TUHH
Hamburg University of Technology

Bachelorarbeit

Markus Huber

Entwicklung eines universellen Visualisierungs- und Datenverwaltungstools für asynchrone Messwerte

1. Oktober 2014

supervised by:
Prof. Dr. Sibylle Schupp
Felix Klöckner
Dr. Frank Hermann

Hamburg University of Technology (TUHH)
Technische Universität Hamburg-Harburg
Institute for Software Systems
21073 Hamburg

TUHH STS
Software
Technology
Systems

Eidesstattliche Erklärung

Ich, Markus Huber, versichere an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel *Entwicklung eines universellen Visualisierungs- und Datenverwaltungstools für asynchrone Messwerte* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, den 1. Oktober 2014

Markus Huber

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

In der Firma WEINMANN Emergency Medical Technology GmbH + Co. KG (Weinmann) wird während der Entwicklung eines Gerätes die Funktionsfähigkeit des Gerätes durch Versuche geprüft und nachgewiesen. Für diese Versuche werden häufig Messungen aufgenommen und ausgewertet.

Auch in der Serienfertigung muss die Funktionstüchtigkeit der einzelnen Geräte in einer Endprüfung nachgewiesen werden. Dazu werden Messungen gemacht, die den Messungen in der Entwicklung sehr ähnlich sind. Sowohl für die Tests in der Entwicklung, als auch für die Endprüfung wird Software geschrieben, die die Messungen steuert. Durch die Ähnlichkeit der Messung ist auch die Software sehr ähnlich. Da die in der Entwicklung geschriebenen Messprogramme jedoch nicht den Qualitätsanforderungen der Endprüfung genügen, müssen dafür bisher komplett neue Programme geschrieben werden.

Durch ein gemeinsam benutztes Softwaretool soll die während der Entwicklung geschriebene Software mit möglichst wenig Anpassungen für die einzelnen Testfälle der Endprüfung verwendet werden können. Dieses Tool soll dem Benutzer automatisch ein Grundgerüst für das Programmieren von Messprogrammen zur Verfügung stellen. So soll das Tool asynchrone Daten unterstützen.

2. Stand der Technik und Grundlagen

2.1. Übliche Messprogramme

Bei einer Messung werden Sensoren an einen Computer angeschlossen. Die Sensoren sind über verschiedenste Busse, beispielsweise I2C oder USB an den Computer angeschlossen. Außerdem werden die Daten unterschiedlich codiert übertragen. Zum Beispiel als Little oder Big Endian, als IEEE-Float oder BCD. Viele Sensoren müssen vor der Datenerfassung konfiguriert werden. Dann lassen sich Parameter wie Abtastrate und Auflösung einstellen. Oft können die Sensoren mehrere Größen messen. In dem Fall muss noch eingestellt werden, welche Daten übertragen werden sollen. Dies ist zum Beispiel bei Flow Sensoren der Fall, die auch die Temperatur und den Druck messen. Die Daten werden mit Hilfe von Software verarbeitet. In der Firma Weinmann wird die Datenverarbeitung und -Visualisierung mit NI LabVIEW (LabVIEW) programmiert. Typische Verarbeitungsaufgaben sind:

Offset Korrektur Bereinigen von Daten um einen Offset, in dem ein konstanter Wert auf die Daten addiert wird.

Skalierung Skalieren von Daten, in dem sie mit einem konstanten Faktor multipliziert werden.

Linearisierung Anpassen von Daten, die von einem Sensor mit einer nichtlinearen Kennlinie aufgenommen wurden. Hierbei wird die inverse Kennlinie auf die Daten angewendet.

Addition Addieren von mehreren Daten miteinander.

Multiplikation Multiplizieren mehrerer Daten miteinander.

Filter Filtern der Daten. Besonders häufig werden Tiefpassfilter verwendet.

Die Aufgaben haben häufig Parameter, die eingegeben werden müssen. Bei der Skalierung ist das der konstante Faktor, bei dem Filter sind das Filterkoeffizienten. In einigen Programmen können diese Parameter zur Laufzeit durch den Benutzer geändert werden. Das hat den Vorteil, dass das Messprogramm nicht bei jeder Parameteränderung neu gestartet werden muss. Bei jedem Neustart vergeht Zeit, bis das Programm gestartet ist und die Sensoren wieder neu konfiguriert sind. Außerdem können Veränderungen im Ergebnis so direkt beobachtet werden.

Nach der Verarbeitung werden die Daten in verschiedenen Diagrammen visualisiert. Die Anzeigemöglichkeiten sind vielfältig, nachfolgend sind die häufigsten aufgelistet:

Kurvendarstellung Die Daten werden in einem Koordinatensystem angezeigt. In der Anzeige bleibt eine Historie vergangener Daten erhalten, so dass eine Kurve mit den aktuellsten Daten entsteht. Hierbei sind zum Beispiel Trends gut ablesbar.

Text Der aktuelle Wert wird als Text ausgegeben. Manchmal werden mehrere Textfelder verwendet um eine Historie der letzten Daten anzeigen zu können. Diese Darstellung ähnelt der eines Protokolls.

Bargraph Die Länge eines Balkens zeigt das letzte Datum an. Für eine Historie können vergangene Daten transparenter angezeigt werden. Statt eines Balkens kann zum Beispiel auch eine Tachonadel angezeigt werden.

Mit Hilfe der Anzeige der Daten kann der Benutzer direkt in die Messung eingreifen.

Um die Daten längerfristig aufzubewahren werden diese üblicherweise in Dateien gespeichert. Die verwendeten Dateiformate sind sehr unterschiedlich. Häufig wird in Comma-separated values (CSV) Dateien geschrieben, die zum Beispiel in Excel bearbeitet und betrachtet werden können. Auch das TDMS-Dateiformat wird verwendet.

Eine Übersicht über das Einsatzgebiet des Tools ist in Abbildung 2.1 gezeigt.

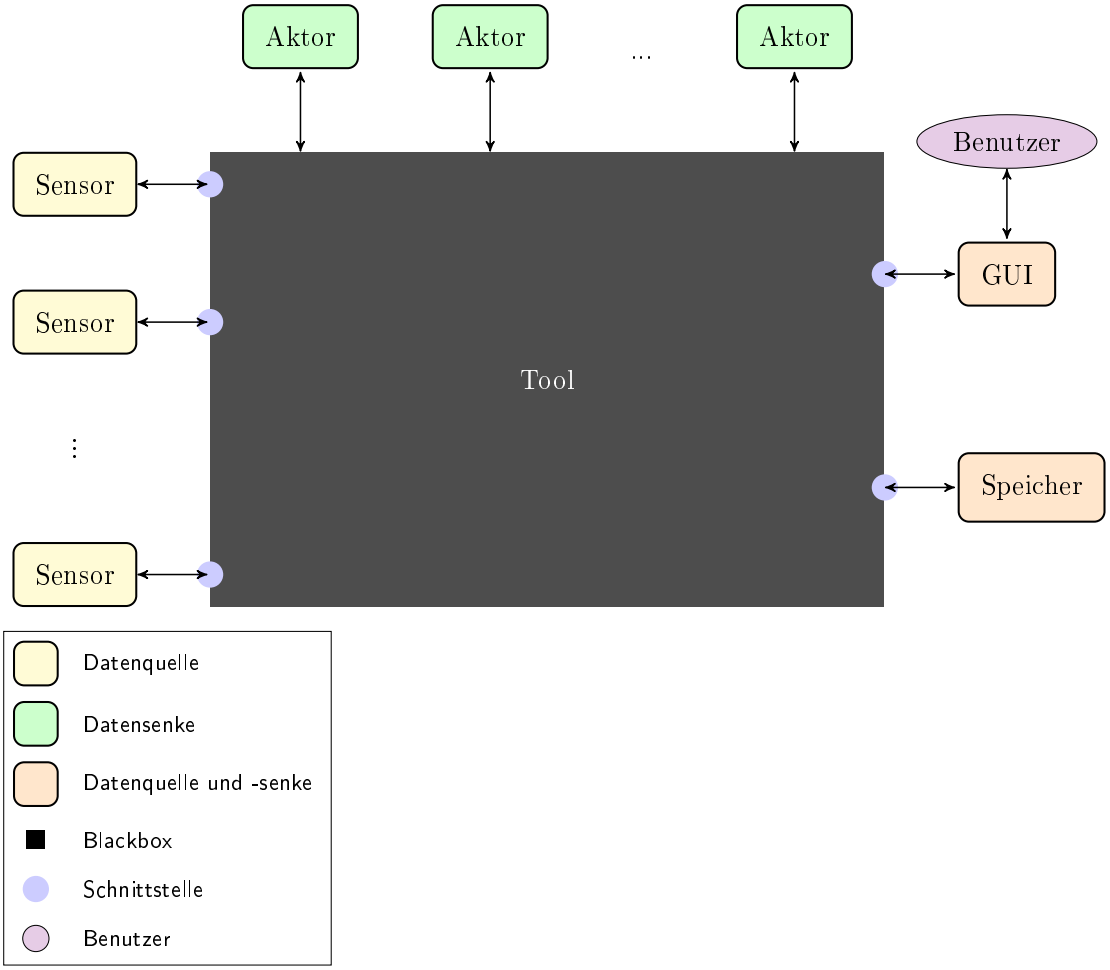

Abbildung 2.1.: Tool Einsatzgebiet

2.2. LabVIEW

Für die Programmierung von Messprogrammen wird bei Weinmann das grafische Programmiersystem LabVIEW wegen seiner umfangreichen Bibliotheken zur Datenerfassung und Anzeige verwendet. Die Programmiersprache heißt G [2]. Durch die Bibliotheken und das grafische Programmieren in Form einer Datenflussbeschreibung können Messprogramme mit geringem zeitlichen Aufwand und wenig Programmierkenntnissen erstellt werden.

Das Prinzip von LabVIEW wird anhand des in Abbildung 2.2 gezeigten Beispielprogrammes erklärt. Das Programm addiert zwei Zahlen miteinander. Jedes LabVIEW Programm, auch Virtuelles Instrument (VI) genannt, besteht aus zwei Teilen, dem Frontpanel und dem Blockdiagramm. Das im Bild rechts gezeigte Frontpanel dient der Ein- und Ausgabe. Hier können Werte in sogenannte Bedienelemente eingegeben werden. Diese Werte kann das Programm als Eingabeparameter verwenden. Insofern entsprechen Bedienelemente Übergabeparametern der meisten anderen Programmiersprachen. Um Ergebnisse anzuzeigen können im Frontpanel auch Anzeigeelemente platziert werden. Diese entsprechen Rückgabewerten.

Das Blockdiagramm ist in der Abbildung links gezeigt. In ihm ist der eigentliche Programmcode hinterlegt. Unter Verwendung der Bedienelemente können hier die Anzeigeelemente angesteuert werden. Das Blockdiagramm kann auch weitere VIs enthalten, wie die meisten Programmiersprachen den Aufbau eines Programmes aus mehreren Funktionen zulassen. Um auf das Frontpanel der sogenannten Sub-VIs zuzugreifen, muss das Sub-VI die Anzeige- und Bedienelemente an seine Anschlussfläche ausführen. Durch verdrahten dieser Anschlussfläche wird der verdrahtete Wert dem Sub-VI entweder als Eingabe übergeben, oder als Rückgabe gesetzt.

In dem Frontpanel des Beispielprogrammes gibt es zwei Bedienelemente, x und y, sowie ein Anzeigeelement x+y. In dem Blockdiagramm sind die beiden Bedienelemente mit den Eingängen der Sub-VI Addieren verbunden. Die Ausgabe der Sub-VI ist mit dem Anzeigeelement verdrahtet. Werden in die Anzeigeelemente beliebige Werte geschrieben und das VI ausgeführt, erscheint im Anzeigeelement die Summe der beiden Werte.

Zur Visualisierung bietet LabVIEW diverse Anzeigeelemente. Das einfachste, die Textanzeige, wurde im eben erläuterten Beispiel zur Ausgabe der Summe benutzt. Die am häufigsten benutzten Diagramme sind in den Abbildungen 2.3, 2.4 und 2.5 gezeigt.

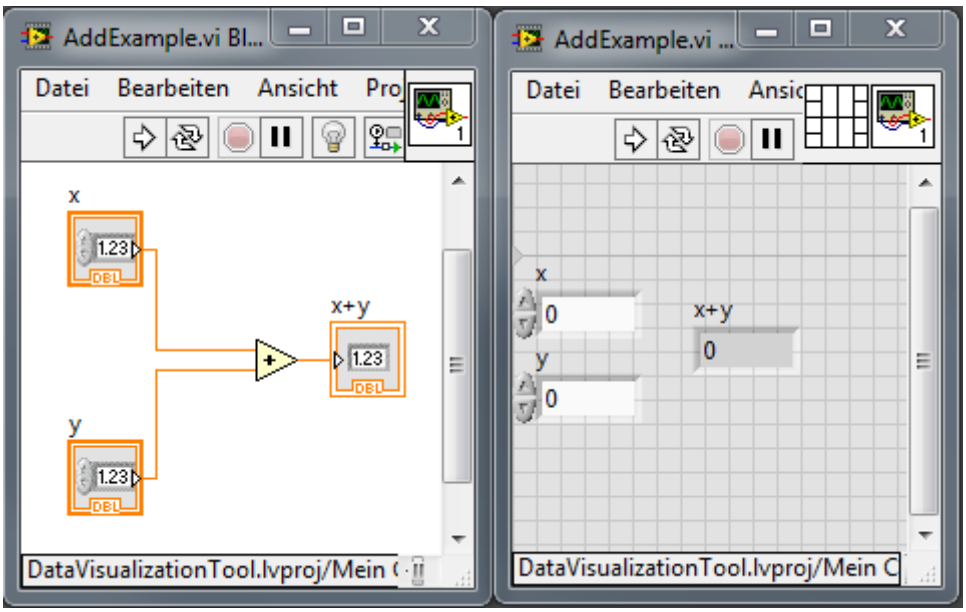

Abbildung 2.2.: LabVIEW Beispielprogramm. Links ist das Blockdiagramm und rechts das Frontpanel gezeigt.

