NTB
INTERSTAATLICHE HOCHSCHULE
FÜR TECHNIK BUCHS

Interstaatliche Hochschule für Technik Buchs (NTB)

MRU Systemtechnik, Automation und Produktion

Fachbereich Embedded Systems und Informationstechnologie

im Rahmen des Projektes

EEROS - an Easy, Elegant, Reliable, Open and Safe robotic software

Implementierung Prototyp

Master-Thesis

von
Julian Specker

Referent	Hr. Prof. Dr. Urs Graf
Experte	Hr. Dr. Bernhard Sprenger
Advisor	Hr. Prof. Dr. Norbert Frei
Version	1.0
Seiten	43
Datum	31. Januar 2013

Zusammenfassung

Die Programmierung von Robotikanwendungen ist komplex, gute Hilfsmittel sind deshalb unverzichtbar. Die auf dem Markt verfügbaren Tools sind entweder teuer oder stellen unnötig grosse Eintrittshürden auf.

In einer Allianz der NTB mit der ZHAW, welche durch die Gebert Rüf Stiftung gefördert wird, soll mit EEROS (Easy, Elegant, Reliable, Open and Safe robotic software) ein Software Framework geschaffen werden, welches den Benutzer ins Zentrum rückt. Ziel ist es, der Wirtschaft ein Hilfsmittel in die Hand zu gegen, mit welchem sich ohne lange Einarbeitungszeit qualitativ hochstehende Produkte im Bereich Automatisierung und Robotik erstellen lassen. Durch den Entscheid, das Resultat der Arbeit als Open Source zu veröffentlichen, soll es einer Community ermöglicht werden, eigene Ideen mit einzubringen.

Das Framework besteht aus vier Hauptbereichen: Control, Sequencer, Error Handling und Safety Framework.

Das Control Framework ist der eigentliche Regelungstechnik-Teil. Tasks im Control Framework werden in Echtzeit ausgeführt.

Im Sequencer Framework werden die übergeordneten Robotik-Aufgaben initiiert, beispielsweise das nächste Ziel geplant.

Im Error Handling Framework werden adäquate Lösungen für aufgetretene Probleme gefunden.

Das Safety Framework überwacht bestimmte interne Signale, Ein- und Ausgänge und stellt durch ein ausgeklügeltes System sicher, dass keine Gefährdung des Benutzers oder des Roboters möglich ist, ohne dabei das Debuggen zu erschweren.

Die in den Konzeptfindungssitzungen definierten Anforderungen wurden im implementierten Prototyp grösstenteils umgesetzt. Die Funktion des Prototyps wurde mit zwei Beispielanwendungen nachgewiesen.

Die erste Beispielanwendung ist ein simpler Positionsregler. Auf einem Motor mit Encoder wird die durch einen zweiten Encoder vorgegebene Position nachgeregelt.

Die zweite Beispielanwendung ist ein Stab balancierender SCARA Roboter. Diese Robotikanwendung erfordert eine Initialisierungssequenz, mehrmaliges Wechseln der Regelcharakteristik, hat diverse Ein- und Ausgänge und bedingt das Triggern eines Watchdogs.

Abstract

The programming of robotics applications is complex. Therefore there is an indispensable need for good tools. Tools available on the market are either expensive or raise unnecessary barriers for potential users.

In an alliance with ZHAW, the NTB develops EEROS, an Easy, Elegant, Reliable, Open and Safe robotic software. The Gebert Rüf Stiftung funds the alliance. Target of the project is to develop a software framework that is build around users needs. The alliance wants to provide the economy with a tool that leads to high quality automation and robotics applications with affordable time invested. EEROS will be released to the community as free open source software. This will empower everyone interested to contribute to the progress.

EEROS consists of four components: Control-, Sequencer-, Error Handling- and Safety-Framework.

Control Framework is the component where all the control tasks take place. It is the only part that executes tasks in real time.

The Sequencer-Framework is responsible for the robotics functions. Planning the next destination would be an example of that.

The Error Handling-Framework reacts to errors from the Sequencer-Framework and finds appropriate actions to resolve them.

The Safety-Framework monitors internal signals, inputs and outputs. It makes sure, neither the programmer nor the robot is at risk. The well-thought-out concept does all that without limiting the programmer in an exceeding fashion.

Most of the functions and wishes stated by the EEROS project team were successfully implemented within that work. The function of the prototype implementation has been proven with two sample applications.

The first sample application is a simple position controller. There is a motor with an attached encoder and a second encoder as reference input. All the movements from the reference encoder is mirrored to the motor.

The second sample application is a bar balancing SCARA robot. This application requires an initialization sequence, repeated transition of the control characteristics and has many in- and outputs.

Inhaltsverzeichnis

1. Motivation

Roboter sind der nächste grosse Boom, darin sind sich die Experten einig. Durch die hohe Komplexität sind gute Tools für die Erstellung von Robotikanwendungen eine Voraussetzung. Die am Markt verfügbaren Produkte sind jedoch aus Sicht des Projektteams aus diversen Aspekten nicht ideal. Zu den Kritikpunkten zählen:

- Einstiegshürde unnötig gross:
  - Webauftritt transportiert den Zweck und die Möglichkeiten des Produkts nicht oder nicht ausreichend.
  - Anleitungen sind nicht oder nicht in der nötigen Qualität vorhanden.
- Kosten zu hoch:
  - Das Produkt erfordert den Kauf von dedizierter Hardware.
  - Das Produkt kostet Lizenzgebühren für die Entwicklungsumgebung, pro Endprodukt oder sogar beides.
- Bedienungskonzept nicht praxistauglich:
  - Der Robotik-Anwender ist kein Informatiker. Es soll ihm somit auch nicht die Denkweise eines Informatikers aufgedrängt werden.
  - Der Robotik-Anwender will sich aber auch nicht einschränken lassen. Es muss für den Anwender möglich sein, eigene Funktionen zu implementieren, an welche der Hersteller des Frameworks möglicherweise nicht gedacht hat.
- Sicherheitsfunktionen sind nicht durchdacht:
  - Das Design des Sicherheitskonzepts wird vollumfänglich dem Benutzer zugemutet.

2. Spezifikation (Konzeptphase)

Innerhalb des Projektteams war schon vor dem Start des Projekts sehr viel Wissen zum Thema Robotik vorhanden. Speziell Herr Prof. Einar Nielsen bringt mit seiner 20-jährigen Erfahrung im Bereich Roboterentwicklung einen enormen Wissensschatz mit. In der Konzeptphase galt es somit, die Anforderungen und Wünsche an das Robotik-Framework zu sammeln und zu ordnen. Im Sinne der Nachvollziehbarkeit listet dieses Kapitel die gefundenen Eigenschaften und Vorschriften.

Das Kapitel spiegelt somit den Wissensstand zum Ende der Konzeptphase wider.

2.1. Systemübersicht

Das System besteht aus den vier Komponenten Control-, Sequencer-, Error Handling- und Safety-Framework. Das System soll in Java implementiert werden und sowohl auf Windows als auch auf Linux lauffähig sein. Eine detaillierte Beschreibung der einzelnen Komponenten des Systems ist Thema der nächsten Abschnitte. Um die geplante Architektur erfassbar zu machen, soll hier dennoch schon eine kurze Beschreibung anhand eines Beispiels vorgezogen werden.

Als Beispiel soll hier der Stab balancierende Roboter dienen, welcher später auch komplett implementiert wurde.

- Control Framework
  - Enthält die nötige Regelungstechnik. Im Beispiel wäre dies beispielsweise der kaskadierte Regler.
  - Berechnungen, Ein- und Ausgaben werden in Echtzeit durchgeführt.
- Sequencer Framework
  - Steuert die übergeordneten Ziele des Roboters.
  - Setzt Reglerstrukturen auf und aktiviert resp. deaktiviert diese.
  - Im Beispiel ist eine typische Aufgabe des Sequenzers der Start der Homing Sequenz und danach der Wechsel zum Positionsregler in welchem der Roboter zur Position fährt, in welcher der Stab aufgenommen werden kann.
- Error Handling Framework

  - Enthält Funktionen, um auf Fehler im Roboter und im Ablauf zu reagieren.

  - Im Beispiel könnte dies sein, dass die Homing Sequenz für eine Achse nicht korrekt abläuft.

- Safety Framework

  - Das Safety Framework stellt sicher, dass keine Gefährdung des Benutzers, des Programmierers oder des Roboters auftreten kann.

  - Im Beispiel steuert das Safety Framework die Bremsen, die Freigabe und den Watchdog. Beispielsweise ist während dem Homing von Achse 1 nur gerade die Bremse und das Enable Signal von Achse 1 deaktiviert resp. aktiviert.

Die Zusammenarbeit der vier Bereiche Control, Sequencer, Error Handling und Safety kann auf unterschiedliche Arten aufgebaut sein. Denkbar ist eine Struktur wie in Abbildung 1, eine Struktur wie in Abbildung 2 oder jede Möglichkeit dazwischen.

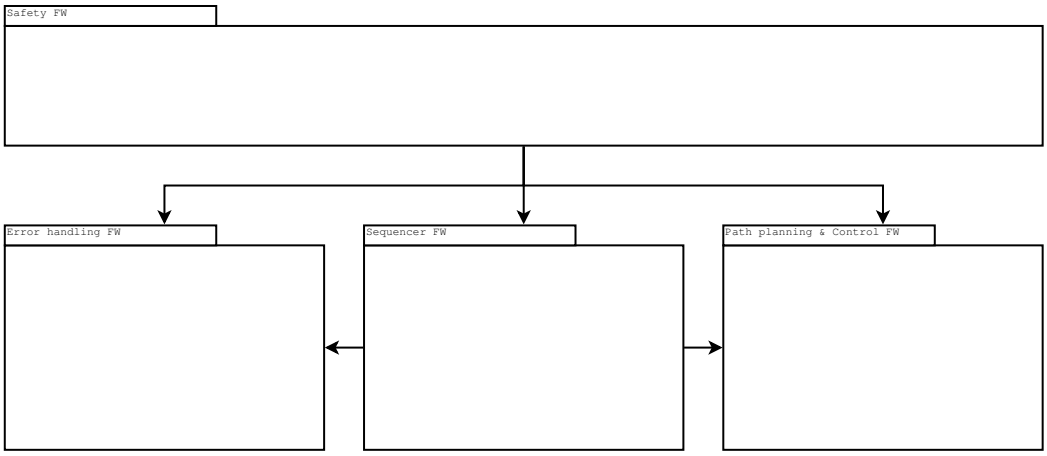

Abbildung 1: Struktur Variante A

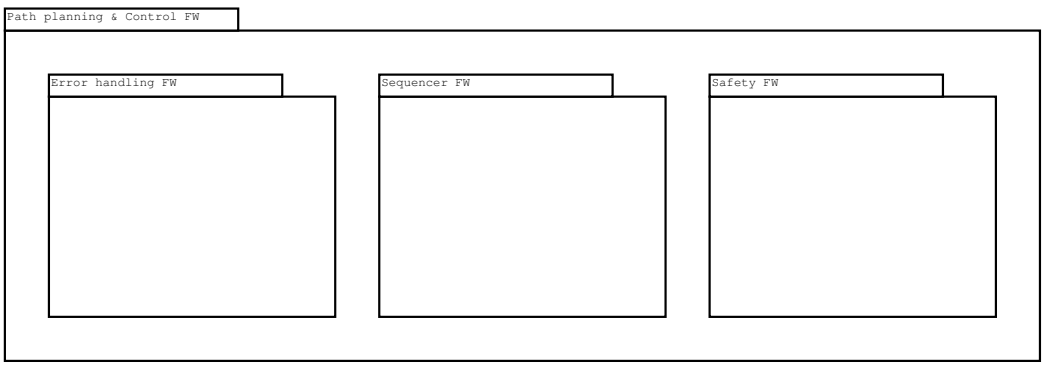

Abbildung 2: Struktur Variante B

2.2. Control Framework

Das Control Framework beinhaltet den eigentlichen Regeltechnik-Teil von EEROS.

2.2.1. Blöcke

- Blöcke mit zeitlichen Bedingungen:
  - Block mit Takteingang → Kann nur ein Ein- oder Ausgangsblock sein.
  - Block mit zeitlichem „Übersetzungsverhältnis“.
  - Asynchroner Block.
  - Blöcke ohne zeitliche Bedingungen, diese geben Zeitbedingung weiter.
  - Der schnellste Block ist NIE ein innerer Block.
- Bei Eingangs- und Ausgangsblöcken muss eine zeitliche Synchronizität definiert werden können (sofern von betroffener Hardware unterstützt).
- Bei Schleifen muss eine zeitliche Trennung eingefügt werden. Dies soll im Normalfall automatisch geschehen, soll aber auch vom Anwender manuell gemacht werden können. Beim automatischen Einfügen sollte dies vor der Summierstelle geschehen.
- Der Pfad, welcher zum Ausgang führt, wird priorisiert berechnet (Jitter klein halten).
- Blöcke müssen verschachtelt werden können, dies bedeutet, dass es Subsysteme gibt.
- Blöcke haben Signal Ein- bzw. Ausgänge.
  - Signale müssen verbunden werden.
- Blöcke haben Parameter. Diese können fest oder verstellbar sein.
  - Parameter müssen nicht verbunden werden.
  - Parameter können Initialwerte haben (zum Beispiel Verstärkung eines P-Blocks).

