GEFEG-Messprogramm

Dipl.-Ing. (FH)
Alexander Kiebler
E-Mail: alexander_kiebler@gmx.net
Tel.: 01625611283

—Lustige Lebensweisheit—
Wo man nehmen will, muss man geben.

Inhaltsverzeichnis

User Manual I

In diesem User Manual lernen Sie einen Motor mit dem GEFEG-Messsystem zu vermessen. Es klärt, wie die Geräte angeschlossen werden müssen, sowie welcher Konfiguration sie bedürfen. In einem weiteren Schritt wird die Anwendung der Software dokumentiert. Gegen Ende finden Sie ein kleines Errata zur Fehlerbeseitigung.

Das gesamte Projekt hatte eine Laufzeit von drei Monaten, in welcher der Softwarekern, die GUI, die Messstrategie und die Softwarearchitektur entwickelt worden ist. Aus Zeitmangel ist die GUI bezüglich des Kerns leider nicht vollständig.

Messaufbau - Anschlüsse und Geräteeinstellungen 1

1.1 Überblick

Bei einem Messprozess arbeiten verschiedene Komponenten zusammen. Zum einen gibt es den zu vermessenden Drehstrommotor, zum anderen die Messgeräte.

Die Messung aller relevanten Daten wird momentan durch zwei Geräte vollzogen. Das eine ist die Leistungsbremse DSP6001 von MAGTROL. Diese misst die Drehzahl und das Drehmoment des Motors und berechnet aus diesen Werten die mechanische Abgabeleistung. Das Andere ist der Power Analyzer. Dieser misst im Falle eines Drehstrommotors die drei Phasenströme, sowie deren Frequenz, die Spannung und die elektrische Aufnahmeleistung des Motors. Auch die Phasenverschiebung zwischen Strom und Spannung wird gemessen, aus welcher sich der komplexe Widerstand und somit die Induktivität des Motors bemessen läßt. Somit erhält man nicht nur alle für den Motoranwender interessanten Daten, sondern auch einen guten Schätzer für die Optimierung des Motors auf eine bestimmte Belastung (Optimierung des Wirkungsgrads für einen bestimmten Arbeitspunkt)

Die Intelligenz liegt in der Steuerung der Messgeräte. Die Software steuert sowohl die Leistungsbremse, als auch den Power Analyzer. Während einer Messung wird somit generell die mechanische Steuerung der Geräte abgeschaltet.

Vorsicht ist geboten, da die Energieversorgung nicht von der Software gesteuert wird. Im Klartext bedeutet dies, dass im Fehlerfall die Energie in Hitze oder in mechanische Energie umgewandelt werden muss. Wir haben uns für die mechanische Variante entschieden, welche zwar höheres Verletzungsrisiko beinhaltet, aber die Geräte vor Brand-fällen schützt.

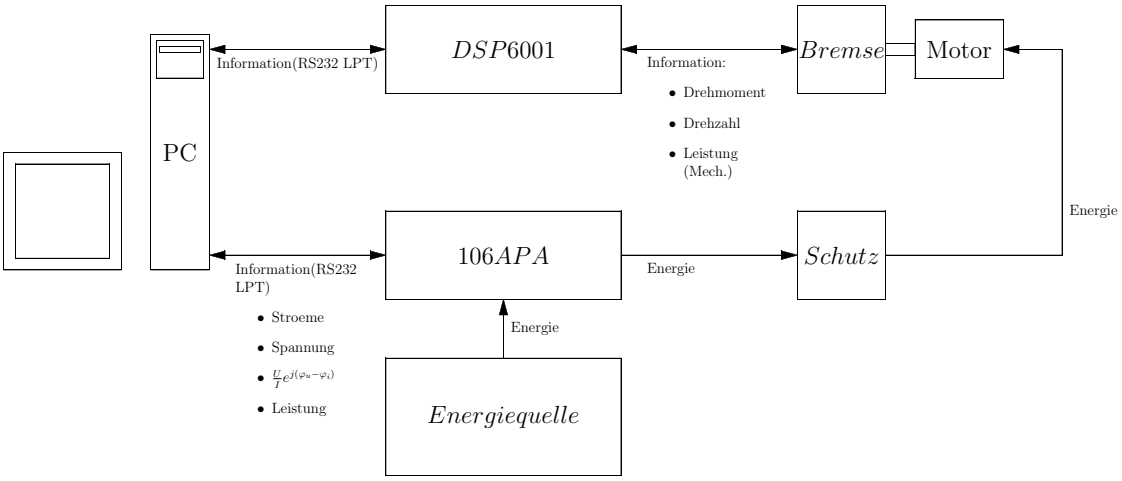
Eine Erweiterung auf ein Temperaturmessgerät ist wünschenswert, jedoch noch nicht vollzogen. Das Temperaturmessgerät TC-08 lieferte sehr dynamische Messdaten, und ist somit nicht Bestandteil des Messsystems geworden.

Von Seiten der Softwarearchitektur ist die Anzahl der Mess und Steuergeräte unerheblich. Somit ist eine Erweiterung zur Steuerung der Energieversorgung und eine Erweiterung auf mehrere Messgeräte kein Problem.