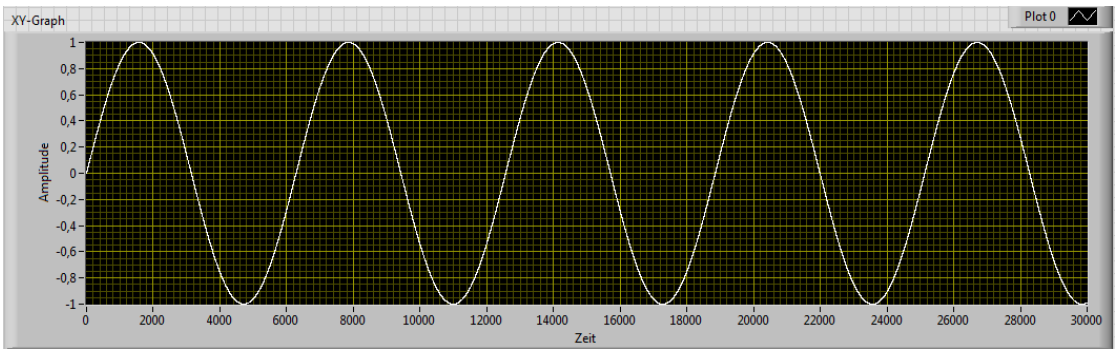

Abbildung 2.3.: Zeigt den Verlauf von Werten in Form von Kurven an.

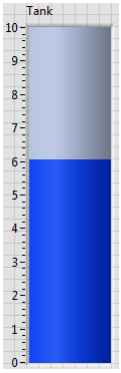

Abbildung 2.4.: Bargraph

Abbildung 2.5.: Tacho

LabVIEW bietet die Möglichkeit VIs in Threads auszuführen. Damit das gleiche VI mehrmals gleichzeitig ausgeführt werden kann, muss es als ablaufinvariant konfiguriert sein. Für die Kommunikation zwischen Threads sind Queues [10] und Melder [9] vorgesehen. Queues sind FIFO-Puffer mit Möglichkeiten zum Legen und Entnehmen. Melder haben einen Status und bieten Funktionen zum Warten auf einen neuen Status und zum Lesen des aktuellen Status.

Seit Version 8.20 unterstützt LabVIEW objektorientierte Programmierung. Um dieses Feature einfacher benutzbar zu machen wird zusätzlich das NI GOOP Development Suite (GOOP) Plugin [3] bereitgestellt. Dies erweitert LabVIEW um Skripte, die das Erstellen von Klassen vereinfachen. Außerdem ermöglicht es das Erstellen eines Unified Modelling Language (UML) Diagrammes und das Generieren von Code aus diesem. Auch das Generieren eines UML Diagrammes aus vorhandenem Code ist möglich.

Für gleichmäßig abgetastete Signale bietet LabVIEW den Datentyp Signalverlauf an. Dieser speichert ein Array von Double-Werten, eine Startzeit als Zeitstempel und das Inkrement als Double-Wert. Ungleichmäßig abgetastete Signale können LabVIEW als Array aus Clustern übergeben werden. Das Cluster besteht aus einem Zeitstempel und einem Double-Wert.

Um in LabVIEW beliebige Datentypen auszutauschen, gibt es in LabVIEW den Variant [8] Datentyp. In ihm wird sowohl der Wert des Ursprungsdatentyps, als auch ein Typdeskriptor [7], binär gespeichert. Der Typdeskriptor beschreibt, wie der Wert interpretiert werden muss.

Eine große Stärke von LabVIEW sind Bibliotheken zur Anzeige von Messdaten in Graphen. Für die Anzeige der Daten bietet LabVIEW den Signalverlaufsgraphen und den XY-Graphen an. Mit dem Signalverlaufsgraph können Signalverläufe, also gleichmäßig abgetastete Signale angezeigt werden. Der XY-Graph dient der Anzeige von beliebigen Kurven.

Um Inhalte im Frontpanel dynamisch zu verändern, gibt es in LabVIEW Subpanels. In sie können Frontpanels anderer VIs geladen werden. So ist es möglich SubVIs in das HauptVI einzubetten.

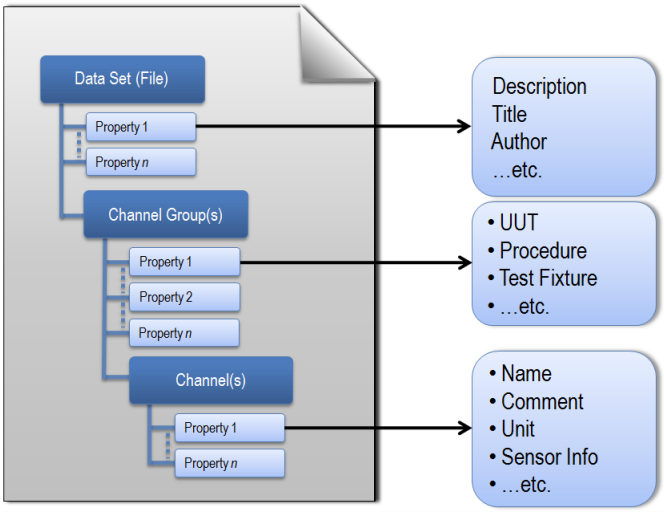

Abbildung 2.6.: Struktur von TDMS Dateien [6].

2.3. TDMS Format

Um Daten in eine Datei zu speichern, bietet LabVIEW Funktionen zur Nutzung des TDMS [4] Dateiformates an. Das TDMS Format ist hierarchisch aufgebaut. Eine Datei enthält beliebig viele Gruppen. Die Gruppen enthalten ihrerseits beliebig viele Kanäle. In die Kanäle können mehrere Daten eines Datentyps geschrieben werden. Unterstützt werden zum Beispiel Double, Zeitstempel, Integer und Strings. Für die Datei, die einzelnen Gruppen und Kanäle können Eigenschaften gesetzt werden. Eine Eigenschaft besteht besteht aus einem Namen, der zur Identifikation benutzt wird, und einem Wert. Der Wert kann wieder mehrere verschiedene Datentypen haben, zum Beispiel die zuvor genannten. Die Werte werden in einem Binärformat gespeichert.

2.4. CSV Format

Der Aufbau von CSV Dateien ähnelt einer Tabelle. Die einzelnen Zeilen werden durch Zeilenumbrüche voneinander getrennt. Die Zellen in den Zeilen werden durch einen weiteren Separator getrennt. Dieser ist häufig das Semikolon. Die Werte der Zellen sind im ASCII Format gespeichert. Für das nachträgliches Hinzufügen von Spalten muss in der bestehenden Datei in jede Zeile der Wert mit dem Separator hinzugefügt werden.

```
      ; Spalte1  ; Spalte2
Zeile1 ; Zelle11 ; Zelle12
Zeile2 ; Zelle21 ; Zelle22
```

Abbildung 2.7.: Struktur von CSV Dateien.

3. Präzisierung der Aufgabenstellung

Im Abschnitt Übliche Messprogramme des Kapitels Stand der Technik und Grundlagen wurde der Typische Aufbau der Messprogramme beschrieben. Es soll ein Tool entwickelt werden, das die Programmierung dieser Messprogramme vereinfacht. Eine generische Anbindung für alle Sensoren zu programmieren, ist aufwendig und übersteigt den Rahmen dieser Arbeit. Die Anbindung eines konkreten Sensors gestaltet sich dank der umfassenden Bibliotheken von LabVIEW zur Messdatenerfassung einfach. Damit die Sensoren in das Tool eingebunden werden können, muss also eine Schnittstelle gefunden werden, mit der so angebundene Sensoren ihre Daten an das Tool übergeben können.

Die Sensordaten werden zum Teil nicht regelmäßig empfangen. Dies ist zum Beispiel bei Benutzereingaben der Fall. Das Tool muss auch für solche Daten eine Schnittstelle anbieten.

Daten werden von den Sensoren teilweise mit einer Geschwindigkeit von bis zu 10 kHz erfasst. Das Tool soll die Datenerfassung nicht ausbremsen, daher muss es eine Verarbeitung der Daten mit diesen Geschwindigkeiten unterstützen.

Das Tool soll es dem Benutzer ermöglichen, die Verarbeitung der Daten zu steuern. Da sehr viele Bearbeitungsformen denkbar sind, die nicht alle im Rahmen dieser Arbeit implementiert werden können, muss es möglich sein, den Funktionsumfang des Tools zu erweitern. LabVIEW bietet bereits Bibliotheken zum Verarbeiten von Daten wie zum Beispiel verschiedene Filter. Es soll eine Schnittstelle angeboten werden, mit der die von LabVIEW angebotene Funktionalität einfach in das Tool integriert werden kann.

Daten können auf viele Arten visualisiert werden. Häufig geschieht dies durch Plotten in ein Koordinatensystem. Aber auch die Anzeige des aktuellen Wertes über ein Textfeld oder eine Bargraph kommt vor. LabVIEW bietet für die Visualisierung bereits verschiedene Funktionen als Bibliothek an. Um den Aufwand für diese Arbeit zu begrenzen, soll eine Schnittstelle zum Einbinden beliebiger Visualisierungen in die Grafische Benutzeroberfläche (GUI) gefunden werden. Außerdem soll zum Demonstrieren der Benutzung dieser Schnittstelle die Anzeige in ein Koordinatensystem implementiert werden.

Es ergeben sich folgende Aufgaben:

- Festlegung eines Grobkonzeptes für die allgemeine Messaufgabe.
- Erarbeiten mehrerer Feinkonzepte und Entscheidung für ein Feinkonzept.
- Implementierung eines Feinkonzeptes.
- Validierung wesentlicher Bestandteile der Implementierung.

4. Grobkonzept

Für die im Abschnitt 2.1 beschriebenen Messprogramme soll eine allgemeine Grobstruktur gefunden werden. Dafür werden voneinander unabhängige Aufgaben identifiziert, die in jedem Messprogramm vorkommen. Diese Aufgaben sind folgend aufgelistet.

- Die Sensoren werden konfiguriert und deren Daten erfasst.
- Die Daten werden verarbeitet.
- Die Verarbeitungsergebnisse werden zum Ansteuern von Aktoren verwendet.
- Die verarbeiteten Daten werden in einem Diagramm visualisiert.
- Benutzereingaben werden zum Konfigurieren des Tools genutzt.
- Die Daten werden in Dateien gespeichert.
- Daten werden aus Dateien gelesen.

Jede der Aufgaben entspricht einem Modul in der Grobstruktur. Die Module sind im Folgenden kurz benannt.

Treiber Erfassung von Messdaten, die zum Beispiel ein Sensor liefert. Treiber werden im Zuge der Arbeit nicht realisiert, aber die Schnittstelle definiert und mit einer Demo verifiziert.

Bearbeitung Operieren auf den Daten, zum Beispiel Skalieren der Werte um einen Faktor.

Regler Steuert die Aktoren in Abhängigkeit von den Eingangsdaten an.

Visualisierung Aufbereitung der Daten zur Anzeige in der GUI.

Konfigurierung Verarbeiten von Benutzereingaben zum Konfigurieren der Bearbeitung.

Aufzeichnung Schreiben von Daten in Dateien.

Wiedergabe Lesen von Daten aus Dateien.

In Abbildung 4.1 wird der Zusammenhang der Module im Tool gezeigt. Die Daten werden in einem Treiber erfasst und anschließend bearbeitet. Diese Bearbeitung kann Operationen wie Addieren, Multiplizieren und Filtern umfassen. Die bearbeiteten Daten können dann in eine Datei aufgezeichnet und parallel dazu dem Benutzer visualisiert werden.

Häufig werden genau die Daten angezeigt, die abgespeichert werden. Prinzipiell können die vom Treiber erfassten Rohdaten aber auch direkt in eine Datei gespeichert werden und weiterverarbeitete Daten werden angezeigt. Zum Beispiel könnten die Rohdaten Strom und Spannung in eine Datei geschrieben werden und das Produkt Strom wird angezeigt. Die Visualisierung muss grundsätzlich also unabhängig von der Aufzeichnung sein.

Im Folgenden werden die Module ausführlich beschrieben.