- Blöcke besitzen einen Konstruktor, einen Destruktor sowie eine „Run“-Methode.
- Blöcke haben eine Verbindung zum Sicherheitssystem.
- Blöcke können Ein- bzw. Ausgänge für „Zeitsignale“ haben.
- Blöcke reichen geschätzte Ausgabezeit weiter.
  - geschätzte „Verarbeitungszeit“ muss bekannt sein.
- Blöcke haben einen Namen.
- SI-Einheit kann gespeichert werden.
  - Oder gehört die SI-Einheit zum Ein- und/oder Ausgang?

Mögliche Blöck sind:

- Algorithmus-Block
- Summierer
- P, I, D, PI, PD, PID
- Scope
- Filter: verschiedene HP und TP Filter, Bandpass Filter, ...
- ADC/DAC
- Interpolator
- Encoderauswertungsblock: FQD, Sin/Cos, etc.
- Kinect
- Laserscanner
- ROS-Interface
- Begrenzer/Limiter → verschiedene Typen (z.B. bei Geschw. Wurzelkurve)
- Signalfluss-Zeitmessung
- Schnittstellen: UART, Ethernet, USB, ...

- Quellen: Step, Sinus, Rechteck, ...
- HMI: Knöpfe, Ausgaben, ...
- Zeit-Trenn-Block (synchron/asynchron)
- Signal-zu-Parameter-Konverter
- File-Output (z.B. für CSV-Dateien)
- Signal-zu-Parameter-Konverter
- Blöcke zum Erstellen von „Signalbündeln“:
  - mit automatischer Interpolation, so dass alle Signale gleichen Zeitstempel haben
  - ohne Interpolation → Signale haben unterschiedliche Zeitstempel
- Synchronisierungsblock (zeitliche Synchronisierung → Interpolieren)

Die Tabelle Standardblöcke zeigt die unterschiedlichen Eigenschaften von den gefundenen Standardblöcken:

Block	Anzahl Ein-/Ausgänge	Taktabhängig	Takteingang	Zeitl. Übers.	Spezielles
P-Glied	1/1	Nein	Nein	Nein	-
Summierer	n/1	Nein	Nein	Nein	-
I, D	1/1	Ja	Nein	Nein	-
PI, PD, PID	1/1	Ja	Nein	Nein	-
Scope	n/1	Ja	offen	offen	HMI
Filter	1/1	Ja	offen	(Ja)	-
ADC	0/0	offen	Ja	Nein	HW Eingang
DAC	1/0	offen	Ja	Nein	HW Ausgang
Interpolator	1/1	Ja	Nein	Ja	ggf. asynchron
Eingabefeld	1/1	Nein	Nein	Nein	HMI, asynchron

Tabelle 1: Standardblöcke

2.2.2. Ein- und Ausgänge

- Ein Ausgang kann mit beliebig vielen Eingängen verbunden werden. Ein Eingang hingegen nur mit genau einem Ausgang.

2.2.3. Zeitbedingungen

- Pro Zeit-synchronem Bereich gibt es genau einen Grundtakt. Dies ist der schnellst in diesem Bereich vorkommende. Es muss aber nicht zwingendermassen der vom Benutzer angegebene sein.
- Alles was auf der Eingangsseite eines Blockes hängt, muss vor diesem berechnet werden.
- Werden neue Werte berechnet, bekommen diese einen neuen Zeitstempel.
- Zeitinformationen:
  - Vorgabe von Takt.
  - Offset zwischen zwei Eingangsblöcken.
  - Events.

2.2.4. Signale

- Jeder Wert hat einen Zeitstempel, dieser wird beim Erfassen und bei Berechnungen definiert.
- Signale enthalten einen Einzelwert oder ein Signalbündel.
- Es gibt zwei Arten von Signalbündeln:
  - alle Werte / Signale haben den gleichen Zeitstempel.
  - Werte / Signale haben unterschiedliche Zeitstempel.
- Es gibt eine zeitliche Gruppierung. Diese definiert:
  - den Takt
  - die Reihenfolge der Ein- und Ausgänge
  - das für den Bereich gültige „Zeitkonzept“:
    - Einlesen im Takt, Ausgeben so schnell wie möglich (jeder Kanal für sich).
    - Einlesen im Takt, Ausgeben im Takt.

    - Einlesen im Takt, Ausgeben im Takt, alle Kanäle direkt hintereinander.

- Signale können einen Zustand haben (aktiv, inaktiv, open loop, ...).
- Signale haben ggf. einen Namen.
- Signale haben eine Einheit.

2.3. Sequencer Framework

Der Sequenzer ist für die übergeordnete Ablaufplanung des Roboters zuständig. Er instanziiert und aktiviert für die aktuelle Aufgabe geeignete Reglerstrukturen im Control Framework. Für die Fehlerbehandlung greift er auf das Error Handling Framework zurück.

Folgende Eigenschaften wurden für den Sequenzer gefunden:

- Der Aufruf einer Sequenz ist blockierend.
- Der Aufruf eines Steps ist blockierend.
- Der Sequenzer ist nicht echtzeitfähig.
- Der Sequenzer ist kein Block.
- Der Sequenzer ist keine State Machine.
    - Die Darstellung wird "Flow of a sequence XYZ" genannt.
    - Dazu gehören ein oder mehrere Darstellungen "Error Handling of a sequence XYZ".

2.4. Error Handling Framework

Das Error Handling Framework ist für die adäquate Fehlerbehandlung von nicht planmässig ablaufenden Sequenzen zuständig.

Folgende Eigenschaften wurden für das Error Handling Framework gefunden:

- Das Error Handling ist kein Block.
- Es gibt generische Error Handling Routinen sowie spezifische, welche zu einem definierten Sequenzer gehören.

2.5. Safety Framework

Das Safety Framework garantiert die Sicherheit des Roboters in allen Betriebszuständen. Folgende Eigenschaften wurden im Brainstorming postuliert:

- Ein Safety Level...
  - besitzt eine eineindeutige Nummer.
  - besitzt einen eineindeutigen Namen.
  - definiert eine Matrix / einen Vektor mit Ausgängen.
  - definiert Bedingungen, welche in einem periodischen Überwachungstask überprüft werden.
- Die Reihenfolge der Nummerierung ist entsprechend den potentiellen Schäden bei Fehlfunktion.
- Zu einem Safety Level gehört eine Liste mit erlaubten Level Änderungen.
- Der aktuelle Safety Level kann von Blöcken im Control Framework und vom Sequencer abgefragt werden.
- Im Safety Framework kommen keine versteckten Zustände vor Abbildung 3.

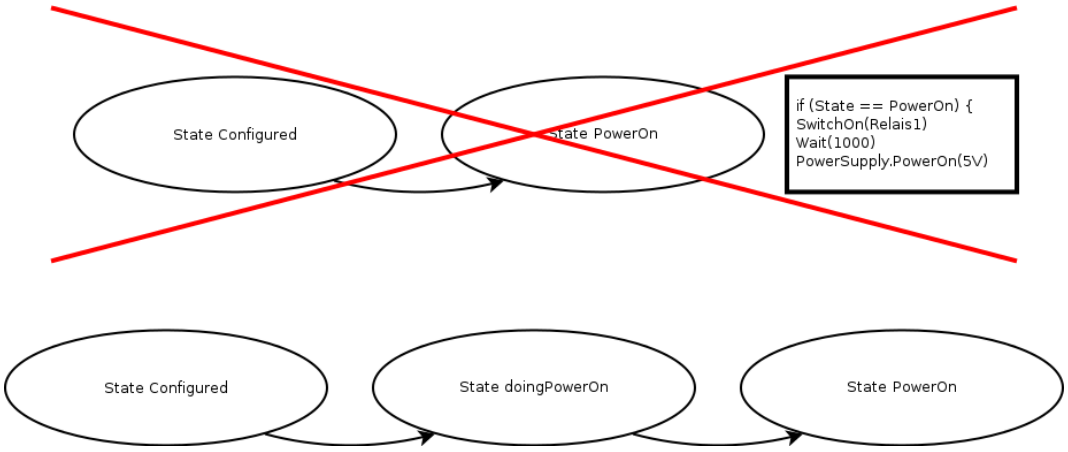

Abbildung 3: Beispiel versteckter Zustand

2.6. Anwendungsfälle

Folgende Anwendungsfälle sollen als Beispiel innerhalb des EEROS Projekts umgesetzt werden:

- Einzelachsregelung
- SCARA
- Vertikal-Knickarm-Roboter: Education-Roboter

Optional können in Zukunft folgende Anwendungsfälle umgesetzt werden:

- Mobiler Roboter: Eurobot
- Präzisionsroboter
- Flugroboter
- Chirurgieroboter

Bei der Implementierung des Frameworks soll ausserdem an den folgenden Anwendungsfall gedacht werden, welcher aber nicht innerhalb des Projektes umgesetzt werden soll:

- Mobiler Roboter: Autonom fahrendes Auto

3. Implementierung Prototyp

Dieses Kapitel dokumentiert die Architektur und die Funktionen des als Schwerpunkt dieser Arbeit erstellten Prototypen. Hauptziel des Prototypen war es, die Machbarkeit der in den Konzeptphase gefundenen Eigenschaften zu überprüfen und nachzuweisen.

3.1. Systemübersicht

Der Prototyp wurde gemäss Vorgaben in Java realisiert. Das Safety Framework steht einerseits den anderen Komponenten als Singleton für Abfragen des Zustandes sowie für Anfragen zum Wechsel auf einen neuen Safety Level zur Verfügung. Andererseits besitzt das Safety Framework einen Block, den Safety Block. Dieser läuft im Real Time Bereich des Control Frameworks mit. Über diesen Safety Block kann das Safety Framework wichtige Eingänge und Ausgänge in Echtzeit schalten, beispielsweise die Bremsen aktivieren, wenn der Not-Aus gedrückt wurde. Zuletzt besitzt das Safety Framework einen Task welcher periodisch Eigenschaften und Signale des restlichen Frameworks überwacht.

Das Sequenzer Framework steuert das Control Framework und ruft bei Bedarf Komponenten des Error Handling Framework auf.

Die Beziehungen sind in Abbildung 4 ersichtlich.

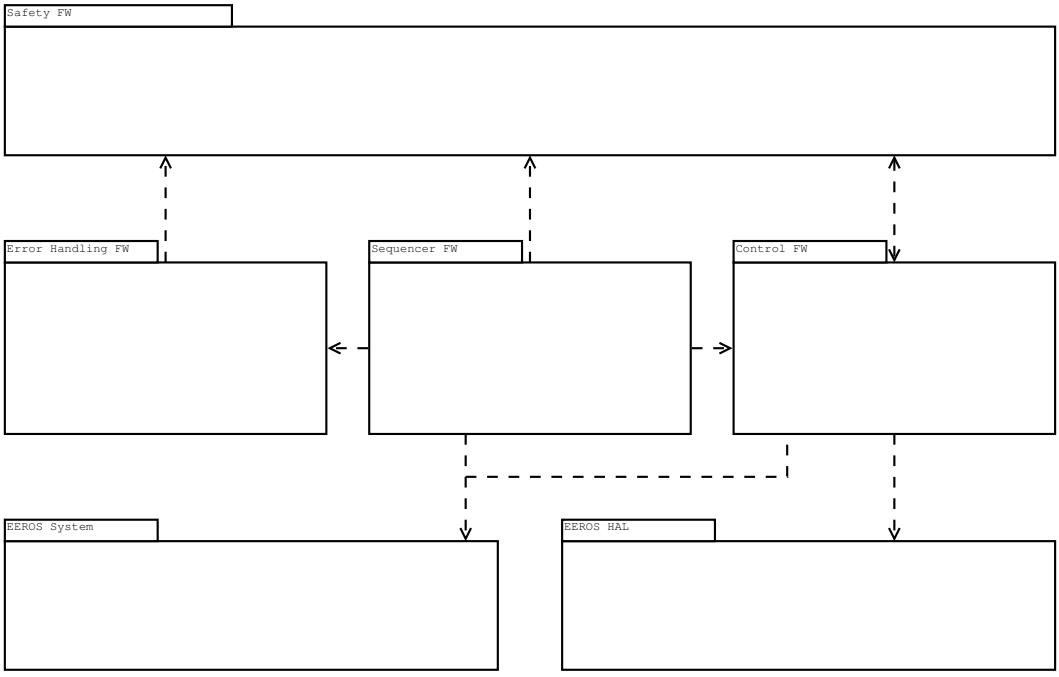

Abbildung 4: Abhängigkeiten zwischen den Komponenten

3.2. Control Framework

Wie vorgängig erwähnt beinhaltet das Control Framework den Regeltechnik-Teil vom EEROS Framework. Es die umfangreichste der vier Komponenten. Für den Prototypen musste bereits eine grosse Anzahl von Klassen, Methoden und Konzepten implementiert werden.