1.2 Genereller Messaufbau

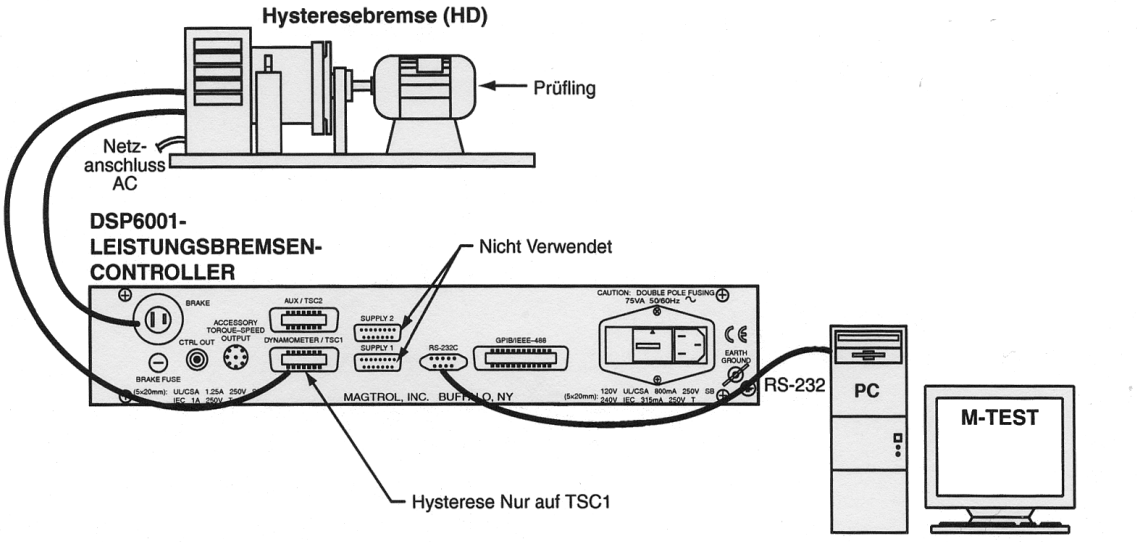
Für den Messaufbau sind folgende Schritte zu beachten:

Die Bremse DSP6001 über die RS232 Schnittstelle mit dem PC verbinden.
Den Power Analyzer 106A über die RS232 Schnittstelle mit dem PC verbinden.
Die Energiequelle mit dem Power Analyzer verbinden (Steckplatte).
Zwischen den Power Analyzer und den Prüfling (Motor) einen Schutzschalter anbringen.



1.3 DSP6001 konfigurieren

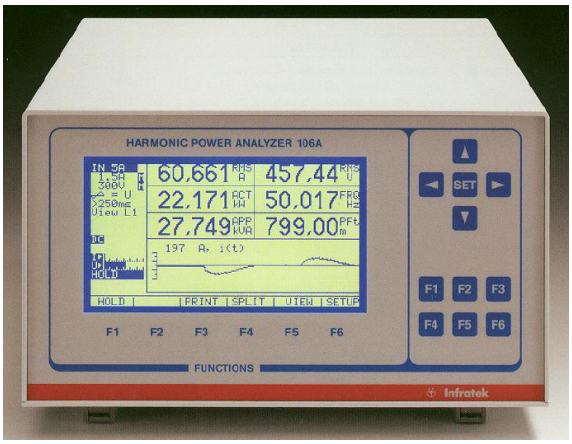
Es wird vorausgesetzt, dass das Gerät einwandfrei arbeitet, und kalibriert ist, da das Kalibrierungsproblem bereits in der Dokumentation des Gerätes gelöst ist.
Für die Software ist es wichtig, dass die Baudrate für den seriellen Anschluss richtig eingestellt ist. Standardeinstellung von der Softwareseite aus ist 19200 Baud, COM1.

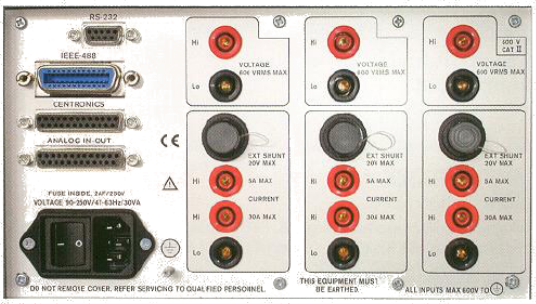


Die weitere Gerätekonfiguration wird automatisch vorgenommen. Das heißt, der gemessene Drehzahlbereich, die Geschwindigkeit der Drehzahlrampe sowie Alarme und Abschaltbedienungen. Auch der PID-Regler wird automatisch gesetzt und in Abhängigkeit der Drehzahl nachgezogen.
Die Regler-werte können im Übrigen mit Hilfe der Softwaremodule sehr einfach bestimmt werden, in dem man einen Drehzahlsprung fährt und sich die Impulsantwort des Systems anzeigen läßt. Aus den abgelesenen Werten können die optimalen Regelparameter einfach bestimmt werden.

1.4 Power Analyzer

Der Power Analyzer 106A von Infratek wird standardmäßig über die RS232 Schnittstelle mit einer Baudrate von 38400, COM2 angesteuert. Die Baudrate der Power Analyzers ist vor jeder Inbetriebnahme neu zu setzen.
Weitere Einstellungen werden automatisch vollzogen. Dazu gehört zum Beispiel das Umschalten des Displays zum Anzeigen der aktuell gemessenen Daten.
Da der Power Analyzer erhaltene Nachrichten nicht bestätigt, ist die Fehleranfälligkeit etwas höher.



