugt nun auf dem Band eine Referenz auf ein Objekt des Typen SignalChannel. Dieses Objekt ist aber nicht dasselbe Obejkt wie in ChannelArray eingelesen wurde, also existieren nun 2 Channels welche eigentlich Identisch sind aber an einem andrem Speicherort liegen. Somit würde bei einem Userinput nur der Channel geändert werden welcher im ChannelArray liegt, nicht aber der Channel der auf dem Band existiert. Dies hat zufolge das die Software nicht mehr ordnungsgemäss funktioniert.
Es muss nun bei jedem Band geschaut werden welcher Channel darauf referenziert ist. Danach muss der richtige Channel im ChannelArray gesucht werden und auf das Band gelegt werden. Das gleiche Prozedere muss auch für die Rules und die Bänder gemacht werden.

JSON Beispiel

```
{
  "channelarray": [
    {
      "channelName": "Magnetfeld",
      "channelLocation": "Motor Labor",
      "channelDiscription": "Windisch",
      "samples": 0,
      "overlap": 10,
      "samplesState": false,
      "overlapState": false,
      "frequency": 50
    },
    {
      "channelName": "Erdbeben",
      "channelLocation": "Erdbeben im VS",
      "channelDiscription": "Erschmatt VS",
      "samples": 0,
      "overlap": 20,
      "samplesState": false,
      "overlapState": false,
      "frequency": 30
    }
  ],
  "rulearray": [
    {
      "ruleName": "Motor",
      "phoneTel": "",
      "phoneSMS": "077",
      "mail": "",
      "message": "Motor ",
      "phoneTelState": false,
      "phoneSMSState": true,
      "mailState": false,
      "ruleBandTable": [
        [
          {
            "channel": {
              "channelName": "Magnetfeld",
              "channelLocation": "Motor Labor",
              "channelDiscription": "Windisch",
              "samples": 0,
              "overlap": 10,
              "samplesState": false,
              "overlapState": false,
              "frequency": 50
            },
            "minFreq": 0,
            "maxFreq": 60,
            "bandRuleAttributesList": [
              {
```

```
            "numberExceedances": [
              30,
              50
            ],
            "alertLevel": 250,
            "rulename": "Motor"
          }
        ],
        "minPower": 0.0,
        "maxPower": 0.0,
        "avgPower": 0.0
      }
    ],
    []
  ]
},
{
  "ruleName": "Erdbeben",
  "phoneTel": "077",
  "phoneSMS": "",
  "mail": "erdbeben@gmail.com",
  "message": "Erdbeben detektiert",
  "phoneTelState": true,
  "phoneSMSState": false,
  "mailState": true,
  "ruleBandTable": [
    [],
    [
      {
        "channel": {
          "channelName": "Erdbeben",
          "channelLocation": "Erdbeben im VS",
          "channelDiscription": "Erschmatt VS",
          "samples": 0,
          "overlap": 20,
          "samplesState": false,
          "overlapState": false,
          "frequency": 30
        },
        "minFreq": 0,
        "maxFreq": 50,
        "bandRuleAttributesList": [
          {
            "numberExceedances": [
              50,
              50
            ],
            "alertLevel": 500,
            "rulename": "Erdbeben"
          }
```

```
            ],
            "minPower": 0.0,
            "maxPower": 0.0,
            "avgPower": 0.0
          }
        ]
      ]
    }
  ],
  "bandarray": [
    {
      "channel": {
        "channelName": "Magnetfeld",
        "channelLocation": "Motor Labor",
        "channelDiscription": "Windisch",
        "samples": 0,
        "overlap": 10,
        "samplesState": false,
        "overlapState": false,
        "frequency": 50
      },
      "minFreq": 0,
      "maxFreq": 60,
      "bandRuleAttributesList": [
        {
          "numberExceedances": [
            30,
            50
          ],
          "alertLevel": 250,
          "rulename": "Motor"
        }
      ],
      "minPower": 0.0,
      "maxPower": 0.0,
      "avgPower": 0.0
    },
    {
      "channel": {
        "channelName": "Magnetfeld",
        "channelLocation": "Motor Labor",
        "channelDiscription": "Windisch",
        "samples": 0,
        "overlap": 10,
        "samplesState": false,
        "overlapState": false,
        "frequency": 50
      },
      "minFreq": 100,
      "maxFreq": 200,
```

```
      "bandRuleAttributesList": [],
      "minPower": 0.0,
      "maxPower": 0.0,
      "avgPower": 0.0
    },
    {
      "channel": {
        "channelName": "Erdbeben",
        "channelLocation": "Erdbeben im VS",
        "channelDiscription": "Erschmatt VS",
        "samples": 0,
        "overlap": 20,
        "samplesState": false,
        "overlapState": false,
        "frequency": 30
      },
      "minFreq": 0,
      "maxFreq": 50,
      "bandRuleAttributesList": [
        {
          "numberExceedances": [
            50,
            50
          ],
          "alertLevel": 500,
          "rulename": "Erdbeben"
        }
      ],
      "minPower": 0.0,
      "maxPower": 0.0,
      "avgPower": 0.0
    }
  ]
}
```

