Requirements and Design Documentation (RDD)

Version 0.4

ESEP – Praktikum – Sommersemester 2017

LANKE

Hartmann	Lennart	2236791	Lennart.Hartmann@haw-hamburg.de
Mendel	Alexander	2188808	Alexander.Mendel@haw-hamburg.de
Eggebrecht	Nils	2247014	Nils.Eggebrecht@haw-hamburg.de
Witte	Karl-Fabian	2246435	Karl-Fabian.Witte@haw-hamburg.de
Veit	Eduard	2227951	Eduard.Veit@haw-hamburg.de

Hamburg, den 2. Juli 2017

Änderungshistorie:

Version	Autor	Datum	Anmerkungen/Änderungen
0.1	K.Witte	04.04.2017	Aus der Vorlage (Version 0.5) von Prof Lehmann doc2tex, um es in Git besser pflegen zu können
0.1.1	K.Witte	11.04.2017	Es wurden Tabellenvorlagen für die Requirements und Use Cases hinzugefügt (Wave und Kite lvl)
0.1.2	A. Mendel	10.05.2017	Komponentendiagramm der Architektur und Beschreibung
0.1.3	A. Mendel	11.05.2017	Use Case Aktivitätsdiagramme eingepflegt
0.2	N. Eggebrecht	01.07.2017	Relogger (Nachtrag von einem anderen)
0.2.1	A. Mendel	30.06.2017	Diverses (Nachtrag von einem anderen)
0.3	L. Hartmann	2.07.2017	Finale Steuerung beschrieben
0.4	K.Witte	02.07.2017	ProfileDetection, Serial, Layer, new Architecture

Inhaltsverzeichnis

1 Teamorganisation

Teamorganisation:

Die Teamorganisation hat sich im Laufe des Projekts stets dem Entwicklungsfortschritt angepasst. Am Anfang des Projekts stand jede Woche ein fester Termin für ein Treffen mit schriftlichem Protokoll fest. Dabei wurden wichtige Punkte für die Architekturplanung und auch schon Designüberlegungen festgehalten.
Ab etwa der Hälfte des Projekts gab es keine festen Termine und keine festen Protokolle mehr, die Organisation fand hauptsächlich mit einem Kanban-Bord statt und sowieso ständig in direkter Zusammenarbeit.

1.1 Verantwortlichkeiten

Mitglied	**Rolle**	**Aufgaben**
Lennart Hartmann	Architekt (Scrum-Master)	Der Architekt gibt in der Architekturplanung Richtlinien vor und hält über den agilen Entwicklungsprozess fest, was sich an der Architektur verändert hat.
Alexander Mendel	Dokumentator	Der Dokumentator kümmert sich um die sorgfältigkeit aller schriftlichen Ausarbeitungen, die dem Kunden vorliegen.
Nils Eggebrecht	Product-Owner	Der Product-Owner legt während des Entwicklungsprozesses fest, in welcher Reihenfolge die Aufgaben bearbeitet werden und wie die Priorisierung ist.
Karl-Fabian Witte	Hauptentwickler	Der Hauptentwickler entwickelt in Zusammenarbeit mit dem Architekten das Softwaredesign.
Eduard Veit	Qualitätsmanager	Der Qualitätsmanager ist für die Einhaltung der Coding-Styles verantwortlich.

1.2 Absprachen

Der Architekt, Herr Hartmann, war im Laufe des Projekts der Hauptansprechpartner für alle Designfragen und Designentscheidungen. Für die jeweiligen Teilgebiete die von anderen umgesetzt wurden, waren entsprechend die Unterteams oder Einzelpersonen verantwortlich. Die Kommunikation lief entweder mündlich in den Räumlichkeiten der Hochschule ab oder sonst intensiv über einen Mobile-Messanger. Für sonstige Kommentare stand eine Kanban-Plattform zur Verfügung.

1.3 Repository-Konzept

Das Repository ist in vier Hauptordner aufgeteilt:

CODE Hier landet der gesamte Sourcecode. Für jede Architekturschicht gibt es mindestens einen Subordner mit den Untermodulen.

DIAGRAM In diesem Ordner sind alle *Architektur-*, *Design-* und *Use-Case*-Diagramme.

DOC In diesem Ordner sind alle Dokumentationen rund um das Projekt, wie *User-Stories*, *How-To's*, *Doxygen* und *RDD* zu finden.

PROTOKOLL In diesem Ordner sind alle sonstigen *Dokumente-*, *Protokolle-* und *Tabellen*.

Aufteilung der Branches Auf dem `master`-Branch wird zu jedem Termin ein funktionierender Stand des Projektes entsprechend der User-Story gepusht. Es gibt einen `develop`-Branch auf den der aktuellste Entwicklungsstand aller `feature`-Branches gemerged wird, sobald diese, entsprechend der aktuellen Aufgabe lauffähig sind.

Commit-messages Die Commitnachrichten sind weitesgehend nach einer im Dokument `how2git.pdf` (im Repositoryordner "DOC\HOW_TO\" zu finden) vorgegebenen Syntax verfasst. (Dieses Dokument befindet sich im Anhang)

2 Projektmanagement

Es wurde das `Scrum`-Verfahren der **Agilen Softwareentwicklung** als Softwareentwicklungsvorgehensmodell festegelegt. Eine genaue Analyse der Aufgabenstellung fand im Laufe des Prozesses statt. Das Team war noch nicht mit der Arbeitsumgebung und der Arbeitsweise vertraut.

2.1 Prozess

Ab dem Quality-Gate – Termin wurden Arbeitspakete, die aus einer User-Story zum nächsten Termin formuliert wurden, als `Sprints` auf einem *Kanban-Bord* festgehalten.

2.2 PSP/Zeitplan/Tracking

Die in das Projekt investierten Arbeitszeiten wurden auf einem *Cloud-Spreadsheet* festgehalten.
Auf dem Online-Kanbanbord wurden Arbeitspakete in Listen eingetragen:

Ideen Hier werden allgemeinnützige Informationen festgehalten, die grundsätzlich für das gesamte Projekt von Relevanz waren.
Es ist eine voraussichtliche Zeiteinschätzung für restliche Termine und das Gesamtprojekt hinterlegt, sowie die etwaige Arbeitszeit pro Woche.
Außerdem ist beschrieben was die festgelegten farblichen Labels für die Arbeitspakete bedeutet.

Backlog In diese Liste werden neue Arbeitspakete eingetragen, die bis zum nächsten Sprint laut *User-Story* bearbeitet werden sollen.

Analyse In dieser Liste befinden sich die Arbeitspakete aus dem Backlog und werden analyisert und es wird zu jedem Paket eine `definition of done` formuliert, die z. B. von einem bereits am Modul arbeitenden Teammitglied geprüft und ggf. ergänzt wird.

Realisation In diese Liste wird ein Arbeitspaket verschoben, wenn es in der Analyse war. Die bearbeitenden Mitglieder tragen sich für das Arbeitspaket ein. Ist die gesamte Aufgabe oder ein Teil der Aufgabe erledigt wird ein kurzer Kommentar zu dem Arbeitspaket veröffentlicht, der umgangssprachlicher als eine `Commit-Message` beschreibt, was geschafft wurde. Auch wenn neue Designentscheidungen getroffen wurden, wird zum Arbeitspaket ein Kommentar hinzugefügt, so dass andere Teammitglieder, die diese Designentscheidungen betrifft informiert sind.

Review Ist ein Arbeitspaket nach der `definition of done` fertiggestellt, landet es in der Review-Liste. Hier segnet der Product Owner oder ein anderes von ihm delegiertes Teammitglied das Arbeitspaket ab oder es wird korrigiert, falls in der Durchsicht Fehler gefunden wurden.

Fertig Ist das Arbeitspaket erfolgreich geprüft worden, wird es in die letzte Liste verschoben und ist somit abgeschlossen.

2.3 Qualitätssicherung

Grundsätzliche Festlegungen zu Code-Conventions sind im Dokument `coding_style` festgehalten (unter “DOC\HOW_TO\”). (siehe Anhang) Es wurde soweit wie möglich zu zweit an einem Arbeitspaket gearbeitet, also *pair programming* betrieben, um Flüchtigkeitsfehler zu vermeiden.

3 Randbedingungen

3.1 Entwicklungsumgebung

Als Entwicklungsumgebung wurde im Labor unter Windows 7 mit QNX-Momentics gearbeitet. “Zuhause” bzw. auf dem eigenen Rechner auch auf anderen Betriebssystemen und mit anderen Entwicklungsumgebungen.

3.2 Werkzeuge

QNX Momentics V7/V6.6

Codeblocks 16

CLion 2017

NetBeans V8.2

Dia 0.9

Magic Draw 18.5

3.3 Sprachen

C

C++

4 Requirements und Use Cases

4.1 Systemebene

Die Use-Case – Tabelle befindet sich im Anhang.

4.1.1 Stakeholder

Die Stakeholder dieses Projektes sind in den zwei Kategorien intern und extern unterteilt. In Tabelle 2 sind diese mit ihren Interessen aufgelistet.

4.1.2 Use Cases

Es sind im folgenden Use-Case – Aktivitätsdiagramme dargestellt, die den Ablauf verdeutlichen. (Abbildung 1 -8)

Tabelle 2: interne und externe Stakeholder des Projekts

interne Stakeholder	**Interessen**
CEO (Management)	- Gewinn - Rufaufwertung der Firma - effiziente und flexible Arbeiter - Einhaltung des Zeitplans - transparenter Einblick in den Entwicklungsprozess
Developer Team	- motivierende und sinnvolle Arbeit - Gehalt - gutes Teamklima
externe Stakeholder	**Interessen**
Kunde	- Projekt hat geforderte Funktion - Projekt hat gewünscht Qualität - geringe Kosten - schnelles Ergebnis - Regelmäßige Kontrolle des Projekts (Zwischenstand) - Einflussnahme im Projekt, neue Funktionen fordern
Anwender	- einfache Bedienung - hohe Sicherheitsstandards im Betrieb
Wartungsservicekraft	- einfache Wartung - geringe Fehleranfälligkeit - hohe Sicherheitsstandards bei Wartung

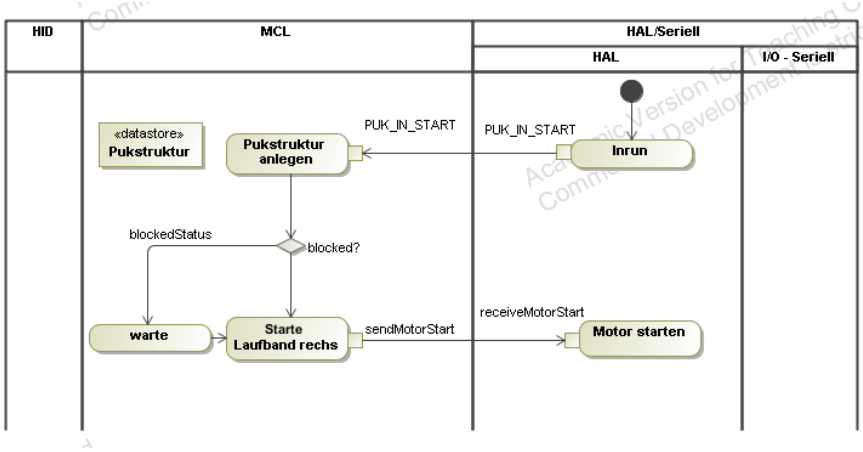


Abbildung 1: Use Case: Puk im Einlauf

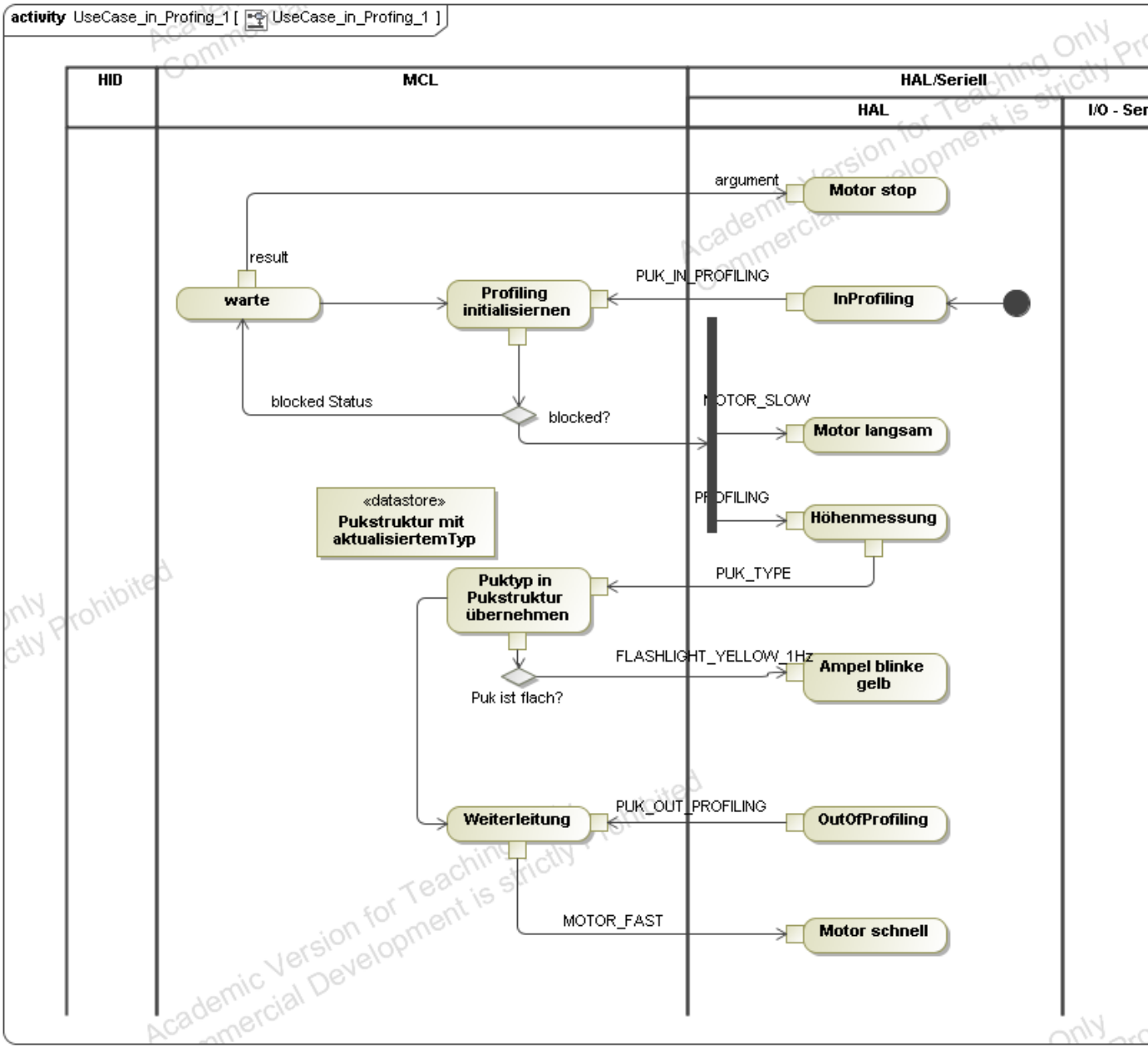


Abbildung 2: Use Case: Puk in Höhenmessung (Profiling)

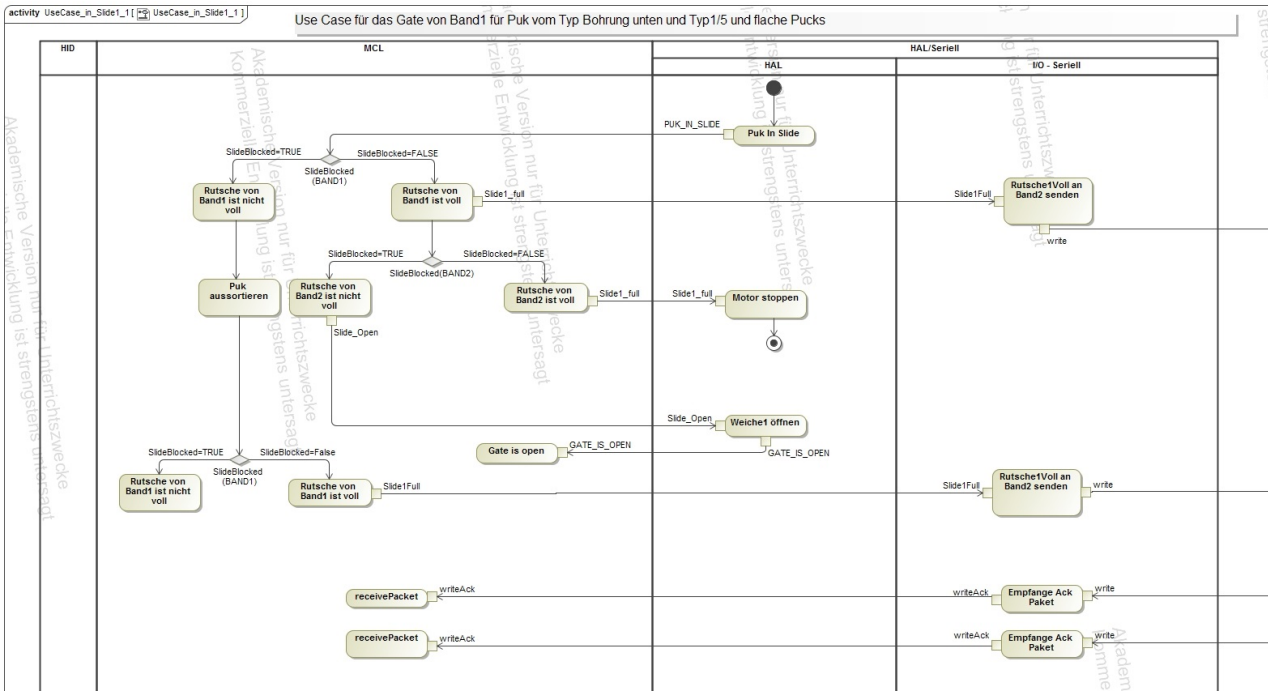


Abbildung 3: Use Case: Puk (Flach/Typ1/Typ5) in Weiche (Band1)