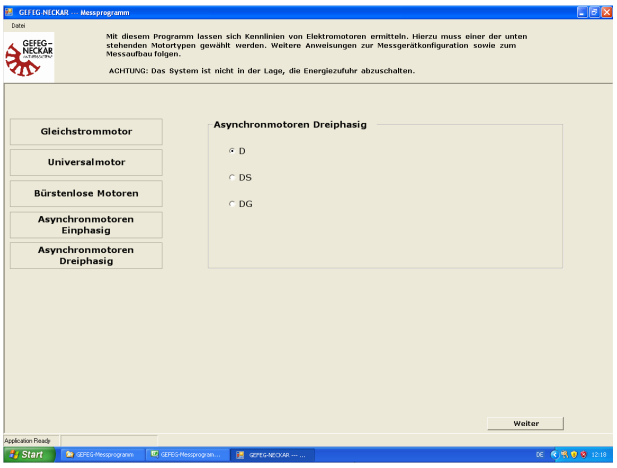


Programmführung 2

2.1 Überblick

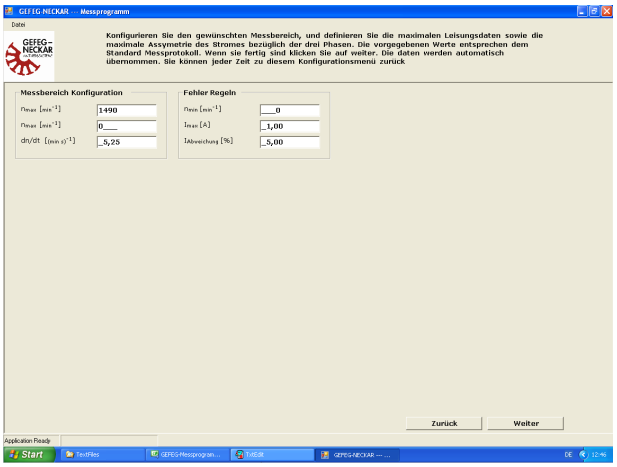
Mit dem Messprogramm werden dynamisch Daten des Drehstrommotors gemessen. Dazu gehören Drehzahl, Drehmoment ,$Z_{ges} = \frac{u}{i} e^{j(\varphi_u - \varphi_i)}$, Phasenströme, Spannung, Frequenz (elektrisch).
Generell bietet der Softwarekern eine Vielzahl von Konfigurationsmöglichkeiten. So zum Beispiel eine Liste von PID-Reglerobjekten, zum Anpassen der Regler-werte für die Bremse.
Regeln zum Abschalten der Bremse können definiert werden. Des Weiteren kann für jedes Gerät die Baudrate frei gewählt werden, sowie das Interface. Die Sprung antworten kann mit Hilfe des Softwarekerns gemessen und angezeigt werden. Und gibt die Möglichkeit, die Daten in verschiedenen Formaten auf die Festplatte zu schreiben.
Leider hatte ich für die GUI lediglich eineinhalb Wochen Zeit, womit sie noch nicht alle interessanten Einstallparameter und Möglichkeiten des Kerns nutzt.

2.2 Programmstart



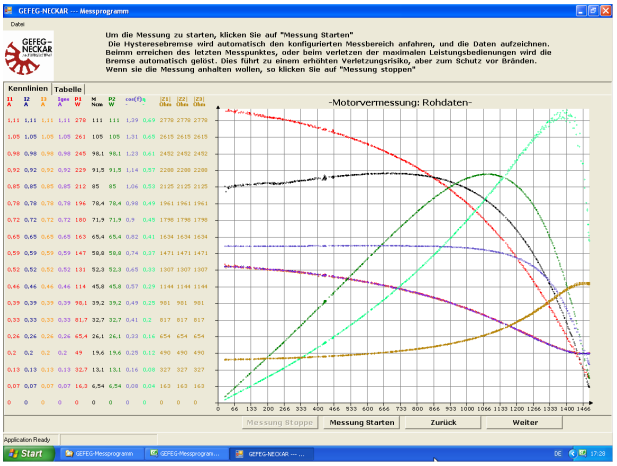
Bei Starten des Programms erscheint im Hauptfenster ein Dialog zum Auswählen des Motortyps. Durch das Berühren mit der Maus kann eine grobe Vorauswahl des Motors getroffen werden. Die firmeninternen Bezeichner selektieren dann genau einen bestimmten Motortypen. Die Auswahl geschieht durch einen Radiobutton. Mit „Weiter “gelangt man zum Konfigurationsmenü des selektierten Motors.
Im oberen Teil der Software befindet sich immer eine Hilfestellung in Textform, sowie eine Menü leiste. Der untere Rand bildet die Status bar.