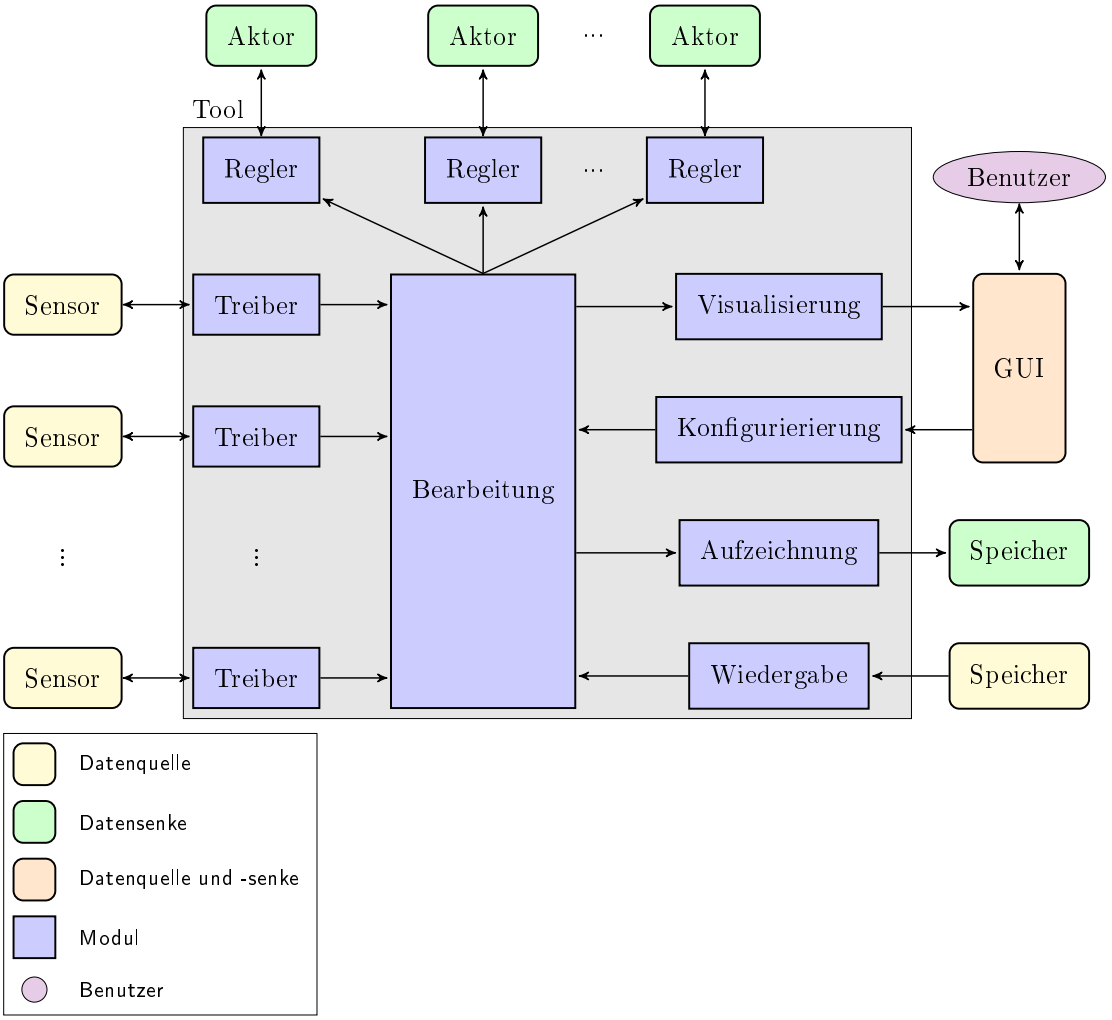

Abbildung 4.1.: Grobstruktur der Messprogramme

4.1. Treiber

Der Treiber ist für das Tool eine Datenquelle. Er stellt die Verbindung zwischen dem Programm und einem physikalisch vorhandenem Sensor her. Sie übernehmen das Konfigurieren und die Datenerfassung von den Sensoren.

4.1.1. Sensoren

Für Sensoren können mehrere Zustände unterschieden werden. Zuerst befindet sich der Sensor im uninitialisierten Zustand. Bei der Initialisierung wird der Sensor konfiguriert. Der nun initialisierte Sensor kann nun Daten an das Tool senden. Manche Sensoren senden Daten automatisch mit einer festen Periodendauer, andere müssen dazu aktiv angesprochen werden. Zum Schluss erwarten manche Sensoren eine Mitteilung, dass die

Messung beendet werden soll.

Die Sensoren messen immer numerische Werte, zu einer bestimmten Zeit. Die empfangenen Daten sind also Zeit-Wert Paare. Sensoren erfassen die Daten mit bis zu 10 kHz.

Allgemein kann über die zeitliche Verteilung der erfassten Daten keine Aussage getroffen werden. Häufig messen die Sensoren die Daten aber in einem festen Intervall beziehungsweise mit einer festen Frequenz. Das sind auch die Sensoren, die am Anfang konfiguriert werden. Sensoren, die keine feste Messfrequenz haben, sind häufig Benutzereingaben. Eine ungleichmäßige zeitliche Verteilung der Daten kommt aber auch bei Sensoren vor, von denen die Messwerte aktiv angefragt werden. Das liegt daran, dass das Betriebssystem die Anfragen mit unterschiedlicher Verzögerung zum Sensor schickt. Abbildung 4.2 zeigt die beiden zuvor beschriebenen zeitlichen Verteilungsarten. Die Amplitude des Sinussignals wird unregelmäßig vorgegeben, das daraus resultierende Sinussignal wird gleichmäßig abgetastet.

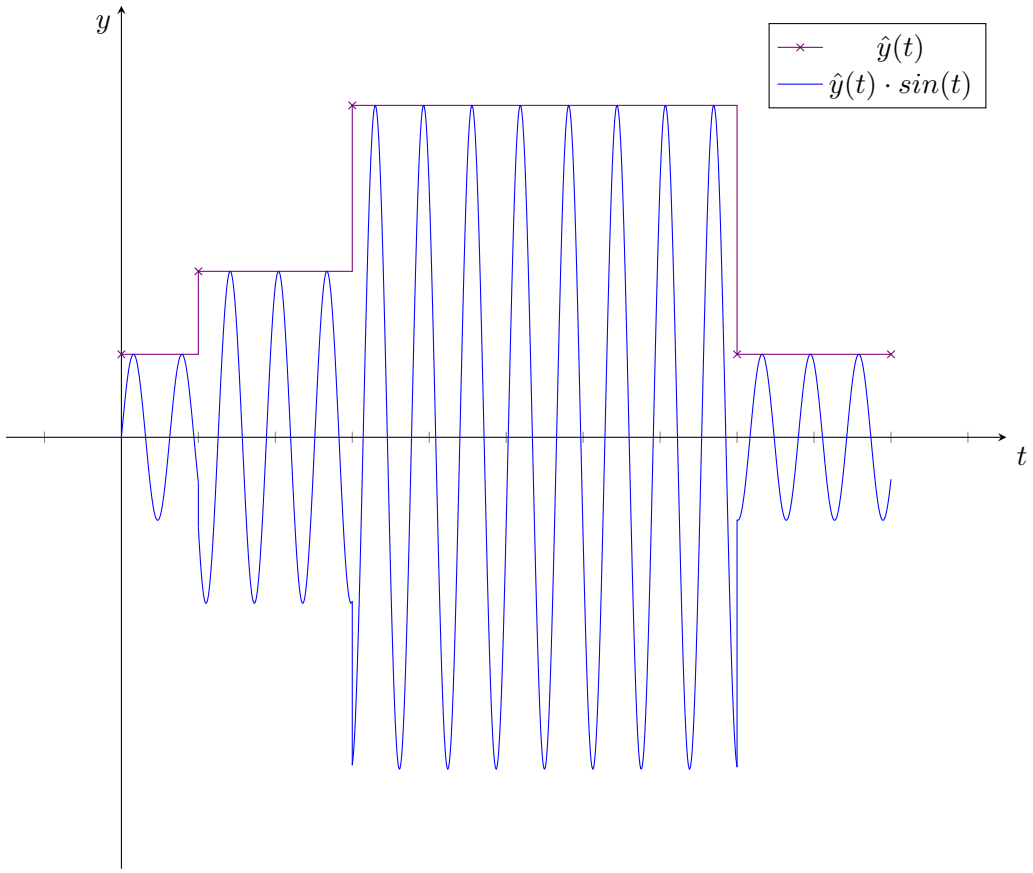

Abbildung 4.2.: Ein Sinussignal wird gleichmäßig abgetastet gemessen. Die Amplitude wird ungleichmäßig geändert

4.1.2. Ablauf

Da die Treiber die Sensoren steuern, können im Treiber die entsprechenden Gegenstücke zu den Zuständen der Sensoren gefunden werden.

Zu Beginn ist der Treiber gestoppt. Bei Start der Messung werden Ressourcen, wie der Port, angefordert und der Sensor konfiguriert. Hier werden beispielsweise die Auflösung, Abtastrate und Messgrößen bestimmt. Nach erfolgreicher Initialisierung des Sensors befindet sich der Treiber im Zustand laufend. Je nach Art des Sensors werden die Daten wiederholt nachgefragt, oder die automatisch geschickten Daten entgegengenommen. Um Änderungen am physikalischem Messaufbau vorzunehmen, können die Treiber häufig pausiert werden. Dann wird die Datenerfassung unterbrochen und derweil entstehende Störungen werden nicht weiter verarbeitet. Beim Stoppen der Messung sendet der Treiber dem Sensor bei Bedarf eine entsprechende Mitteilung und gibt schließlich alle angeforderten Ressourcen, wie zum Beispiel einen seriellen Port, frei.

Die im Zustand laufend empfangene Daten sind im Allgemeinen zu unterschiedlichen Zeitpunkten und mit unterschiedlicher Frequenz aufgenommen. Aufgabe der Treiber ist es, die asynchronen Daten an die Bearbeitung weiterzuleiten. Um eine direkte Weiterverarbeitung und Live-Visualisierung zu ermöglichen, werden die Daten, sobald sie im Treiber erfasst wurden, einzeln an die Bearbeitung übergeben. Die Bearbeitung erhält also bereits Daten, obwohl die Treiber noch weitere Daten erfassen.

4.2. Bearbeitung

Häufig müssen die vom Sensor erfassten Rohdaten zur Informationsgewinnung weiterverarbeitet werden. Zum Beispiel können für eine Leistungsmessung Strom und Spannung erfasst werden. Die eigentlich interessante Größe Leistung wird erst durch Multiplikation von Strom und Spannung berechnet. Außerdem werden Daten oft gefiltert, oder um einen Offset bereinigt. In den meisten Fällen werden mehrere Modifikationen an den Daten vorgenommen, wie zum Beispiel zuerst um einen Offset bereinigt, dann multipliziert und anschließend gefiltert.

Die Eingangsdaten der Bearbeitung sind im allgemeinen Asynchron. Durch ein synchronisieren der Daten gehen Informationen der Ursprungsdaten verloren. Bei manchen Bearbeitungen, wie zum Beispiel dem Multiplizieren von zwei Sensorsignalen, muss das in Kauf genommen werden. Wenn nur Einzeldaten manipuliert werden ist das aber nicht notwendig. Den meisten Bearbeitungen, wie der Skalierung, arbeiten mit einzelnen Daten. In dem Fall können die nach und nach vom Treiber empfangenen Daten auch nach und nach der Visualisierung und der Aufzeichnung übergeben werden.

Die Ausgabe der Bearbeitung kann, je nach Eingangsdaten und Verarbeitung, asynchron sein. Diese asynchronen Daten werden an die Visualisierung und die Aufzeichnung weitergeleitet.

Die Bearbeitung soll die Aufzeichnung und die Visualisierung getrennt voneinander ansprechen können. Das hat den Vorteil, dass eine Messung auch ohne das Speichern in eine Datei angezeigt werden kann. Außerdem kann die Bearbeitung dann Rohdaten

speichern und unabhängig davon die verarbeiteten Daten anzeigen lassen. Der Benutzer hat dann also mehr Möglichkeiten, die Messung an seine Bedürfnisse anzupassen.

4.2.1. Bearbeitungsschritte

Die Bearbeitung wird in mehrere Bearbeitungsschritte unterteilt. Der Bearbeitungsablauf wird vorgegeben, in dem mehrere Bearbeitungsschritte aneinander gereiht werden. Die einzelnen Bearbeitungsschritte können also immer wiederverwendet werden. Das spart Programmieraufwand. Abbildung 4.3 zeigt einen Beispielhaften Bearbeitungsablauf, der aus den beiden Bearbeitungsschritten Offset und Skalierung aufgebaut ist.

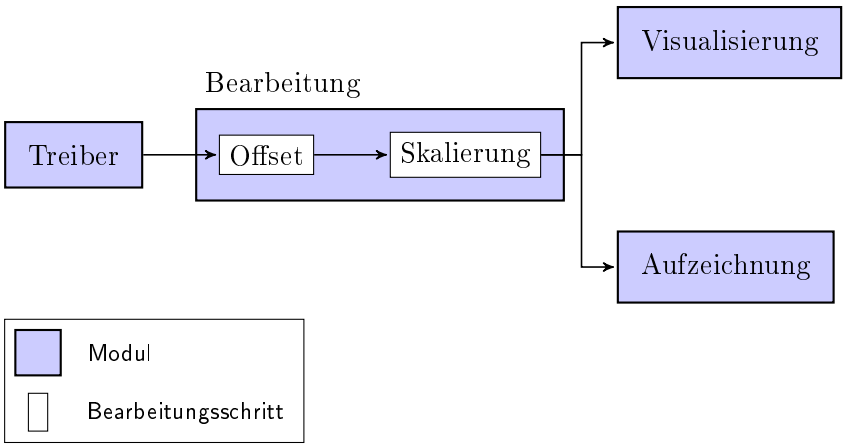

Abbildung 4.3.: Ein Beispiel für einen Bearbeitungsablauf. Die Daten werden über den Treiber empfangen, dann um einen Offset bereinigt und anschließend skaliert. Die bearbeiteten Daten werden abgespeichert und angezeigt.

Es gibt Bearbeitungsschritte, die zwei zueinander synchrone Signale benötigen. Die Multiplikation von zwei Signalen ist ein Beispiel. Da die Eingangssignale im Allgemeinen asynchron zueinander sind, muss der jeweilige Bearbeitungsschritt die Signale dann synchronisieren. In Abbildung 4.4 ist ein solcher Bearbeitungsablauf gezeigt. Die Multiplikation muss hier beide Signale miteinander synchronisieren.

4.3. Regler

Der Regler ist für das Tool eine Datensenke. Er ist die Schnittstelle von dem Tool zu einem Aktor. Der Aktor wird abhängig von dem Eingangssignal angesteuert. Das Eingangssignal des Reglers kann ungleichmäßig sein. Der Regler muss also auch mit ungleichmäßigen Eingangssignalen umgehen können. Um mehrere Aktoren kombinieren zu können müssen diese unabhängig voneinander durch die Bearbeitung ansprechbar sein. Die Signale für die Regler sind im Allgemeinen asynchron zueinander.