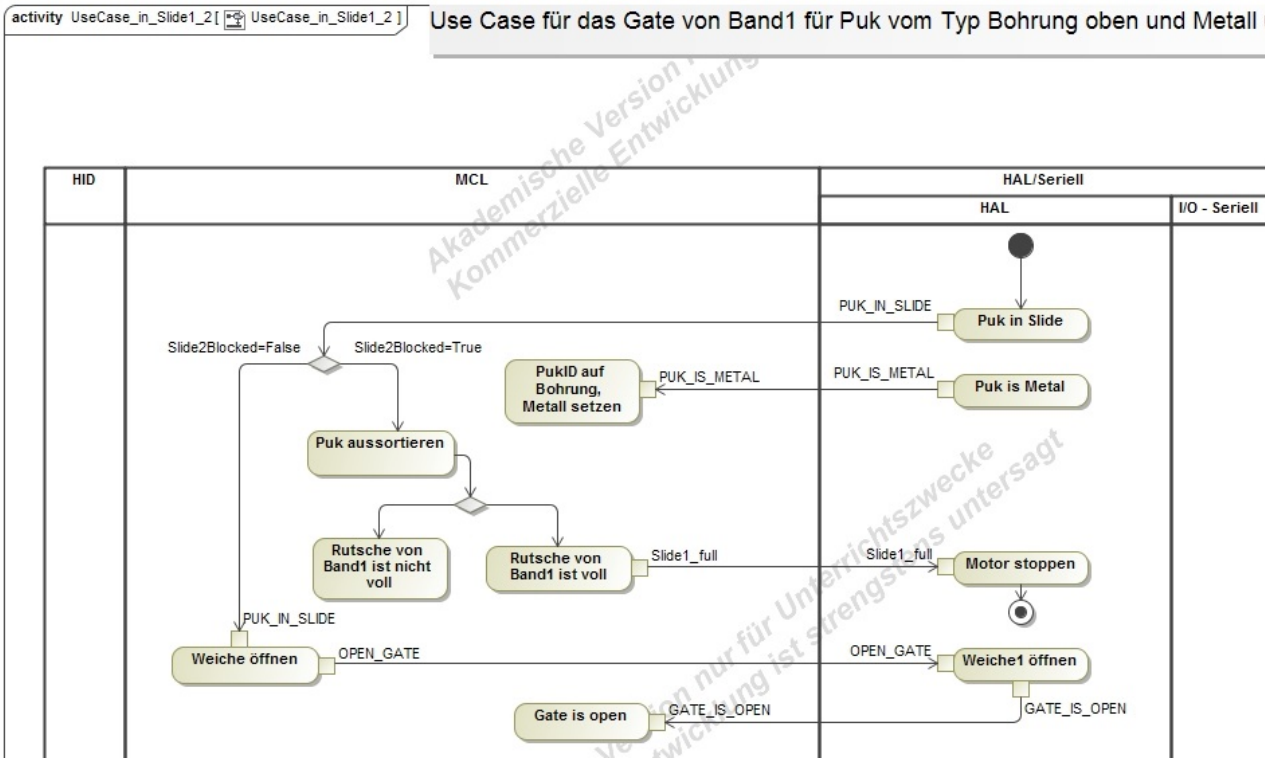


Abbildung 4: Use Case: Puk (Bohrung/Metall/Typ2/Typ4) in Weiche (Band1)

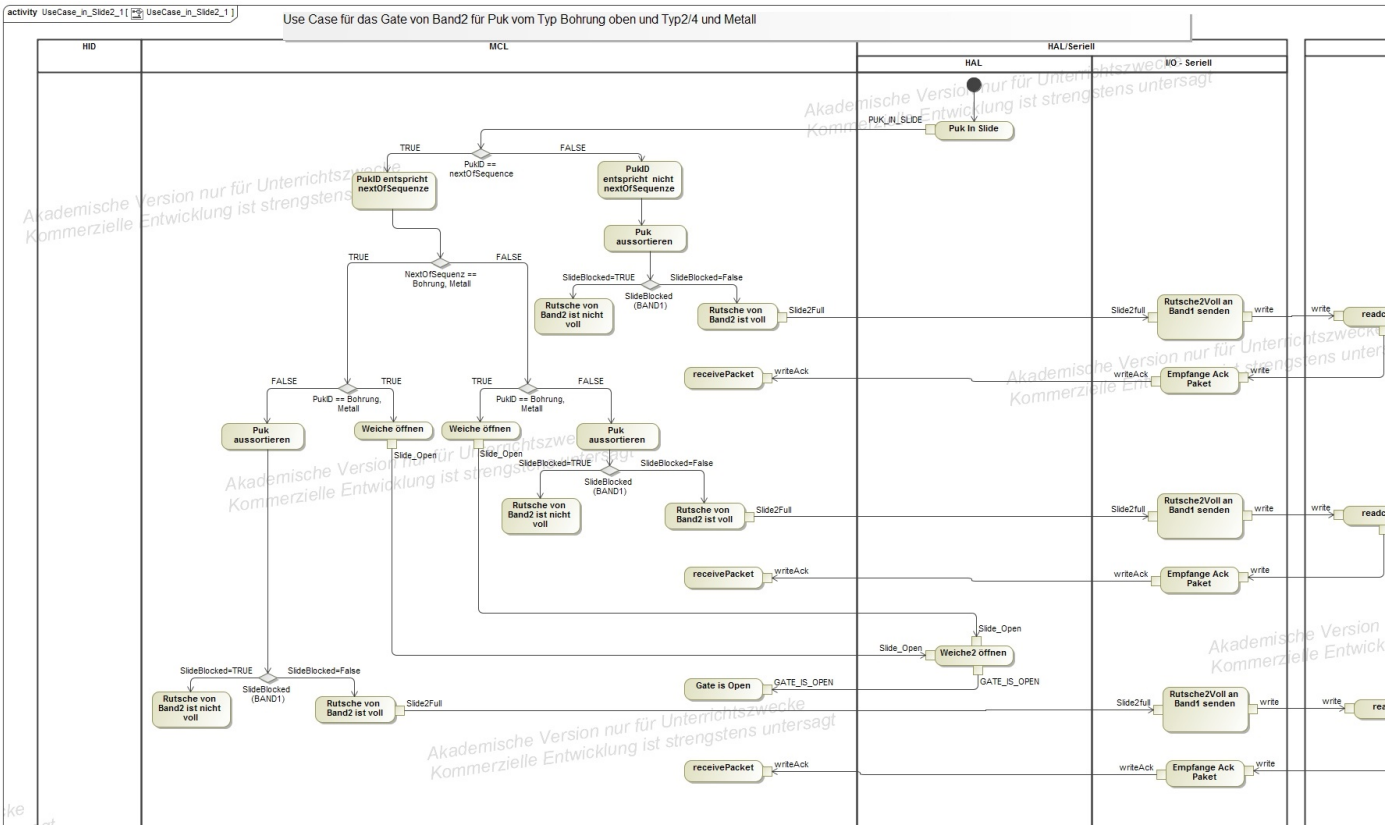


Abbildung 5: Use Case: Puk (Bohrung/Metall/Typ2/Typ4) in Weiche (Band2)

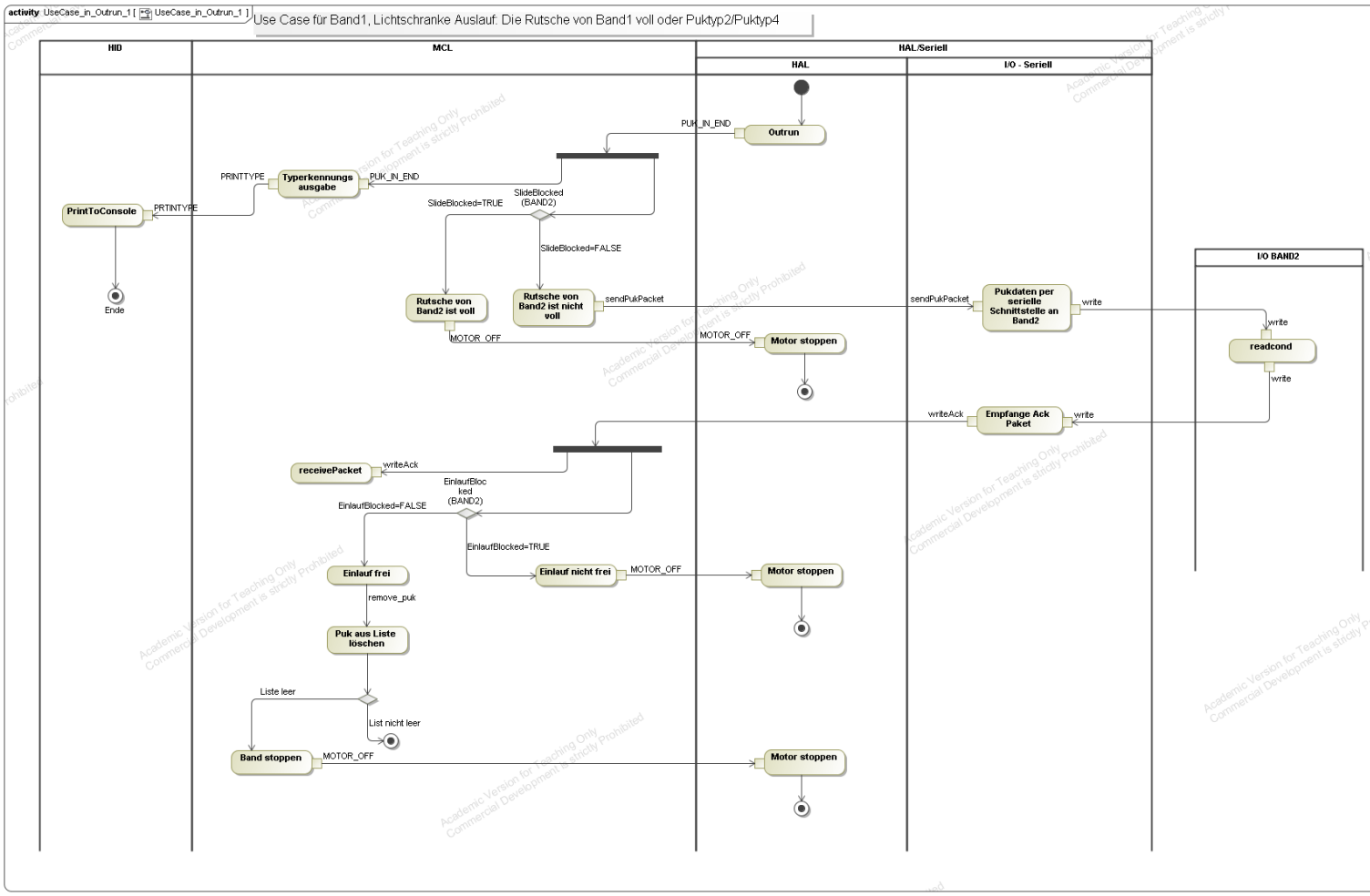


Abbildung 6: Use Case: Rutsche von Band1 voll oder Puk Typ2/Typ4 im Auslauf (Band1)

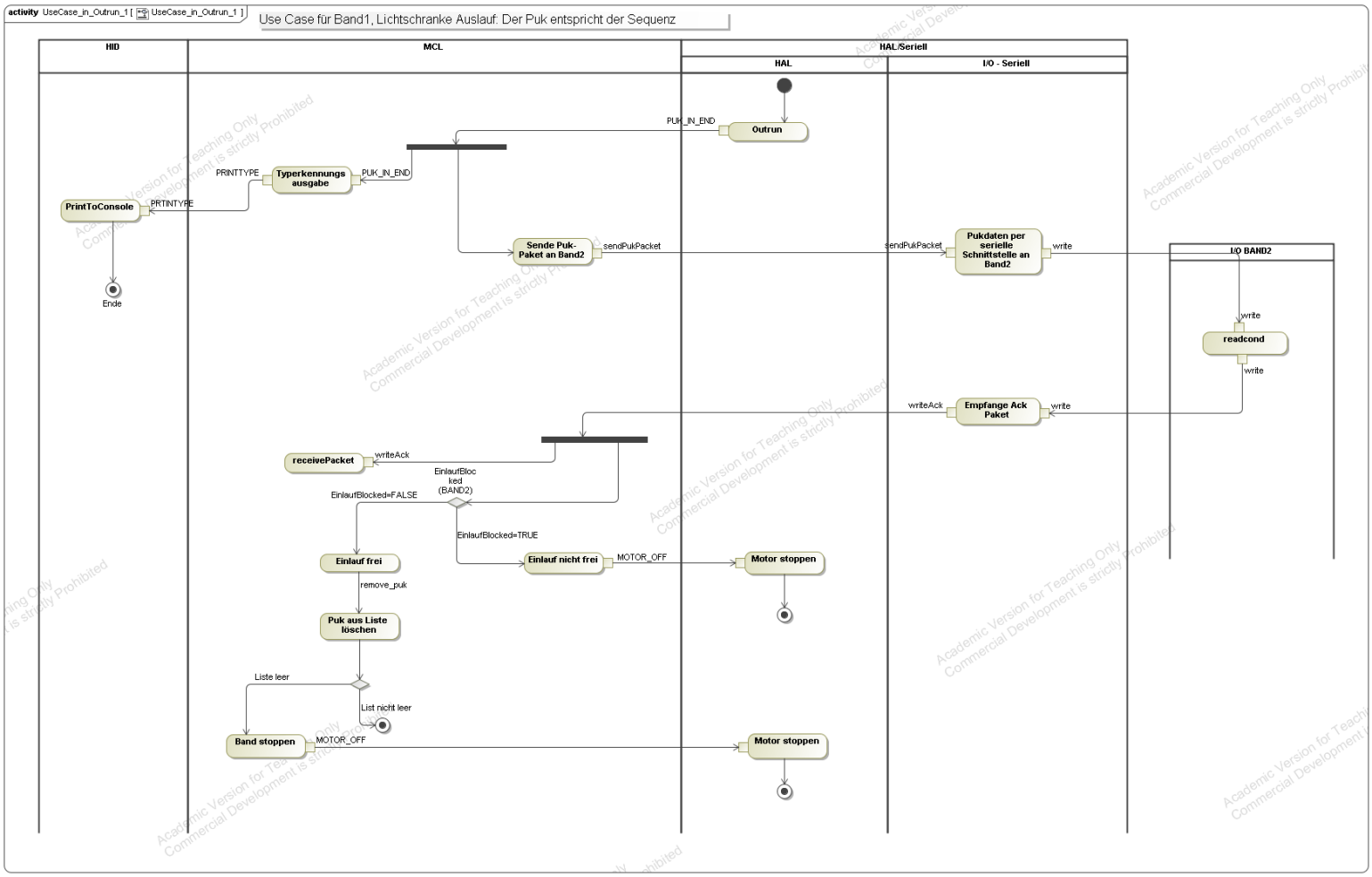


Abbildung 7: Use Case: Puk entspricht der Sequenz (Band1)

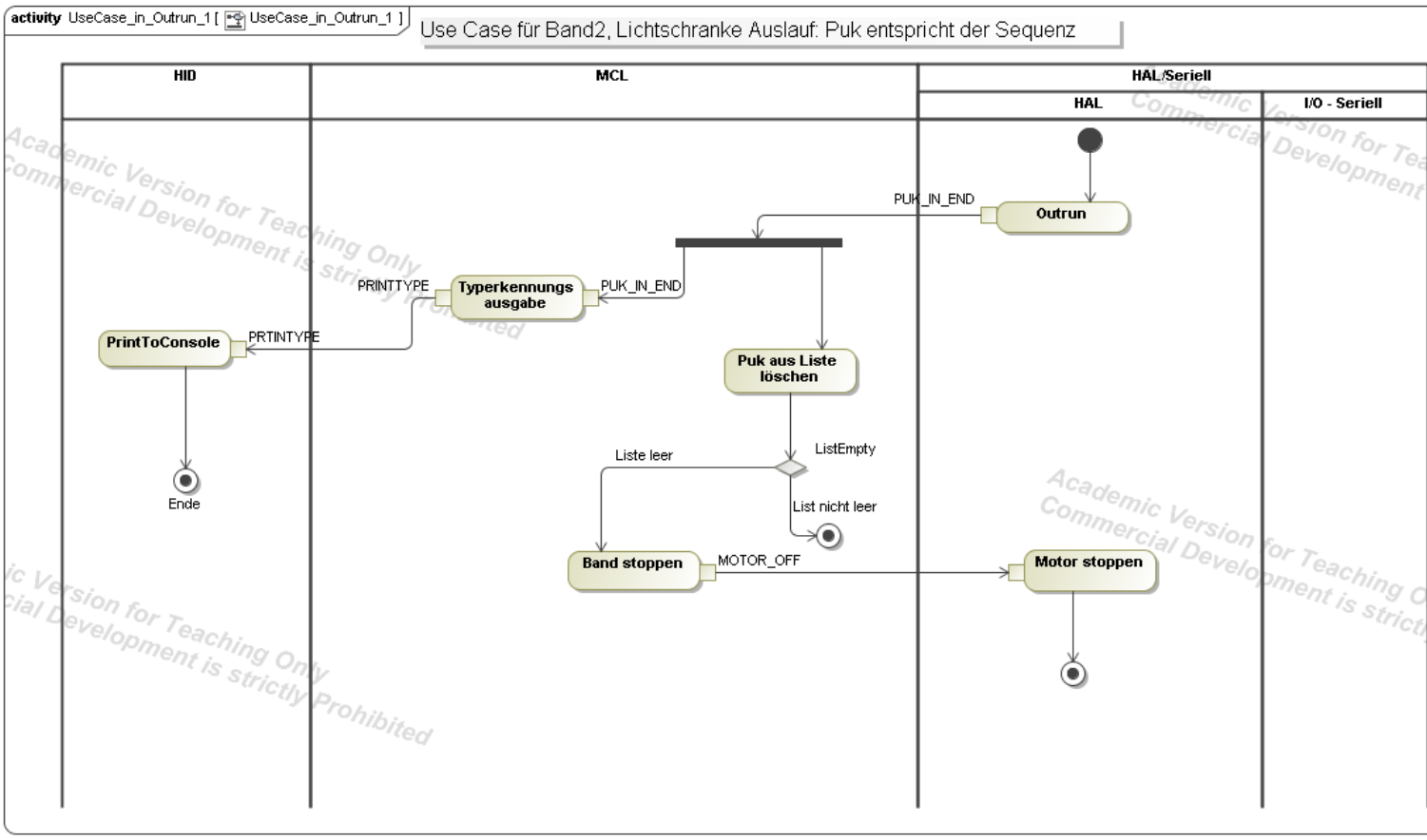


Abbildung 8: Use Case: Puk entspricht der Sequenz (Band2)

4.1.3 Systemkontext

HAL *(Hardware Abstraction Layer)* Hierzu gehören
Sensoren: Lichtschranken, Metaldetektor, Höhenmessungssensor, Bedientasten
und Aktoren: Warnleuchten, LEDs, Laufbandmotor und Weiche.