2.3 Konfiguration der Messung



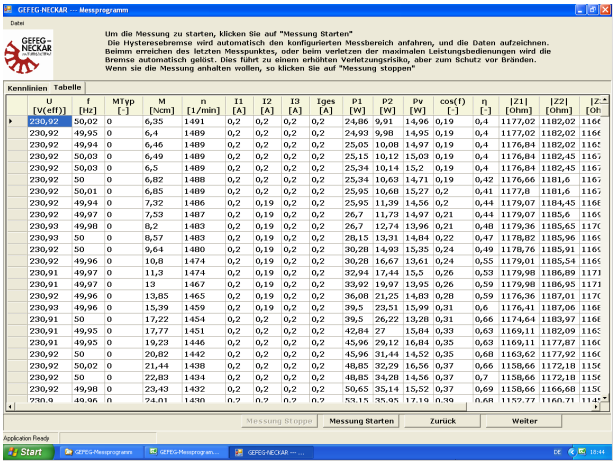
In dem folgenden Menü können der Messbereich und die Abbruchbedingungen konfiguriert werde.
Messbereich Konfiguration:
Gestartet wird mit der maximal angegebenen Drehzahl n_{max}. Dann wird eine Rampe gefahren bis zur minimalen Drehzahl n_{min}. Die Steigung der Rampe ist in Drehzahl pro Sekunde definiert $\frac{dn}{dt}$. Eine Messung braucht somit $\frac{(n_{max}-n_{min})dt}{60dn}$ Minuten.
Abbruchsbedingungen:
Aus Sicherheitsgründen kann eine minimale Drehzahl n_{min} in der Untergruppe „Abbruchsbedingungen “angegeben werden. Diese hat ihren Nutzen für den Fall, dass sich zum Beispiel ein Kleidungsstück im Motor verfängt, und diesen Abrupt stoppt. Intern kann dies die Software registrieren, und bei Erweiterung auf eine Energiequellensteuerung adäquat reagieren. Mit der maximalen Stromforderung kann der maximale Strom definiert werden. Bei dessen Überschreitung erzeugt die Software einen Fehler. Die Stromassymetrie definiert, wie weit der Strom einer Phase des Dreiphasenmotors vom Mittel derselben abweichen darf, bevor die Software einen Fehler generiert.
An dieser Stelle sind noch einige weitere Konfigurationsmöglichkeiten vorgesehen, welche bereits einwandfrei vom Softwarekern unterstützt werden.
Beim klick auf den „Weiter “Button werden die eingestellten Daten direkt übernommen.

2.4 Messung starten und im Koordinatensystem anzeigen



Hier kann die Messung gestartet werden. Sie können jeder Zeit die Messung sowohl über die Software, als auch über das Hardware gerät stoppen. Die Taste zum entkoppeln der Bremse des DSP6001 bleibt als einzige aktiv.
Der weiße, gerasterte Bereich ist die Leinwand. Diese kann in x und y-Richtung skaliert werden. Auch Translation und Zoom werden unterstützt. Mit „STRG+x | y+Scrollen“wird skaliert, „Scrollen“ist ein Zoom. Mit „MouseDown-Right“und ziehen wird entsprechend eine Translation ausgeführt.
Bei jeder Transformation wird die Beschriftung automatisch neu berechnet, so dass sich die Nummerierungen nicht gegenseitig überschreiben.
Die Anzeige der Messdaten geschieht in „Echtzeit“wobei Messen und Malen in Threads entkoppelt sind.
Wird die Messung beendet, so wird automatisch eine Transformation berechnet, welche die gemessenen Daten vollständig darstellt.
Die Beschriftungsfarben der y-Achse entsprechen den Farben ihrer Kurven.
Über das Menü Datei können Messdaten von der Festplatte geladen oder auf diese geschrieben werden.
Der „Zurück “Button führt zum vorherigen Konfigurationsmenü.

2.5 Messung starten und tabellarisch anzeigen

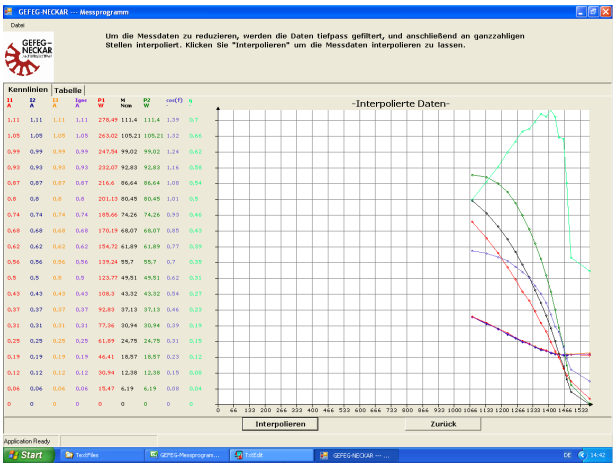


Wechselt man das Tab von „Kennlinien “auf „Tabelle “so gelangt man zu der Tabellenansicht.
Hier können die Messdaten editiert werden. Das Editieren von Messdaten verändert keine physikalisch abhängende Werte. Erhöht man zum Beispiel die Drehzahl durch editieren des entsprechenden Wertes bei konstantem Moment, so bleibt die mechanische Leistung konstant.
Das Erstellen der Tabelle erfolgt immer erst nach der Messung, also nicht in „Echtzeit “.
Zu jedem Record (entspricht einer Tabellenzeile) können Kommentare angelegt und gespeichert werden. Die Anornung der Records, entspricht ihrer Messreihenfolge über der Zeit.
Der Weiter Button führt auf eine momentan leere Seite welche zum Einstellen der Interpolationsregeln gedacht ist.

2.6 Einstellen der Interpolationsregeln

Momentan werden von Seiten der GUI keine Einstellungen unterstützt. Vorgesehen und im Kern implementiert sind zwei Glättungsfilter (Fensterfunktionen: Gauss-Weichzeichner 1D,Hamming), deren Ordnung zur Laufzeit bestimmt werden kann, sowie die Schrittweite der Stützpunkte bezüglich der veränderlichen Variable.
Auch eine Selektierung der unabhängigen Variable ist vorgesehen.

2.7 Messdaten Interpolieren



Um die Messdaten zu interpolieren genügt es, den „Interpolieren “Button zu drücken. Die Messdaten werden dann geordnet, tiefpass gefiltert und schließlich interpoliert.
Die unabhängige Variable ist auf expliziten Kundenwunsch das Drehmoment geworden. Dadurch dass die Drehzahl-Drehmomentkennlinie nicht eindeutig invertierbar ist, kann nicht die gesamte Kurve berechnet werden.
Eine andere Lösung wäre eine Diskretisierung entlang der Kurve, in dem man die Kurve in gleichlange Teilstücke zerlegt (Euklidische Norm als lineare Annäherung), oder eine Diskretisierung entlang der Drehzahlachse
Auch hier können die Daten in der Tabellenansicht unter den gleichen Bedingungen editiert werden.