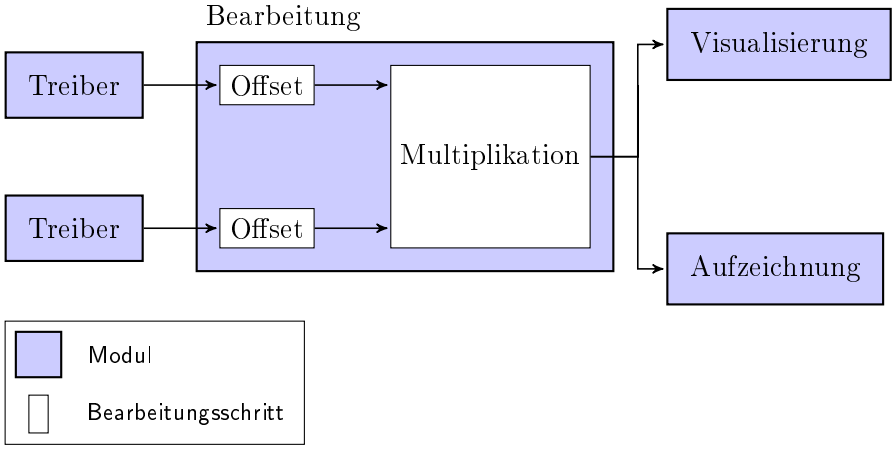

Abbildung 4.4.: Ein Beispiel für einen Bearbeitungsablauf. Zwei Signale werden über zwei Treiber empfangen. Diese werden jeweils um einen Offset bereinigt. Im Anschluss werden beide Daten miteinander multipliziert. Dazu müssen beide Signale synchronisiert werden. Zum Schluss wird das Signal visualisiert und abgespeichert

4.3.1. Aktoren

Ähnlich wie Sensoren müssen auch die Aktoren häufig erst initialisiert werden. Im Anschluss werden ihnen Stellwerte übermittelt. Dies sind immer numerische Werte, die bei einem analogen Aktor zum Beispiel einem Spannungsniveau entsprechen. Abschließend müssen einige Aktoren in eine Ausgangsposition gebracht werden.

4.3.2. Ablauf

Um die Aktoren anzusprechen haben die Regler einen entsprechenden Ablauf. Sie fordern bei Start des Programmes Ressourcen an und initialisieren den Aktor. Im Betrieb wird der Aktor basierend auf den Eingangswerten angesteuert. Wie im Treiber ist auch im Aktor ein pausieren sinnvoll, damit der Benutzer Einstellungen ändern kann, wenn der Aktor nicht das gewünschte Verhalten hat. Bei Beendigung des Programmes werden die Aktoren in eine Ausgangsstellung gebracht und die angeforderten Ressourcen freigegeben.

4.4. Visualisierung

Um schon während der Aufnahme die Messung beobachten zu können, soll die Visualisierung Daten Live, also während der Erfassung, anzeigen. Das ist besonders nützlich um Fehler, wie falsche Einstellungen, frühzeitig zu erkennen und zu beheben. Vor der eigentlichen Aufnahme wird daher häufig die Messung ohne das Abspeichern in eine Datei gestartet um Filterkoeffizienten einzustellen. Deshalb muss die Visualisierung unabhängig von Dateien funktionieren. Typischerweise werden die eingehenden Daten in ein Koordi-

natensystem als Kurve geplottet, es gibt aber diverse Visualisierungsmöglichkeiten. Die Visualisierung muss also eine Schnittstelle bereitstellen um beliebige Diagramme in der GUI anzuzeigen.

Von der Bearbeitung werden im Allgemeinen asynchrone Daten erhalten. Diese werden in ein Diagramm gelegt, das in der GUI dem Benutzer angezeigt wird. Die asynchronen Daten werden in der GUI synchronisiert angezeigt. Die Synchronisation geschieht im Betriebssystem automatisch und muss nicht gesondert implementiert werden.

4.5. Konfigurierung

Um Einstellungen an der Bearbeitung vornehmen zu können, muss der Benutzer eine Möglichkeit zum Anzeigen und Ändern der aktuellen Einstellungen haben. Hier kann der Benutzer zum Beispiel Filterkoeffizienten ändern, oder die Offsetkorrektur anpassen. Änderungen an diesen Einstellungen sollen zur Laufzeit übernommen werden, so dass der Benutzer die Auswirkung seiner Änderungen direkt beobachten kann.

4.6. Aufzeichnung

Um die Ergebnisse einer Messung aufzubewahren ist es notwendig diese speichern zu können. Manchmal werden hierbei direkt die vom Treiber ausgegebenen Rohdaten gespeichert, manchmal aber auch die bereits verarbeiteten Daten. Die bereits bearbeiteten Daten eignen sich zum schnellen Nachvollziehen von Ergebnissen ohne den Bearbeitungsablauf neu aufzubauen. Rohdaten sind nützlich um im Nachhinein zusätzliche Informationen zu gewinnen, oder um einen Fehlerhaften Bearbeitungsablauf korrigiert noch einmal zu durchlaufen. In einigen Fällen werden auch sowohl Rohdaten als auch bearbeitete Daten gespeichert. Daher ist es wichtig, dass die zu speichernden Daten in der Bearbeitung ausgewählt werden können. Diese Daten sollen ohne Informationsverlust gespeichert werden. Das heißt, dass ungleichmäßige Daten nicht vorher gleichmäßig Interpoliert werden sollen. Außerdem müssen die Daten, im Fall von mehreren asynchronen Signalen, ohne vorgehende Synchronisation abgespeichert werden können.

4.7. Wiedergabe

Um die gespeicherten Messungen zu betrachten, sollen die in der Aufzeichnung gespeicherten Daten wieder ausgelesen werden können. Die Daten werden nicht in Echtzeit erfasst, sondern liegen bereits vor und müssen nur nachgeschlagen werden. Wenn der Benutzer zum Beispiel nur einen kleinen Abschnitt der Datei betrachten will, soll dieser in der Datei gezielt gesucht werden können.

5. Feinkonzept

In diesem Kapitel werden Konzepte für die Umsetzung der Bestandteile und Aufgaben der zuvor identifizierten Module vorgestellt. Nach den folgenden Kriterien wird dann das Feinkonzept ausgewählt:

Umsetzbarkeit Die Konzepte müssen so gewählt sein, dass sie in der Bachelorarbeit umsetzbar sind.

Performance Da die Daten mit bis zu 10 kHz erfasst werden, müssen auch die einzelnen Module diese Geschwindigkeit unterstützen.

Speichereffizienz Sowohl die Größe der Dateien auf der Festplatte, als auch die der Daten im Arbeitsspeicher, soll möglichst klein gehalten werden.

Bedienbarkeit Das Tool soll möglichst einfach Bedienbar sein. Das ist für die Akzeptanz durch den Anwender besonders wichtig.

5.1. Treiber

5.1.1. Datumsformat

Ein Datum ist ein Zeit Wert Paar. Für die Zeit wird der LabVIEW Datentyp Zeitstempel verwendet. Der Datentyp des Wertes ist nicht immer einheitlich, aber immer numerisch. Der Wert soll durch einen Datentyp dargestellt werden, der für möglichst viele Messaufgaben verwendet werden kann. Gleichzeitig ist darauf zu achten, dass der Wert möglichst wenig Speicher belegen soll und einfach zu verarbeiten sein muss.

Der Wert kann für eine sehr generische Implementierung des Datums durch einen Variant repräsentiert werden. Dadurch kann jeder Datentyp in ein Signal gelegt werden. Die Bearbeitung müssten dann allerdings für jedes Datum den Datentyp überprüfen, um eine entsprechende Verarbeitung vorzunehmen. Das heißt, dass auch für jeden Datentyp ein entsprechender Algorithmus implementiert werden muss. Prinzipiell ist es mit dem Variant möglich, verschiedene Datentypen in einem Signal zu haben. Bearbeitungsschritte zu schreiben, die auch mit solchen variierenden Datentypen arbeiten können, ist sehr aufwändig. Da Bearbeitungsschritte nicht alle Datentypen unterstützen können, muss der Nutzer aufpassen, welche Bearbeitungsschritte er für bestimmte Datentypen verwenden kann. Außerdem belegt der Variant zusätzlichen Speicherplatz, um zu dem eigentlichen Wert auch den Datentyp des Wertes zu speichern. Gerade bei selbstgeschriebenen Klassen ist dieser Datentyp-Identifikator sehr lang. Zu der Speicherineffizienz kommt, dass auch der Prozessor durch das Interpretieren des Datentyp-Identifikators zusätzlich ausgelastet wird.

Da alle Messwerte numerisch sind, ist die Verwendung des LabVIEW-Double, eine Gleitkommazahl mit doppelter Präzision gemäß IEEE 754 [1], sinnvoller. Dies ist der am häufigsten vorkommende Messwert. Andere numerische Datentypen, wie Integer, können

auch durch einen Double dargestellt werden, allerdings zum Teil mit eingeschränkter Genauigkeit. Durch das Benutzen von nur einem Datentyp muss der Datentyp-Identifikator nicht mit gespeichert werden. Da der Wert ohne Umweg über den Variant direkt als Double Datentyp gespeichert wird entfällt das Überführen in einen anderen Datentyp, also das Interpretieren des Identifikators. Die Bearbeitungsschritte müssen nur mit diesem einen Datentyp arbeiten können. Das vereinfacht das Programmieren von Bearbeitungsschritten.

5.1.2. Signalarten

Gleichmäßig abgetastete Signale können anders als ungleichmäßig abgetastete Signale mit Start und Intervall statt den Zeitpunkten beschrieben werden. Da gleichmäßig abgetastete Signale aber nur spezielle ungleichmäßig abgetastete Signale sind, können sie auch als ungleichmäßig abgetastete Signale dargestellt werden.

Wird nicht zwischen mehreren Signalarten unterschieden, wird nur das allgemeinere ungleichmäßige Signal implementiert. Da es dann nur eine Einzige Datenaustauschmöglichkeit gibt, müssen die Module auch nur diese verstehen. Die Modulschnittstelle und die Module vereinfachen sich dadurch. Zu jedem Wert muss ein Zeitstempel abgespeichert werden, auch wenn die zeitliche Verteilung der Daten durch einen einfacheren Zusammenhang gegeben ist.

Daher wird zwischen gleich- und ungleichmäßigen Signalen unterschieden. Beide Signale bieten ein Interface, das dem der ungleichmäßig abgetasteten Signale, also nehmen und legen von Daten, gleicht. Durch Benutzung dieses Interfaces können Module automatisch mit beiden Signalarten umgehen. Aufwändigere Module können eine optimierte Bearbeitung für gleichmäßig abgetastete Signale implementieren. Außerdem werden die gleichmäßig abgetasteten Signale so effizienter gespeichert.

5.2. Bearbeitung

5.2.1. Parameterinhalt

Die Bearbeitungsschritte müssen Daten mit dem Folgenden Bearbeitungsschritt austauschen. Unter Umständen ist es sinnvoll auch eine Kommunikation in die entgegengesetzte Richtung zuzulassen, zum Beispiel um Daten gezielt von vorgehenden Bearbeitungsschritten anzufragen.

Wenn der Parameter nur aus den Daten bestehen, verringert sich der Programmieraufwand. Allerdings muss durch die fehlende Kommunikationsmöglichkeit auch bei betrachten eines kleinen Ausschnitts einer Datei die gesamte Datei geladen werden.

Enthalten die Parameter zusätzlich zu den Daten auch eine Möglichkeit zur gezielten Anfrage eines bestimmten Zeitbereichs, müssen nur die Bereiche der Datei ausgelesen werden, die auch angezeigt werden. Zusätzlich kann der Parameter den Zustand der vorherigen Bearbeitungsschritte weiterleiten. So können Bearbeitungsschritte sich zum Beispiel automatisch pausieren, wenn die vorgehende Bearbeitungsschritte pausiert sind, und Ressourcen sparen.

5.2.2. Benutzerinterface

Um möglichst einfach neue Bearbeitungsschritte implementieren zu können und den Bearbeitungsschritten eine klare Struktur zu geben, soll dem Benutzer ein Interface geboten werden. In dem Interface soll Funktionalitäten, die in Bearbeitungsschritten häufig vorkommt zusammengefasst werden. Durch erstellen eines solchen Interfaces kann der Benutzer auf ein Grundgerüst gewarteter Komponenten zurückgreifen. Um die Erweiterung noch einfacher zu gestalten, sollen auch weitere Interfaces bereitgestellt werden, die zum Beispiel die Implementierung von Bearbeitungsschritten mit genau einem Eingangssignal und einem Ausgangssignal vereinfachen. Häufig wiederkehrende Funktionalität sind:

- Konfigurieren durch den Benutzer,
- Zugriff auf Ein- und Ausgabesignale und
- Ablaufkontrolle.

Mit der Ablaufkontrolle ist gemeint, dass die Bearbeitungsschritte in Phasen, wie die Allokation von Ressourcen, aufgeteilt werden können, deren Ablauf durch das Interface gesteuert wird.

5.2.3. Ablaufgenerierung

Der Benutzer muss dem Bearbeitungsmodul die auszuführenden Bearbeitungsschritte und ihre Reihenfolge vorgeben können.

Diese Generierung könnte zur Laufzeit erfolgen. Dann kann der Benutzer in der GUI des Tools Bearbeitungsschritte aus einer Bibliothek auswählen und aneinander reihen. Das Tool wäre dann sehr flexibel. Die Bedienung des Tools wird aber auch kompliziert. Außerdem ist das Programmieren einer solchen Laufzeitlösung sehr aufwendig. Es müssten viele Elemente implementiert werden, die LabVIEW durch seine Datenflussorientierung ohnehin schon bietet. Eine solche Lösung ist nur sinnvoll, wenn ein Executable generiert werden sollte, dass unabhängig von LabVIEW bedienbar ist.