Serielle Schnittstelle Kommunikation zwischen Gemebox 1 und 2:
Packettypen: System-Status-Updates oder Pukdaten.

HDI *(Human-Device Interface)* Die Schnittstelle zur Konsolenausgabe und Konsoleneingabe.

4.1.4 Anforderungen

Typ	Resultat regulär	gewünschte Rutsche voll
Bohrung unten	Band1/Band2	Band2
Flach	Band1 (und gelb blinkt)	Band2 (und gelb blinkt)
Bohrung	falls Sequenz falsch: Band2	-
Bohrung-Metall	falls Sequenz falsch: Band2	-
Typ 1/5	Band1	Band2
Typ 2/4	Band2	Band1
unbek. Lieg Objekt-ULO	Band1/Band2	Band2/Band1

Abbildung 9: Puks

Typ	Grund des Leuchtens
gelbe Leuchte blinkt	Flache Werkstücke auf Band1 erkannt. (wird aussortiert)
grün leuchtend	Bandanlage in Betrieb
rot 1Hz	anstehend unquittiert
Brot 0,5Hz	gegangen quittiert
rot leuchtend	anstehend quittiert

Abbildung 10: Leuchten

5 Design

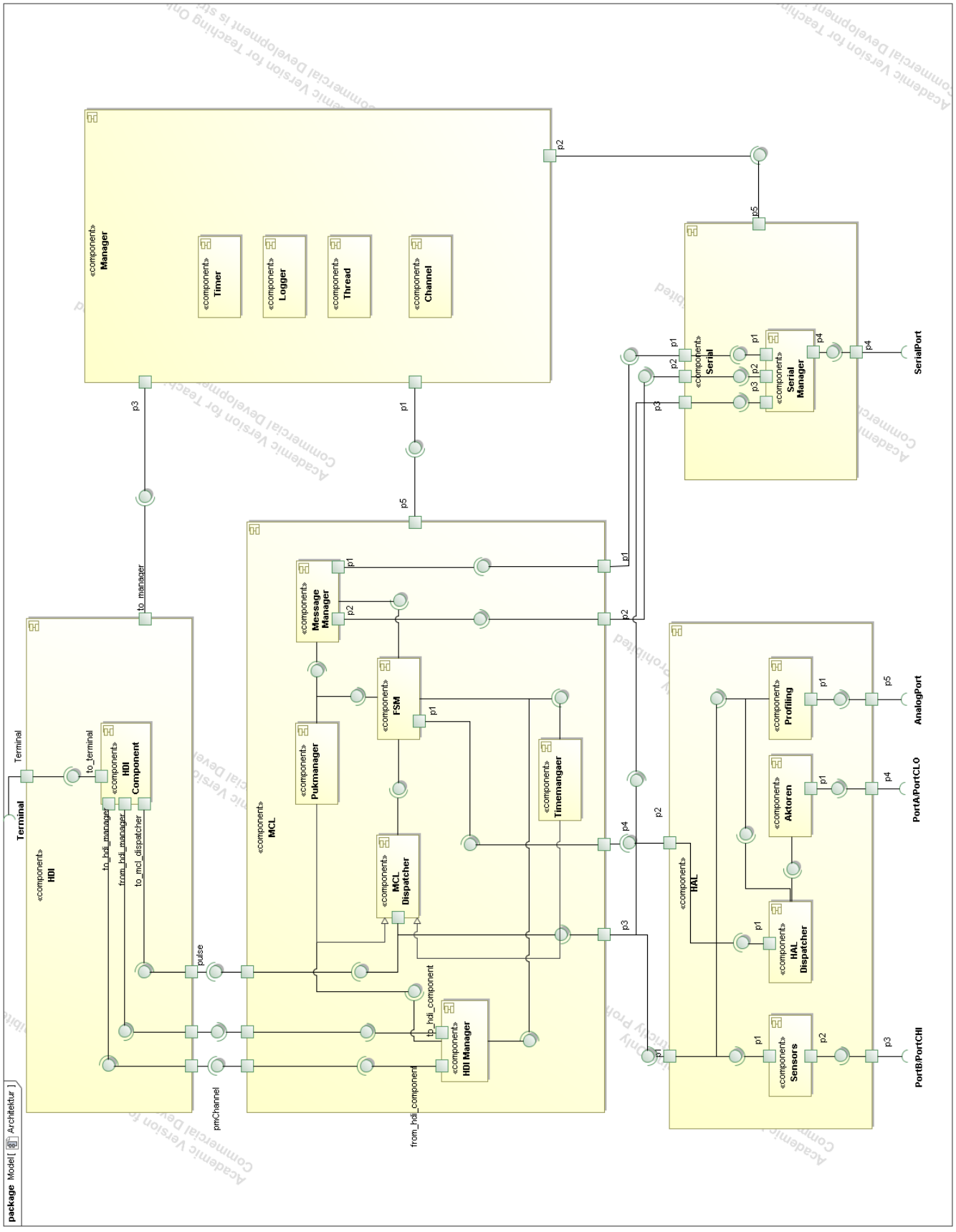


Abbildung 11: Komponentendiagramm der Architektur

5.1 System Architektur

Übersicht über die Architektur
Die Architektur ist gemäß der Aufgabenbereiche in drei Schichten ausgelegt. Die Kommunikation zwischen diesen geschieht über **Pulse Messages**. Hierbei stehen 8 Bit zur Übermittlung von Instruktionen als `enum` zur Verfügung, sowie 32 Bit, welche zum übergeben der Referenz auf zu manipulierende Daten genutzt werden. Ein blockierendes Verhalten wird somit vermieden.

HAL
Die **HAL** (Hardware Abstraction Layer) abstrahiert die Hardware des Förderbandes, indem sie zwischen `enum` und Belegungen der Kontroll-Register der für die Ports A-C genutzten I/O-Karten übersetzt. Die Komponente `Sensors` reagiert auf Digitale Inputs, indem sie bei Updates die geänderten Bits in einer ISR identifiziert und der Kontrollkomponente (**MCL**) deren Semantik in Form eines Signalcodes übermittelt. Sie ist insbesondere für die Erkennung gedrückter Buttons, Metall-Erkennung und Pegelwechseln an Lichtschranken zuständig. Der `Dispatcher` hingegen wandelt von der **MCL** eingehende Kontrollcodes und steuert mit der Komponente `Aktoren` durch Überschreiben der zuständigen Kontrollregister die Aktorik an. Die Komponente *Profiling* wird ebenfalls vom `Dispatcher` angesteuert, nimmt aber in so Fern eine Sonderstellung ein, als sie Autonom durch wiederholte Interaktion mit der analogen I/O-Schnittstelle ein Profil bestimmt und dessen Erkennungscode in als Referenz erhaltene Daten einträgt.

MCL
Diese **MCL** (Master Control Layer) steuert und überwacht alle Abläufe der Anlage. Sie hält und verwaltet die Daten und Timer und regelt mit Pulse Messages alle angrenzenden Schichten. Die Komponente `Dispatcher` verarbeitet alle Eingehenden Pulse Messages, indem sie Transitionen der **FSM** steuert. Sie reagiert sowohl auf Signale anderer Schichten, als auch auf abgelaufene Software-Timer, die über den über einen gemeinsamen `Channel` eingehen.

Die **FSM** (Finite State Machines) hält den aktuellen Betriebszustand der Anlage und bestimmt ob und in welcher Weise auf Eingaben reagiert wird. Die zugehörigen Funktionsaufrufe steuern Timer und Komponenten der eigenen Schicht und schicken Pulse an angrenzende Schichten. Der `TimerManager` erlaubt das Beanspruchen und Stoppen von Timern. Bei Initialisierung kann eine bestimmte Strecke oder Dauer bis zum Interrupt festgelegt werden. Um auf Änderungen der Geschwindigkeit des Förderbandes zu reagieren, können darüber hinaus alle im Modus „Weg“ arbeitenden Timer gleichzeitig neu parametrisiert werden.
Die Komponente `PukManager` verwaltet die auf der Anlage befindlichen Puks als Struktur in einer **Liste**. Sie hat Referenzen auf die als Nächstes in den für Zugriffe relevanten Positionen erwarteten Puks, sodass die Verteilung auf der Anlage mit minimalem Aufwand die abgebildet werden kann. Sie allein ist für Instanziierung und Löschung zuständig. Wir verwenden das *Shared-Memory-Konzept*. Innerhalb einer Anlage werden Referenzen verwendet. Kopiert wird nur beim Transfer auf die zweite Anlage. Die individuelle Puk-Struktur enthält sämtliche zum physikalischen Objekt erfassten Daten. Sie enthält darüber hinaus Referenzen auf zugehörige minimal- und maximal-Timer für die Erwartete Dauer bis zur nächsten erfassbaren Position sowie eine vorläufige Entscheidung über die ggf. zu nutzende Weiche zum Ausleiten.
Der `SerialManager` erlaubt den **FSMs** mehrerer Anlagen zu kommunizieren, um Benachrichtigungen über Fehlerzustände auszutauschen und blockierte Eingänge/Rutschen zu melden. Er ermöglicht ferner beim Transfer von Puks zum angrenzenden Band die korrespondierende Datenstruktur zu übergeben.

Der `HDI-Manager` dient der Ansteuerung der **HDI**. Er soll insbesondere Puk-Strukturen und Messwerte der Kalibrierung für die Anzeige aufbereiten.
HDI
Die Human Device Interface - Layer steuert die QNX-Konsole. Neben der Anzeige von Daten soll sie gegebenenfalls selbstständig die Semantik Tastatureingaben selbständig identifizieren und **MCL** per Pulse benachrichtigen können.

5.1.1 Nachtrag

Da der HDI eine sehr kleine Funktion in unserem Design hat, wurde dieser Layer in den `MCL-Layer` mit eingebunden. Somit erhalten wir eine *Zweilayer*-Struktur. Die neue Architektur ist in Abbildung 12 dargestellt.

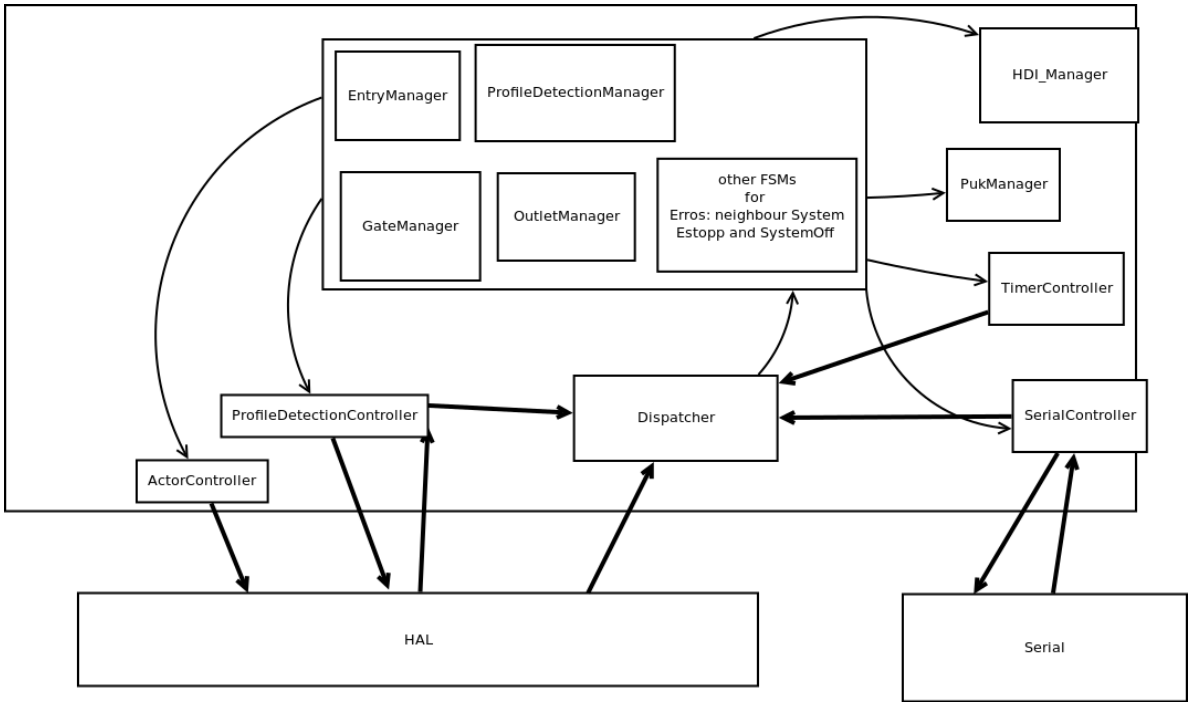


Abbildung 12: ZweischichtenArchtiektur: kräftige, gerade Pfeile: Kommunikation über PulseMessages (Channel), dünne,runde Pfeile: Funktionsaufrufe. Dies ist eine vereinfachung der Architektur und sollte als Komponentendiagramm interpretiert werden. Dabei ist der obere Layer die `MCL`. Die `HAL` und die `Serial` ist im Anhang als Klassendiagramm dargestellt und somit wird hier darauf verzichtet. Zwischen den Komponenten, welche keine FSMs sind (Controller, HDI/PukManager), wird ebenfalls über Funktionsaufrufe kommuniziert.

Der Vorteil dieser Architektur ist die quasi (linare) Unabhängigkeit der Signale zu den Statemachine's, sodass es einfach ist, eine neue Messstation einzubauen. Man muss nur entsprechend eine neue Statemachine hinzufügen und die Zulieferer entsprechend erweitern.

FSM-Modelle: siehe Anhang (State Machine's) HAL und Serial: Klassendiagramme: siehe Anhang(Hal/Serial)

6 Implementierung

6.1 Wrapping

Abbildung 13 und 14 beschreiben die gewrappten Funktionen des Threads und QNX-PulseChannels und deren Hilfsklassen, was nötig ist, um folgende Bilder zu verstehen. Zudem wrden in der statischen Klasse `mgt` die `chrono` Bibliothek gewrappt, um mit ihr einfacher den Softwaretimer zu implementieren. So wurde eine Funktion zum berechnen der Zeitdifferenz `mgt::elapse(timePoint &last)` erstellt.