2.8 Layout verhalten

Die Software passt all ihre Steuer und Anzeigeelemente automatisch an die Bildschirmauflösung, sowie an die aktuelle Fenstergröße an. Dies geschieht durch eine eigene Layoutengine. Das Konzept ist wesentlich benutzerfreundlicher als die Standardlayoutklassen, da es kein Steuerelement in Abhängigkeit der Fenstergröße oder Bildschirmauflösung unplatziert, sondern lediglich in den Höhen und Längenverhältnissen zoomt und die Position des Elements dementsprechend anpasst.

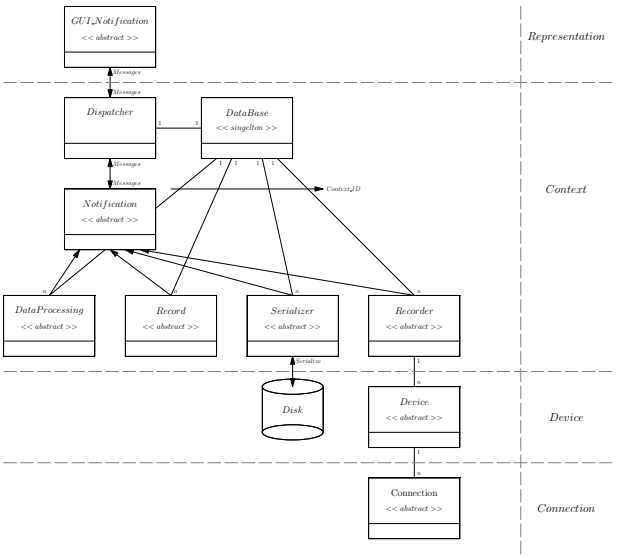
Programmer Guide II

Der Programmerguide beschreibt die grundlegende Struktur, sowie die verwendeten Konzepte des Messprogramms. Er richtet sich in erster Linie an Entwickler, welche gedenken, die Software zu warten oder weiterzuentwickeln. Das Softwarekonzept des Kerns ist mir sehr gut gelungen. Dagegen ist die GUI eine direkte Implementierung ohne Konzept. Sie läßt sich aber sehr leicht in ein tolles Konzept gießen, da die Struktur des Kerns eine Architektur nahe legt.
Aus eigener Sicht hätte ich den Softwarekern gerne in eine dll gepackt, und diese dann getiet. Dies hätte den Vorteil die GUI unabhängig vom Kern updaten zu können. Des Weiteren teilt diese Vorgehensweise die Komplexität entweder parallel auf zwei Programmiererköpfe oder sequentiell, so dass der Programmierer die Konzentration auf einen Teil der Software konzentriern kann.
Des Weiteren könnte die Messung so auch über die Konsole gestartet werden.

Strukturelle Übersicht 3

3.1 Das Vier-Schichtenmodell

Die Software ist in vier Schichten unterteilt. Die unterste ist der Hardware am nächsten, wohingegen die oberste die GUI darstellt.



Representation:
Die Representationschicht hat mehrere Aufgaben. Sie muss die gemessenen Daten anzeigen können, sowie eine Editierung der Daten ermöglichen. Des Weiteren fängt sie wie üblich Fehler ab. Alle Schichten unterhalb der GUI also der Kern sind aus Effizienzgründen ohne Fehlerabfragen implementiert. Dieses Konzept habe ich aus anderen GUI-Kern tupeln übernommen. Wenn man den Kern in eine dll packt, und diesen dann für eine Konsole tiet, könnte man dies über Wrapperfunktionen machen, welche die Fehlerüberprüfungen übernehmen.
Eine weitere wichtige Aufgabe der Representationschicht ist es, den Programmablauf zu steuern. Definitionsgemäß: Die GUI benutzt den Kern fehlerfrei. Dies tut sie über den Dispatcher.
Context:
Die Contextschicht speichert einen Messcontext. Das bedeutet sie hält die Daten, welche für eine Messung erzeugt werden. Somit kann der Kern mehrere Messungen parallel im Speicher halten. Jede Instanz welche zu einem Context gehört, ist durch die sogenannte context_id in O(1) Laufzeit eindeutig dereferenzierbar. Der Context ist global und von jedem Punkt der Software aus erreichbar. Des Weiteren stellt die Contextschicht den Dispatcher zur Verfügung. Durch den Dispatcher kann man jeder Instanz der Context und Repräsentationschicht Nachrichten schicken. Dies ermöglicht eine hervorragende Trennung von Kern und Repräsentationschicht.

Device:
Die Deviceschicht repräsentiert ein Softwaregerät. Auf ein Softwaregerät kann mit Device::write(...) geschrieben wer-

den. Wird aus Hardwaresicht, auf das Softwaredevice gschrieben, so löst das Softwaredevice ein Softwareinterrupt aus oder eben eine Funktionspointer. Für jedes Device ist der Befehlssatz eindeutig. Das Schreiben auf das Softwaredevice erledigt das Schreiben auf das korrespondierende Hardwaredevice. Es gibt genau ein Device, welche statisch ist. Es ist also rein softwarespezifisch. Hinter diesem Device ist nie ein Hardwaregerät zu finden. Die Befehle, welche darauf geschrieben werden, werden als Geräteantwort der Software übergeben. Dies ist eine sehr wichtige Funktion, da sie zur Synchronistaion eingesetzt wird. Das statische Device::write(...) ähnelt der LoopBack Adresse des TCP/IP Protokolls.