4 Ergebnisse Validierung

Um die einwandfreie Funktion der Software zu gewährleisten, wird das Programm über den kompletten Entstehungsprozess laufend geprüft. Hierbei gibt es viele unterschiedliche Aspekte zu beachten, die genau geprüft werden müssen. Besonders muss die Eingabe von kritischen oder wichtigen Werten fehlerfrei und sauber funktionieren, damit im späteren Dauerbetrieb der Software keine schwerwiegenden Probleme auftauchen.

Das Programm ist in zwei verschiede Testbereiche unterteilt. Der erste Testbereich wird durch das GUI gebildet. Es wird anhand eines zuvor besprochen «Testprotokolls» geprüft. So ist sichergestellt, dass alle Useroberflächen die gleiche Qualität in Bezug auf Fehlerfreiheit aufweisen. Der zweite Teil des Testens bezieht sich auf den Teil des Programms, der für den User nicht direkt sichtbar ist. Hier wird die Funktionalität des MVC überprüft.

4.1 Testen des GUI

Im Teil eins wird darauf geachtet, dass die Eingabe für den User einen Sinn macht. Grundsätzlich ist in Eingabefeldern, bei welchen Zahlen erforderlich sind, die Eingabe von Buchstaben und Sonderzeichen nicht erlaubt. Bei der Eingabe von Punkt oder Komma ist dies unterschiedlich zu handhaben. Falls Punkt und oder Komma keinen technischen Nutzen haben, wird der User über eine Warnigmessagebox auf seine fehlerhafte Eingabe aufmerksam gemacht und um eine Korrektur gebeten. In allgemeinen Textfeldern ist keine Beschränkung von Nöten, ausser es wird explizit im fachlichen Pflichtenheft darauf verwiesen. So wird sichergestellt, dass alle Eingabefelder im GUI nur Eingabemöglichkeiten aufweisen, welche die Software auch weiter verarbeiten kann.

Für die Funktionsfähigkeit der Schaltflächen im GUI wird ebenfalls auf das fachliche Pflichtenheft verwiesen. In diesem ist beschrieben, welche spezifische Funktion jede einzelne Useroberfläche aufweisen soll und muss. Da diese Funktionen überall unterschiedlich sind, ist hier eine einheitliche Prüfungsrichtline wie bei den Eingabefeldern nicht möglich. Der jeweilige Softwarebeauftragte ist hier verpflichtet, die Umsetzung nach Pflichtenheft genau zu realisieren.

Im Allgemeinen wird die Software vorzu geprüft. Dies bedeutet, dass der Programmierer nach der Umsetzung einer Schaltfläche oder Eingabefläche alle Funktionen überprüft. Hierbei wird parallel auch die Eingabe von nicht erlaubten Elementen überprüft und entsprechend behandelt.

4.1.1 Vorgehen GUI testen Protokoll

Testprotokoll Eingabe		
FFT Settings window 1		
Test	**Funktion erfüllt**	**Handling**
Gross- Kleinschreibung	NEIN	WarningMessageBox
Leerzeichen	NEIN	WarningMessageBox
Umlaute	NEIN	WarningMessageBox
Sonderzeichen	NEIN	WarningMessageBox
Punkt/Komma	NEIN	WarningMessageBox

Testprotokoll Eingabe		
FFT Settings window 2		
Test	**Funktion erfüllt**	**Handling**
Gross- Kleinschreibung	JA	-
Leerzeichen	JA	-
Umlaute	JA	-
Sonderzeichen	JA	-
Punkt/Komma	JA	-

Testprotokoll Eingabe		
Band		
Test	**Funktion erfüllt**	**Handling**
Gross- Kleinschreibung	NEIN	WarningMessageBox
Leerzeichen	NEIN	WarningMessageBox
Umlaute	NEIN	WarningMessageBox
Sonderzeichen	NEIN	WarningMessageBox
Punkt/Komma	NEIN	WarningMessageBox

Testprotokoll Eingabe		
Rule Configurator (Message)		
Test	**Funktion erfüllt**	**Handling**
Gross- Kleinschreibung	JA except for Phone numbers	Phone number WarningMessageBox
Leerzeichen	JA except for Phone numbers	Phone number WarningMessageBox
Umlaute	JA except for Phone numbers	Phone number WarningMessageBox
Sonderzeichen	JA except for Phone numbers	Phone number WarningMessageBox
Punkt/Komma	JA except for Phone numbers	Phone number WarningMessageBox