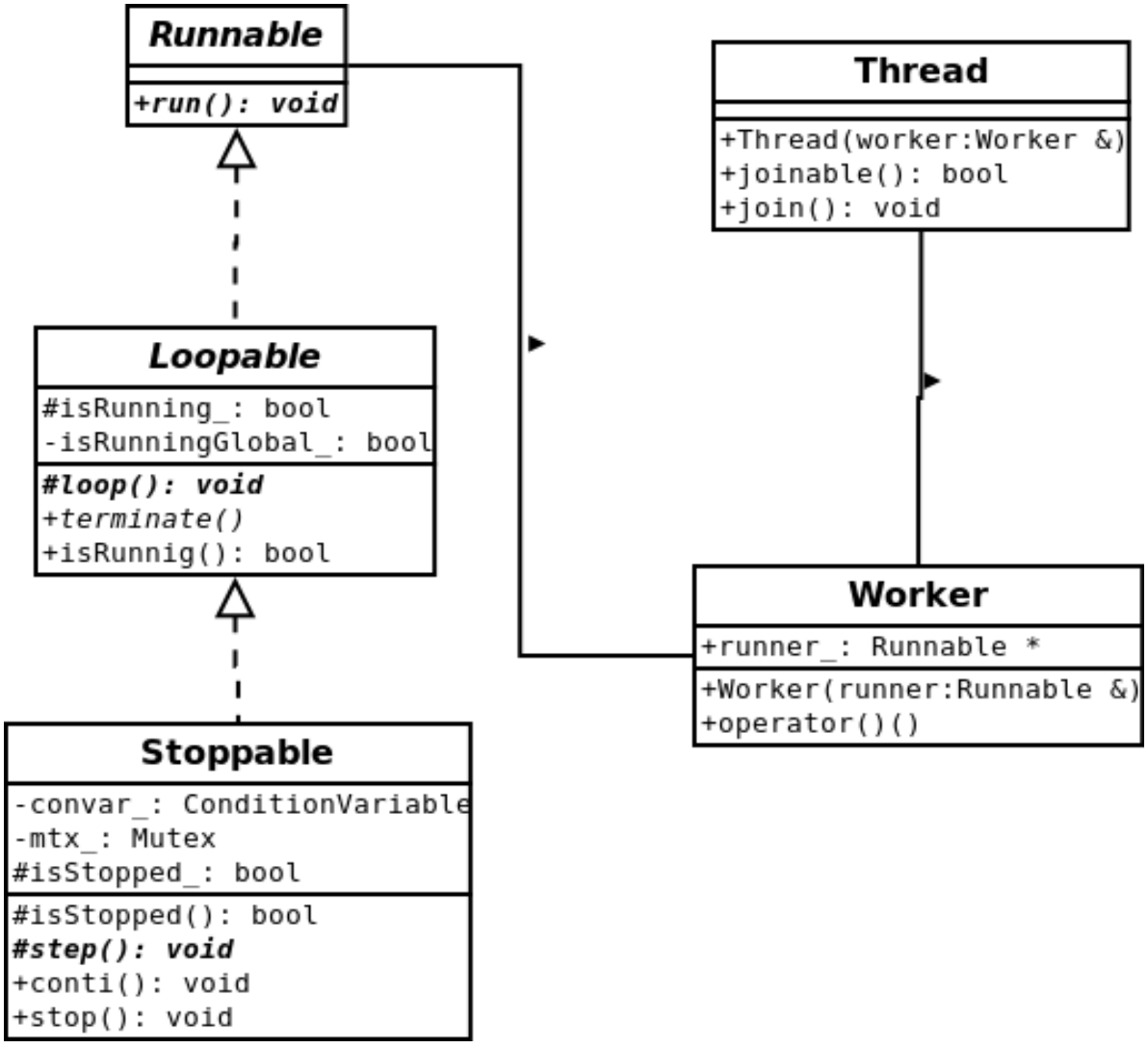


Abbildung 13: Threadwrapper mit seinen Hilfsklassen.

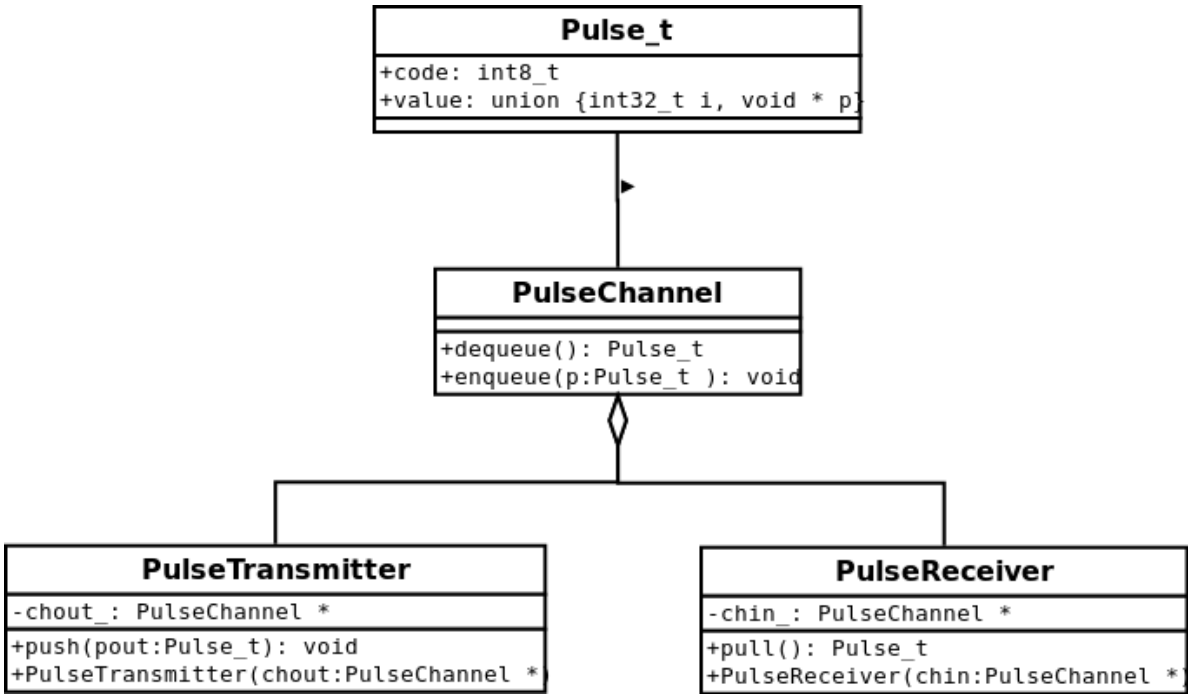


Abbildung 14: Channelwrapper undseine Hilfsklassen

6.2 Layerconnection und Communication

Pro Layer existiert ein Sigelton Entity Object, welches den Hauptthread der Layer verwaltet (meist Dispatcher) und die Channels der Komunikation hält. In der Main werden diese Objecte instaziiert und die Layer mit über die Channel miteinander verbunden. Erst wenn das geschehen ist, wird eine entsprechende init-Methode ausgeführt, um die Hauptthreads und weiteren Klassen zu instanziieren und zu starten. In Abbildung 16 sind die Klassen in UML dargestellt.
Die Kommunikation der Layer erfolgt dann über die Channel, welche diese Entity-Singeltons haben. Dabei gehört der Channel dem, der auf diesen hörcht. Die Kommuniktion erfolgt dabei auf dem Prinzip der PulseChannel von QNX. Dafür wurde eine vereinfachung der Pulsestruktur gewrappt. `Pulse_t` besteht aus einem identifizierungs Code `code` (8bit) und einem inhaltswert `value`, welcher als Ganzzahl oder als Pointer verwendet werden kann. In Abbildung **??** sind die Interfaces nochmal dargestellt.

6.3 Timer

Es werden im Code Software-Timer statt Hardware verwendet, da diese einfacher zu implentieren sind und das System ausreichend Ressourcen für diese verfügbar hat. Die Software-Timer werden asynchron verwendet, um keine anderen Prozesse zu blockieren. Außerdem ist der Code so weitgehend betriebssystemunabhängig und kann so auch auf anderen Plattformen eingesetzt werden. Da sich alle Timer in der MCL befinden, ist dies auch vorzuziehen. Leichte Ungenauigkeiten der Software-Timer im Zeitverhalten sind aufgrund der Echtzeitanforderungen im halb-Sekunden Bereich unproblematisch.
Die Grundidee ist der laufende Thread der TimerController, der pro Software-Tackt (Thread schläft

Abbildung 15: layer Klassendiagramm: Die Großen Entity-Klassen, um die Layer miteinander zu verbinden und den Layerhauptthread zu verwalten.

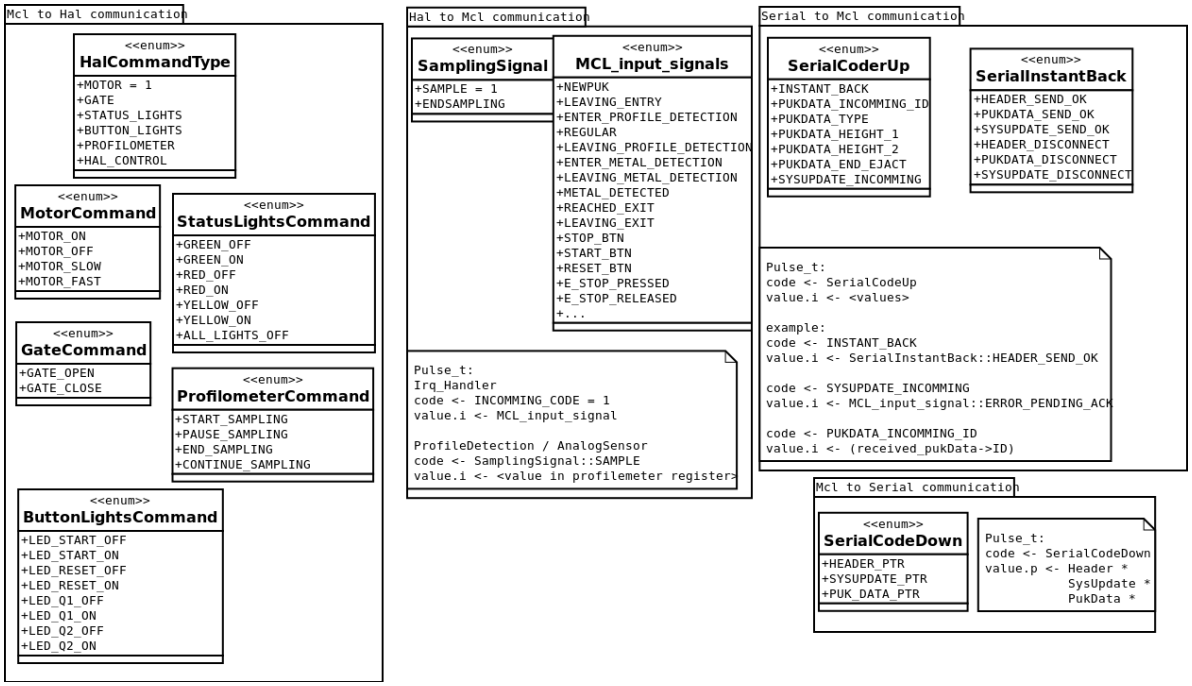


Abbildung 16: layer Klassendiagramm: Die Interfaces zwischen den Layern

eine bestimmte Zeit an Millisekunden) alle verwalteten TimerObjekte, neu berechnet. Die TimerObjecte sind einfache Entitäten, welche, die Daten, wie die abzulaufende Restzeit (`delta`) und den Typen (Distance oder Zeit Timer). Der TimerController wertet diese Informationen aus und berechnet bei jedem Tick die neue Restzeit. Wenn diese abgelaufen ist, wird der ebenfalls im Timer gehaltene Rückgabewert in den MCL-Hauptchannel geschrieben. Jedes TimerObjekt muss mit der Factoryfunktion `getTimer()` des TimerControllerObjectes erzeugt werden, damit diese Timer auch von diesem bearbeitet wird. Ein Timer wird neu aufgezogen, indem das entsprechende MCL_signal in der Methode TimerController::resetTimer(Mcl_signal ret) verwendet wird.

6.4 Interrupts

In der HAL werden einkommende Signal auf Port B und C entsprechend ausmaskiert und ausgewertet, ob es sich um eine steigende Flanke handelt oder um eine fallende. Das Ergebinis wird auf das entsprechende MCL_input_signal gemappt und nach oben gesendet.

6.5 Profilerkennung

Die Profilerkennung startet beim Eintritt in den Bereich (unterbrechung der Lichtschranke), misst den Puk von der Mitte bis über den Puk hinaus, wird beendet beim Austreten des Bereiches (Lichtschranke geschlossen) und die ist wie folgt aufgebaut:
In der MCL: Der ProfileController holt sich den Puk in seinen Speicher und schickt der HAL den Befehl zum Starten der Profilmessung runter.
In der HAL: Es wird der AnalogSensorthread aufgeweckt. Es wird zunächst das Register, in dem sich der akutelle Wert befindet nach den Hardwarespezifikation ausgelesen. Dann wird ein

Medianfaltungsfilter über die letzten fünf Werte gestellt, um die Fluktuation zu minimieren, und dieses Mittel dann an die MCL bzw. den ProfilDetectionController geschickt.
: In der MCL: Dieser sammelt diese Werte in einer Liste von Buckets so, dass immer der Wert mit dem gewichteten Mittel der letzten Werte verglichen wird. Die letzten Werte sind hier werte, die im Toleranzbereich des Mittels lagen. Liegt der Wert im Toleranzbereich des Mittels, wird der Zähler für dieses Mittel hochgezählt (und das Mittel wird neu berechnet). Liegt der Wert nicht im Tolleranzbereich, wird ein neuer bucket in die Liste gesetzt und von dort neu verglichen. So erhält man eine zeitlich korrektsortierte BucketList. Zum Beenden des Samplingvorgangs schickt der ProfileDetectionController einen Stopp Befehl an die Hal.
In der HAL: Im AnalogSensorObject wird ein Flag auf true gesetzt, was das Auslesen der Registers unterbricht und eine (Ende der Profilemessung) Nachricht an den ProfileDetectionController schickt. Im Anschluss schickt sich der Thread selber in den Schlaf.
In der MCL: In der MCL leitet diese Nachricht die Auswertung der Bucketliste ein. Zuerst wird geguckt, ob der Zähler der Buckets groß genug ist. Bei zu kleinem Zähler wird dieser Bucket gelöscht, um fallende und steigende Sinussteigung (entstanden durch die Mittelung in der HAL) zu dezimieren. Aus den übrigen Werten wird jetzt das Maximum und das Minimum gesucht und damit die entsprechende Höhe berechnet, denn es wird auch immer die Höhe des Laufbandes selber mit erfasst. Es wird das Minimum von den Werten abgezogen und alle Werte, die im Tolleranzbereich des Minimums und des Maximums sich befinden, werden aus der Liste gelöscht. Ist die Listengröße nur null, so ist der Puk flach. Ist die Größe eins, so ist der Puk gebohrt. Ist der Puk größer zwei, so muss es ein Bitcodierter sein. Die Liste wird in diesem Fall nochmal durch die Höhe minus Tolleranzbereich Ganzzahlig geteilt. Die nun quasi bitcodierte Liste wird nun in eine Ganzzahl umgewandelt und das Ergebnis wird direkt in den Puk geschrieben.

6.6 Serielle Kommunikation

Die Gemebox hat zwei Serielle Schnittstellen, was hier ausgenutzt wird. Eine Schnittstelle für eingehende Nachrichten und die andere Schnittstelle dient zum Senden von Nachrichten. Dabei wird auf jede gesendete Nachricht im SerielLayer mit einem bestätigendem ACK gewartet. Das Lesen (Empfangen) von ACK ist hierbei Zeitbeschränkt. Beim Überschreiten der Zeit, wird eine Fehlermeldung hoch an die MCL geschickt. Das Protokoll ist entsprechend Aufgebaut (siehe hierfür Abbildung 21 oben rechts im Anhang):
In der MCL: (SerialController) Zuerst wird ein Header gezeugt, welcher die Art der kommenden Information beschreibt. Eine Referenz des Headers wird mit entsprechenden Code (HEADER_PTR) nach unten geschickt. Das folgende Packet wird im Anschluss entsprechend runtergeschickt. Um Speicherleaks zu vermeiden, werden die Pakete oben in Listen verwaltet und wenn ein SEND_OK gelöscht.
In der Serial: Die von der MCL kommenden Referenzen werden entsprechend einfach über das Kabel weitergeleitet.
Beim Empfangen werden die einzelnen Pakete, wird ein ACK auf dem gleichen Kabel gesendet und wegen der Natur des Replayloggers, in seine Einzelteile zerlegt und stückweise in die MCL geschickt.

6.7 Log and Replay

Durch das Loggen der eingehenden Signale des Channels vom **MCL**, kann ein kompletter Ablauf der einmal auf einem Laufband abgelaufen ist, beliebig oft abgespielt werden. Es wird die aktuelle

Systemzeit beim Start des Ablaufs und beim Eingang jedes Signals getrackt. Außerdem wird der Enum- Wert des Signals und der Wert, welcher Sender es in den Channel geschrieben hat, gespeichert.
Zum abspielen der Datei, muss das Programm neu ausgeführt werden und beim Start des Laufbands über Tastaureingabe angegeben werden, dass der Replay-Modus ausgeführt werden soll und die entsprechende Log- Datei ausgewählt werden.
Nach der Auswahl der Datei werden die Signale mit dem richtigen Timing, welches durch die Differenzbildung der Zeit, von dem Signal zuvor, berechnet wird, wieder in den Channel vom **MCL** geschrieben.

6.8 Steuerung (FSM)

Die Steuerung des Systems ist durch vier orthogonale Finite State Machines realisiert. Jede dieser FSMs überwacht/steuert das Verhalten, welches durch das physikalische Eintreffen/Fehlen eines Objekts am der State Machine zugeordneten Ort zum definierten Zeitpunkt beabsichtigt ist, indem sie den einer zugehörigen Hardware-Komponente Controller anspricht, eine Transaktion an einen Manager delegiert und/oder Flags im Shared Memory setzt. Letzteres erlaubt es notwendige Informationen über blockierte Rampen/Übergänge sowie Fehlerzustände global verfügbar zu machen.