Der Dreh- und Angelpunkt für das Verständnis des Control Frameworks ist die Klasse Block. Der Name ist als Metapher für klassische Regleraufbauten mit diskreten Reglern gedacht.

Ein Block hat 0 bis n Ein- und Ausgänge. Diese werden durch die Klassen Input und Output abgebildet. Ein Eingang wird an einen Ausgang angeschlossen, worauf ein Signal weitergereicht werden kann.

Die eigentliche Funktion eines Blocks wird in der Methode run() implementiert. Diese run-Methode wird dann periodisch aufgerufen. Wie oft dies pro Sekunde passiert, hängt davon ab, zu welcher Zeitdomäne (TimeDomain) der Block gehört. Die Planung, in welcher Reihenfolge die einzelnen Blocks aufgerufen werden müssen, damit das Signal in möglichst kurzer Zeit durchläuft, ist die Aufgabe des Schedulers.

3.2.1. Blöcke

Block ist eine abstrakte Klasse mit den Daten welche jeder Block unabhängig von seiner Funktion enthalten muss:

- name (String)
- Anzahl Eingänge (int)
- Anzahl Ausgänge (int)
- Zeitdomäne (TimeDomain)

Davon abgeleitet die abstrakten Klassen Block1i1o, Block2i10, ...

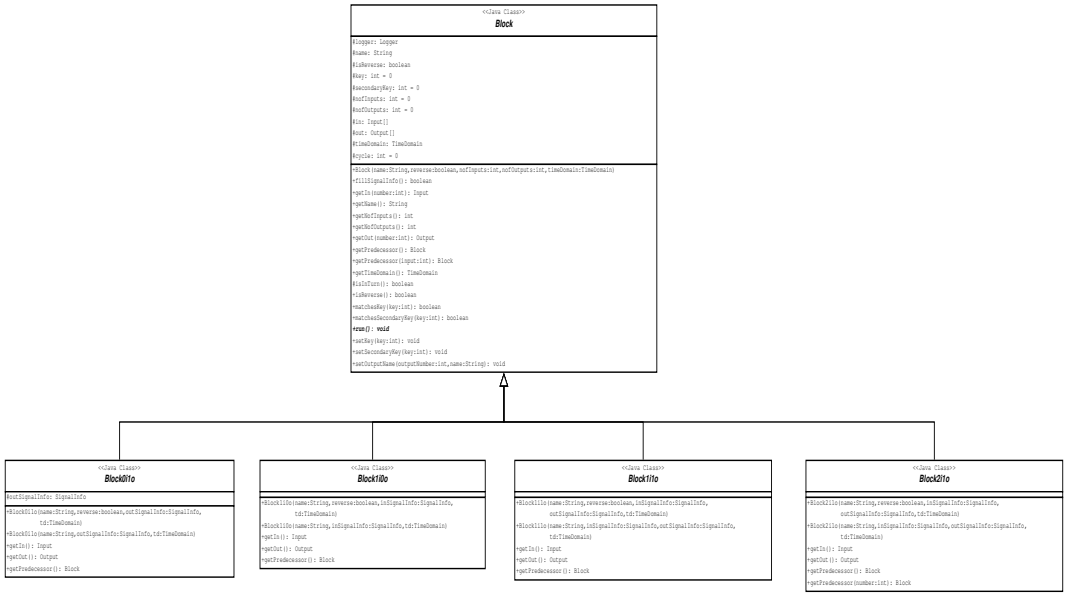

Abbildung 5: Klassendiagramm prototype.control.blocks

Von den abstrakten Klassen Block1i1o, Block2i1o, ... werden wiederum die konkreten Implementierungen abgeleitet:

- I abgeleitet von Block1i1o
  - run() Methode enthält die eigentliche Implementierung

Diese Situation wird in Abbildung 6 gezeigt.

Natürlich ist auch eine konkrete Implementierung möglich, welche direkt von Block ableitet. Die abstrakten Klassen vereinfachen lediglich die Implementierung. CombinedBlock ist ein Beispiel, welches direkt von Block erbt [Abbildung 7].

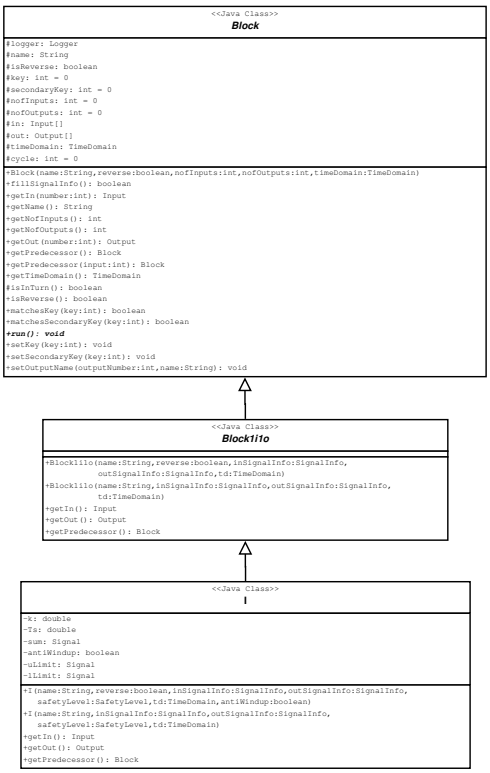

Abbildung 6: I erbt von Block1i1o welche wiederum von Block erbt

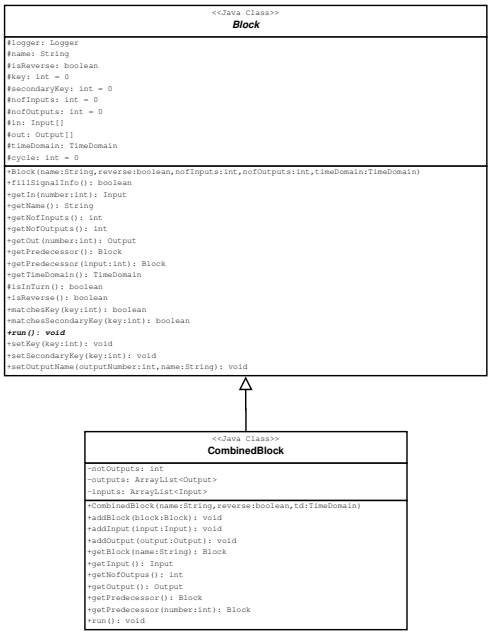

Abbildung 7: CombinedBlock erbt direkt von Block

3.2.2. Klasse Signal und die davon abgeleiteten Klassen ScalarSignal, MatrixSignal und BundledSignal

Die abstrakte Klasse Signal definiert die von den abgeleiteten Klassen zu implementierenden Methoden. Dadurch können die meisten Aktionen, welche auf ein Signal angewendet werden müssen, implementiert werden, ohne dass der Laufzeittyp bekannt sein muss. Ein Flag isValid zeigt, ob das Signal gültig ist.

Von der Klasse Signal werden aktuell drei Klassen abgeleitet:

- ScalarSignal
  - 1 double Wert
  - 1 Einheit (Units)
  - 1 Zeitstempel (Time)
  - 1 Name (String)
- MatrixSignal

  - nxm Matrix mit double Werten
  - 1 Einheit (Units)
  - 1 Zeitstempel (Time)
  - 1 Name (String)

- BundledSignal
  - 0-n UnarySignale
  - 0-n MatrixSignale
  - 0-n BundledSignale

3.2.3. SignalInfo

- Klasse mit Informationen über das Signal
  - SignalType (UnarySignal, MatrixSignal, BundledSignal)
  - Einheiten (Units)
  - Anzahl Zeilen und Spalten (nofRows, nofCols)
- Mit der Methode matches(SignalInfo): boolean kann geprüft werden, ob die aktuelle Instanz kompatibel mit der SignalInfo ist, welche als Parameter übergeben wird.

3.2.4. SignalFactory

- SignalFactory erstellt aus einer SignalInfo Instanz ein Signal

3.2.5. Scheduler

Abbildung 8: Klasse Scheduler

Der Scheduler erhält vom Sequencer den Auftrag, die Blöcke im Control Framework in der richtigen Reihenfolge auszuführen. Dazu beginnt er bei den Ausgangsblöcken und arbeitet sich zu den Eingangsblöcken durch. Da jeder Block zu einer TimeDomain gehört, werden der TimeDomain in der geforderten Reihenfolge alle zugehörenden Blöcke in den runStack gelegt. Wenn der Sequencer die Methode startScheduledTask aufruft, erstellt der Scheduler eine Liste mit TimeDomains, ordnet diese und weist jede TimeDomain an, einen Timer zu starten.

Wichtige Attribute:

- Liste mit TimeDomains (ArrayList timeDomains)
  - Wird in der Methode schedule(Block) mit den in den Blocks vorhandenen TimeDomains gefüllt
- Temporäre Liste mit Blöcken (ArrayList tempList)
  - Enthält Kandidaten für den nächsten Scheduling-Lauf.
- Forcier Liste mit Blöcken (ArrayList forceList)
  - Kann mit der Methode addBlockToForceList(Block) gefüllt werden. Elemente darin werden zuerst geplant.

Wichtige Methoden:

- Die Methode "startScheduledTasks(): void" startet einen neuen Timer, welcher periodisch alle run() Methoden der Blöcke dieser TimeDomain ausführt.

- Die Methode "schedule(Block): void" fügt den übergebenen Block in die tempList und beginnt danach alle Blocks in der korrekten Reihenfolge in der tempList zur entsprechenden TimeDomain hinzuzufügen.

3.2.6. Scheduler Algorithmus

Start bei Ausgangsblock. Falls mehrere Ausgangsblöcke vorhanden sind, werden diese alle in die tempList gelegt und mit demjenigen gestartet, welcher die grösste Distanz zum nächsten Input hat.

Nun wird immer der Block, welcher mit dem Input des aktuellen Blocks verbunden ist, auf den runStack gelegt. Dieser wird dann zum aktuellen Block und somit wird wieder der nächste betrachtet. Bei Blöcken mit mehreren Eingängen werden die Blöcke welche sich an den Eingängen 2 – n hängen in die tempList gelegt. Der Block am Eingang 1 wird gleich behandelt wie ein vorgeschalteter Block eines Blocks mit nur einem Eingang.

Abgebrochen wird die Schleife, wenn entweder ein Eingangsblock wie ein AD Konverter erreicht wird oder ein Block bereits vom Scheduler verplant wurde.

Danach wird von allen Blöcken in der tempList der Abstand zum nächsten Eingangsblock ermittelt. Derjenige Block mit dem grössten Abstand wird als nächstes als aktueller Block definiert und die Schleife beginnt von vorne. Diese äussere Schleife wird so lange ausgeführt, bis die tempList leer ist.

Idealerweise erhält der Benutzer am Schluss noch die Möglichkeit, die Reihenfolge manuell zu ändern.

3.2.7. Verhalten beim Übergang von einer Zeit-Domäne zu einer anderen

Diejenigen Blöcke welche nicht mit dem höchsten Takt im System ausgeführt werden, werden nur alle n mal ausgeführt, wobei n dem Teilverhältnis (höchster Takt im System) / (Takt der Time Domain) entspricht.

3.2.8. TimeDomain

Abbildung 9: Klasse TimeDomain

Enthält den Namen, eine Unterscheidung RT oder nicht, den Grundtakt und den Teiler. Die TimeDomain enthält ausserdem ein Array mit Blöcken, welche vom ExecutorService ausgeführt werden sollen.

Wichtige Attribute:

- Liste mit TimeDomains (ArrayList timeDomains)
  - Wird in der Methode schedule(Block) mit den in den Blocks vorhandenen TimeDomains gefüllt
- Temporäre Liste mit Blöcken (ArrayList tempList)
  - Enthält Kandidaten für den nächsten Scheduling-Lauf.
- Forcier Liste mit Blöcken (ArrayList forceList)
  - Kann mit der Methode addBlockToForceList(Block) gefüllt werden. Elemente darin werden zuerst geplant.

Wichtige Methoden:

- Der Konstruktor prüft mit Hilfe der Funktion "validateDivisor(TimeDomain)", ob der konfigurierte Divisor gültig ist. Gültig sind Divisoren welche ein ganzzahliges Vielfaches aller kleineren Divisoren und welche selbst Teiler aller grösseren Divisoren sind.