Da das Ziel dieser Arbeit das Hinzufügen von Funktionen zur Messdatenerfassung zu LabVIEW und nicht das Programmieren einer eigenen Oberfläche ist, wird der Bearbeitungsablauf zur Kompilationszeit von LabVIEW erzeugt. Der Benutzer gibt den Bearbeitungsablauf vor, in dem er VIs in einem Blockdiagramm miteinander verdrahtet. Das entspricht der üblichen LabVIEW Programmierung. Der Benutzer muss sich also nicht um gewöhnen. Die Funktionalität zur Beschreibung eines Datenflusses wird von LabVIEW übernommen und muss nicht neu implementiert werden.

5.2.4. Modulansteuerung

Der Benutzer muss im Bearbeitungsablauf auf die Eingaben, wie Treiber und Wiedergabe zugreifen können. Außerdem muss der Benutzer die Übergabe von Daten an einen Regler, die Visualisierung oder die Aufzeichnung anordnen können.

Dazu werden spezielle Bearbeitungsschritte implementiert, die diese Module ansprechen. Das ist möglich, da sich die Modulansteuerung und die Bearbeitungsschritte stark

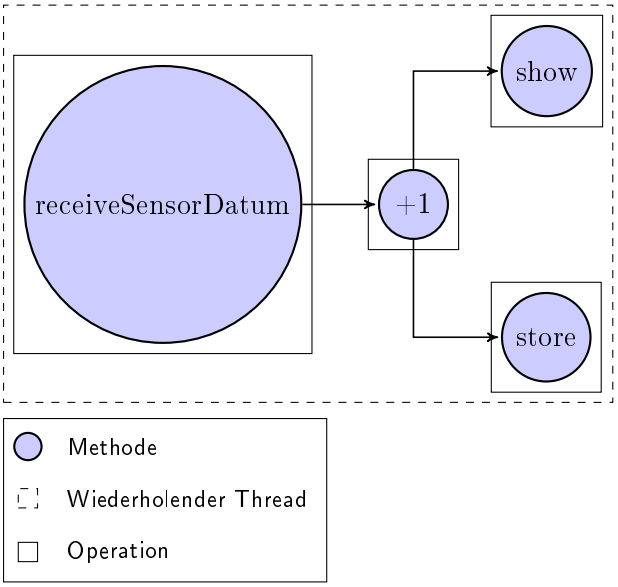

Abbildung 5.1.: Beispielbearbeitungsablauf. Immer wenn ein neues Datum vom Sensor empfangen wird, wird der Wert des Datums um Eins erhöht. Anschließend wird das Datum sowohl angezeigt, als auch abgespeichert.

ähneln. Die Bearbeitungsschritte haben zum Beispiel ähnliche Phasen zu den Zuständen der Treiber und Regler. Für den Benutzer vereinfacht sich der Aufbau eines Bearbeitungsablaufes, da er nicht zwischen Modulansteuerung und Bearbeitungsschritten unterscheiden muss. Da die Modulansteuerung in der Bearbeitung beliebig platziert werden kann, können die Module sehr flexibel in die Bearbeitung eingebunden werden. Für das hinzufügen neuer Module kann die Schnittstelle der Bearbeitungsschritte verwendet werden. Der Programmieraufwand reduziert sich dadurch.

5.2.5. Kontrolle der Bearbeitungsschritte

Der gespeicherte Bearbeitungsablauf wird für jedes erfasste Datum durchlaufen. Abbildung 5.1 veranschaulicht den Ablauf. Der längste Pfad der Bearbeitungskette beschränkt die Ausführungsgeschwindigkeit der gesamten Bearbeitungskette. Die einzelnen Bearbeitungsschritte arbeiten nur mit einzelnen Daten und nicht mit dem ganzen Signal. Das vereinfacht die Implementation dieser Bearbeitungsschritte, beschränkt aber die Funktion. Bearbeitungsschritte, die mehr als ein Signal als Eingang haben, sind so zum Beispiel schwierig zu implementieren. Bearbeitungsschritte mit asynchronen Eingängen sind so nicht implementierbar. Die Signale müssten also bereits synchronisiert sein. Die Daten könnten auch im Fall von gleichmäßig abgetasteten Signalen nicht komprimiert werden, da die Daten einzeln verarbeitet werden. Ebenso kann in dem Fall auf Attribute wie Startzeitpunkt und Inkrement nicht zugegriffen werden.

Gerade um die Performance der gesamten Bearbeitungskette zu erhöhen, läuft jeder

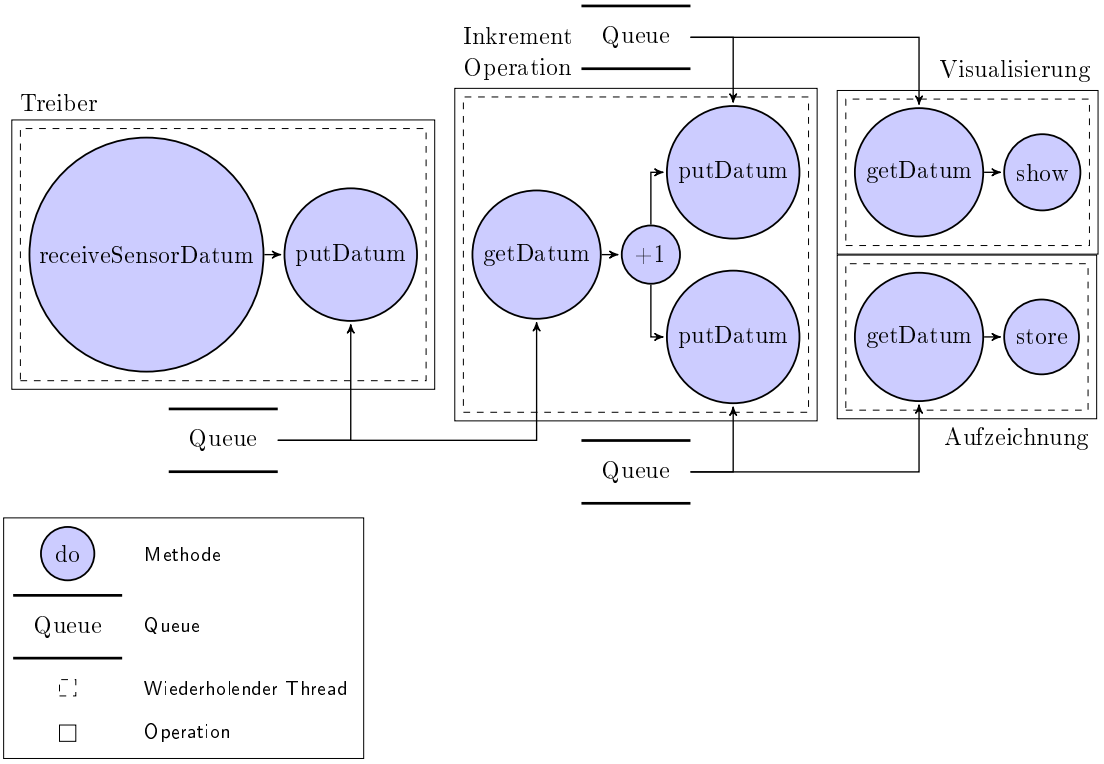

Abbildung 5.2.: Beispiel für in Threads laufende Bearbeitungsschritten. Bei Programmstart werden die Threads gestartet und Queues zur Kommunikation zwischen den Threads konstruiert. Der Treiber liest Daten vom Sensor aus und legt sie in die Ausgangsqueue. Die Inkrement Operation erhöht den Wert jedes empfangenen Datums um Eins und legt das Datum in zwei Ausgangs Queues. Die Visualisierung zeigt die empfangenen Daten an. Die Aufzeichnung speichert die Daten in eine Datei.

Bearbeitungsschritt in einem eigenem Thread. Diese werden bei Programmstart gestartet. In diesen Threads werden auf neue Daten in den Eingangssignalen gewartet. Kommen neue Daten an, werden diese modifiziert und in die Ausgangssignale gelegt. Da die Signale Threads miteinander verbinden, müssen sie durch Queues realisiert werden. Für die Spezialfälle, in denen die Bearbeitungsschritte nur jedes Einzeldatum modifizieren, können speziellere Interfaces zur Implementierung benutzt werden. Mit diesen wirkt die Implementierung eines Bearbeitungsschrittes, wie das einfache modifizieren jedes eingehenden Datums. Ansonsten ist es den Bearbeitungsschritten überlassen, wann Daten aus dem Signal entnommen werden. Das vereinfacht die Implementierung von Bearbeitungsschritten, die mehrere Signale als Eingang haben, oder ein bestimmten zeitlichen Ablauf benötigen, der von dem Empfang von Daten unabhängig ist. Die Synchronisation der Daten ist den Bearbeitungsschritten selbst überlassen. Dadurch sind sie flexibler. In Abbildung 5.2 wird der Ablauf mit Threads veranschaulicht.

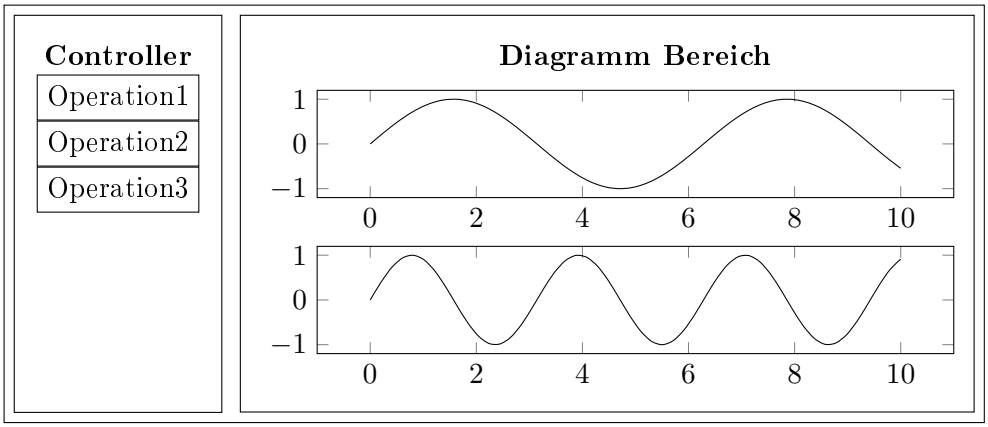

Abbildung 5.3.: Schema der GUI. Links ist der Kontrollbereich mit allen Bearbeitungsschritten zu sehen. Rechts der Diagrammbereich, der alle Ausgaben zusammenfasst.

5.3. Visualisierung

5.3.1. Schnittstelle

Um beliebige Inhalte in einem Frontpanel dynamisch anzuzeigen, bietet LabVIEW die Subpanels an. In sie kann das Frontpanel eines anderen VIs zur Laufzeit eingebettet werden. Diese Funktionalität wird als Schnittstelle benutzt, mit der ein Visualisierungs-Bearbeitungsschritt der Visualisierung ein Diagramm zur Anzeige übergeben kann. Mit dieser Schnittstelle kann der Programmierer des Visualisierungs-Bearbeitungsschrittes das Diagramm wie ein Frontpanel zusammenstellen und LabVIEW Bibliotheken, wie den XY-Graph verwenden. Die Schnittstelle ist vom anzuzeigendem Inhalt völlig unabhängig.

Die Visualisierung verwaltet die Subpanels und arrangiert sie untereinander in einem Bereich der GUI. In Abbildung 5.3 ist rechts dieser Bereich mit zwei Diagrammen schematisch dargestellt.

5.4. Konfigurierung

Da die zu konfigurierende Bearbeitung aus mehreren voneinander unabhängigen Bearbeitungsschritten besteht, muss auch die Konfiguration für jeden Bearbeitungsschritt einzeln einstellbar sein. Dazu wird dem Benutzer eine Liste der Bearbeitungsschritte in der GUI präsentiert. In Abbildung 5.3 ist die Liste links mit den Beispielbearbeitungsschritten Operation1, Operation2 und Operation3 abgebildet.

5.5. Aufzeichnung und Wiedergabe

5.5.1. Dateiformat

Die Messungen sollen in Dateien gespeichert werden. Dazu ist ein Dateiformat notwendig, das auch ungleichmäßig abgetastete Signale unterstützt. Außerdem soll die Datei möglichst schnell schreib- und lesbar sein.

Die Daten werden bislang häufig als CSV Dateien gespeichert. Würde dieses Format verwendet werden, wären also ein Großteil der bereits aufgenommenen Daten mit dem Tool kompatibel. Für die Verwendung von CSV Dateien bietet LabVIEW Funktionen an. Excel, das CSV Dateien öffnen kann, ist auf den meisten Computern installiert und bietet einige Funktionen um die Datei mit geringem Aufwand zu interpretieren, wie Diagramme oder bedingte Formatierung. Da die Werte als ASCII gespeichert werden, wird der Speicher nicht so effizient, wie bei binären Formaten genutzt. In CSV Dateien können Messungen nur mit erheblichem Aufwand zu bestehenden Dateien hinzugefügt werden. Mehrere Signale in eine Datei zu schreiben ist nicht Möglich, wenn die Signale nicht synchron sind. In dem Fall kann die Datei nämlich nicht sequenziell geschrieben werden. Eigenschaften, also Name-Wert Paare, sind nicht vorgesehen.

Das Tool wird die Dateien im TDMS-Dateiformat anlegen. Das TDMS-Dateiformat wurde von National Instruments speziell für das schnelle Speichern von Messdaten entwickelt. LabVIEW stellt für dieses Format Bibliotheken in LabVIEW bereit. Das speichern in TDMS Dateien kann für LabVIEW Messkarten grafisch in deren Konfigurationsmenü aktiviert werden. Es existiert ein Excel Add-In[5] und eine DLL zum einbinden in beliebigen Programmen. Durch die hierarchische Struktur ist das Format ohne Anpassungen für eventuelle spätere Erweiterungen verwendbar.