Ein Multiplexer-Modul gewährleistet die korrekte Zuordung der eingehenden Signale. Die Nutzung eines einzigen Channels als Quelle aller transitionen bewirkender Steuersignale gewährleistet eine vollständige Synchronisierung und Pufferung. Alle Transaktionen sind folglich atomar.
Die Zuständigkeit der aufgrund ihrer erweiterten Funktionalität als EntryManager, ProfileDetectionManager, GateManager und Outletmanager bezeichenten FSM-Module korrespondiert mit den Orten, an denen aufgrund der an der Anlage befindlichen, orthogonal zur Laufrichtung des Bandes ausgerichteten Lichtschranken Informationen über die Position eines Objektes gewonnen werden können.
Informationen über zwischen Lichtschranken entfernte bzw. zugefügte Objekte werden aus Timeouts gewonnen. Jeder Puk hat zu diesem zweck Referenzen auf zwei Timer deren Dauer bei Verlassen einer Position mit der minimalen/maximalen benötigten Zeit zum erreichen der nächsten feststellbaren Position reinitialisiert wird. Das Ausbleiben eines Minimal-Timeouts vor Eintreffen eines Objektes wird als Hinzufügen, das Ablaufen des Maximal-Timers als Entfernen eines Objektes interpretiert.
Die Laufzeit aller erzeugten Timer kann und bei Änderung der Geschwindigkeit des Laufbandes aufgrund ihrer Erzeugung durch eine spezielle Factory gleichzeitig angepasst werden.

Jeder Automat erfasst an der zugeordneten Position aufgetrende Fehler und ist ebenfalls für deren Auflösung verantwortlich. Da die Anlage im Fehlerfall der Betrieb stoppt und alle Signale serialisiert eingehen, befindet sich stehts nur eine FSM im Fehlerzustand. Die Auflösung erfolgt bei Erkennen des Reset-Signals, welches an alle FSMs witergemeldet wird.

Übergeodnete Zustände werden nur durch Fehler auf der seriell verbundenen Anlage, oder durch die Verwendung von Druckknöpfen erreicht. Sie sind so umgesetzt, dass sie durch das Setzen von als Guards agierenden Flags eine Weiterleitung von Signalen an untergeordnete FSMs unterbinden.

Dies betrifft den E-STOP, welcher das System anhält und das Druchlaufen einer vorgegebenen Sequenz von Button-Events erzwingt bevor die reguläre Weiterleitung Signale wieder aufgenommen

wird, die Unterbrechung aufgrund eines Fehlerzustandes auf andererem Band, das das Verhalten Warnleuchten angleicht und jede Fortsetzung des Betriebs unterbindet, sowie ein An/Aus, das lediglich den Betrieb unterbricht.

Weitere Signale, die keine Transition bewirken, aber aufgrund des Entwurfs nach Zuständigkeit zur Verarbeitung ausgewertet werden, sind Benachrichtigungen über eingegange Messwerte.

7 Testen

Beim Testen in der Implementierungsphase hat die Funktionsfähigkeit des regulären Betriebsablaufs Priorität. Erst wenn diese gegeben ist sind Fehlerfälle zu berücksichtigen, um die Komplexität bei Problemen gering zu halten. Dieses Vorgehen kann in der Praxis an der Hardware nur bei Vermeidung von gefahrbringenden Zusätänden angenommen werden.
Die Softwarekomponenten sind zuerst bei der Implementierung durch simulierte Signale zu testen, die innerhalb der HAL definiert und zeitgesteuert ausgeführt, d.h. an Komponenten weitergegeben werden.
Nach den erfolgreichen Simulations-Tests werden zusammenhängende Komponenten auch an der Hardware getestet.

7.1 Abnahmetest

Schriftlich ausformulierter Abnahmetest: Siehe Anhang `Abnahmetest`.

Nr	Test	Ausgangslage	Durchführung	Erwartung	Auswertung
Betriebsmodi					
1	Anlage EIN	Anlage aus, Aktoren aus, Band und Rutschen leer	Start-Button betätigen	Ampel schaltet auf Grün	
				Start-Tastenbeleuchtung ein	
				Bandlauf startet bei Unterbrechung der Lichtschranke im Einlauf	
2	Anlage AUS	Anlage EIN aktiv	Stop-Button betätigen	Ampel aus	
				Tastenbeleuchtung aus	
			Unterbrechung von Lichtschranke Einlauf Lichtschranke Profiling Lichtschranke Weiche Lichtschranke Rutsche	Sensorik wird ignoriert	
3	Anlage-Stop (e-stop)	Anlage EIN aktiv	E-Stop einrasten	Ampel Rot	
				Tastenbeleuchtung aus	
				Sensorik wird ignoriert	
				Taster werden ignoriert	
			E-Stop herausziehen	Ampel Rot (blinkend)	
				Taster außer Reset werden ignoriert	
				Tastenbeleuchtung Reset ein (blinkend), andere aus	
				Sensorik wird ignoriert	
			Reset betätigen	Anlage EIN	

Abbildung 17: Betriebsmodi

Nr	Test	Ausgangslage	Durchführung	Erwartung	Auswertung
Bandlauf					
Typerkennung					
4	Bohrung unten	Anlage EIN, Rutschen und Band frei bei jedem Durchgang	Entsprechenden Typ in Einlauf legen	Konsole: Bohrung unten	
5	Flach			Konsole: Flach	
6	Bohrung (nicht Metall)			Konsole: Bohrung (nicht Metall)	
7	Bohrung Metall			Konsole: Metall	
8	Profiltyp 1 / 5			Konsole: Profiltyp 1 / 5	
9	Profiltyp 2 / 4			Konsole: Profiltyp 2 /4	
10	Unbekanntes Objekt		Objekt mit anderen Abmaßen in Einlauf legen	Konsole: bezeichnende Fehlermeldung	
Regulärer Betriebsablauf					
11	Korrekte Reihenfolge erkennen	Anlage EIN, Rutschen und Band frei	Abwechselnd einen Puk außerhalb der Sequenz und einen innerhalb einlegen	aussoritert	
				Außerhalb: Auf Band 2	
				Innerhalb: Auf Band 1	
				Außerhalb: Auf Band 2	
				Innerhalb: Auf Band 1	
				Außerhalb: Auf Band 2	
				Innerhalb: Auf Band 1	
				Fehlermeldung weil voll, Anlage Stop	
Fehlerfälle					
12	Beide Rutschen voll	Anlage Ein	beide Rutschen im Normalbetrieb volllaufen lassen	Laufbänder stoppen	
				Anlage geht in Anlage Stop	
				Konsole: bezeichnende Fehlermeldung	
13	Rutsche 1 voll, obwohl Sequenz nicht komplett	Anlage Ein, Rutsche 2 frei	Die Lichtschranke an der Rutsche 1 wird manuell für 10 Sekunden unterbrochen	Laufbänder stoppen	
				Anlage geht in Anlage Stop	
				Konsole: Zeitverletzung	
		Rutsche 2 belegt durch Normalbetrieb	Puk außerhalb Sequenz einlegen	Puk in Rutsche 1 aussortiert	
				Konsole: Sequenz unterbrochen	
				Konsole: Fehlermeldung Rutschen voll	

Abbildung 18: Bandlauf

Nr	Test	Ausgangslage	Durchführung	Erwartung	Auswertung
Zeitverletzungen					
14	Falscher Gegenstand oder Puk erreicht verfrüht seine nächste Station	Puk in Einlauf, Bereich bis Profiling frei	nächsten In-Position Sensor händisch vorzeitig auslösen	Fehlermeldung, Anlage Stop	
		Puk aus Profiling, Bereich bis Rutsche frei		Fehlermeldung, Anlage Stop	
		Puk zum Aussortieren an Weiche		Fehlermeldung, Anlage Stop	
		Puk zum Weiterleiten an Band 2 an Weiche		Fehlermeldung, Anlage Stop	
15	Blockade oder vermisster Puk	Puk in Einlauf, Bereich bis Profiling frei	Puk wegnehmen bzw. verzögern um mehr als 1 cm	Fehlermeldung, Anlage Stop	
		Puk aus Profiling, Bereich bis Rutsche frei		Fehlermeldung, Anlage Stop	
		Puk zum Aussortieren an Weiche		Fehlermeldung, Anlage Stop	
		Puk zum Weiterleiten an Band 2 an Weiche		Fehlermeldung, Anlage Stop	
Log & Replay					
16	Ablauf von #11 im Logging prüfen	#11 zuvor durchgeführt		Ablauf ist nachvollziehbar dargestellt	
17	Ablauf von #11 abspielen			Ablauf entspricht der Aufzeichnung	

Abbildung 19: Zeitverletzungen und Log & Replay

7.2 Testprotokolle und Auswertungen

letztes Testprotokoll vor Abnahmetest hier einheften.

7.3 Lessons Learned

Die Bewerkstelligung des Projekts brachte einige Aufgaben mit sich, die bisher niemand im Team bewältigen musste. Zum einen ist der Umfang der Aufgabe deutlich größer und umfasst mehrere Termine und zum anderen ist dementsprechend die Größenordnung des Projekts und der Teams größer als gewohnt.
Die wichtigste Hürde, die zuerst genommen werden musste, war die Organisation in den großen Teams. Die Absprache und Kommunikation bringt einen exponentiell steigenden schwierigkeitsgrad mit sich, wenn man es gewohnt ist, sich mit nur einem Partner abzusprechen.
Durch die wegweisenden Hinweise der Aufgabenstellung und der Empfehlungen der Professoren und Erfahrungen anderer Studenten wurde der agile Entwicklungsprozess eingeschlagen und der Fokus auf die Modellierung und allgemein ausformulierte Planung der Anforderung gelegt.
Da man es nicht in kurzer Zeit schafft einen tiefen Einstieg in die komplexen Anforderungen der Aufgabe zu machen, fiel die Analyse anfangs schwer und war nicht überschaubar, bis eine grobe Architekturidee festgelegt wurde.

Krampfhafte Termineinhaltung, Ordnung und Struktur der Protokolle und Dokumentation aller Fortschritte wurde anfangs stark eingehalten, aber später für zu Zeitraubend gehalten, da die Teamgröße schlicht und einfach zu klein war, um bei der erzeugten Dokumentation mitzuhalten. Der Umstieg auf ein Kanban-Bord hat die Aufgabenverteilung und Übersicht des Projekts stark vereinfacht.
Im Stress,die vorgeschlagenen Meilensteinen einzuhalten, haben wir eigenforumlierte User-Stories zu Terminen verfasst, die eher unserem Projektforschritt entsprachen.
Die Kommunikation in der Gruppe war am effektivsten, wenn im großen Team zusammen gearbeitet wurde. Falls Fragen entstanden waren, konnte direkt geklärt werden, wer diese beantworten kann, oder die Entscheidung treffen soll. Dabei kam es allerdings auch zu Ablenkung und Unkonzentriertheit, die nach einer gewissen Zeit die Energie geraubt hat. Hinzu kam das Erlernen einer neuen Programmiersprache und der Umgang mit einer neuen Entwicklungsumgebung, die ihre Tücken mit sich bringt. Der reibungslose Umgang mit der Entwicklungsumgebung sollte im vorhinein gefördert werden, da die Probleme nebenbei sehr zeitraubend sein konnten, wenn die Hauptkonzentration eigentlich auf einem anderen Problem lagen. Der wichtigste Faktor, der uns trotz den Hürden durch das Projekt gebracht hat, waren die Teilerfolge, an denen anfassbare Fortschritte zu sehen waren.

8 Anhang

8.1 Glossar

Eindeutige Begriffserklärungen

Begriffsglossar ESE-Projektaufgabe

Nils Eggebrecht
Lennart Hartmann
Alexander Mendel
Eduard Veit
Karl-Fabian Witte

[30. März 2017]

Dieses Dokument ist vorerst Abgeschlossen. Es werden keine Änderungen benötigt.

1 Glossar

Begriff	Erläuterung	Quellen
Artefekte	Artefakte eines Softwareprojekts bezeichnet man als Arbeitsereignisse (Work Products). Dazu gehören nicht nur Arbeitsschritte der Softwareimplementierung, sondern auch Vorarbeiten (Vorbereitung der Architektur, des Designs) und kleine Milestones, die zum Erreichen des Ziels erforderlich sind. Ein Artefakt kann in Form eines Dokuments, Models (Use-Case), Element eines Modells festgehalten werden, so dass jedes Teammitglied ständig Informationen zur Planung des Projekts abrufen kann.	1. https://blog.flavia-it.de/artefakte-in-softwareprojekten/ [30. März 2017, 9:30]

Begriff	**Erläuterung**	**Quellen**
STL Threads	STL (Standard Template Library) ist ein Paket von Template-Klassen (z. B. data structures: vetors, lists, queues und stacks). Kategorien: • I/O • String and character handling • Mathematical • Time, date, and localization • Dynamic allocation • Miscellanous • Wide-character functions Die Threadklasse (std::thread) repräsentiert ausführbare Threads und Multithreading im gleichen Adressspace. (Joinable)	1. https://www.tutorialspoint.com/cplusplus/cpp_standard_library.htm [30. März 2017, 9:30] 2. http://www.cplusplus.com/reference/thread/thread/ [30. März 2017, 9:31]
HAL	Der HAL (Hardware Abstraction Layer) ist eine logische Zwischenschicht im Betriebssystem. Sie vereinfacht die Übertragung zwischen Betriebssystem und kapselt die Eigenschaften der Zielplattform (MMU, Memory, Times, Port/Devices...). Die HAL abstrahiert Eigenschaften einer Plattform zu einer Programmierschnittstelle (API), wodurch die Architekturen der Plattformen gleich aussehen. Bei Hardwareänderung muss also nur die HAL-Schicht verändert werden. Die Funktion eines HAL gibt es nicht nur bei Betriebssystemen, sondern auch dort, wo Schichten einer Systemarchitektur voneinander getrennt werden müssen.	1. http://www.itwissen.info/Hardware-Abstraktionsschicht-hardware-abstraction-layer-HAL.html [30. März 2017, 9:30]

Begriff	Erläuterung	Quellen
Petri-Netze	Ein Petri-Netz (auch Stellen-/Transitions-Netz) ist eine Art einer Modellierung von Abläufen mit nebenläufigen Prozessen mit kausalen Beziehungen. Knoten repräsentieren Bedingungen, Zustände, Aktivitäten, die Knotenmarkierung repräsentiert den veränderlichen Zustand des Systems. Kanten verbinden Knoten mit Vor- und Nachbedingung. Die Anwendung umfasst unter anderen die Modellierung von realen oder abstrakten Automaten und Maschinen, oder auch Verhalten von Hardware-Komponenten. (Detailliertere Informationen unter dem Quell-Link)	1. http://www2.cs.uni-paderborn.de/cs/ag-hauenschild/lehre/WS06_07/modellierung/download/mod620.pdf [30. März 2017, 9:30]
UML	Die (Unified Modeling Language) ist eine standardisierte grafische Sprach für Modelle von Systemen und insbesondere von Software-Teilen zur Dokumentation, Konstruktion und Spezifikation. Sprich: In Diagrammen werden Zusammenhänge zwischen Softwareteilen o.Ä. mithilfe der Sprechelemente dargestellt.	1. http://www.torsten-horn.de/techdocs/uml.htm [17. April 2016, 11:30] 2. http://www.itwissen.info/definition/lexikon/unified-modelling-language-UML.html [17. April 2016, 11:30] 3. https://de.wikipedia.org/wiki/Unified_Modeling_Language [17. April 2016, 11:00]

8.2 Abkürzungen

Begriff	**Erläuterung**
MCL **Master Control Layer**	Logikschicht des Automaten.
HAL **Hardware Abstraction Layer**	Hardwareschicht. Sensorik, Aktorik, serielle Schnittstelle.
HDI **Human Device Interface**	Konsole/Konsolenein - und ausgabe.
EntryManager	Dies ist der Zustandsautomat, der die Lichtschranke `Einlauf Werkstück` behandelt Zustände sind: **Ready** und **Done**.
ProfileDetectionManager	Dies ist der Zustandsautomat, der die Lichtschranke `Werkstück in Höhenmessung` und `Höhenmessung` behandelt. Zustände sind: **Idle**, **Ready**, **processing**, **MissingWorkpiece** und **UntrackedWorkpiece**.
GateManager	Dies ist der Zustandsautomat, der die Lichtschranke `Werkstück in WeicheWerkstück Metall` behandelt. Zustände sind: **Idle**, **Ready**, **processing**, **ejecting**,**Done**, **rampError**,**Acknowledged**, **MissingWorkpiece** und **UntrackedWorkpiece**.
OutletManager	Dies ist der Zustandsautomat, der die Lichtschranke `Auslauf Werkstück` behandelt. Zustände sind: **Idle**, **Ready**, **expecting**, **MissingWorkpiece** und **UntrackedWorkpiece**.
SystemOffFSM	Dies ist der Zustandsautomat, der den allgemeinen Systemzustand behandelt. Zustände sind: **on** und **off**.
EStopFSM	Dies ist der Zustandsautomat der auf den E-Stop reagier. Zustände sind: **ok**, **Acknowledged** und **EStopped**.

8.3 How-To-Git

The git must flow

a git survival kit - german version

kafawi

1. Mai 2017

Dieses Dokument ist speziell für das Praktikum des ESE Moduls 2017 des Studienganges Technische Informatik der HAW Hamburg erstellt worden und dient der Zusamenarbeit der Gruppe LANKE. Es werden hier die Grundideen mit dem Umgang mit git und dem ¨flow¨ erklärt. Die Commit Message Struktur sowie die Branchstruktur wird hier beschlossen.