- Die Methode "startScheduledTasks(): void" startet einen neue Timer welcher periodisch alle run() Methoden der Blöcke dieser TimeDomain ausführt

- Die Methode "schedule(Block): void" fügt den übergebenen Block in die tempList und beginnt danach alle Blocks in der korrekten Reihenfolge in der tempList zur entsprechenden TimeDomain hinzuzufügen.

3.2.9. Übergang von einer TimeDomain in eine andere

Denkbar sind drei Konzepte

Konzept 1: Eine Pipe pro Verbindung

Pro Signal (ScalarSignal, MatrixSignal oder BundledSignal) wird ein Ein- und ein Ausgangsblock verwendet. Dazwischen wird eine Pipe dediziert für dieses Signal verwendet.

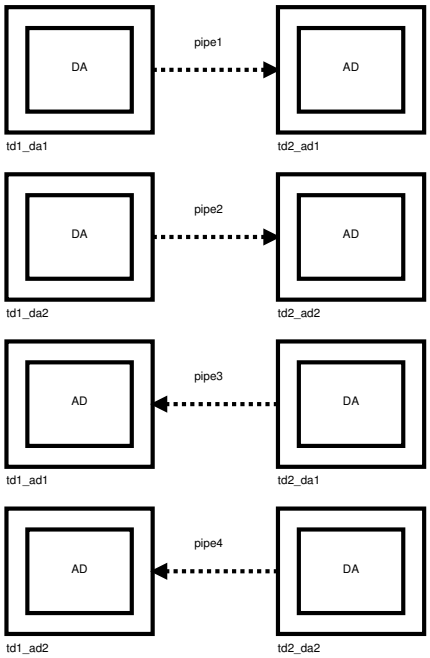

Abbildung 10: Übergang TimeDomain Konzept 1

Vorteile:

- Einfache Implementierung.

Nachteile:

- Thread welcher die Pipe öffnet bleibt stehen, bis die Gegenseite die Pipe ebenfalls öffnet.

- Grosse Anzahl von Pipes nötig.

Variante von Konzept 1: Asynchron laufender Thread zum Öffnen der Pipes verwenden.

Konzept 2: Eine Pipe pro Signalrichtung, die Ein- und Ausgangsblöcke schreiben und lesen direkt in der Pipe

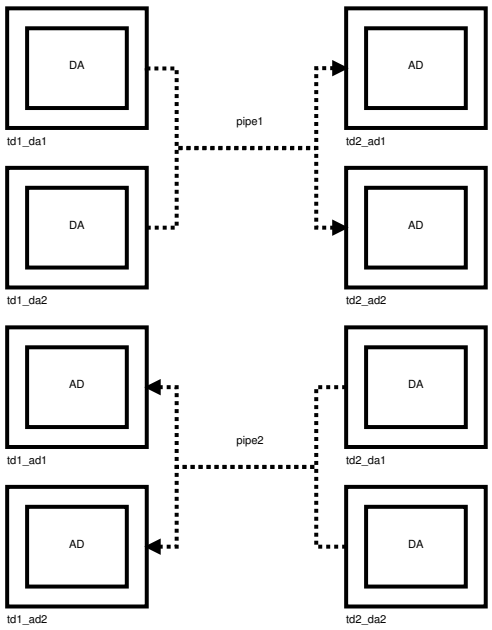

Abbildung 11: Übergang TimeDomain Konzept 2

Mehrere Ausgangsblöcke können in dieselbe Pipe schreiben und mehrere Eingangsblöcke von derselben Pipe lesen.

Vorteile:

- Nur zwei Pipes zwischen zwei TimeDomains.

Nachteile:

- Grösserer Overhead durch Protokoll.
- Anfälliger auf Locks

Konzept 3: Asynchron laufende In / Out Blöcke

Es wird pro Verbindung zwischen zwei TimeDomains je ein Ausgangs- und ein Eingangblock definiert. Dieser bündelt mehrere Signale und sendet diese über eine Pipe pro Richtung zum zugehörigen Eingangsblock.

Vorteile:

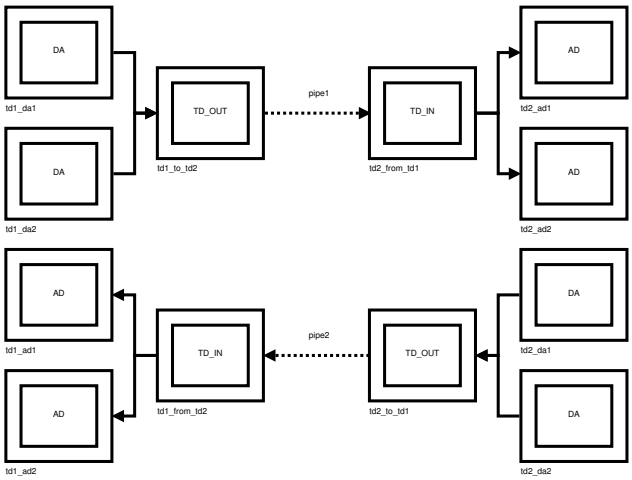

Abbildung 12: Übergang TimeDomain Konzept 3

- Anzahl Pipes ist kleiner.

Nachteile:

- Kompliziertere Struktur.

Diese Variante kann als Variante von Konzept 1a gesehen werden.

3.3. Sequenzer

Der Sequenzer ist für die Planung der übergeordneten Robotik-Aufgaben zuständig. Er baut die Reglerstruktur im Control Framework nach den Anforderungen des aktuellen Schrittes in der Sequenz auf und startet den Regler. Ein Beispiel einer Sequenz von Schritten (Sequence of Steps) ist in Abbildung Abbildung 13 dargestellt.

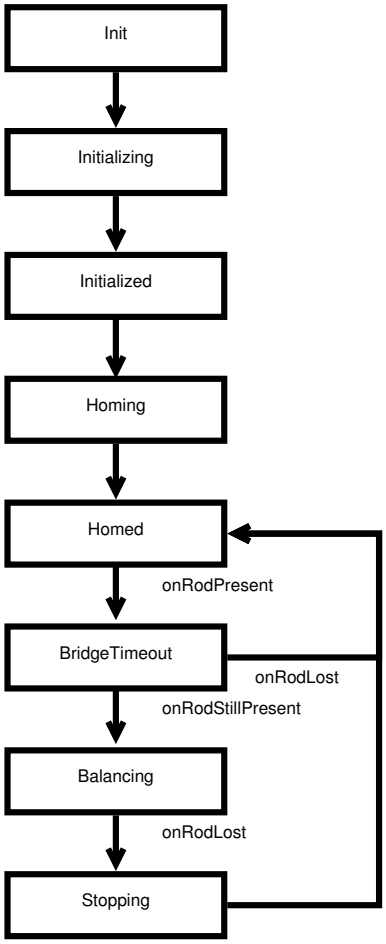

Abbildung 13: Beispiel einer Sequence of Steps

Die einzelnen Sequenzen werden als Ableitung von StepFunction implementiert. Jede Sequenz hat somit die Eigenschaften TimeOut, Status, ErrorHandler sowie die Funktion run(). Die Klasse StepFunction ist in Abbildung Abbildung 13 abgebildet.

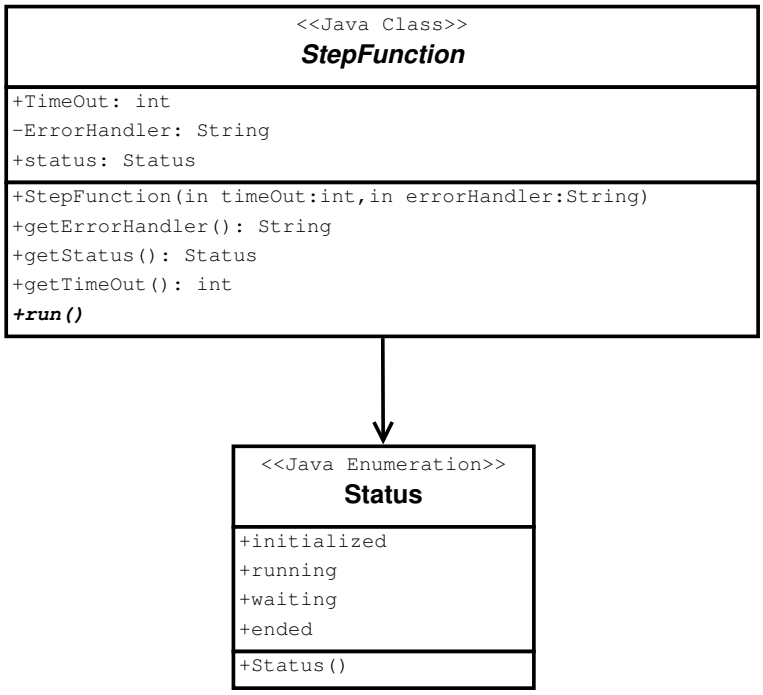

Abbildung 14: StepFunction

Damit der MainSequencer den neuen Sequenzer aufrufen kann und ein sicherer Wechsel zwischen den Sequenzen möglich ist, muss eine Sequenz noch bei der Enumeration Steps hinzugefügt werden. Der Konstruktor dieser Enumeration erfordert zwei Übergabeparameter: eine StepFunction und ein String. Der String ist dabei eine mit Kommas getrennte Liste von erlaubten Transitionen. Nur Sequenzen welche hier angegeben wurden, können nach dieser Sequenz aufgerufen werden. Die Abbildung Abbildung 15 zeigt die Klasse SequenzerSteps und die Enumeration Steps, beide wurden jedoch der Übersichtlichkeit halber auf die drei Sequenzen Init, Initializing und Initialized gekürzt.

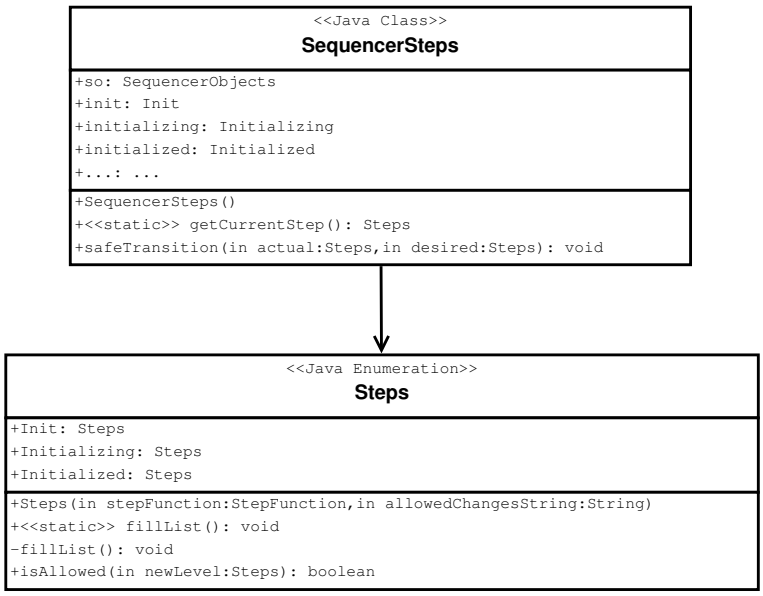

Abbildung 15: SequencerSteps

Um einen Sequenzer Schritt zu erstellen sind somit folgende Schritte notwendig:

- Erstellen einer Klasse welche von StepFunction ableitet, bspw. Initializing.
- Statische Deklaration einer Variable vom Typ der Klasse, bspw. initializing vom Typ Initializing
- Erstellen eines neuen Elements in der Enumeration Steps mit den Übergabeparametern StepFunction und String, bspw. Initializing(initializing, "Initialized");

Im Programmablauf ruft der MainSequencer immer den aktuellen Sequenzer auf. Diesen erhält er von SequencerSteps.getCurrentStep(). Danach hat der aufgerufene Sequenzer die bei seiner Initialisierung definierte Zeit zur Verfügung, seine Aufgaben zu erfüllen. Endet der Sequenzer-Thread nicht in dieser Zeit, wird er beendet und ein ErrorHandler wird gestartet. Mit der Funktion ErrorHandler.getErrorHandler(StepException e) wird der zur auslösenden Sequenz passende ErrorHandler gewählt.

3.4. Error Handling Framework

Das Error Handling Framework ist für die adäquate Fehlerbehandlung von nicht planmässig ablaufenden Sequenzen zuständig.