6. Umsetzung

In den vorgehenden Kapiteln wurden die einzelnen Module identifiziert und Konzepte für diese gefunden. In der Umsetzung werden die Module mit den gefundenen Konzepten implementiert.

Die Klassen und ihre Zusammenhänge, die in diesem Kapitel beschrieben werden, sind in Abbildung 6.1 zusammengefasst. Ein mit GOOP aus dem geschriebenen Code generiertes UML Diagramm befindet sich im Anhang als Abbildung A.1.

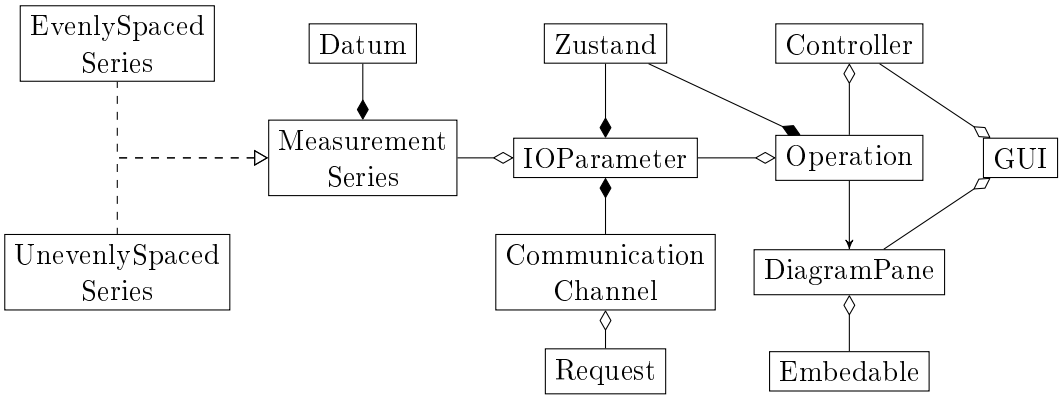

Abbildung 6.1.: Klassenstruktur

6.1. Treiber

6.1.1. Datum

Für ein einheitliches Datenformat wird die Klasse `Datum` programmiert. Sie besteht aus den beiden Membervariablen Zeit, als Zeitstempel, und Wert, als Double. Außerdem gibt es für beide Variablen die Getter und Setter Methode.

6.1.2. Signal

Um mehrere Daten zwischen zwei Bearbeitungsschritten auszutauschen, wird die Klasse `MeasurementSeries` implementiert. Sie ist ein Interface mit den beiden Methoden `Datum receiveDatum()` und `putDatum(Datum datum)`. Zur Vereinfachung der Programmierung der Bearbeitungsschritte, das die empfangenen Daten streng monoton wachsend sind, also $\forall i, j \in \mathbb{N} : i > j \Rightarrow \text{Zeit}(i) > \text{Zeit}(j)$. Werden demnach zwei Daten empfangen, wurde das zuerst empfangene Datum auch früher erfasst.

Für beliebig abgetastete Signale ist die Klasse `UnevenlySpacedSeries` als Realisierung der `MeasurementSeries` gedacht. In der Methode `putDatum` wird das übergebene `Datum` in eine Queue gelegt. Bei Aufruf der Methode `receiveDatum` wird das am längsten in der

Queue liegende `Datum` aus ihr entfernt und zurückgegeben. Um die Monotonie sicherzustellen, wird beim legen von Daten in die `UnevenlySpacedSeries` der Zeitstempel des Datums auf Monotonie geprüft. Wenn das Kriterium nicht erfüllt ist, wird das Datum nicht in die Queue gelegt. In dem Fall wird ein Fehler zurückgegeben.

```
public class UnevenlySpacedSeries extends MeasurementSeries{
   private Queue<Datum> queue=new Queue<Datum>();
   private Timestamp lastTime=null;
   public void putDatum(Datum datum)throws Exception{
      if(lastTime==null || lastTime.isBefore(datum.getTime())){
         queue.put(datum);
         lastTime=datum.getTime();
      } else {
         //Die Zeit ist nicht streng monoton wachsend
         throw new Exception("The time of the datum is invalid");
      }
   }
}
```

Um gleichmäßig abgetastete Messreihen effizienter darstellen zu können, gibt es zusätzlich die Klasse `EvenlySpacedSeries`. Die `EvenlySpacedSeries` hat eine Queue um die Werte der Daten abzuspeichern. Außerdem hat sie einen Startzeitpunkt und ein Inkrement. Beim Legen des ersten Datums wird der Startzeitpunkt auf den Zeitwert des Datums gesetzt. Für alle weiteren Daten wird geprüft, zu welchem Index das `Datum` am nächsten ist: $\min_{n\in\mathbb{N}}(\text{Start} + \text{Inkrement} \cdot n - \text{Datum}_{\text{Zeit}})$. Sollte für diesen Index schon ein Wert geschrieben sein, wird ein Fehler ausgegeben. Ansonsten wird der Wert des Datums dorthin geschrieben. Die Plätze zwischen dem vorher und zuletzt gelegtem Wert werden mit dem vorher gelegten Wert aufgefüllt. Durch diese Vorgehensweise wird Datenverlust, der Eventuell zwischen Sensor und Betriebssystem auftritt, automatisch korrigiert.

6.2. Bearbeitung

Die Bearbeitung gibt der Benutzer durch mehrere Bearbeitungsschritte vor. Gemeinsame Funktionalität der Bearbeitungsschritte wird in der Klasse `Operation` gebündelt. Die Klasse `IOParameter` dient dazu sie zu verbinden.

6.2.1. Parameter

Der `IOParameter` bietet die Möglichkeit Daten auszutauschen, Daten anzufragen und den Zustand der erzeugenden Operation zu erfahren. Er bündelt also eine `MeasurementSeries`, einen Zustand und einen `CommunicationChannel`.

Mit dem `CommunicationChannel` kann eine Operation den am Eingang verfügbaren Zeitbereich sehen. Daten in diesem Bereich können dann über ihn angefragt werden. Der `CommunicationChannel` benötigt daher zwei Variablen, in denen die untere und obere Grenze des Zeitbereichs vermerkt sind. Ist die obere Grenze kleiner als die untere Grenze

handelt es sich um ein Live-Signal, dass gerade von einem Sensor erfasst wird. In dem Fall können keine Anfragen gestellt werden. Um einen bestimmten zeitlichen Bereich anzufragen kann eine `Request` generiert werden. Diese besteht aus zwei Zeitstempel, die die Grenzen des gewünschten Bereiches enthalten.

6.2.2. Operation

Jede `Operation` verwaltet ihre Ein- und Ausgabe Parameter. Außerdem registriert sie sich in ihrem Konstruktor bei der Konfigurierung. Sie bietet auch eine Methode zum Zugriff auf den Diagrammbereich der GUI. Um das Verhalten einer `Operation` durch Zustände zu beschreiben, wird ein Zustandsautomat für sie implementiert.

Die Zustände werden dazu auf den folgenden gemeinsamen Nenner gebracht, die Events ergeben sich daraus:

$$\text{Zustände} = \{\text{Beendet}, \text{Laufend}, \text{Pausiert}\}, \text{Events} = \{\text{Start}, \text{Stopp}, \text{Pausieren}, \text{Fortsetzen}\}$$

Für jedes Event wird eine entsprechende Methode aufgerufen, die überschrieben werden kann um das Verhalten der Operation zu bestimmen. Zum Beispiel wird vor dem Wechsel in den Zustand Pausiert die Methode `onPause()` aufgerufen. Außerdem wird die Methode `run()` im Zustand Laufend aufgerufen. Dieser wird ein boolscher Melder übergeben, der anfänglich den Wert `false` hat. Der Melder hat den Wert `false`, so lange die Operation laufen soll. Wenn die Operation den Zustand laufend verlassen soll, wird der Melder auf `true` gesetzt. Zusätzlich gibt es die Methode `setup()`, die bei Eintritt in den Zustandsautomaten aufgerufen wird.

In Abbildung 6.2 ist der Aufbau dargestellt. Unerwartete Events lassen den Zustandsautomaten in seinem Zustand verharren. Dadurch ist der Zustandsautomat vollständig und der Benutzer kann kein Event triggern, das in einem bestimmten Zustand nicht definiert ist. Außerdem wird dem Benutzer die Möglichkeit eines Resets gegeben. Der Reset entspricht in der Ausgabe dem aufeinanderfolgendem Triggern der Events Stopp und Start.

Die überschreibbaren Methoden, die im Zustandsautomat aufgerufen werden, können Fehler zurückgeben. In diesem Fall wird der Zustandsautomat heruntergefahren, also das herunterfahren Event ausgelöst. Außerdem wird der Fehler dem Benutzer angezeigt.

```
public abstract class Operation{
   private enum state;
   protected void executeState(enum state);
   public void operate(){
      boolean shutdown=false;
      while(!shutdown){
         try{
            executeState(state);
         } catch(Exception e) {
            enum decision=GUI.reportError(e);
            if(decision==shutdown){
               shutdown=true;
```

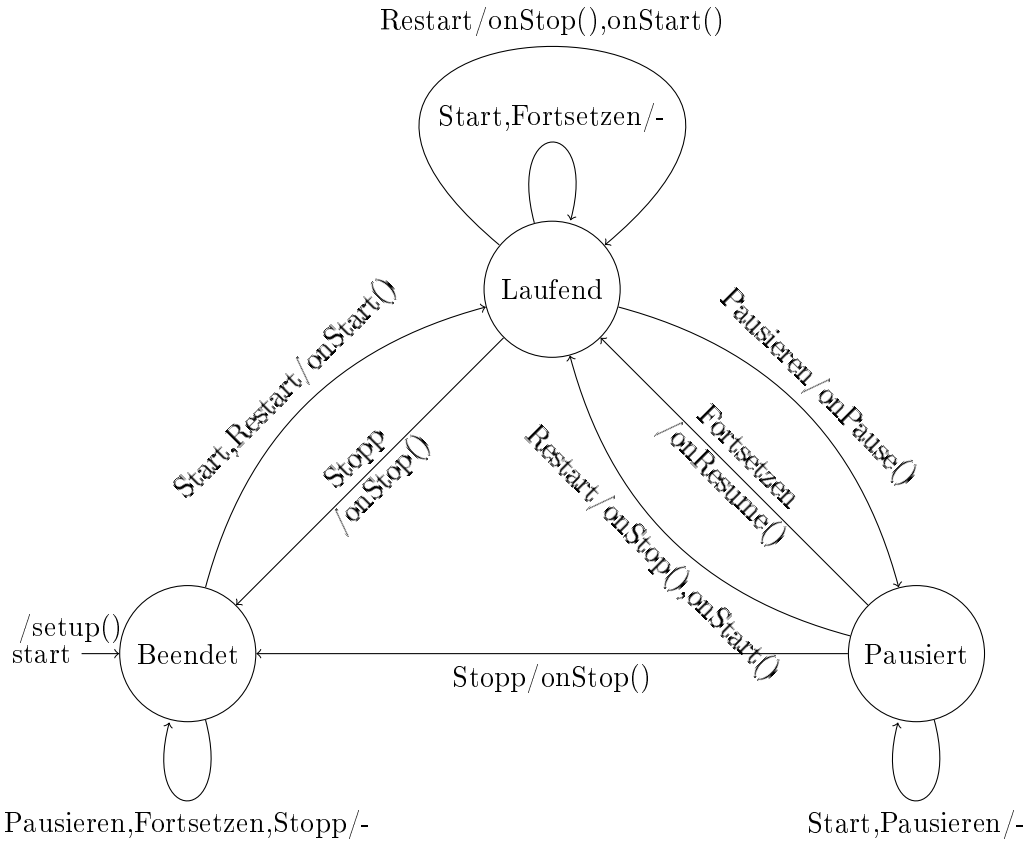

Abbildung 6.2.: Zustandsautomat der Operationen

```
            }
        }
    }
  }
}
```

6.2.3. Benutzerinterface

Damit möglichst einfach neue Bearbeitungsschritte hinzugefügt werden können, werden von der `Operation` speziellere Klassen abgeleitet, die für häufig vorkommende Operationsarten Vereinfachungen haben. Abbildung 6.3 zeigt ein Vererbungsdiagramm dieser Operationen. Im Folgenden wird die Funktionalität erklärt.

Driver Eine Operation, die keinen Eingang hat. Geeignet um einen Sensor in das Tool einzubinden.

SIZOOperation (Single Input Zero Output) Hat genau eine Eingabe und keine Ausgabe. Stellt eine Methode bereit um ohne Arrayindizierung auf die Eingabe zuzugreifen. Eignet sich zum Beispiel für Regler oder Aufzeichnungen.

SOOperation (Single Output) Hat nur einen Ausgang. Gerade Operationen die mit Messreihen rechnen, haben meistens nur einen Ausgang.

VisulatizationOperation Registriert automatisch das Frontpanel eines VIs für die Anzeige als Diagramm. Zur Verwendung als Visualisierungsmodul gedacht.

SISOOperation (Single Input Single Output) Hat genau eine Eingabe und eine Ausgabe. Stellt für den Zugriff auf diese Elemente Methoden bereit.

DatumByDatumOperation Eine Operation, die das Empfangen und Legen von Daten in die Ein und Ausgabe Messreihen implementiert. Für jedes empfangene Datum wird eine überschreibbare Methode aufgerufen, die das Datum modifiziert. Das Resultat dieser Methode wird in die Messreihe gelegt.

ValueByValueOperation Im Gegensatz zur DatumByDatumOperation werden hier nicht mehr ganze Daten, sondern nur die Werte der Daten modifiziert. Die Zeiten werden also unverändert übernommen.