Inhaltsverzeichnis

1 Einleitung

Dies soll ein Crashkurs in Sachen Git für dieses Projekt darstellen. Es wird sich hier auf das Nötigste beschränkt. Für tiefergreifende Informationen sollten externe Quellen verwendet werden, wie [Chacon and Straub, 2014], welches online frei zur verfügung steht.

Der Quellcode in den Listings zeigt nur die Gitbefehle, wie sie in einem Terminal verwendet werden. Die Ausgaben werden nicht aufgeführt. Als Prompt wird \$ verwendet. Davor steht der aktuelle Branch (checktout branch) in Klammern $(branchname)$\$. Kommentare werden mit einem Voranngeführten $\#$ eingeleitet.

```
1 (master)$ git checkout develop
2 # Update develop
3 (develop)$ git pull
4 (develop)$ git checkout -b feature/myfeature
5 (feature/myfeature)$ ...
```

Zunächst wird der Workflow, welcher in diesem Projekt praktiziert werden soll, erklärt. Darauf folgen die Konventionen im Umgag mit Git, wie die Struktur der Commit Messages. Darauf Folgt eine Auffrischung der Git Befehle. Im Letzten Kapitel werden nützliche Tricks und Kniffe vorgestellt.

2 Workflow

Wegen der sehr kleinen Anzahl von Entwicklern, wird ein zentralisierter Workflow angewendet, d.h. wir haben ein Referenzrepro auf einem Server, welcher uns den aktuellen Stand für unsere lokalen Repros anbietet. Dabei haben wir zwei Hauptbranches und mehrere dynamische Featurebranches, auf denen hauptsächlich Entwickelt wird. Dargestellt ist dies in der Abbildung 1.

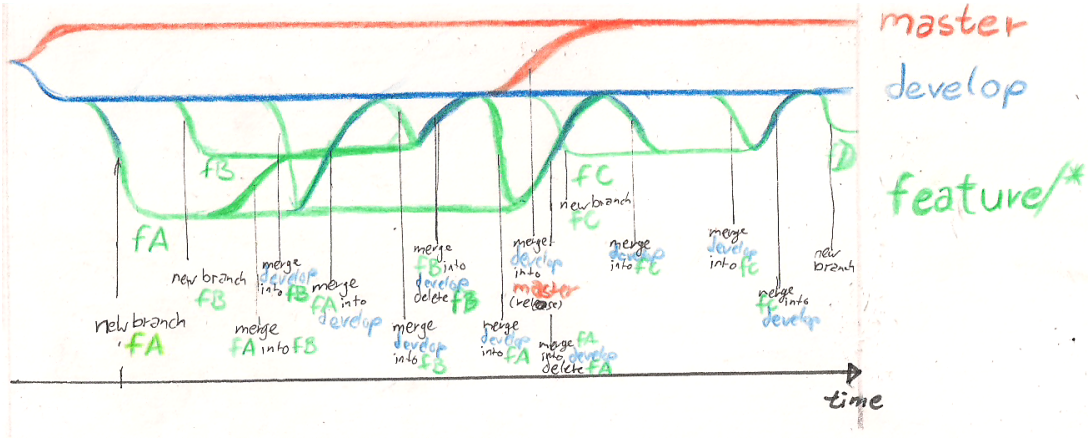


Abbildung 1: Der Workflow, wie er in diesem Projekt stattfinden soll, ist hier dargestellt. Der Masterbranch `master` ist für die Releases bestimmt und nur der `develop` wird kurz vor einem Release gemerged. Vom Entwicklungsbranch `develop` werden die jeweiligen Featurebanches `feature/*` abgeleitet und auf diesen wird in kleinen Teams gearbeitet. Wenn ein Feature fertig gestellt wird, wird dieses auf dem mit dem `remote` upgedateten `develop` eingepflegt. Wird der Featurebranch nicht mehr gebraucht, wird dieser lokal und im remote gelöscht (`git branch -dr origin/feature/*`).

master Dieser `branch` ist nur für Realeses bestimmt, und wird nur mit dem `develop` gemerged. (neuster Stand des Projektes für den nächsten Praktikumstermine)

develop Stellt die Basis für jedes `feature/*` da. Jedes einzelne `feature/*` kann in den `develop` gemerged werden, sobald dieser für annehmbar (qualitativer Inhalt) befunden wurde.

feature/* Hier werden die einzelnen Tasks in Code umgewandelt. Diese `branches` leiten sich von den dem `develop` ab und werde in diesen bei der Fertigstellung des `feature/*` zurückgeführt und ggf. gelöscht. Es kann zwischen den `feature/*` auch gemerged werden, falls diese ein bestimmtes `feature/fA` dringend braucht, welches sich noch nicht auf dem develop befindet, da es z.B. in der Review befindet. Vor jedem merge in den `develop` muss das `feature/myF` den aktuellen Stand des `origin/develop` in sich integrieren (mergen).

2.1 Zusammenarbeit im Workflow

Wenn mehrere Entwickler an einem `feature/*` arbeiten, dann müssen sie sich miteinander Absprechen, bzw. vor jedem Sprint, wird der aktuelle Stand dieses `origin/feature/*` auf den lokalen `branches` angeglichen.

2.1.1 Erstellung eines features und veröffentlichen

Im nebenstehen Listing wird ein Ablauf beschrieben, wie man ein neuen Featurebranch anlegt und diesen veröffentlicht. Dabei kann es auch vorkommen, dass ein anderer Entwickler einen Featurebranch mit gleicher Intention angelegt hat und diesen vorher schon veröffentlicht hat. Dann muss man seinen eigenen Branch mit dem schon vorhandenen zusammenführen. Wichtig ist, dass jeder `feature/*` von einem aktuellen `develop` abstammt.

```
(master)$ git checkout develop
# Update develop
(develop)$ git pull
# create and checkout new branch
(develop)$ git checkout -b feature/fA
# working some on some files and stage them
(feature/fA)$ ...
(feature/fA)$ git commit
# looking for new branches in remote
(feature/fA)$ git fetch
(feature/fA)$ git branch -a
# two possibilities
#     1. my feature is not a existing topic
#        -> publish
(feature/fA)$ git push -u origin feature/fA
#     2. my partner already has a fAA-branch
(feature/fA)$ git checkout --track origin/feature/fAA
(feature/fAA)$ git merge feature/fA
# delete my branch
(feature/fAA)$ git branch -d feature/fA
(feature/fAA)$ git push
```

2.1.2 Arbeiten an einem öffentlichen Feature

Es wird zunächst der featurebranch lokal auf den neusten Stand gebracht. Dann sollte man erstmal angucken, was sich in geändert hat. Nach getaner Arbeit wird `feature/*` aktualisiert und und entsprechend getestet. Erst dann wird `feature/*` auf das remote gepusht.

```
# update develop
(develop)$ git pull
# get featurebranch fA
# if not existing
    (develop)$ git checkout --track origin/feature/fA
# else
    (develop)$ git checkout feature/fA
    (feature/fA)$ git pull
# whats new?
(feature/fA)$ git log --since=1.weeks
# starting with Sprint and commit
(feature/fA)$ ...
(feature/fA)$ git commit
# update from repro
(feature/fA)$ git fetch
# while a new version of fA is online
    (feature/fA)$ git merge origin/feature/fA
    # testing after merge
    (feature/fA)$ ...
    (feature/fA)$ git commit
    (feature/fA)$ git fetch
# publish / push
(feature/fA)$ git push
```

2.1.3 Feature in den develop mergen

Das Feature ist aktuell und fertig geprüft, sodass es in den `develop` überführt werden kann. So wird zunächst der lokale `develop` geupdatet und anschließend in den `feature/*` eingefügt. Dann werden Tests durchgeführt, ggf. Konflikte gelöst und anschließend wird der `feature/*` in den `develop` einpflegt. Wenn `feature/*` nicht mehr gebraucht wird, so wird dieser vom lokalen und remote gelöscht.

```
(feature/fA)$ git checkout develop
# Update develop
(develop)$ git pull
# do :
    #switch to feature
    (develop)$ git checkout feature/fA
    # merge devolop into feature
    (feature/fA)$ git merge develop
    # testing
    (feature/fA)$ ...
    (feature/fA)$ git commit
    (feature/fA)$ git checkout develop
    # update develop again
    (develop)$ pull
# while ( develop changes) -> do
(develop)$ git merge feature/fA
# if feature is no longer needed
    # delete lokal
    (develop)$ git branch -d feature/fA
    # delete remote
    (develop)$ git push origin :feature/fA
    (develop)$ git branch -dr origin/feature/fA
```

2.1.4 Realese

Zu einer gewissen Zeit wird dem `master` der `develop` einverleibt. Dabei sollte sich auf `develop` ein vorzeigbarer Snapshot des Projektes befinden. Gegebenen falls kann man hier dem Commit einen Tag geben. Es wird aber ein kommentierter Tag bevorzugt.

```
# update develop
(develop)$ git pull
# update master
(develop)$ git checkout master
(master)$ git pull
# merge develop into master
(master)$ git merge develop
# publish
(master)$ git push
# tagging
(master)$ git tag -a v0.1
(master)$ git push origin v0.1
```

3 Konventionen

An diese Regeln sollte sich jeder Entwickler halten. Diese Regeln sorgen dafür, dass sich der Workflow einstellen kann und es zu keinen komischen Ausfällen kommt.

- update before you work (update = pull = fetch + merge)
- update before you merge and push
- never rebase in an public branch
- never use git commit –amend to a published commit
- write commit messages in an editor with a template
- all feature branches start with `feature/`

3.1 Git Commit Message

Es wird die Grundstrukur der AngularJS Communitiy [AngularJS, 2017] verwendet und auf unsere Bedürfnisse angepasst. Die Commit messages werden in Englisch verfasst.

3.1.1 Struktur / template

Ein Template ist in der Datei `.gitmessage` im Wurzelverzeichnis unseres Projektes und wird mit `git config commit.template ".gitmessage"` eingepflegt. Diese Vorlage wird in deinen Editor geladen, wenn ein Commit ohne "-m" Flag aufgerufen wird.

```
<type>(<scope>): <subject>
BLANKLINE
<body>
BLANKLINE
<footer>
--- EXAMPLE ---
docs(RDD): Add stakeholder list

Stakeholder are a hint, how we have to
designe our Project and how we have to
behave.

Close stakeholder card
```

head: Hier werden die wichtigesten Informationen zusammengefasst
- Er darf maximal 50 Zeichen lang sein

<type>: Typ des Commits, welche da sind:
- docs: Änderungen in der Documentation
- feat: generell Änderung am Produktcode
- test: Tests: keine Änderung am Produktcode
- refac: Refactoring am Produktcode
- fix: gefixte Bugs
- style: änderung am Style (Einrückung etc.)

<scope>: optional
- Datei, Module, Layer auf die sich die Änderung bezeiht
- Wenn mehrere betroffen sind: (*) und Liste im <body>

<subject>: Was bewirkt die Änderung im Kern?
- " If applied, this commit will <subject> "
- Imparativ und Präsens
- Beginnend mit einem Großbuchstaben
- Endet ohne Punkt

<body>: Warum wurde die Änderung gemacht?
- Imparativ und Präsens.
- Listen werden mit [-] unterteilt (z.B. betroffene Datein).
- Maximal 72 Zeichen pro Zeile.

<footer>: optional
- Tickets oder Referenzen zu Artikeln.
- BREAKING CHANGES: kurze Beschreibung.
- Maximal 72 Zeichen pro Zeile.

4 Cheat sheet

Local Changes

```
# List changed files in workingdir
git status

# Show changes to tracked files
# repro vs. working dir
git diff

# Stage current changes for commit
git add *

# Add just some changes
git add -p <file>

# Commit all staged files
git commit

# Commit alle changed tracked files
git commit -a

# Commit with detail info (diff)
git commit -v

# Change last commit
# to rewrite the commit msg
# (forbitten for  pubished commits)
git commit --amend

# Remove files (modifiations)
git rm <file>

# Rename files
git mv <form> <to>

# Remove files from stage
git rest HEAD <file>

# Undo all modifications in working dir
git checkout -- <file>

# Revert a Commit (makes a new commit)
git revert <commithash>
```

History and info

```
# Show all commits
git log

# Show changes for one file
git log -p <file>

# Show changers of file
git blame <file>
```

Config and Init

```
# set Username and adress
git config --global user.name <name>
git config --global user.email <email>

# Clone an existing repo
git clone <adr to repro>

# Create a new local repo
git init
```

Branches and Merge

```
# List all existing branches
git branch -av

# Switch to branch (set HEAD pointer)
git checkout <branch>

# Create a new branch
(base)$ git branch <new>

# Create a new trackingbranch from remote
git checkout --track <remote/branch>

# delete local branch
git branch -d <branch>

# Merge branch into base
(base)$ git merge <branch>

# use mergetool to resolve conflicts
git mergetool
```

Remote

```
# Show info of remote
git remote show <remote>

# Add new remote
git remote add <alias> <url>

# Download all changes from remote
# without merge them into HEAD
git fetch <remote>

# Download all changes from remote
# and merge them directly
git pull <remote>

# Publish local changes to remote
# Create a new branch
git push <remote> <branch>

# Delete branch on remote
git branch -dr <remote/branch>
```

5 Tipps und Tricks

5.1 Autocompletion

Mit einem kleinen Tool, kann man mit `TAB` Gitbefehle vervollständigen lassen. Unter Windows ist dies meist schon im Git-Packet enthalten. Linuxer mit bash können das mit folgendem Vorgehen einrichten.

```
# download in to $HOME
$ cd ~
$ curl -OL https://raw.githubusercontent.com/git/git/master/contrib/completion/git-completion.
    bash
$ mv ~/git-completion.bash ~/.git-completion.bash
```

Ändere deine `./bashrc`, indem du Folgendes einfügst:

```
if [-f ~/.git-completion.bash];then
  source ~/.git-completion.bash
fi
```

5.2 Alias

Setze aliase wo immer du kannst.

Beispiele:

```
# nice logg
git config --global alias.logg "
log␣--graph␣--decorate␣--oneline␣--no-merges␣--all"

# diff Stage vs HEAD
git config --global alias.difs "diff␣--staged"

# update develop
git config --global alias.updev "
!git␣checkout␣develop␣&&␣git␣pull␣&&␣git␣checkout␣-
&&␣:"
```

Einfacher ist es aber direkt in die Datei `/.gitconfig` reinzuschreiben.

5.3 ssh-key

Lege dir einen SSH-key an, um nicht immer wieder bei jedem `push` deinen Namen und Passwort einzugeben.

Die Anleitung findet man bei den jeweiligen Anbietern der Remoteserver: bitbucket

5.4 Tools

Setze Tools ein und lege sie fest:

```
git config --global core.editor vim
git config --global merge.tool vimdiff
```

Und rufe ggf. die Hilfe von git auf:

```
git help <cmd>
git <cmd> --help
man git -<cmd>
```

5.5 Dose vs Unix

Das alte Lied von den Zeilenenden. Um dies zu vermeiden, setze folgendes um:

```
# UNIX
git config --global core.autocrlf input
# WINDOS
git config --global core.autocrlf true
```

5.6 persönliches .gitignore

Da fast jeder Entwickler seine eigenen Werkzeuge verwendet, sollten die Nebenprodukte dieser nicht von Git gesehen werden.

Um dies zu verhindern, sollte eine Datei `/.global-gitignore` erstellt und git mit
`git config -global core.excludesfile ' /.global-gitignore'`
eingepflegt werden.

Gute Templates findet man unter github.

Literatur

AngularJS. Contributing to AngularJS. https://github.com/angular/angular.js/blob/master/CONTRIBUTING.md#commit (2017-04-28), 2017.

Scott Chacon and Ben Straub. *Pro Git*. Apress, Berkely, CA, USA, 2nd edition, 2014. ISBN 1484200772, 9781484200773. https://git-scm.com/book/en/v2 (2017-04-28).

8.4 Coding-style

C++ Coding Style

LANKE

2. Mai 2017

Um die Garantie zu geben, dass die Qualität des Quellcodes eigehalten wird, ist eine Richtlinie für den Code zu schreiben, unabdingbar. Auf Basis von den Codebeispielen der Professoren, unseren Erfahrungen und allgemeinen Vereinbarungen in der Programmierer Szene ist hier für das Praktikum ESE eine solche Richtline erstellt worden.

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument legt die Konventionen für den Quellcode fest. Zuerst wird das Sprachbild (die Namensgebung, das Layout und die Struktur im Code) festgelegt. Danach wird über das Dokumentieren gesprochen. Das Testen eines Modules wird abschließend besprochen.

2 C++ Sprachbild

2.1 Namengebung

Alle Namen sind Englisch.
Typen: **C**amel**C**ase
- Klasse (class)
- Struktur (struct)
- Aufzählung (enum)
  - Singular (Status statt Stati)
- typedef

Variablen: **c**amel**C**ase
private Attribute: **c**amel**C**ase
- suffix_

Konstanten: **UPPER_CASE**
Funktionen: Verb + Nomen, **c**amel**C**ase
Flags: **c**amel**C**ase
- **is**Prefix,
- keine Negation
- > isRunning anstatt isNotRunning

```
#define UNIVERCITY_NAME "HAW Hamburg"
class UniAccount
{
  public:
    Status getStatus(void) ;

  private:
    Status status_ ;
    bool isStudent_ ;
}

enum Status
{
   STUDENT
  ,PROF
}
```

2.2 Layout

white space

Einrückung: für jeden Block
- Leerzeichen statt Tabs
- 2 oder 3 Leerzeichen

Zeilen zwischen... :

Funktionen: eine Leerzeile

logische Aufteilung: Zeilenkommentar statt Leerzeile

Lesbarkeit: ist das oberste Ziel
- **Klammern:** je nach Lesbarkeit
- **Operantionen:** zwischen Operanten und Operatoren ein Leerzeichen
- **Listen:** hinter jedem Komma ein Leerzeichen (Ausnahme: wenn Liste unteinander geschrieben wird)
- **Zuweisungen:** es dürfen mit Leerzeichen Tabellen gemimt werden.