Connection:

Die Connectionschicht sorgt für den Treiberzugriff. Jedes Device hat mindestens eine Connection. Die Connection gibt zum Beispiel die Datenrate vor, mit welcher auf ein Gerät geschrieben wird oder die gerätespezifischen Paritätsbits. Sie verallgemeintert das Schreiben auf ein Hardwaregerät für das Softwaredevice.

Softwarekern 4

Der Kern der Software besteht aus den untersten drei Schichten. Context, Davice und Connection. Der Kern verallgemeinert eine Messung. Dies geschieht durch einen abstrakten Recorder, abstractes Record, abstracter Serializer etc. . Je nach instanziertem Recorder kann eine unterschiedliche Messung vollzogen werden. Zum Beispiel das Fahren einer Drehzahlrampe, aber auch ein Sprung für die Sprungantwort oder das Anfahren einzelner vorgegebener Punkte ist so leicht möglich. So genügt es zum Beispiel für eine neue Art der Messung (oder Messtrategie) einen neuen Recorder zu implementieren, in dem man die Schnittstelle implementiert. Der neue Recorder fügt sich automatisch in die Software ein. Sprich die gemessenen Daten können wie zuvor auf die Festplatte geschrieben, durch die GUI angezeigt und durch die DataProcessingunti gefiltert werden werden. Für den neuen Recorder bleibt die gesamte Funktionelität der Software verfügbar. Dies gilt für jede Klasse des Kerns außer dem Dispatcher.
Jede Komponente des Kerns, ausgenommen des Dispatchers, ist einzeln austauscchbar da die Schnittstellen wohl definiert sind.
Wenn man also zum Beispiel einen neuen Serializer für ein neues Datenformat schreibt, in dem man die entsprechende Schnittstelle implementiert, so ist dieser Serializer für jeden gemessenen Datensatz verfügbar und unabhängig von den davor neu implementierten Recordern.

4.1 Die globale Datenbasis

Das Fundament des Softwarekerns ist die Klasse DataBase. Sie ist eine singelton Klasse und hält alle relevandten Instanzen wie zum Beispiel eine Recorderinstanz, eine Recordinstanz und vieles mehr. Die Record_id dereferenziert alle Instanzen, welche zu einem Kontext gehören. Natürlich gibt es mehrere verschiedene Context_ids. Für jeden Kontext genau eine.
Um die Datenbasis global verfügbar zu machen, bekommt sie eine statische Wrapper, welche einen öffentlichen Zugriff auf die singelton Klasse erlaubt.

```
static class DBW
{
      private static DataBase glob_dat_base = DataBase.get_instance();
      public static DataBase GlobDataBase
      {
          get { return glob_dat_base; }
      }
}
```

Der Zugriff auf die Elemente der Klasse DataBase erfolgt von jedem beliebigen Punkt der Software aus wie folgt:

```
DBW.GlobDataBase.<Element>
```

4.2 Der Dispatcher

Einer der wichtigesten Elemente der Software ist der Dipspatcher. Ich habe es ab und an erlebt, dass für die Kommunikation zwischen zwei Instanzen für jede Nachricht einzeln funktionspointer benutzt werden. Ich halte aus zwei Gründen nicht viel davon. Der erste:

Mit steigender Komplexität der Software wird die Vorgehensweise unglaublich unübersichtlich und die Software wird die Eigenschaft der Wartbarkeit verlieren.
Der zweite Grund ist, dass man beim Aufruf eine Funktionspointers nicht genau weiß, wie lange genau der Code hinter dem Funktionpointer zur Abarbeitung benötigt.
Aus diesem Grund habe ich den Dispatcher implementiert. Er löst diese Probleme beispielhaft.
Der Dispatcher besteht aus drei Klassen. Die Hauptklasse ist der GUI_Dispatcher. Es gibt in der Software genau eine Instanz dieser Klasse: Diese Instanz befindet sich in der globalen Datenbasis. Der Zugriff auf den Dispatcher erfolgt also durch folgenden Code:

```
DBW.GlobDataBase._dispatcher.<Methode>
```

Mit dem Dispatcher kann man dann Nachrichten an Instanzen versenden. Folgende Methoden sind möglich:

```
public void write_message(int context_id, CONTEXT_TYPE c_type, string message)
public void write_int_message(int context_id, CONTEXT_TYPE c_type, int val)
public void write_double_message(int context_id, CONTEXT_TYPE c_type, double val)
```

Diese Methoden rufen dann in der entsprechenden Instanz ihre korrespondierenden get Methoden auf:

```
public void get_message( string message);
public void get_int_message( int val);
public void get_double_message( double val);
```

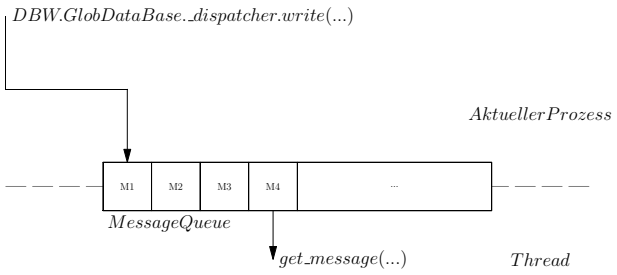
Um die Instanzen einer Klasse dem Dispatcher verfügbar zu machen, müssen ein paar Dinge erledigt werden. Erstens, die neue Klasse muss entweder von dem Interface Notification oder von GUI_Notification abgeleitet werden. Dann muss die Funktion

```
public void _write2context()
{...
}
```

entsprechend modifiziert werden. Als letztes müssen sie ihren Typ als CONTEXT_TYP hinzufügen.

```
public enum CONTEXT_TYPE { RECORD = 0, SERIALIZER = 1, ...
```

Nun können Sie die Instanzen ihrer neuen Klasse durch den Dispatcher global erreichen.