Tritt im Ablauf einer Sequenz ein Fehler auf oder spricht das Timeout an, wird ein Fehler geworfen. Dieser Fehler wird im MainSequencer aufgefangen und dient nun der Initialisierung eines ErrorHandlers. Dazu wird die statische Funktion ErrorHandler.getErrorHandler(StepExcepetion) welche einen ErrorHandler zurückgibt benutzt. Dies entspricht dem bekannten Factory Pattern. Der zurückgelieferte Error Handler kann nun ein generischer oder ein speziell für den aktuellen Schritt der Sequenz erstellter sein. Abbildung Abbildung 16 zeigt die Klassenstruktur mit einem generischen (DefaultErroHandler) und einem spezifischen (InitializingErrorHandler). Unterschiedlich bei diesen Klassen ist die Implementierung der Funktionen tryResolve() und tryResolve(StepException).

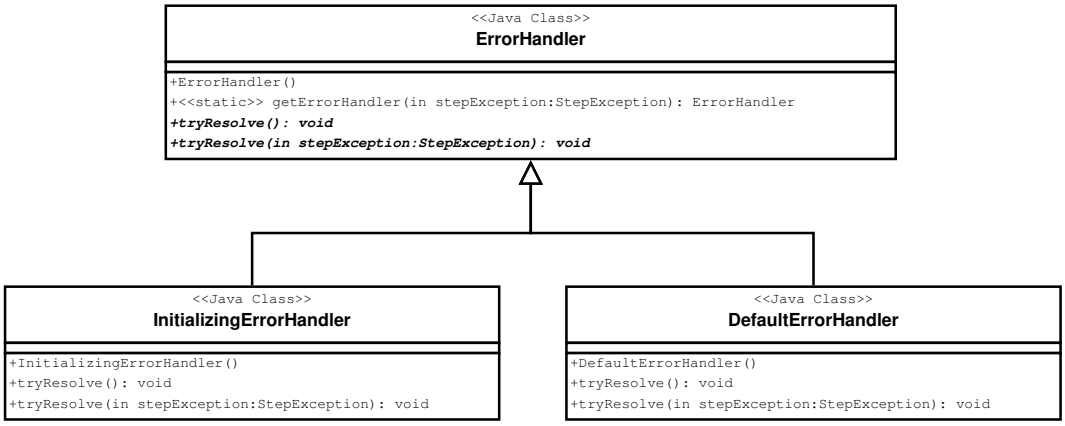

Abbildung 16: Klasse ErrorHandler

3.5. Safety Framework

Das Safety Framework stellt sicher, dass weder der Programmierer, der Benutzer noch der Roboter gefährdet werden kann. Die Kombination zweier Konzepte stellt dies sicher. Einerseits stellt das Safety Framework den aktuellen SafetyLevel allen Interessenten zur Verfügung. Beispielsweise kann damit ein Limitierungs-Block seine Limits abhängig vom aktuellen SafetyLevel einstellen. Für die schnelle Reaktion auf Änderungen bei Sicherheitseingängen und die Ausgabe von sicherheitsrelevanten Signalen steht der SafetyBlock zur Verfügung. Dieser erbt die Funktionalität der Klasse Block, stellt aber noch weitere Funktionalität zur Verfügung. Beispielsweise prüft er, dass auch wirklich alle seine Ein- und Ausgänge verbunden sind.

3.5.1. Safety

Die Klasse Safety enthält einen SafetyBlock sowie den aktuellen SafetyLevel. Sie wird als Singleton verwendet und stellt Funktionen zum Lesen des aktuellen SafetyLevels, zum sicheren Wechsel des SafetyLevels und zum Zugriff auf den SafetyBlock zur Verfügung.

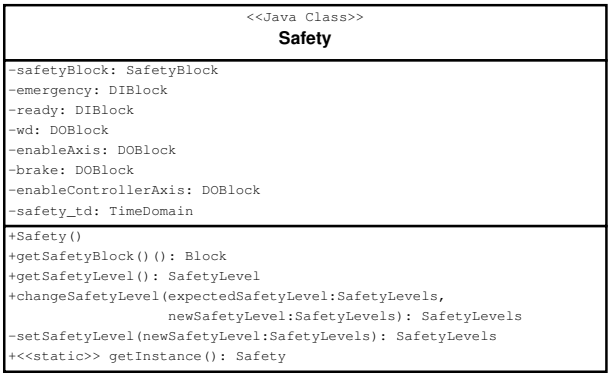

Abbildung 17: Safety

3.5.2. SafetyLevels

Die Enumeration SafetyLevels hat einen Konstruktor, welcher zu jedem Ausgang den im zu instanziierenden SafetyLevel den Wert erfordert. Die SafetyLevels werden somit an einer zentralen Stelle definiert und es wird jedem SafetyLevel direkt ein Set von Eigenschaften mitgegeben.

```
Name            Axis0,  Axis1,  Axis2,  Axis3,  FS      Allowed
NotConfig       (  false,  false,  false,  false,  false,  "Configurated"  ),
Configurated    (  false,  false,  false,  false,  false,  "NotConfig,TestingAxis0,TestingAxis1,Test
TestingAxis0    (  true,   false,  false,  false,  false,  "Configurated"  ),
TestingAxis1    (  false,  true,   false,  false,  false,  "Configurated"  ),
TestingAxis2    (  false,  false,  true,   false,  false,  "Configurated"  ),
TestingAxis3    (  false,  false,  false,  true,   false,  "Configurated"  ),
Initialised     (  true,   true,   true,   true,   false,  "NotConfig,Configurated,Running" ),
Running         (  true,   true,   true,   true,   true,   "NotConfig,Initialised" );
```

Abbildung 18: Definiton SafetyLevels

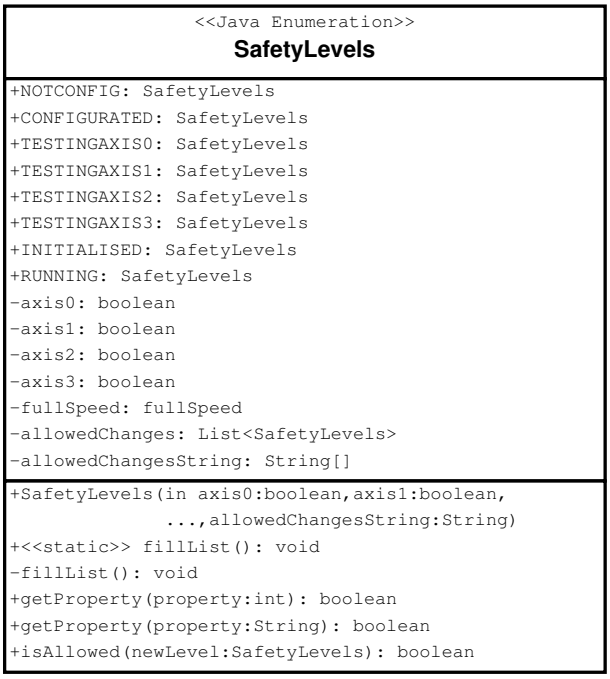

Abbildung 19: SafetyLevels

3.5.3. SafetyBlock

Bei jedem Aufruf der run() Methode wird:

- Geprüft, ob alle Ein- und Ausgänge des SafetyBlock verbunden sind. Ist dies nicht der Fall, werden alle Ausgänge auf 0, resp. auf den pro Ausgang konfigurierten SafeLevel gesetzt.

- Vom aktuellen SafetyLevel die Eigenschaften abgefragt und die Ausgänge entsprechend gesetzt.

- Definierte Aktionen auf Eingangssignale ausgeführt.

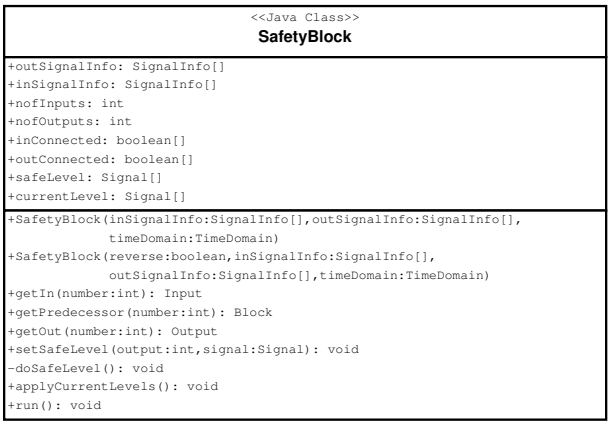

Abbildung 20: SafetyBlock

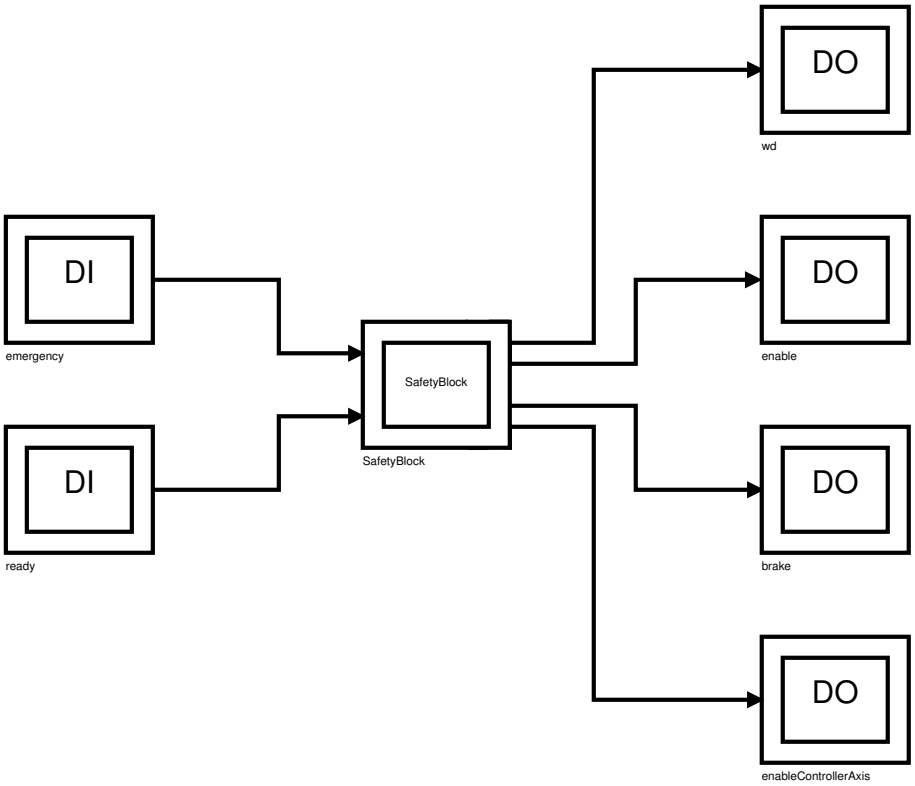

Abbildung 21: Verbindungen zum SafetyBlock

3.6. Native Interface

Um die geforderte Echtzeitfähigkeit zu erreichen, muss das Framework auf einem Real Time System ausgeführt werden können. Weiter ist gefordert, dass EEROS Plattform-unabhängig ist. Denkbar ist es somit, EEROS sowohl auf einer energiesparenden ARM Architektur für einfachere Regelcharakteristiken einzusetzen aber auch auf einem leistungsfähigen x86 System

für sehr aufwendige Regleraufbauten. Nicht zu vernachlässigen ist natürlich auch das Entwicklungssystem und die Entwicklungsumgebung. Als Build Infrastruktur wird GNU GCC for Java respektive GCJ verwendet. Dieser Compiler ist durch seine flexible Architektur mit Trennung in Frontend und Backend für beinahe alle Plattformen einsetzbar. Damit ist auch die Entscheidung zugunsten CNI schnell gefällt, da CNI von den GCJ Entwicklern gegenüber JNI klar favorisiert wird.

3.6.1. Real Time Threads

Nur wenige Teile von EEROS müssen in Echtzeit ausgeführt werden. Es sind ausschliesslich Teile des Control Frameworks sowie Teile des Safety Frameworks. Alle diese Komponenten sind Teil einer TimeDomain. Was somit für die periodische Ausführung übergeben wird, ist somit eine TimeDomain.

Angenommen, es werden folgende TimeDomains am System angemeldet:

1. Sequencer 100
2. Safety 1
3. TimeDomain 1 10
4. TimeDomain 2 20

Wie soll dann die Abhängigkeit zwischen den TimeDomains sein?

Möglichkeit 1: Kette

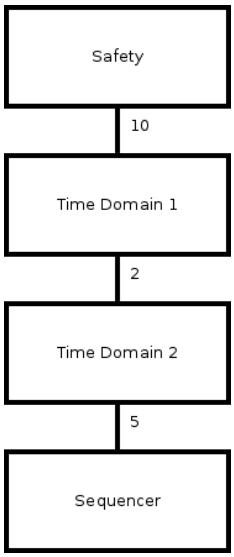

Abbildung 22: Möglichkeit 1: Kette

Jede TimeDomain steht in Relation zu einer anderen TimeDomain welche entweder den gleichen Divisor hat oder um eine Ganzzahl kleiner.