Blöcke

öffnende Blockklammer: {
- **Typen:** in neuer Zeile
- **Schleifen:** direkt dahinter oder neue Zeile
- **Bedingungen:** direkt dahinter oder neue Zeile

schließende Blockklammer: }
- in der Einrückebene, in der sie geöffnet wurden
- sind einziges Zeichen in Zeile

Zeilenumbruch

Zeilenzeichenlimit: 80 Zeichen

Deklaration: je eine Zeile
- wenn das Zeichenlimit pro Zeile nicht reicht...
  - **Parameter-/Argumentenliste:** wo die Klammer sich öffnet
  - **Zuweisung, Berechnung:** wo = beginnt

```
1  class UniAccount
2  {
3    public:
4      void work(int time){
5        // get infos
6        int freeTime      = freeTime_ ;
7        Status motivation = getMotivation();
8        // test if she/he is capable to work
9        if (freeTime > time && motivation => OK){
10         workTime_ += time;
11         freeTime_ -= time;
12       }
13       return;
14     }
15
16     int calcSomthing ( int a, int b, int c,
17                        int d, int e, inf f
18     ){
19       int thisReturnValueNameLong = a + b + c
20                                    + d + e + f;
21       return thisReturnValueNameLong;
22     }
23
24   private:
25     workTime_ ;
26     freeTime_ ;
27 }
28
29 enum Status
30 {
31    DEPRESSED
32   ,OK
33   ,MOTIVATED
34 }
```

3 Struktur

Module: Teile Implementation vom Interface
- **Module.h:** Interface
  - verwende include guards (MODULE_H_)
  - Deklarationen
  - Templates
  - inline function Definitionen
- **Module.cpp:** Implementation
  - Funktionsdefinitionen
  - strikte innere Klassen
- **Ausnahme:** Eigene Bibliotheken sind in einem Header definiert.

Weiteres: Was soll noch eingehalten werden.
- keine magic numbers (lieber Konstanten)
- vermeide namespaces
- order of includes: most spacific first
- wenn du einen Header nur im cpp file benötigst, füge diesen auch nur dort ein
- wenn möglich, verwende forward declaration
- erstelle den Ctor
- verwende RAII (Ressource hängt von Lebzeit des Objekts ab)
- versuche die rule of three
  - Konstruktor und passender Destruktor
  - Kopierkonstruktor
  - assigment operator

Module.h

```
#ifndef MODULE_H_
#define MODULE_H_

class SomeClass;

class Module
{
  public:
    // rule of three
    Module(SomeClass) ; // ctor
    virtual ~Module() ; // destructor
    Module(const Module&) ; // copy constructor
    Module& operator= (const Module&); //
    assigment operator

    void printSomething(void) const;

  private:
    SomeClass &handle_ ;
}
#endif /* MODULE_H_ */
```

Module.cpp

```
#include "Module.h"
#include "SomeClass.h"

#include <iostream>

using namespace std;

// RAII begin
Module::Module(SomeClass cl)
: handle_(cl)
{
  //ctor
}

Module::~Module(){
  release (handle_);
}
// RAII end

Module::printSomething(void){
  cout << handle_->getSomthing() << endl;
}
```

4 Dokumentation

CODE DOKUMENTATION

Die Dokumentation wird mit dOxygen erzeugt. Dafür wird mit Tags auf Informationen gesammelt. DOxygen holt sich die Informationen aus nebenstehenden Kommentarstrukturen. Funktionen und Typen werden mit dem Block erklärt. Variablen, Attribute und Konstanten werden mit /**< ...*/ beschrieben. Die Beschreibung erfolgt fast ausschließlich in der Headerdatei des Modules. Erstellt wird die Dokumentation mit den Befehlen:

```
$ doxygen <config-file>
```

```
/** @file <filename>
 *  @brief <short description of module>
 *
 *  <detailes, longer explaination, pattern, componente>
 *  @author <author 1>
 *  @author <author 2>
 */

/**
 * @class <description of class>
 */
class Module
 {
 ...
/**
 *  @details description of function
 *  @param a <description of 1st parameter>
 *  @param b
 *       <description of 2nd parameter>
 *  @return <description of retrunvalue>
 */
int  func(int a, int b);

/// single line

int x_; /**< decription of x */
```

Die verwendetet Tags für dieses Projekt sind:

tag	Rendering
@file	Name des Files
@brief	kurze Beschreibung des Moduls
@author	Name des Authors
@class @enum @struct	Beschreibung des Typen
@details	kurze Beschreibung der Funktion
@param <par>	Beschreibung des Parameters <par> in Funktionen
@return	Beschreibung des Rückgabewertes

LIZENZ

Unser Project läuft unter der MIT Lizenz, welche in der Textdatei `LICENSE.txt` beschrieben ist. Jedes File muss den folgenden Text im header haben.

```
/**
 *   ...
 *  Embedded System Engineering SoSe 2017
 *  Copyright (c) 2017  LANKE devs
 *  This software is licensed by MIT License.
 *  See LICENSE.txt for details.
 */
```

5 Testen

Wie ein Unittest aussieht, ist jedem Entwickler freigestellt. Er hat somit die alleinige Verantwortung, dass sein Code richtig funktioniert. Die Testfiles müssen in dem Unterverzeichnis `test` abgelegt werden, und sollten im Team veröffentlicht werden. Die Tests sollten alle Ausnahmefälle und die Funktion des Modules Testen. Eventuell werden auch mehrere Komponenten gleichzeitig getestet. Bevor es an die Hardware geht, sollte der Code in Software reibungslos laufen.

Mögliche Testwerkzeuge sind einmal ein Testmodul mit Mainfunktion oder man verwedet C++Unit.

kurze Empfehlungen

Wenn Pattern verwendet werden, sollten die Namen der Pattern bzw. deren Komponenten in den Namen der Klassen wiederzufinden sein.

8.5 Hal

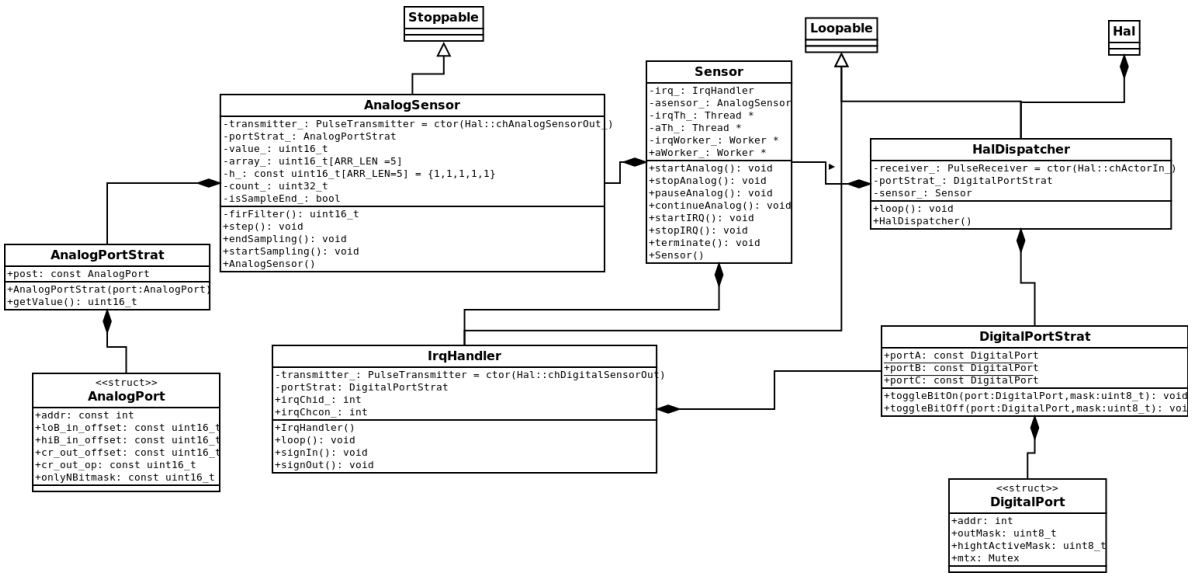


Abbildung 20: Hal Klassendiagramm: Dispatcher wertet Eingänge aus der Mcl aus, die DigitalPort-Strat verwedet informationen aus der Entity DigiPort um auf die Hardware zuzugreifen. Der Sensor ist ein Multiplexer zu der IRQ und dem Höhensensor (AnalogSensor).

8.6 Serial

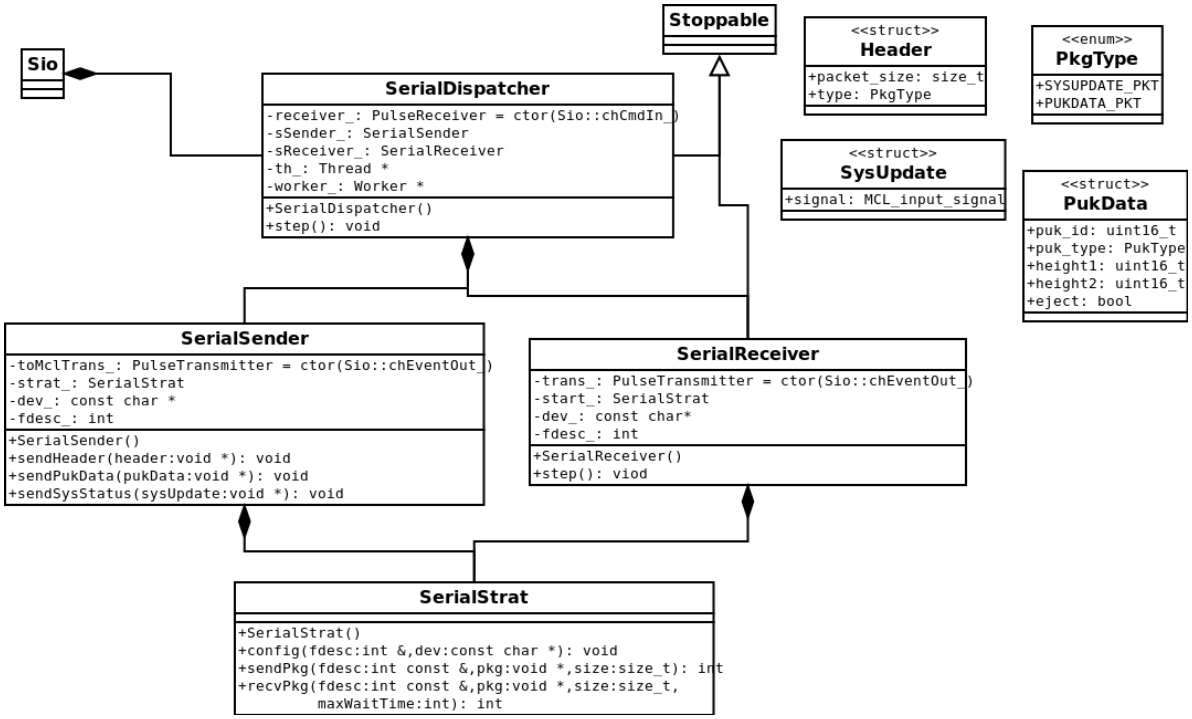
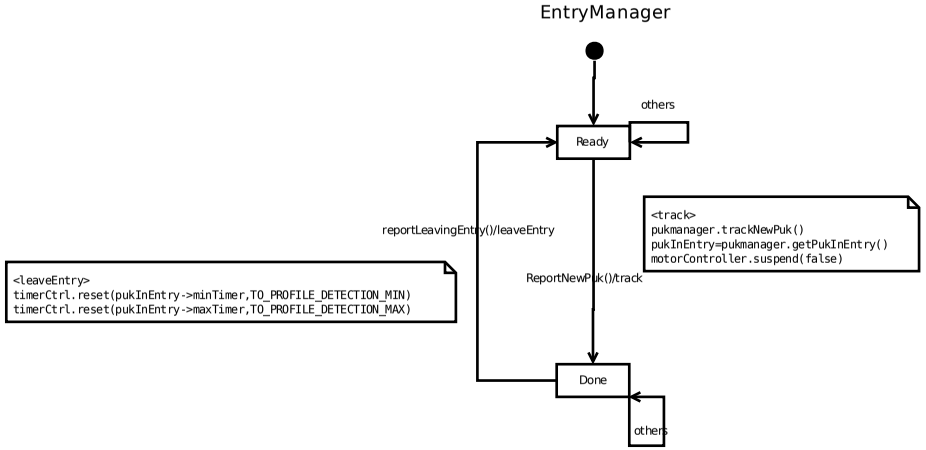
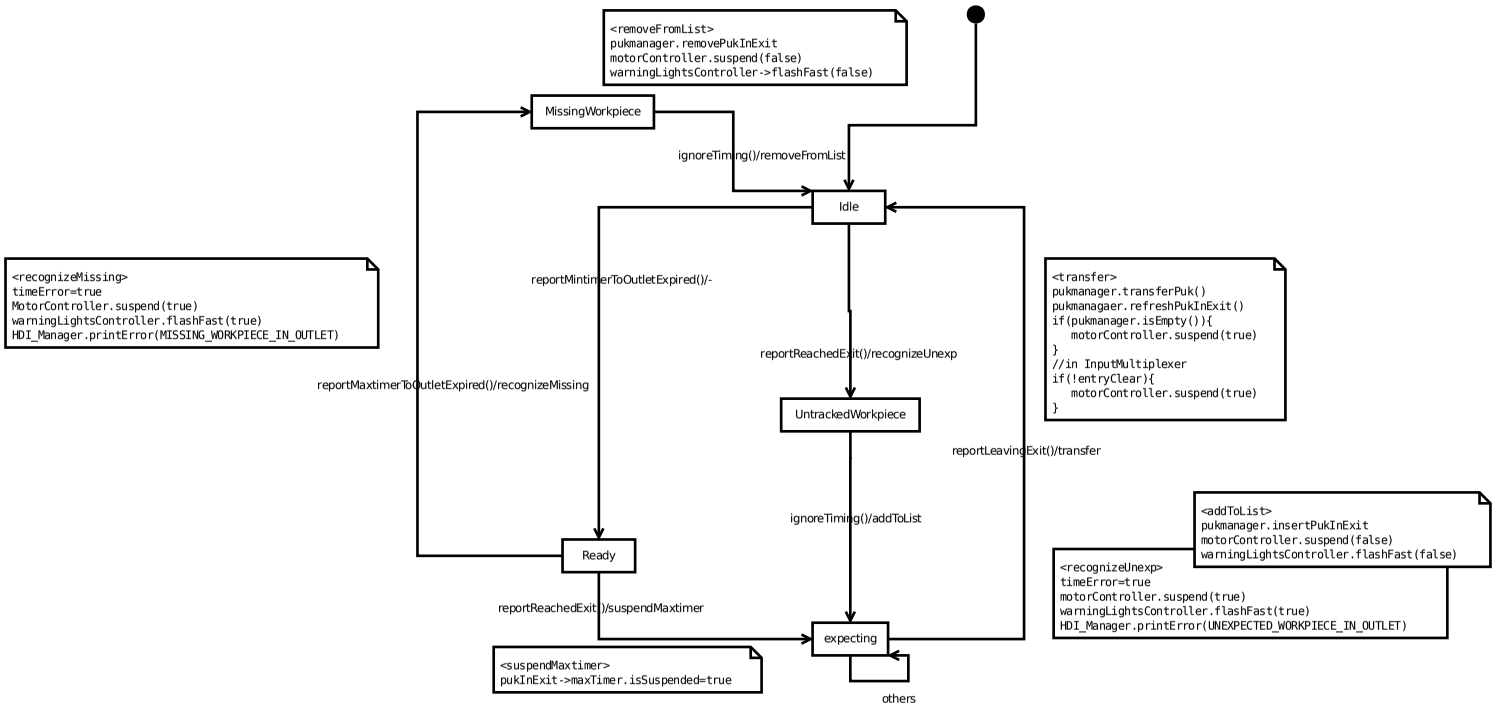


Abbildung 21: Serial Klassendiagramm: Dispatcher wertet Eingänge aus der Mcl aus, die SerialSender und SerialTransmitter interagieren über SerialStrat mit dem Betriebsystem zum Nachbarsystem. Oben rechts stehen die Protokollstrukturen.