4.2 Test der Models

Da Models nicht wie das GUI über die Eingabe testbar sind, ist für die Überprüfung ein «Testprogramm» zuständig.
Im jeweiligen «Testprogramm» (Neue Testklasse) wird eine Main Methode erstellt.

```
public class LoggerTest {
    public static void main(String[] args) {
        String txtTestPath = System.getProperty("user.home");
        boolean testfailure = false; // for checking if test has errors
```

Abbildung 4.1: Neue Klasse LoggerTest für Funktionstest Logger

Zusätzlich werden Instanzen des zu überprüfenden Models erstellt. Im weiteren Verlauf werden Testdaten über die jeweiligen Methoden und Setter in das zu prüfende Model geladen. In diesem werden die Daten weiter verarbeitet.
Abschliessend werden die im Model verarbeiteten Daten über einen Getter zurückgelesen und ausgewertet.

```
// Testing setting negative frequency:
String path = "\\user";
logger.setLogPath(path);; // Set to a valid value
//Read back the path and check if its equal to the set path
if(logger.getLogPath() != path) {
    System.out.printf("Error by setting the path. Expected path, Actual path: %s, %s \n", path, logger.getLogPath());
    testfailure = true;
}
```

Abbildung 4.2: Getter and Setter

Wird ein Fehler registriert, wird dieser über eine Ausgabe (System.out.printf) in der Console dem Softwarebeauftragten mitgeteilt. Eine Ausgabe in der Console findet ebenfalls statt, wenn der Test ordnungsgemäss verlaufen ist und keine Beanstandungen gefunden wurden.

5 Schlussbemerkungen

In der Arbeit wurde das Thema eines Realtime Monitors untersucht. Dieser wurde, wie bereits beschrieben, mithilfe von einer Software-Applikation umgesetzt.
Gemäss Zielsetzung konnte eine funktionierende Applikation erstellt werden mit folgenden Eigenschaften:
In der Applikation können beliebig Channels hinzugefügt werden und mit Usereingaben speziell auf die Bedürfnisse des Kunden angepasst werden. Weiter könne pro Channel verschiedene frequenzabhängige Bänder definiert werden, welche eigens überwacht und ausgewertet werden. Es können individuelle Regeln gemacht werden, welche ausgewählte Bänder überwacht und auswertet. All diese Einstellungen werden im Logger abgespeichert und dem User in Echtzeit zur Verfügung gestellt. Damit kann er überwachen was alles zu welcher Zeit geschehen ist und regelmässig die Daten auswerten. Zusätzlich wird beim Starten der Applikation ein .json File erstellt. In diesem sind die erstellten Channel, Bänder und Regeln mit allen getätigten Einstellungen abgespeichert. Ausserdem kann in diesem File mit der Variabel «GUI»entschieden werden ob die Software mit oder ohne GUI aufstarten soll. Diese Einstellungen sind essenziell für den Betrieb auf einem Embedded System.
Die Regeln sollen direkt Alarm auslösen falls die eingestellte Grenze überschritten wurde. Jedoch konnte der Notification Handler nicht implementiert werden. Im Moment löst die Software Alarm aus, jedoch wird der User nicht informiert. Ausserdem wurde die Software nur mittels statischer Funktionen getestet. Daraus folgt, dass es unklar ist, ob die Signale bei zeitlicher Änderungen noch richtig ausgewertet und angezeigt werden können.

Mit dieser Arbeit ist jedoch der Grundbaustein und eine gute Basis für weiterführende Verbesserungen, beziehungsweise weitere Ausbaumöglichkeiten, gelegt. Nach dem Verbessern, gemäss den Fehlern von oben, und Fertigstellen der bestehenden Software kommt der nächste Schritt, die komplette Testung in der realen Welt. Es werden verschiedene Sensoren angeschlossen und eigens erstellte Signale eingelesen. Die Auswertung des Eingabesignals im GUI, sowie die Messages vom Notification Handler, können somit überprüft werden. Als nächster Schritt wäre eine Transformation der Software auf einem Embedded System, wie zum Beispiel dem Raspberry Pi, möglich. Auch dort muss die Software erneut getestet werden, da kein GUI angezeigt werden kann und somit die Channels und die benötigten Bänder vor dem Verbauen, mittels GSON File, erstellt und validiert werden müssen. Nach erfolgreichem Abschluss der genannten Tests ist die Software bereit für den Endverbraucher und könnte ohne Probleme benutzt werden.

Literatur

[1] S. Gorenflo, *Fachinput Pro2e Teil1*, 2021.
[2] M. Meyer, *Signalverarbeitung: Analoge und digitale Signale, Systeme und Filter*, de, 8. Aufl. Springer Vieweg, 2017, ISBN: 978-3-658-18320-2. DOI: 10.1007/978-3-658-18321-9.