Vorteile:

- Klare Aufteilung

Nachteile:

- Was passiert bei Abmelden einer bestimmten TimeDomain resp. beim späteren Hinzufügen?

Möglichkeit 2: 1 Master

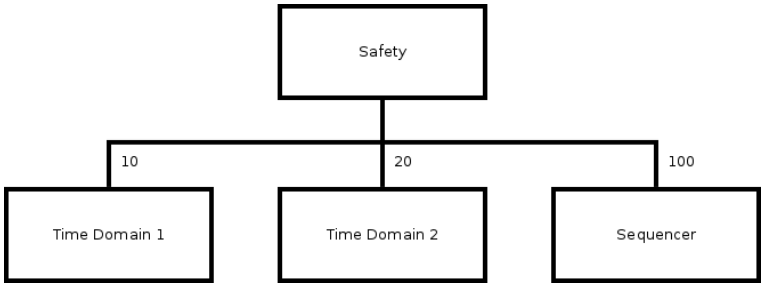

Abbildung 23: Möglichkeit 2: 1 Master

Es gibt im System eine Master TimeDomain. Alle anderen TimeDomains stehen in einem festen Verhältnis zu dieser.

Aufgrund der Nachteile von Möglichkeit 1 wurde die Möglichkeit 2 implementiert.

3.6.2. Hardware Interface

Der Zugriff auf die Hardware ist der zweite Bereich, bei welchem das Native Interface benötigt wird. Als Gerätetreiber Infrastruktur wurde das Comedi Framework gewählt. Es bietet echtzeitfähige Treiber für populäre PCI / PCIe Karten wie sie beispielsweise von National Instrument angeboten werden.

Von der Seite Framework wird auf die Klasse EerosHal zugegriffen. Diese Klasse bietet Funktionen, welche als Parameter nur einen String mit dem Namen der anzusprechenden Ressource und gegebenenfalls den zu schreibenden Wert erfordern.

Die Funtionen übersetzen dann mit Hilfe einer Map aus ComediSettings den Namen in die von JComediInterface benötigten Parameter dev, subdev, ch und gegebenenfalls chA, chB und chI.

JComediInterface hat in Java ausschliesslich Prototypen von Funktionen mit dem Keyword native. Die Implementierungen dieser Funktionen sind in der gleichnamigen C++ Klasse zu finden.

JComediInterface ist abhängig von der Klasse ComediInterface welche dann die von der Bibliothek comedilib zur Verfügung gestellten Funktionen benutzt. Die Situation wird durch Abbildung Abbildung 24 dargestellt.

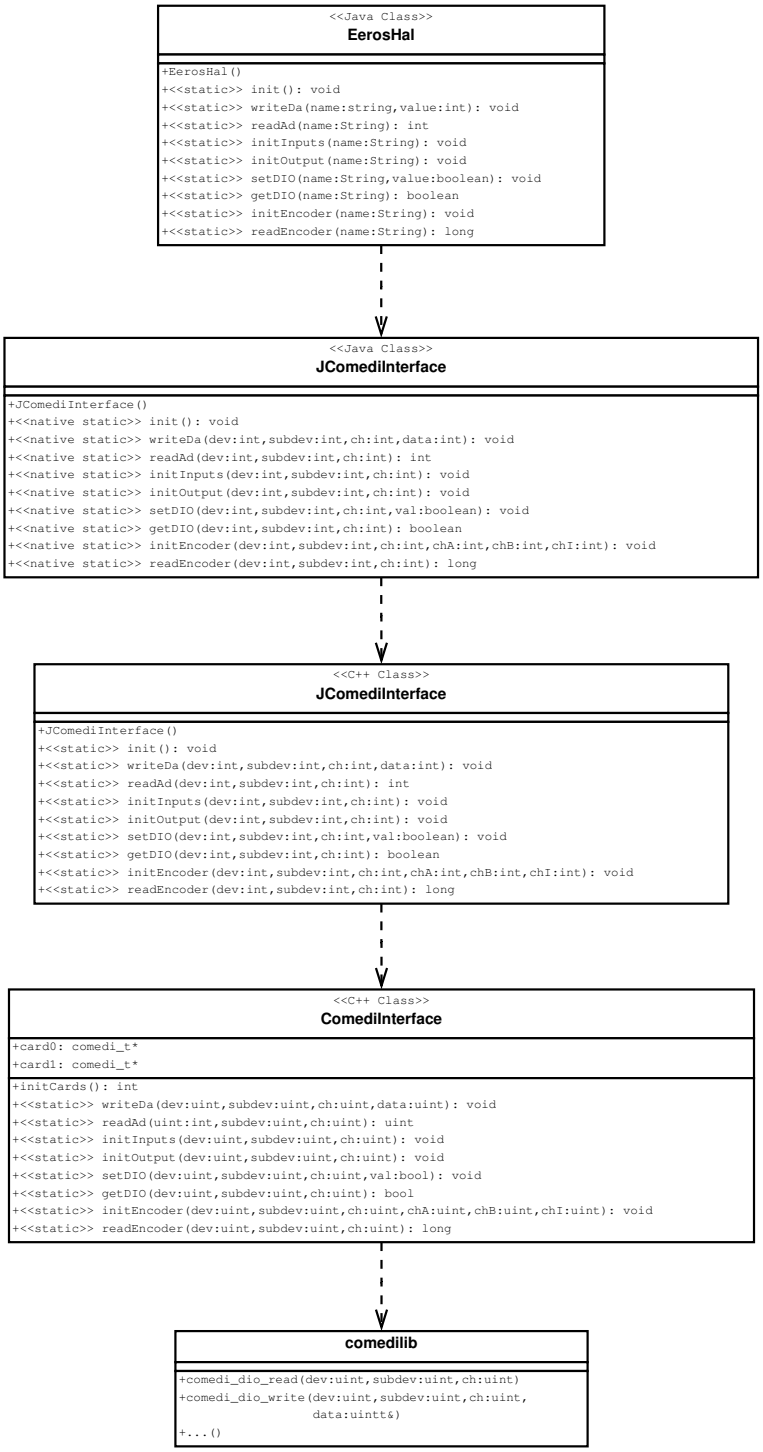

Abbildung 24: Hardware Interface

3.6.3. Build Vorgang

Abbildung 25 zeigt den Buildvorgang des Prototyps auf einem Linux System. Gestartet wird der Vorgang durch den Befehl make, welcher die Build Konfiguration aus dem Makefile liest. Zuerst werden die benötigten Java Bibliotheken, welche als jar-File vorhanden sind, einzeln zu .so-Files kompiliert. Danach werden alle .java, .cpp und .h Files zusammen kompiliert und direkt mit den vorher erstellten .so Bibliotheken sowie den benötigten System Bibliotheken zusammen gelinkt.

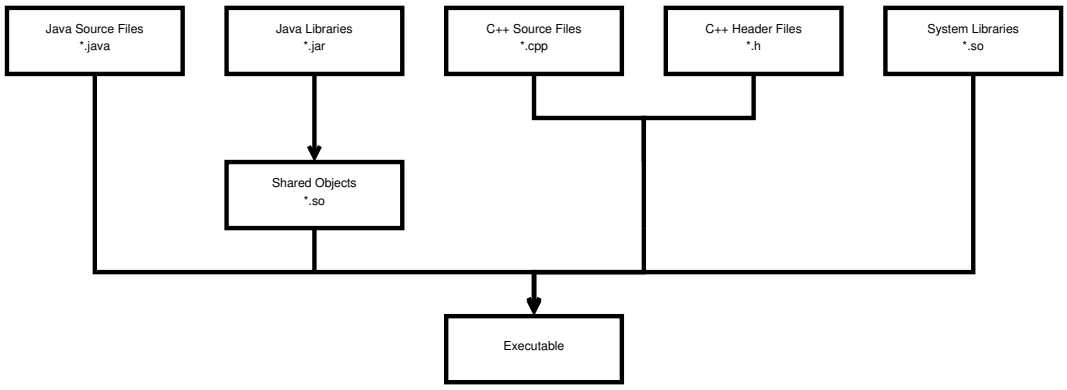

Abbildung 25: Build Vorgang

3.7. Anwendungsfälle

Um die Funktionalität vom EEROS Framework zu erproben und nachzuweisen, wurden zwei Robotikanwendungen erstellt.

3.7.1. SimpleController

Die Anwendung SimpleController ist die nahezu einfachst mögliche Anwendung, in der Regelungstechnik benötigt wird und nicht nur gesteuert wird.

Es wird ein Motor mit Encoder mittels einer Positionsregelung exakt dem Sollwert eines zweiten Encoders nachgeführt. Der Anwender kann somit den Encoder 2 bewegen, was zur exakt gleichen Bewegung beim Motor führt. Abbildung Abbildung 26 zeigt die Reglerstruktur. Ein Verdrahtungsplan und eine Auflistung der verwendeten Instrumente sind im Anhang zu finden.

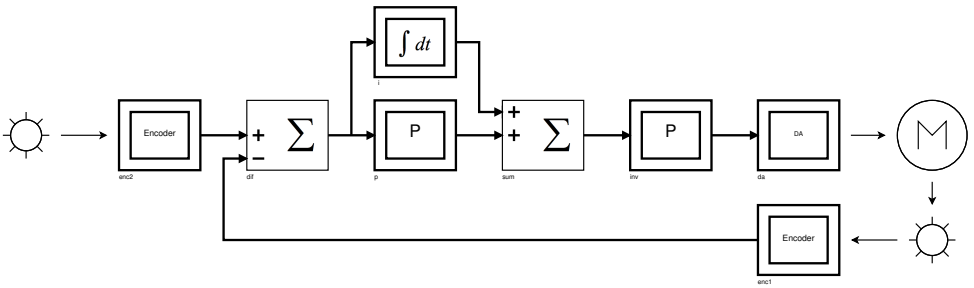

Abbildung 26: SimpleController

3.7.2. Stab balancierender Roboter

Die Anwendung Stab balancierender Roboter weist neben der Fähigkeit, Regeltechnik-Aufgaben auszuführen, zusätzlich nach, dass auch Robotik-Aufgaben durchgeführt werden können. Abbildung Abbildung 27 zeigt ein moderat vereinfachtes Diagramm der Reglerstruktur.

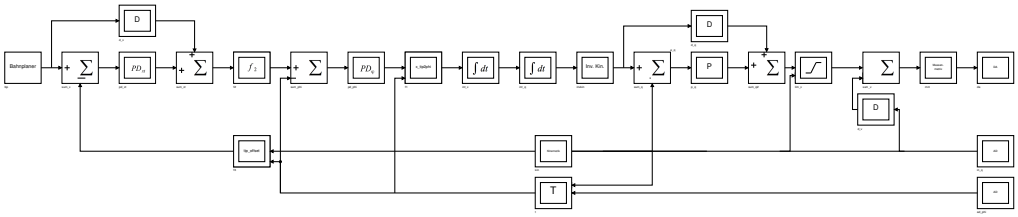

Abbildung 27: Stab balancierender Roboter

3.8. Hilfsprogramme

Damit in EEROS implementierte Robotikanwendungen effizient in Betrieb genommen werden können, sind rund um das Kern Framework noch einige zusätzliche Hilfsprogramme nötig oder zumindest hilfreich. Die nachfolgenden Unterkapitel listen die im Rahmen dieser Arbeit erstellten Hilfsprogramme.

3.8.1. HMI

Das HMI ermöglicht eine rudimentäre Steuerung des Roboters. Wird in der Konfigurationsdatei EerosConfig.xml das Feld Hmi auf true gesetzt, muss jede Transition von einem SafetyLevel oder einem SequencerStep auf der HMI freigegeben werden. Die HMI zeigt jeweils den aktuellen und den angeforderten Zustand an.

Abbildung 28: HMI

3.8.2. Scope

Das Hilfsprogramm Scope zeigt Signale innerhalb des Control Framework oder Ein- und Ausgangssignale wie ein klassisches Scope an.