6.2.4. Bearbeitungsschritte

Folgend sind die Operationen mit Beschreibung aufgelistet, die im Rahmen der Bachelorarbeit zum Demonstrieren der Funktionalität des Framworks implementiert wurden:

ScaleOperation Multipliziert den Wert jedes empfangenen Datums um einen konfigurierbaren Wert.

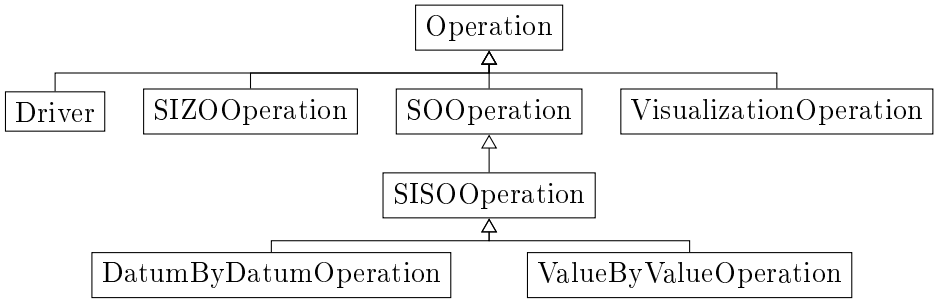

Abbildung 6.3.: Operationsarten

OffsetOperation Addiert auf den Wert jedes empfangenen Datums einen konfigurierbaren Wert.

StoreOperation Speichert die Eingangsmessreihe in eine TDMS-Datei.

LoadOperation Lädt eine Messreihe aus einer TDMS-Datei.

ForkOperation Erzeugt zwei Kopien der Eingangsmessreihe und legt jedes empfangene Datum in die beiden Messreihen.

XYVisualization Visualisiert die empfangenen Daten in einem XY-Diagramm.

SineGenerator Ein Treiber, der keinen Sensor anspricht. Stattdessen wird ein Sinussignal als gleichmäßig abgetastete Messreihe ausgegeben.

UnevenSineGenerator Wie der SineGenerator, gibt jedoch eine ungleichmäßig abgetastete Messreihe aus.

6.3. Visualisierung

Für die Visualisierung wird die Klasse `DiagramPane` erstellt. Immer, wenn ein Diagramm visualisiert werden soll, wird dem `DiagramPane` eine Referenz auf das VI übergeben, das das Diagramm im Frontpanel enthält.

6.3.1. Schnittstelle

Für die Übergabe der Referenz an das `DiagramPane` wird die Klasse `Embedable` implementiert. Ihr kann im Konstruktor eine Referenz auf ein VI übergeben werden, deren Frontpanel bei Aufruf der Methode `embed(Subpanel destination)` in das spezifizierte Subpanel eingebettet wird. Ein `Embedable` kann dem `DiagramPane` mit der Methode `register(Embedable embedable)` übergeben werden.

6.3.2. Diagramm Bereich

Das `DiagramPane` muss bei Aufruf der `register(Embedable embedable)` Methode ein neues Subpanel erstellen und das `Embedable` dort einbetten. Für das dynamische erstellen von Subpanels, oder das führen einer Liste mit beliebigem Inhalt stellt LabVIEW keine Funktionen zur Verfügung.

6.4. Konfigurierung

Die Klasse `Controller` kontrolliert die Zustandsänderung und die Einstellungen einer `Operation`.

6.4.1. Zustandsänderung

Die Konfiguration öffnet bei Rechtsklick ein Kontextmenü mit den Events

- starten,
- stoppen,
- pausieren,
- fortsetzen und
- resetten.

Bei Auswahl eines Punktes wird die Methode `raiseEvent(Enum event)` mit dem ausgewählten Event als Parameter ausgeführt.

6.4.2. Einstellungen

Bei doppelklick auf einen Eintrag wird die Methode `callConfig` der jeweiligen `Operation` aufgerufen. Diese ruft ein überschreibbares VI als Pop-Up auf. In dem Frontpanel des VIs können Einstellungen der `Operation` geändert werden. Der Einstellungsdialog kann einen Fehler zurückgeben, der darauf hinweist, dass die Benutzereingabe falsch ist. Dieser Fehler wird dem Benutzer angezeigt und der Einstellungsdialog öffnet sich von neuem. In folgendem Beispiel prüft die `DivideOperation` den einstellbaren Divisor auf den Wert Null und gibt entsprechend einen Fehler zurück.

```
public class DivideOperation extends Operation{
   private double divisor=1;
   protected void configure(double input)throws Exception{
      if(input==0){
         //0 ist eine ungueltige Eingabe
         throw new Exception("0 is not allowed");
      } else {
         divisor=input;
      }
```

```
   }
}
public abstract class Operation{
   protected void configure(double input)throws Exception;
   public void callConfig(){
      boolean validInput=false;
      while(!validInput){
         try{
            configure(GUI.getUserInput());
            validInput=true;
         } catch(Exception e) {
            GUI.reportError(e);
         }
      }
   }
}
```

In Abbildung 6.4 ist eine Beispielmessung gezeigt. Im linken Bereich ist der Controller zu sehen, im rechten werden die Diagramme angezeigt. Die Hintergrundfarbe der einzelnen Operationen im Controller zeigt den Zustand an. Grün steht für laufend, Rot für gestoppt und Blau für pausiert. Über einen Rechtsklick auf die Operation kann der Zustand geändert werden.

6.5. Aufzeichnung und Wiedergabe

6.5.1. Messdatei

Das Eingangssignal der Aufzeichnung wird in eine TDMS-Datei geschrieben. Der Benutzer legt fest wie die Gruppe und der Kanal heißt, in den die Messung geschrieben wird. Ist das Eingangssignal gleichmäßig abgetastet, werden die Properties Inkrement und Start des Kanals auf die entsprechenden Werte der Messreihe gesetzt. Die Werte der Daten werden in den Kanal geschrieben. Ist das Signal ungleichmäßig abgetastet, werden die Werte in den ausgewählten Kanal geschrieben. Die Zeitpunkte werden in den Kanal geschrieben, der wie der ausgewählte Kanal, nur mit dem Suffix "_Time", heißt. Bei der Wiedergabe wird aus diesen Informationen wieder eine Messreihe erstellt.

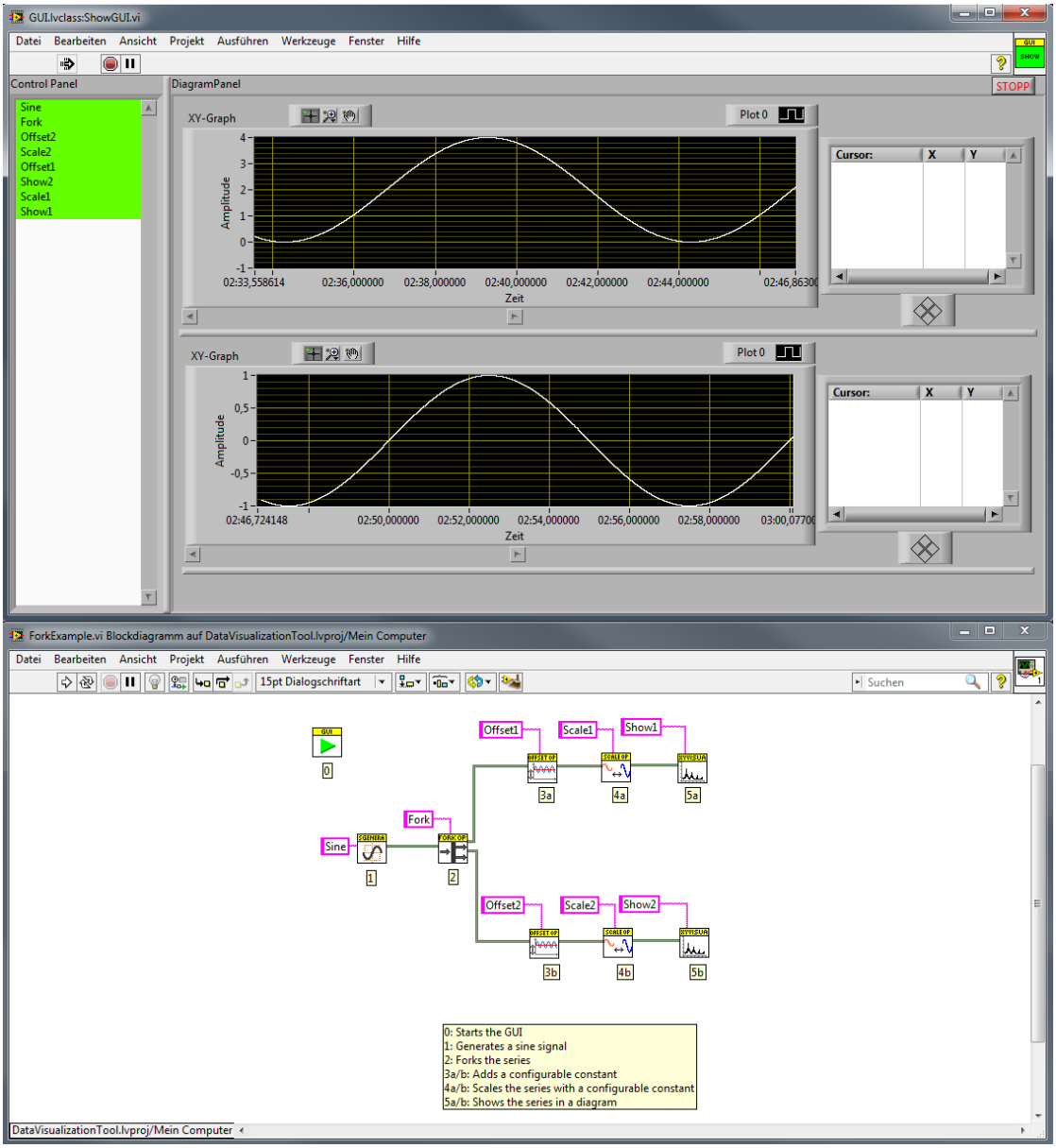

Abbildung 6.4.: Die GUI in einem Beispiel. Ein Sinussignal wird generiert. Dieses wird in zwei Bearbeitungsstränge aufgeteilt. In jedem Strang wird das Signal um einen Offset bereinigt, skaliert und schließlich angezeigt. Das oben angezeigte Signal wurde um Eins nach oben korrigiert und dann mit Vier skaliert. Das untere Signal ist um Null korrigiert und mit Eins skaliert worden.

7. Validierung

Um die Korrektheit der Implementierung zu zeigen und die Performance des Tools zu zeigen, werden die wichtigen Module des Tools getestet. Folgend sind die Tests für die Module beschrieben.

7.1. Zustandsautomat

Beschreibung Der Zustandsautomat muss als zentrale Steuereinheit von Operationen richtig implementiert sein (siehe Abbildung 6.2).

Testdurchführung Zum testen des Zustandsautomaten wird eine Operation abgeleitet, die jeden Zustand loggt. Ein Test-VI löst Events aus und überprüft das Log.

Um alle Zweige aus Abbildung 6.2 zu durchlaufen und so eine möglichst große Abdeckung sicherzustellen, werden die folgenden Events nacheinander ausgeführt: Start, Pausieren, Start, Pausieren, Fortsetzen, Restart, Start, Fortsetzen, Stopp, Pausieren, Fortsetzen, Stopp, Start, Pausieren, Restart, Pausieren, Stopp.

Die erwartete Ausgabe nach Abbildung 6.2 ist: onStart(), onPause(), -, -, onResume(), onStop(), onStart(), -, -, onStop(), -, -, -, onStart(), onPause(), onStop(), onStart(), onPause(), onStop().

Ergebnis Die Reihenfolge der geloggten Zustände stimmt überein. Der Zustandsautomat ist korrekt implementiert.

7.2. Dateieingabe

Beschreibung Die Dateioperationen sollen die Dateien am Anfang auf Richtigkeit prüfen. Wenn die Datei ungültig ist, soll schon dann ein Fehler geworfen werden. Dadurch sollen Fehler während des Lesens vermieden werden. Es muss geprüft werden, ob Fehler der Datei erkannt werden.

Testdurchführung Für das Lesen von Dateien muss folgendes gelten:

- Die Datei muss existieren.
- Der gewählte Kanal muss existieren und den Datentyp Double haben.
- Die Datei hat entweder Start und Inkrement, oder einen weiteren Kanal für die Zeiten.
- Wenn sie einen Zeitkanal hat, müssen Zeitkanal und Wertkanal gleich viele Einträge enthalten.

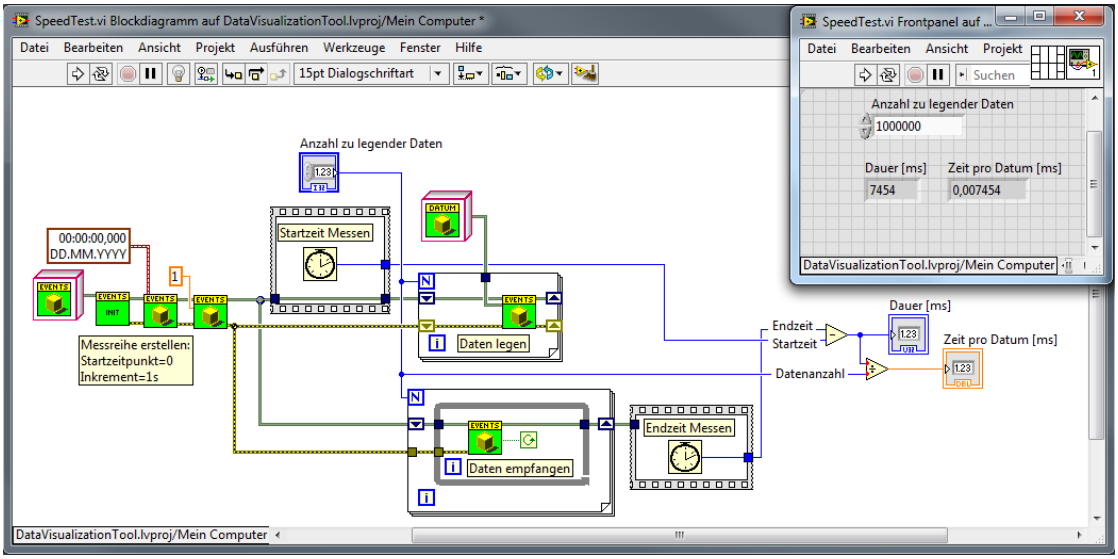

Abbildung 7.1.: Messreihen Geschwindigkeitstest

- Wenn vorhanden muss Start den Typ Zeitstempel und Inkrement den Typ Double haben.
- Sind Start und Inkrement nicht vorhanden, muss der Zeitkanal den Typ Zeitstempel haben.