Der Dispatcher ruft beim Versenden einer Nachricht nicht direkt den entsprechenden Funktionepointer auf. Sondern die write Methoden des Dispatchers tragen eine Nachricht in eine Warteschlange ein. Dann kehrt die writefunktion auch schon zurück. Somit haben die write Funktionsaufrufe eine konstante Laufzeit, unabhängig davon wie komplex der Code in der entsprechenden get Methode ist.
Die entsprechende get Methode wird dann in einem eigenen Thread abgearbeitet. Mit einem Dualcore Prozessor also parallel.
Achtung, vergessen Sie nicht den Aufruf zu marshallen.

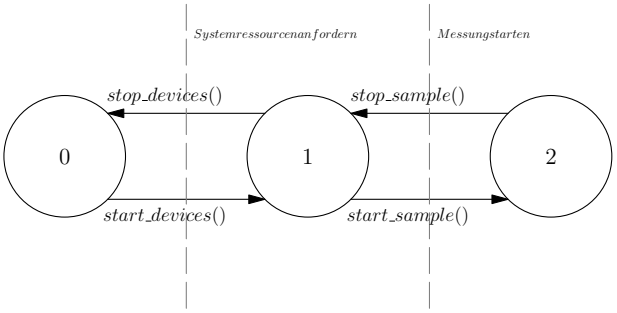
4.3 Das Record

Im Record befinden sich die gemessenen Daten. Ein Recorder schreibt immer in rein Record. Das Record schreibt in eine Tabelle aus Zeilen und Spalten. Auch hier gilt, beim Erzeugen einer neuen Recordklasse, fügt sich diese automatisch in die Software ein, sofern man die Schnitstelle richtig implementiert.
Die Viewerklassen zum Darstellen der Daten lesen immer aus einem Record. Diese Vorgehensweise entkoppelt den Recorder von der GUI und stellt sicher, dass nicht die gesamte Software umgeschrieben werden muss, wenn man eine neue Recorderklasse schreibt.
Bitte umgehen Sie diese Vorgehensweise nicht. Wenn Sie Softwareabhängigkeiten zwischen der GUI und dem Recorder erzeugen, ist das nie eine gute Lösung, da Sie dann für jede neue Messung neue graphische Klassen zum Darstellen schreiben müssen.

4.4 Der Recorder

Der Recorder ist der arbeitende Teil der Software. Er sendet die Befehle an die Geräte, und führt die Messung durch. Des Weiteren schreibt er in das entsprechende Record. Das Interface des Recorders ist als FSM (Finite State machine) zu implementieren.

Die Maschine hat drei Zustände.



Bei dem Übergang zum ersten Zustand sollte die Systemressourcen angefordert werden. Dabei werden die Software Devices instanziert, welche wiederum eine Connection benötigen. Es darf sich immer nur ein Recorder nicht im Zustand 0 befinden, da es sonst zum Systemresourcenkonflikt kommt.
Beim Aufruf von start_sample, beginnt der Recorder die Messung zu starten.

Der Recorder behandelt die durch die instanzierten SoftwareDevices ausgelösten Softwareinterrupts. Hierzu wird ein Array von Funktionspointern instanziert, welches eine Funktion für jedes Sofwaredevice hat. Wenn also ein Hardwaregerät auf ein Softwaredevice schreibt, dann wird eine entsprechende Funktiion im Recorder aufgerufen. Somit gehört zu jedem Gerät, eine Funktion. Diese Vorgehensweise habe ich mir aus dem Linuxkernel abgeschaut und sie ist genau so effizient wie schön zu programmmieren,da die Information dadurch weitestgehend getrennt ist.

Es ist sogar so, dass beim Aufruf einer Interruptfunktion durch ein Gerät der zuvor ausgeführte Befehl, welcher an das Gerät geschickt wurde, exakt bestimmt werden kann. Auch hier kommen Messages zum Einsatz, welche aus einem Befehl bestehen, bis an die Verbindungsschicht heruntergereicht werden und dann beim Hardwareinterrupt mit Daten gefüllt werden.

Dannach wird die gesamte Message wieder hochgereicht und die Information aus Befehl und Antwort bleibt am Stück. sowie genau einem Gerät zugeordnet.

Jede Interruptfunktion, welche auf die Hardware reagiert, bekommt also ein _device_message Objekt. In diesem Objekt befinden sich sowohl der gesendete Befehl, als auch das dazugehörende Datum.

4.5 Der Serializer

Die Serializerklassen sind dazu da, die Messdaten auf die Festplatte zu schreiben und von dort wieder zu lesen. Bisher wird ein Binärfile und ein Excelfile unterstützt. Ein Serializer schreibt immer von einem Record. Dies stellt sicher, dass das Schreiben auf die Festplatte unabhängig von den GUI-Klassen sowie dem Recorder ist.