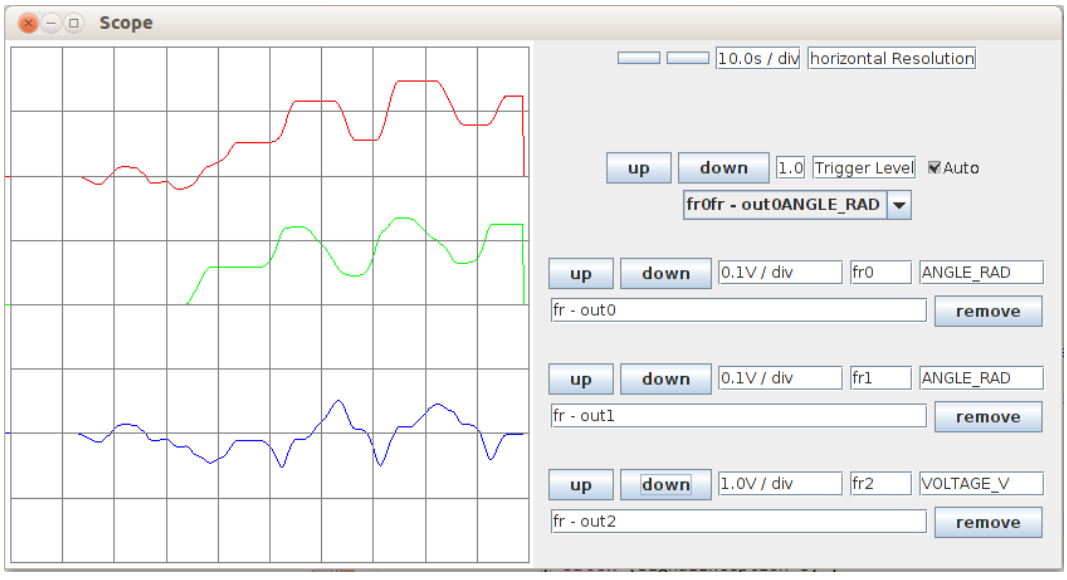

Abbildung 29: Scope

3.8.3. Controller Visualisierung

Für die visuelle Kontrolle der erstellten Reglerarchitektur sowie für das Debugging wurde eine simple Visualisierung implementiert. Die Abbildung ControllerVisualisation zeigt einen Ausschnitt einer Visualisierung.

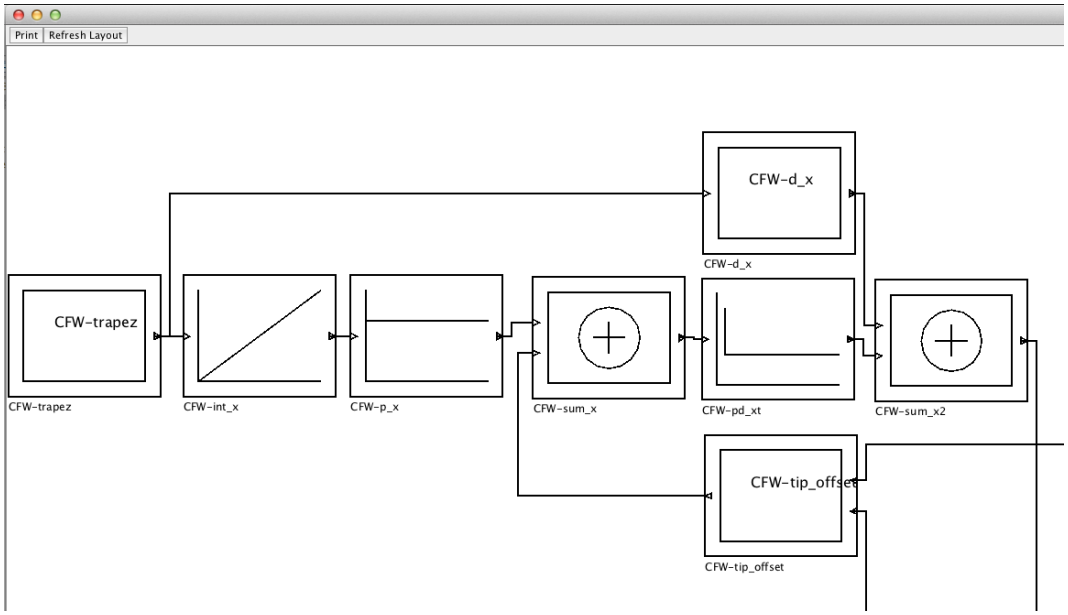

Abbildung 30: ControllerVisualisation

3.8.4. BarBalancingRobotVisualisation

Wenn kein Roboter zur Verfügung steht, muss der Roboter ebenfalls in Software simuliert werden. Damit die Reaktion des Roboters auf unterschiedliche Regelparameter sichtbar wird, wurde das Hilfsprogramm BarBalancingRobotVisualisation erstellt. Es simuliert die beiden Achsen des SCARA Roboters und den zu balancierenden Stab.

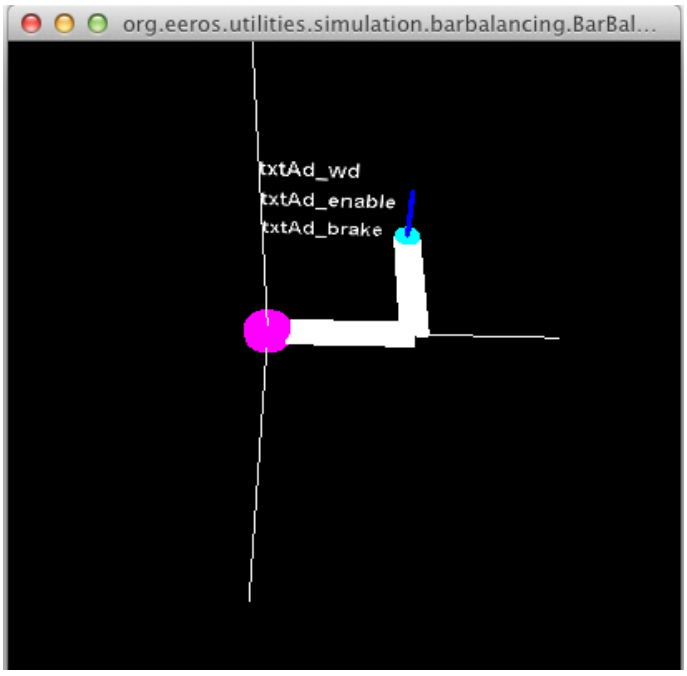

Abbildung 31: 3D Visualisierung

4. Offene Punkte und Kompromisse im Prototyp

Dieser Abschnitt widmet sich den bereits bekannten Kompromissen welche beim Prototyp eingegangen werden mussten. Auch offene Designentscheide und Nachteile der gewählten Implementierungen werden aufgelistet.

Ziel ist es, transparent über den Prototyp zu informieren. Das Kapitel soll als Input für das nachfolgende Designreview und den anschliessenden Designentscheid dienen.

4.1. Systemübersicht

Während der Implementierung des Prototyps sind zahlreiche Fragen aufgetreten, welche aufgrund des sonst schon beträchtlichen Umfangs der Arbeit offen geblieben sind. Einige davon betreffen das Systemdesign, andere die Zuverlässigkeit oder die Performance. Die folgende Aufzählung listet die offenen Fragen:

- Analyse der Auslastung des Systems oder einzelner Tasks. Zu diesem Zweck gibt es eine Reihe nützlicher Tools und Methoden [byt, Mic01].
- Alle Komponenten welche als Bibliothek für Benutzeranwendungen dienen werden, sollten mit Unit Tests geprüft werden.
- Wie lässt sich die Echtzeitfähigkeit eines Programmes nachweisen?

Die bei der Implementierung des Prototyps verwendeten Bibliotheken sind ein weiterer Punkt, welcher noch analysiert werden muss. Zu klären ist dabei, wie stabil und verlässlich die Bibliothek implementiert wurde und ob die benötigte Funktionalität nicht besser im Rahmen des EEROS Projektes implementiert werden sollte. Benutzt wurden folgende Bibliotheken:

- EJML — Bibliothek für Berechnungen von Matrizen.
  http://code.google.com/p/efficient-java-matrix-library/
  Wird im Control Framework benutzt (MatrixSignal).
- vecmath — Bibliothek Vektor Operationen.
  http://java.net/projects/vecmath
  Wird von EJML benutzt.
- piccolo2D — Bibliothek mit 2D GUI Funktionen.
  http://www.piccolo2d.org/
  Wird in ControllerVisualisation benutzt.
- Java3D — Bibliothek mit 3D GUI Funktionen.

http://java3d.java.net/
Wird in BarBalancingRobotVisualisation benutzt.

- JSONArray — Bibliothek Funktionen zum Umgang mit JSON Dateien.
  http://www.json.org/javadoc/org/json/JSONArray.html
  Wird in ControllerVisualisation benutzt.

4.2. Control Framework

4.2.1. Signale

- BundledSignal ist noch nicht implementiert.
- MatrixSignal beruht auf EJML. Es werden in jedem Schritt neue Objekte erzeugt.

4.2.2. Blocks

- Manche run() Methoden sind noch nicht implementiert.
- Vor einem Produktiveinsatz müssten alle Blöcke noch mit Unit Tests überprüft werden.
- Der Marker isValid wird von den meisten Blöcken noch nicht ausgewertet.
- Der aktuelle SafetyLevel wird von den meisten Blöcken noch nicht ausgewertet.

4.2.3. Scheduler

- Ob der verwendete Algorithmus bei allen Reglerstrukturen korrekt funktioniert muss anhand einer grösseren Anzahl Beispiele noch verifiziert werden.

4.2.4. TimeDomain

- Die Unterscheidung, ob Echtzeitfähigkeit in der betrachteten TimeDomain erforderlich ist, ist zurzeit noch nicht implementiert. Später

soll anhand dieser Unterscheidung entschieden werden, welche Art von Executor gewählt wird.

4.3. Sequencer Framework

Die Erstellung von SequencerSteps sollte vereinfacht werden. Dazu ist eine Kapselung nötig.

Wechsel zwischen den verschiedenen Sequencern (bspw. Wechsel der Regelcharakteristik):

- Alle Blöcke bei welchen der Eingang umschaltbar sein muss, besitzen statt eines Inputs einen SwitchableInput.
- Ein Sequencer kann einen weiteren Sequencer starten.
- Der neu gestartete Sequencer baut die neue Regelcharakteristik auf und startet diese.
- Sobald die neue Regelcharakteristik aufgebaut ist und gültige Werte liefert, wird der SwitchableInput umgeschaltet.

4.4. Error Handling Framework

Eine Kapselung würde die Implementierung von neuen ErrorHandlern vereinfachen.

4.5. Safety Framework

- Überwachung von TimeDomains
  - welche Möglichkeiten bestehen für einen Datenaustausch?
- Bei Überarbeitung folgende Idee prüfen:
  - SafetySystem ist gegeben, man muss es erweitern, um die eigene Funktionalität einzubringen. So kann nichts vergessen werden.
- Der SafetyTask besteht zurzeit nur aus dem Aufruf des SafetyBlocks. Für die Überwachung von zusätzlichen Werten müsste ein Mechanismus zur Verhinderung von Blockierungen eingebaut werden.

4.6. Native Interface

- Der Aufruf von nativen Echtzeit Threads funktioniert aktuell noch nicht. Dies hat mit dem nötigen Memory Lock zu tun. In der Prototyp-Implementierung ist die Memory-Allokation innerhalb des Threads nötig. Somit funktioniert eine Lösung, welche einen Memory Lock erfordert, prinzipiell nicht.

- Shared Memory für den Datenaustausch zwischen unterschiedlichen TimeDomains muss noch implementiert werden.

4.7. Anwendungsfälle

Die Anwendung SimpleController funktioniert einwandfrei. Sie wurde dokumentiert und dem Team präsentiert. Zum Zeitpunkt der Abgabe dieser Dokumentation ist die Inbetriebnahme des Stab balancierenden Roboters noch in Arbeit. Die Positionsregelung auf fest vorgegebene Punkte im Raum funktioniert einwandfrei. Das Balancieren des Pendels wurde bis jetzt jedoch erst in der Simulation erfolgreich getestet.

4.8. Hilfsprogramme

Für ein effizientes Programmieren und Benutzen des Frameworks werden vermutlich noch mehrere Hilfsprogramme benötigt werden. Folgendes Programm ist bereits angedacht:

- Hardware Zuweisungs-Tool

  - Der Benutzer soll nach der Installation einer Hardware-Schnittstelle eine Software starten können, in welcher er die neu verfügbaren Ressourcen benennt. Diese könnten dann aus dem EEROS Framework direkt ansprechbar implementiert werden.

5. Ausblick

Die hier vorliegende Arbeit ist wie vorgängig erwähnt nur ein kleiner Teil des Gesamtprojekts EEROS. Im Rahmen dieser Arbeit konnte gezeigt werden, dass sich die Anforderungen und Wünsche der Projektleitung realisieren lassen. Um das Ziel zu erreichen, sind aber noch viele Schritte notwendig. Die aktuelle Implementierung ist erst ein Prototyp. Er soll nun durch das Projektteam eingängig auseinander genommen werden und jeder Design-Entscheid nochmals hinterfragt werden. Dabei wird die Liste mit den bereits bekannten offenen Punkten vom vorherigen Kapitel nochmals ergänzt. Danach folgt die Implementierung der ersten zu veröffentlichenden Version. Ich hoffe natürlich, dass es ein wesentlicher Teil des Prototyps und insbesonders des Control Frameworks in dieses Produkt schaffen wird. Neben der reinen Software Implementierung werden die Arbeiten in Beispielanwendungen einen nicht unwesentlichen Anteil der investierten Arbeiten einnehmen. Erst durch die Implementierung und Dokumentation von Anwendungen werden die Möglichkeiten des Frameworks transparent. Dies ist ein wichtiger Schritt zu einer grösseren Nutzerbasis. Zuletzt darf auch das Marketing und die Vernetzung mit den entsprechenden Interessensgruppen nicht vernachlässigt werden. Erst wenn EEROS in Produkten der Industrie im Einsatz ist, wird die Zielsetzung vollständig erreicht sein.

6. Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich während der Masterarbeit fachlich und persönlich unterstützt haben.

Es ist mir bewusst, dass eine Aufzählung aller Personen welche eine persönliche Honorierung verdient hätten, hier nicht möglich ist. Dennoch möchte ich hier einige speziell erwähnen:

Einen speziellen Dank gebührt dem EEROS Projektteam, welches sich aus Prof. Dr. Urs Graf, Prof. Einar Nielsen, B.Sc Martin Züger sowie Ing. Claudia Visentin zusammensetzt.

Ebenfalls bedanken möchte ich mich bei meinem Arbeitgeber welcher trotz der vielen Abwesenheiten stets positiv zu meiner Weiterbildung gestanden ist.

Schlussendlich will ich nicht meine Partnerin und meine Familienangehörigen vergessen, welche mir viel Verständnis und Unterstützung während der ganzen Dauer des Studiums entgegengebracht haben.

Ehrenwörtliche Versicherung

Hiermit bestätige ich ehrenwörtliche, die vorliegende Arbeit ausschliesslich unter Verwendung der aufgeführten Quellen selbständig ausgeführt zu haben.

Horn, 31.01.2013

Julian Specker

7. Verzeichnisse

Literatur

[AB06] Abel, D. ; Bollig, A.: *Rapid Control Prototyping: Methoden und Anwendungen*. Springer, 2006 http://books.google.ch/books?id=JmGN0ckbENkC. – ISBN 9783540295242

[Abb] Abbott, David Schleef; Frank Mori Hess; Herman Bruyninckx; Bernd Porr; I.: *The Control and Measurement Device Interface handbook for Comedilib 0.10.1*. http://www.comedi.org/doc/index.html

[Ber04] Berger, Benjamin: *Realisierung einer prototypischen Hardwarelösung für ein inverses Pendel*, TU Chemnitz, Diplomarbeit, 2004

[byt] http://asm.ow2.org/eclipse/index.html

[Hag06] Hagen, W. von: *The Definitive Guide to GCC*. Apress, 2006 (Definitive Guide Series). http://books.google.ch/books?id=wQ6r3UTivJgC. – ISBN 9781590595855

[Hau07] Haun, M.: *Handbuch Robotik: Programmieren und Einsatz intelligenter Roboter*. Springer, 2007 (VDI-Buch). http://books.google.ch/books?id=-hn7Vgz5XGEC. – ISBN 9783540255086

[Her03] Herold, H.: *Make*. Addison-Wesley, 2003 (Open source library). http://books.google.ch/books?id=kgiw5OlEV3sC. – ISBN 9783827320957

[Mar09] Martin, R.C.: *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2009 (Robert C. Martin Series). http://books.google.ch/books?id=dwSfGQAACAAJ. – ISBN 9780132350884

[Mic01] http://192.9.162.55/docs/books/performance/1st_edition/html/JPMeasurement.fm.html

[Mö03] Mössenböck, H.: *Sprechen Sie Java? Eine Einführung in das systematische Programmieren*. Dpunkt.Verlag GmbH, 2003 (Dpunkt-Lehrbuch). http://books.google.ch/books?id=jMvpAAAACAAJ. – ISBN 9783898642316

[ntb] *NTB Programmierrichtlinien fuer Java*. http://wiki.ntb.ch/infoportal/software:programmierrichtlinien:java

[rtW] *Real-Time Linux Wiki*. https://rt.wiki.kernel.org/index.

php/Main_Page

[SK08] Siciliano, B. ; Khatib, O.: *Springer Handbook of Robotics*. Springer, 2008 (Gale virtual reference library). http://books.google.ch/books?id=Xpgi5gSuBxsC. – ISBN 9783540239574

[Sta09] Stark, G.: *Robotik mit MATLAB*. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2009 (Lehrbücher zur Informatik). http://books.google.ch/books?id=ATkl_GxtD6wC. – ISBN 9783446419629

[Tro] Tromey, Tom ; Free Software Foundation (Hrsg.): *GNU gcj*. Free Software Foundation

[wik12a] ; Wikipedia (Veranst.): *Remote Method Invocation*. http://de.wikipedia.org/wiki/Remote_Method_Invocation. Version: Mai 2012

[wik12b] ; Wikipedia (Veranst.): *Socket*. http://de.wikipedia.org/wiki/Socket_(Software). Version: Juli 2012

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

KDE K Desktop Environment

SQL Structured Query Language

Bash Bourne-again shell

NRT Non Realtime

RT Realtime

SW-RT Software Realtime, auch Pseudo Realtime

PRT Pseudo Realtime, auch Software Realtime

ppcfw Path planning and Control FW

ppc Path planning and Control

FW frame work

sfw Safety frame work

ehfw Error handling frame work

eh Error handling

A. Anhang

A.1. Definitionen

Um eindeutige Begriffe zu verwenden, hier eine Liste mit Definitionen:

Bahnplaner	Der inkrementelle Bahnplaner berechnet den Weg vom aktuellen Standort des Roboters zu Ziel.
Exception Handling FW	Ist zuständig für die Abhandlung von nicht vorgesehenen Situationen.
Non Realtime	Alle Komponenten des Frameworks welche nicht zu einer garantierten Zeit ausgeführt werden müssen.
Control FW	Beinhaltet die eigentliche Regelungstechnik innerhalb des EEROS Frameworks. Alle RT Tasks sind Teil des Control Frameworks.
Periodischer Task	Ein Task welcher einer strikten Zeitbedingung unterstellt ist.
Scheduler	Der Scheduler ist ein Teil des Path planning and Control Frameworks und ist für die korrekte Reihenfolge der Run Methoden der einzelnen Blöcke zuständig.
Sequencer	Der Sequencer ist einer der vier Hauptteile von EEROS . Er bestimmt den korrekten Ablauf der Aktionen. Dazu bildet er den für die aktuelle Aufgabe korrekten Regler innerhalb des Path planning and Control Frameworks und setzt diesen unter Einhaltung von Übergaberegeln aktiv.
Umhängen	Ein neuer Block wird eingefügt und neu verdrahtet. Ist nur im Sequencer möglich und somit kein RT Task.
Umschalten	Beispielsweise das Umschalten eines Signals bei einem Umschaltblock. Ist innerhalb des Path planning and Control FW möglich und somit RT fähig.

A.1.1. Arten von Aufgaben, Threads, Tasks etc.

Englisch	**Deutsch**	**Beschrieb**
Sequence	Sequenz	Eine Gruppe von Sequenzen und Steps. In Sequenzen können Regler aufgesetzt werden, basierend auf Ereignissen Aktionen ausgeführt und sonstige Robotik-Aufgaben gelöst werden.

Step	Schritt	Die kleinste Einheit welche dem Sequencer bekannt ist.
Process	Prozess	Ein Prozess besteht aus einem oder mehreren Threads. Jeder Prozess hat eigene zugeteilte Ressourcen.
Task	Task	Windows Entsprechung zum Prozess bei Unix. Begriff wird in EEROS nicht verwendet.
Thread	Ausführungsfaden	Bestehen innerhalb eines Prozesses mehrere Ausführungsfäden, teilen sich diese die Ressourcen. Jeder Ausführungsfaden hat jedoch einen eigenen Stack. Ein Thread startet, kann unterbrochen werden und endet nach der kompletten Ausführung seines Codes.
PeriodicalExecutor	-	Ein PeriodicalExecutor startet periodisch einen Runnable.
Runnable	-	Das Interface Runnable wird zur Definition eines Ablaufs implementiert. Ein Runnable Objekt kann einem PeriodicalExecutor oder einem Thread übergeben werden.

A.1.2. Signale

Kartesische Koordinaten $\underline{x} = \begin{pmatrix} x \\ y \\ z \\ \alpha \end{pmatrix}$

Polarkordinaten $\phi = \Phi = \begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \end{pmatrix}$

verallgemeinerte Kraft $Q = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix}$

A.2. Software Toolchain

A.2.1. Entwicklung

Zweck	Software	Windows	Linux	OSX
IDE	Eclipse Indigo for Java Developers	X	X	X
Visualisierung	Piccolo2D. Java	X	X	X

A.2.2. Dokumentation

Zweck	Software	Windows	Linux	OSX
Dokumentation allg.	LibreOffice 3.5.1	X	X	X
Prozesse / Strukturen visualisieren	Dia 0.97.2	X	X	X
LibreOffice Dokumente zu Latex konvertieren	Writer2latex 1.2	X	X	X
Latex Editor	TexMaker 3.3.2	X	X	X
SVG to Latex	Inkscape 0.48.2	X	X	X
Latex Distribution	TeX Live	X	X	
Latex Distribution	MacTex 2011			X

A.2.3. Hilfsmittel

Zweck	Software	Windows	Linux	OSX
Versionskontrolle	TortoiseSVN 1.7.6	X		
Versionskontrolle	RabbitVCS 0.15.0.5		X	
Versionskontrolle	SmartSVN			X

A.2.4. Make

Make ist ein Tool zur automatischen Generierung von Programmen. Da es für die Erstellung des EEROS Frameworks eine hohe Wichtigkeit besitzt, werden hier die wichtigsten Befehle dokumentiert. Die Befehle stammen aus Herold, H. Make [Her03].

- Kommentare werden mit # eingeleitet
- Eine Zeile kann mit \ fortgesetzt werden

- Ein Eintrag besteht aus einer Abhängigkeitsbeschreibung mit Kommandos
  - Abhängigkeitsbeschreibung darf nicht mit Tabulatorzeichen beginnen
  - Kommandozeilen sind mit einem Tabulatorzeichen einzurücken
  - Gültig ist auch: Abhängigkeitsbeschreibung ; Kommandozeile
- Make überprüft die Abhängigkeiten anhand des Zeitstempels
- Wird make ohne Argumente aufgerufen, führt er den ersten Befehl aus (und alle Abhängigkeiten)
- Argument -n: nur Simulation
- Argument -s: Silent
- Makronamen = String
- Makronamen sind als Konvention GROSS geschrieben und dürfen auch Zahlen und Underscores enthalten
- Zugriff auf Makros: ${Makroname} oder $(Makroname); empfohlen {}

A.3. SimpleController

Attribute	Description
PC DAQ	NI PCIe-6251
Motor Drive	Maxon ADS 50/5
Motor with Encoder	Maxon 134952 + HEDL-5540
Encoder	HEDL-5540
DC Power Supply	+15V DC, 1A

Tabelle 2: SimpleController: benutzte Hardware

From (Source)	To (Sink)
NI PCIe-6251 Pin 22 (AO 0)	ADS 50/5 Signal Pin 1 (Set + val.)
NI PCIe-6251 Pin 56 (AO GND)	ADS 50/5 Signal Pin 2 (Set - val.)
NI PCIe-6251 Pin 56 (AO GND)	ADS 50/5 Signal Pin 4 (GND)
DC Power Supply +	ADS 50/5 Power Pin 4 (Volt +)
DC Power Supply -	ADS 50/5 Power Pin 5 (Pwr. GND)
ADS 50/5 Power Pin 1 (Motor +)	Maxon 134952 + HEDL-5540 +
ADS 50/5 Power Pin 1 (Motor -)	Maxon 134952 + HEDL-5540 -
NI PCIe-6251 Pin 8 (+ 5V)	Maxon 134952 + HEDL-5540 VCC
NI PCIe-6251 Pin 8 (+ 5V)	ADS 50/5 Signal Pin 3 (Enable)
NI PCIe-6251 Pin 7 (DGND)	Maxon 134952 + HEDL-5540 GND
Maxon 134952 + HEDL-5540 A	NI PCIe-6251 Pin 37 (PFI 8)
Maxon 134952 + HEDL-5540 B	NI PCIe-6251 Pin 45 (PFI 10)
Maxon 134952 + HEDL-5540 I	NI PCIe-6251 Pin 3 (PFI 9)
NI PCIe-6251 Pin 8 (+ 5V)	HEDL-5540 VCC
NI PCIe-6251 Pin 7 (DGND)	HEDL-5540 GND
HEDL-5540 A	NI PCIe-6251 Pin 42 (PFI 3)
HEDL-5540 B	NI PCIe-6251 Pin 46 (PFI 11)
HEDL-5540 I	NI PCIe-6251 Pin 41 (PFI 4)

Tabelle 3: SimpleController: Verdrahtungsplan