Für jede Fehlermöglichkeit wird eine Datei mit dem Fehler erstellt. Jede dieser Dateien wird mit der Wiedergabe geöffnet. Wenn mehrere Fehler in einer Datei vorhanden sind, reicht es einen zu entdecken. Daher ist es nicht nötig alle Kombinationen von Fehlern zu betrachten.

Ergebnis Alle aufgelisteten Fehler wurden erkannt. Die Datei wird richtig überprüft.

7.3. Messreihenperformance

Beschreibung Die Messreihe soll Daten von einer Operation an die nächste weiterleiten. Für die Performance des Tools ist es wichtig, dass die Verzögerung, die durch die Datenweiterleitung entsteht, gering ist.

Testdurchführung Eine `UnevenlySpacedSeries` wird konstruiert. In sie werden von einem Thread Daten gelegt. Von einem anderen Thread werden die Daten empfangen. Die Verzögerung zwischen legen des ersten Datums und empfangen des letzten Datums wird gemessen. Das Programm ist in Abbildung 7.1 gezeigt. Für die `EvenlySpacedSeries` wird zusätzlich zu dem eben beschriebenen Test ein Test durchlaufen, in dem nur die Werte der Daten gelegt und empfangen werden.

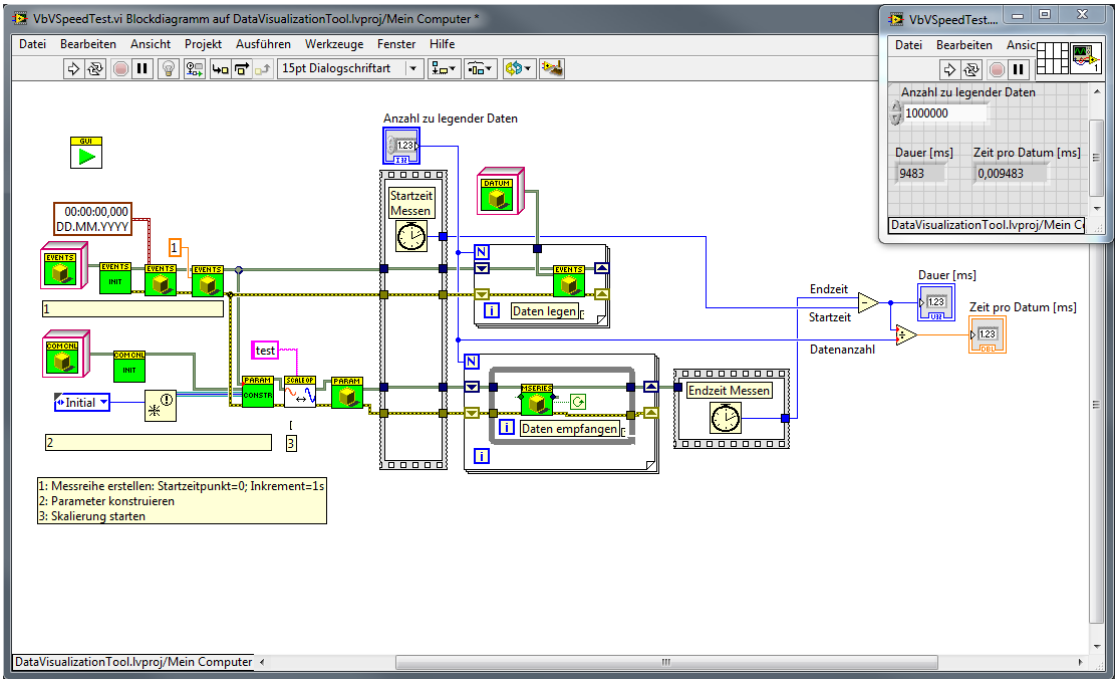

Abbildung 7.2.: Skalierung Geschwindigkeitstest

Ergebnis In dem durchgeführten Test wurden 1.000.000 Daten gelegt und empfangen. Für die `EvenlySpacedSeries` betrug die Verzögerung 7,573 Sekunden. Das ergibt eine durchschnittliche Verzögerung pro Datum von 7,573 Mikrosekunden.

Die `UnevenlySpacedSeries` hatte eine Verzögerung von 4,537 Sekunden. Das ergibt eine durchschnittliche Verzögerung pro Datum von 4,537 Mikrosekunden.

Das die `EvenlySpacedSeries` länger braucht, liegt daran, dass die beim empfangen von Daten die Zeit erst berechnet werden muss. Werden der `EvenlySpacedSeries` statt der Daten nur die Werte übergeben und empfangen, beträgt die Verzögerung nur 3,517 Sekunden, also 3,517 Mikrosekunden pro Datum.

7.4. Operationsperformance

Beschreibung Um die Performance der Bearbeitung zu bestimmen, wird die Verzögerung von Daten durch die Skalierung bestimmt. Da die Daten mit bis zu 10 kHz ankommen, sollte die Verzögerung deutlich unter 100 Mikrosekunden liegen.

Testdurchführung Ein Parameter wird erstellt und der Skalierung als Eingang übergeben. In die Messreihe des Eingangsparameters werden von einem Thread Daten gelegt. Es wird auf Daten in der Messreihe des Ausgangsparameters gewartet. Sind genau so viele Daten empfangen, wie gelegt wurden, wird die Zeit gestoppt und der Test beendet. Das Programm ist in Abbildung 7.2 gezeigt.

Ergebnis In dem durchgeführten Test wurden 1.000.000 Daten gelegt und empfangen. Die Verzögerung betrug 9,483 Sekunden. Das ergibt eine durchschnittliche Verzögerung pro Datum von 9,483 Mikrosekunden. Das ist ein Zehntel der Periodendauer und somit für die Anwendung bis 10 kHz ausreichend.

8. Zusammenfassung und Ausblick

8.1. Zusammenfassung

In der Arbeit ist es gelungen Grobkonzepte zu finden, die die allgemeine Messaufgabe modular zusammenfassen. Das Grobkonzept ermöglicht es, die typischen Messaufgaben in mehreren Modulen unabhängig voneinander zu betrachten. Die Schnittstellen der Module und ihre Vernetzung wurden bestimmt und so ausgelegt, dass auch asynchrone Signale verarbeitet werden können.

Für die einzelnen Module wurden wichtige Bestandteile und Aufgaben identifiziert. Für diese wurden Umsetzungsmöglichkeiten erarbeitet. Aus den einzelnen Umsetzungsmöglichkeiten wurde ein Feinkonzept nach den Kriterien Umsetzbarkeit, Performance, Speichereffizienz und Bedienbarkeit ausgewählt.

Das Tool wurde gemäß den Konzepten in LabVIEW objektorientiert implementiert. Bei der Implementation wurden LabVIEW möglichst viele LabVIEW Bibliotheken benutzt, um eine gute Integration in LabVIEW zu erhalten. Die Schnittstellen wurden wenn Möglich an LabVIEW Schnittstellen angelehnt. Das Tool lässt sich so einfach um LabVIEW Funktionen erweitern.

Die Validierung zeigt, dass grundlegende Bestandteile des Tools für Abtastraten um 10 kHz geeignet sind. Außerdem weist die Implementierung keine Fehler in den wesentlichen Bestandteilen der Implementierung auf.

8.2. Ausblick

Das entwickelte Tool bietet dem Benutzer ein Framework, mit dem die Programmierung neuer Operationen vereinfacht wird. Um dem Framework mehr Funktionalität zu bieten, können auf Basis dieses Frameworks weitere Operationen hinzugefügt werden. Die so neu programmierten Operationen können in einer Bibliothek verwaltet werden, um die künftige Programmierung zu vereinfachen.

Um die Performance bei der Anzeige von Dateien zu erhöhen, wurden Anfragen eingeführt. Die Visualisierung kann so bei der Datei genau den zeitlichen Bereich anfragen, der auch angezeigt werden soll. Da die Anzeige nicht nur im Bereich beschränkt ist, sondern durch eine begrenzte Pixelanzahl auch in der Auflösung, könnte zu der Anfrage eine Auflösung hinzugefügt werden. Wenn dann ein großer Bereich einer zeitlich sehr hoch aufgelösten Datei angezeigt wird, müssen nicht alle Daten des gewählten Bereichs ausgelesen und verarbeitet werden. Dadurch kann die Performance beim betrachten von Dateien weiter gesteigert werden.

Das Tool sollte in der Praxis für Messungen eingesetzt werden. Das von den Benutzern entstehende Feedback, besonders zur Gestaltung der GUI und weiteren sinnvollen Operationen, kann zur Weiterentwicklung des Tools genutzt werden.

A. UML-Struktur

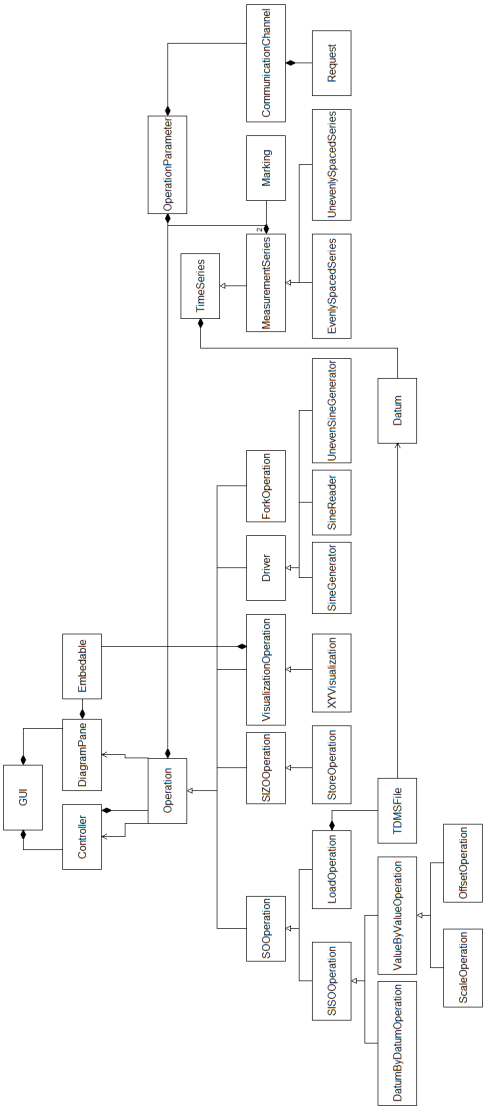

Abbildung A.1.: Von GOOP erzeugte UML-Struktur des Tools

Begriffe

Weinmann WEINMANN Emergency Medical Technology GmbH + Co. KG

LabVIEW NI LabVIEW

VI Virtuelles Instrument

GUI Grafische Benutzeroberfläche

GOOP NI GOOP Development Suite

UML Unified Modelling Language

CSV Comma-separated values

Literatur

[1] IEEE Computer Society, Hrsg. *IEEE Standard for Binary Floating-Point Arithmetic*. 29. 08. 2008.

[2] National Instruments. *BridgeVIEW und LabVIEW. Referenzhandbuch zur Programmierung in G*. 1998. URL: http://www.ni.com/pdf/manuals/321989a.pdf (besucht am 01. 09. 2014).

[3] National Instruments. *NI GOOP Development Suite. Tools to Expand the Usability of the OO Features in LabVIEW*. URL: http://sine.ni.com/nips/cds/view/p/lang/de/nid/209038# (besucht am 01. 09. 2014).

[4] National Instruments. *NI-TDMS-Dateiformat*. 08.07.2013. URL: http://www.ni.com/white-paper/3727/de/ (besucht am 01. 09. 2014).

[5] National Instruments. *TDM Excel Add-In for Microsoft Excel*. 31.7.2014. URL: http://www.ni.com/example/27944/en/.

[6] National Instruments. *TDMS-Dateistruktur*. 08.07.2013. URL: http://www.ni.com/white-paper/app/largeimage?lang=de\&imageurl=\%2Fcms\%2Fimages\%2Fdevzone\%2Ftut\%2FTDMS_with_Properties.png (besucht am 01. 09. 2014).

[7] National Instruments. *Typdeskriptoren*. Juni 2012. URL: http://zone.ni.com/reference/de-XX/help/371361J-0113/lvconcepts/type_descriptors/ (besucht am 01. 09. 2014).

[8] National Instruments. *Variant Data in LabVIEW. Mastering a Higher-Level Way to Work with Data*. 17.10.2012. URL: http://www.ni.com/white-paper/4998/en/ (besucht am 01. 09. 2014).

[9] National Instruments. *What Is a Notifier?* 03.03.1998. URL: http://digital.ni.com/public.nsf/allkb/B9398355D9550EAF862566F20009DE19?OpenDocument (besucht am 01. 09. 2014).

[10] National Instruments. *What Is a Queue?* 03.03.1998. URL: http://digital.ni.com/public.nsf/allkb/DD7DBD9B10E3E537862565BC006CC2E4?OpenDocument (besucht am 01. 09. 2014).