4.6 Das Softwaredevice und die Connections

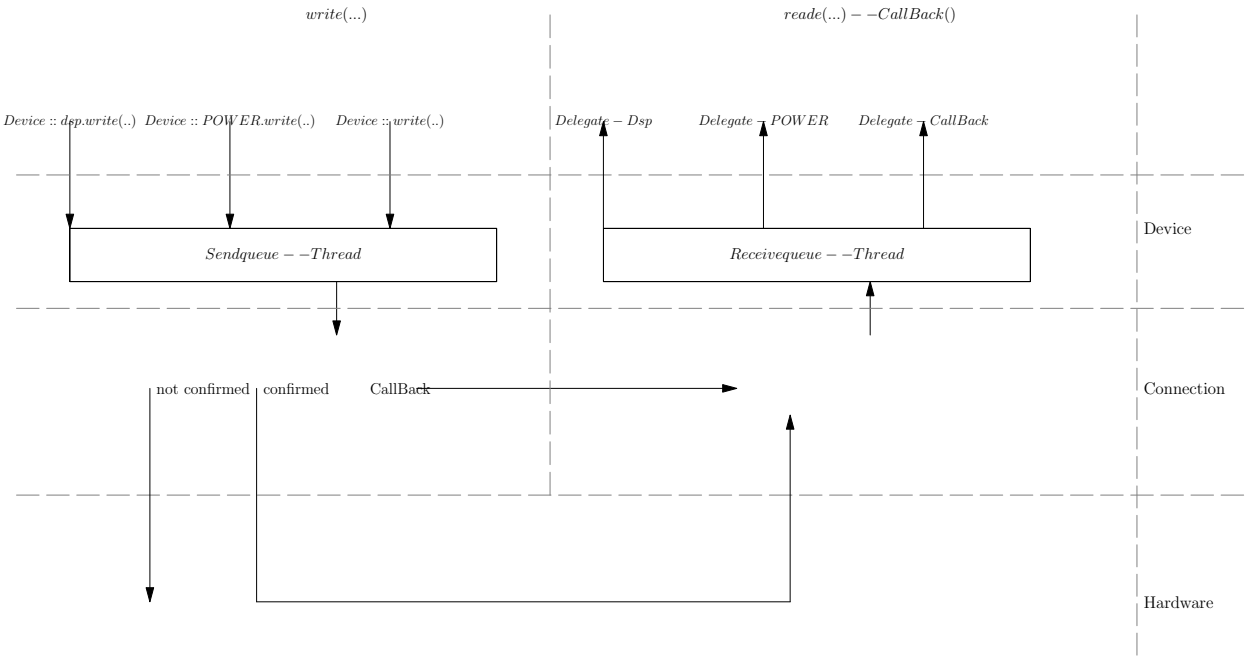
Ein Softwaredevice schreibt Befehle in Form von Messages auf ein Hardwaregerät, oder ruft eine entsprechende Callbackfunktion auf, wenn auf das Softwaredevice geschrieben wird.

Generell gibt es drei Arten von Messages. Confirmed, not confirmed und callback. Die confirmed Nachrichten erwarten vom Gerät eine Antwort, welche sie dementsprechend in die Message mit aufnehmen. Not confirmed Nachrichten tuen das nicht, sondern sie schreiben auf das Gerät, und warten eine bestimmte Zeit um sicherzustellen, dass der Befehl übernommen wurde.

Um die Callbackmessage zu erleutern muss etwas tiefer geblickt werden. Jedes Softwaredevice schreibt ihre Message in einen Senderqueue, und kehrt dann zurück. Ein weiterer Thread bearbeitet diese Nachricht, und reicht sie an die entsprechende Connection weiter. Beim Empfang einer Nachricht, wird nicht direkt ein Softwareinterrupt ausgelöst, sondern es wird eine Message in den Empfangsqueue geschrieben. Ein weiterer Thread ruft dann die entsprechenden Interruptfunktionen auf.

Um nun direkt in den Empfangsqueue schreiben zu können, gibt es die Callbackmessage. Befindet sich eine Callbackmessage im Senderqueue, wird sie nicht an eine Connection weitergereicht, sondern direkt in den Empfangsqueue übernommen.

Auf diese Weise kann man zum Beispiel den Recorder dazu veranlassen, seine Threads zu synchronisieren.



Einige GUI-Komponenten 5

5.1 Karthesian View

Die Karthesian Viewklasse bringt Daten aus einem Record auf den Bildschirm. Sie kann beliebig viele Kurven malen, sowohl Punktweise, als auch gestrichelt. Sie hat eine Methode um Kurven hinzuzufügen. Diese bekommt eine Farbe, sowie zwei Spalten des Recordes. Die eine für die x, die andere für die y-Achse. Des Weiteren bekommt sie eine Anweisung, wie die Kurve gemalt werden soll, und einen Skalierungswert.
Als Transformationen werden Zoom, Translation und Skalierungen unterstützt.
Die Klasse besitzt eine eigene Methode, um aus den gemessenen Daten eine Transformation zu errechnen, welche alle Punkte auf die Leinwand malt.
Es können Mehrere dieser karthesischen Koordinatensysteme gleichzeitig instanziert und bemalt werden. Sie ist also Threadsafe.

5.2 Table View

Diese Klasse nutzt die DataGridView um die Daten darzustellen. Eine einfache Anbindung der DataGridView an das Recordinterface der Software.

Der Prüfstand 6

Der Arbeitsplatz beim programmieren mit Architekturskizzen. Man erkennt schön den Prüfstand. Auf der linken Seite des Bildes befindet sich die Hochspannungsquelle.

Das Arbeiten bei GEFEG-Neckar ist sehr entspannt. Ich habe dort sehr gerne gearbeitet und reichlich Überstunden geleistet. Abschließend ist zu sagen, dass das Projekt sehr gut gelungen ist.