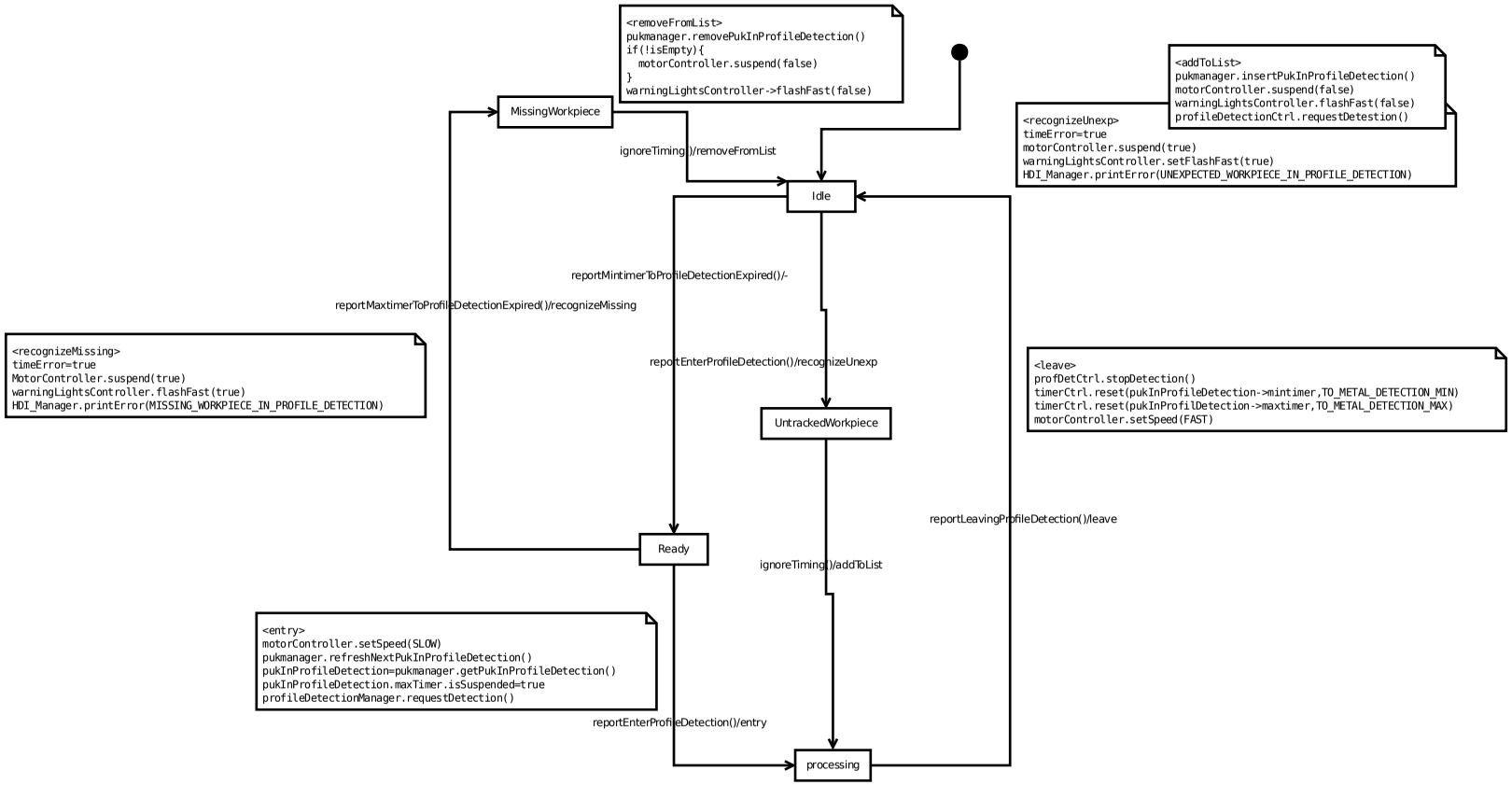
8.7 State Machines

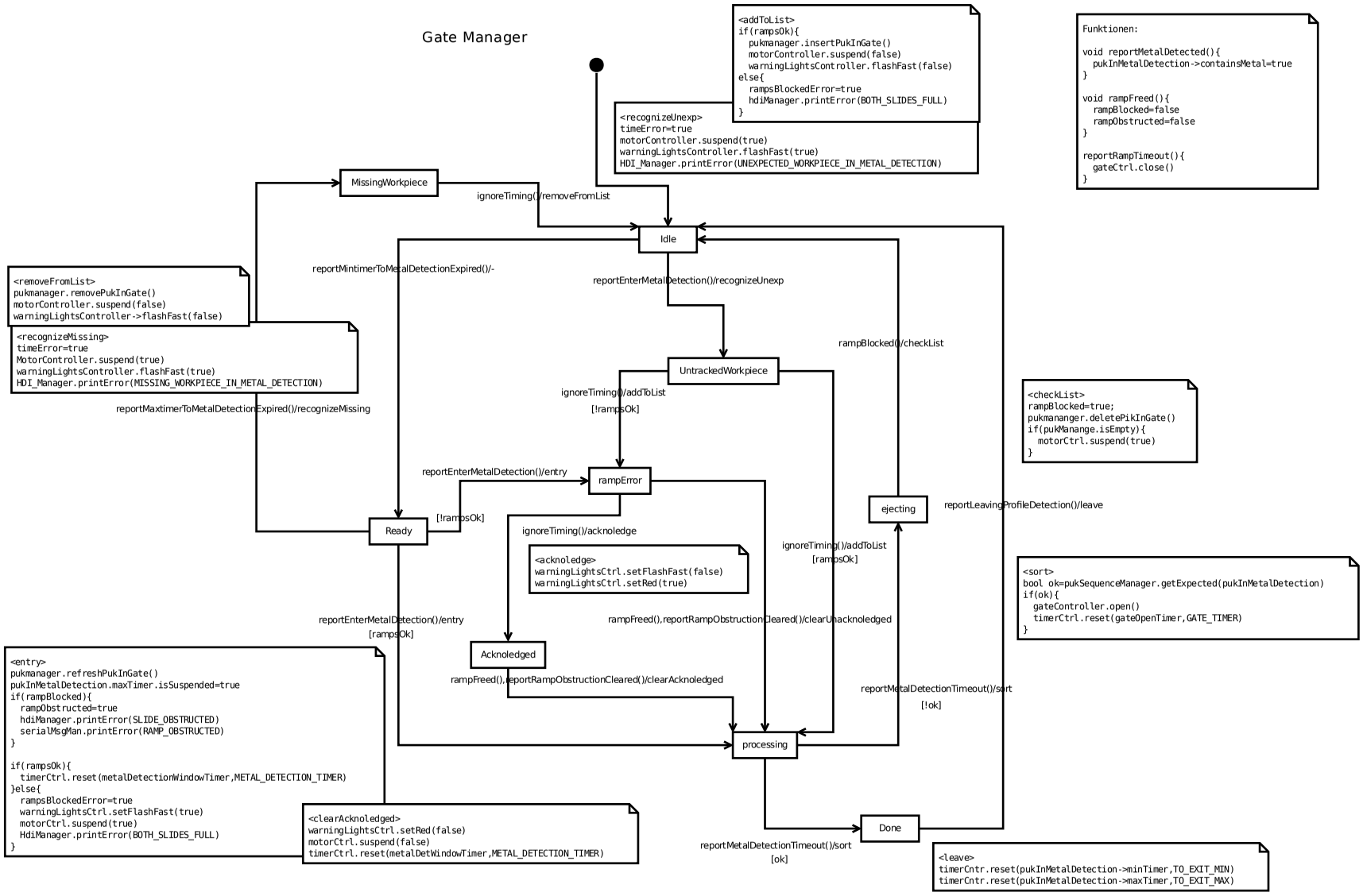

EntryManager
others
Ready
reportLeavingEntry()/leaveEntry
<track>
pukmanager.trackNewPuk()
pukInEntry=pukmanager.getPukInEntry()
motorController.suspend(false)
<leaveEntry>
timerCtrl.reset(pukInEntry->minTimer,TO_PROFILE_DETECTION_MIN)
timerCtrl.reset(pukInEntry->maxTimer,TO_PROFILE_DETECTION_MAX)
ReportNewPuk()/track
Done
others

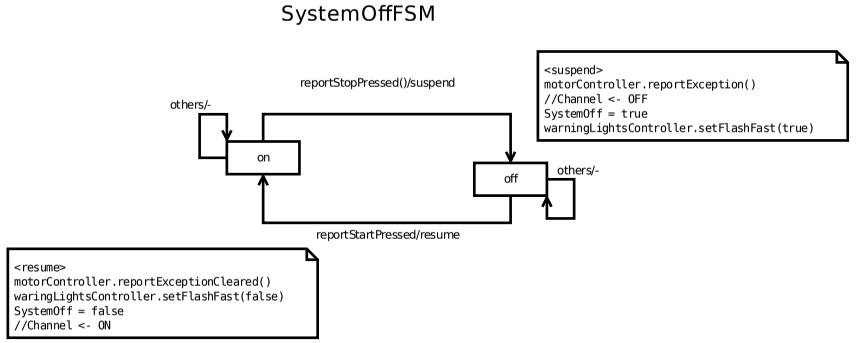
OutletManager



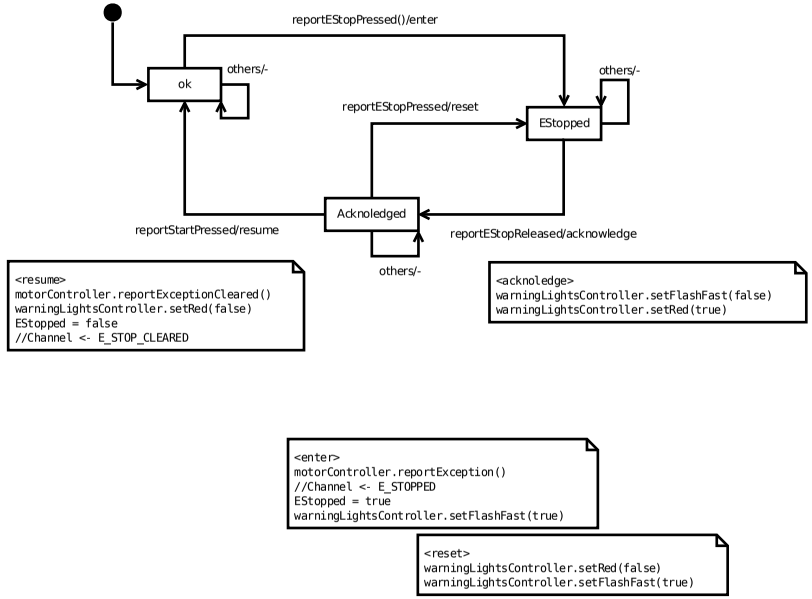
ProfileDetectionManager




Gate Manager
<addToList>
if(rampsOk){
pukmanager.insertPukInGate()
motorController.suspend(false)
warningLightsController.flashFast(false)
else{
rampsBlockedError=true
hdiManager.printError(BOTH_SLIDES_FULL)
}
Funktionen:
void reportMetalDetected(){
pukInMetalDetection->containsMetal=true
}
void rampFreed(){
rampBlocked=false
rampObstructed=false
}
reportRampTimeout(){
gateCtrl.close()
}
<recognizeUnexp>
timeError=true
motorController.suspend(true)
warningLightsController.flashFast(true)
HDI_Manager.printError(UNEXPECTED_WORKPIECE_IN_METAL_DETECTION)
MissingWorkpiece
ignoreTiming()/removeFromList
Idle
reportMintimerToMetalDetectionExpired()/-
reportEnterMetalDetection()/recognizeUnexp
<removeFromList>
pukmanager.removePukInGate()
motorController.suspend(false)
warningLightsController->flashFast(false)
<recognizeMissing>
timeError=true
MotorController.suspend(true)
warningLightsController.flashFast(true)
HDI_Manager.printError(MISSING_WORKPIECE_IN_METAL_DETECTION)
reportMaxtimerToMetalDetectionExpired()/recognizeMissing
rampBlocked()/checkList
UntrackedWorkpiece
ignoreTiming()/addToList
[!rampsOk]
<checkList>
rampBlocked=true;
pukmananger.deletePikInGate()
if(pukMaange.isEmpty){
motorCtrl.suspend(true)
}
reportEnterMetalDetection()/entry
rampError
ejecting
reportLeavingProfileDetection()/leave
[!rampsOk]
Ready
ignoreTiming()/ackoledge
ignoreTiming()/addToList
[rampsOk]
<ackoledge>
warningLightsCtrl.setFlashFast(false)
warningLightsCtrl.setRed(true)
<sort>
bool ok=pukSequenceManager.getExpected(pukInMetalDetection)
if(ok){
gateController.open()
timerCtrl.reset(gateOpenTimer,GATE_TIMER)
}
reportEnterMetalDetection()/entry
[rampsOk]
rampFreed(),reportRampObstructionCleared()/clearUnackoledged
Ackoledged
<entry>
pukmanager.refreshPukInGate()
pukInMetalDetection.maxTimer.isSuspended=true
if(rampBlocked){
rampObstructed=true
hdiManager.printError(SLIDE_OBSTRUCTED)
serialMsgMan.printError(RAMP_OBSTRUCTED)
}
if(rampsOk){
timerCtrl.reset(metalDetectionWindowTimer,METAL_DETECTION_TIMER)
}else{
rampsBlockedError=true
warningLightsCtrl.setFlashFast(true)
motorCtrl.suspend(true)
HdiManager.printError(BOTH_SLIDES_FULL)
}
rampFreed(),reportRampObstructionCleared()/clearAckoledged
reportMetalDetectionTimeout()/sort
[!ok]
<clearAckoledged>
warningLightsCtrl.setRed(false)
motorCtrl.suspend(false)
timerCtrl.reset(metalDetWindowTimer,METAL_DETECTION_TIMER)
processing
Done
reportMetalDetectionTimeout()/sort
[ok]
<leave>
timerCntr.reset(pukInMetalDetection->minTimer,TO_EXIT_MIN)
timerCntr.reset(pukInMetalDetection->maxTimer,TO_EXIT_MAX)


SystemOffFSM
reportStopPressed()/suspend
<suspend>
motorController.reportException()
//Channel <- OFF
SystemOff = true
warningLightsController.setFlashFast(true)
others/-
on
off
others/-
reportStartPressed/resume
<resume>
motorController.reportExceptionCleared()
waringLightsController.setFlashFast(false)
SystemOff = false
//Channel <- ON

EStopFSM


reportEStopPressed()/enter
ok
others/-
others/-
reportEStopPressed/reset
EStopped
Acknoledged
reportStartPressed/resume
reportEStopReleased/acknowledge
others/-
<resume>
motorController.reportExceptionCleared()
warningLightsController.setRed(false)
EStopped = false
//Channel <- E_STOP_CLEARED
<acknoledge>
warningLightsController.setFlashFast(false)
warningLightsController.setRed(true)
<enter>
motorController.reportException()
//Channel <- E_STOPPED
EStopped = true
warningLightsController.setFlashFast(true)
<reset>
warningLightsController.setRed(false)
warningLightsController.setFlashFast(true)

8.8 Use-Case's

Ziel des Funktionstest	Durchführung	Beobachtung
Puck mit Bohrung nach unten erkennen	Legen Sie ein Puck mit Bohrung nach unten auf den Anfang von Band1.	die Typerkennung wird auf der Konsole ausgegeben und der Puck wird auf Band1 oder Band2 aussortiert
Pucks in richtiger Reihenfolge erkennen	Legen Sie mehrere Pucks, mit einem erlaubten Abstand, in gewünschter Reihenfolge, am Anfang, auf das Band1	die Typerkennung wird auf der Konsole ausgegeben und die Pucks die der gewünschten Reihenfolge entsprechen durchlaufen das Band2 bis zum Ende und auf der Konsole werden ID, Typ, Höhen-Messwert von Band1 und von Band2 ausgegeben
Pucks in falscher Reihenfolge erkennen	Legen Sie mehrere Pucks mit einem erlaubten Abstand, in nicht gewünschter Reihenfolge, am Anfang, auf das Band1.	die Typerkennung wird auf der Konsole ausgegeben und die Pucks die nicht der gewünschten Reihenfolge entsprechen werden auf Band2 aussortiert, nur Pucks welche der gewünschten Reihenfolge entsprechen durchlaufen das Band2 bis zum Ende
flachen Puck erkennen	Legen Sie einen flachen Puck, am Anfang, auf Band1.	die gelbe Lampe blinkt und der flache Puck wird auf Band1 aussortiert
beide Laufbänder sind leer	Legen Sie keinen Puck auf Band1, sodass sich auf einem band kein Puck befindet.	grüne Lampe geht an und alle leeren Laufbänder stehen still
Puck verschwindet	Entfernen Sie einen Puck vom Laufband	beide Laufbänder stoppen und eine Fehlermeldung wird ausgegeben
Puck irregulär hinzugefügt	Legen Sie einen Puck nicht wie gewünscht, am Anfang von Band1 ein, sondern an einem andern Punkt des Laufbandes	beide Laufbänder stoppen und eine Fehlermeldung wird ausgegeben
beide Rutschen sind voll	Legen Sie so viele Pucks ein, dass beide Rutschen voll sind und dadurch kein Platz ist ein weiteren Puck auszusortieren	beide Laufbänder stoppen und eine Fehlermeldung wird ausgegeben
Laufband einschalten	Betätigen Sie die Ein Taste, wenn das Laufband aus ist	die Anlage schaltet sich ein, der Testlauf startet und die LED für die Ein Taste wird eingeschaltet
Laufband- Stopp	Betätigen Sie die Stopp Taste, wenn das Laufband an ist	Laufband Stoppt
Fehler Quittierung	Betätigen Sie die Reset Taste, wenn das Laufband an ist und einen Fehler meldet, um den Fehler zu Quittieren	Laufband wird neu gestartet

Ziel des Funktionstest	Durchführung	Beobachtung
Neustart nach E-Stopp	Betätigen Sie die Reset Taste um das Laufband nach einem E-Stopp gestoppt wurde und der E-Stopp- Schalter wieder zurückgestellt wurde	Laufband läuft wieder los und die grüne Lampe wird wieder eingeschaltet und die LED für die Ein/ Aus Taste wird eingeschaltet

8.9 Abnahmetest

Abnahmetests

Betriebsmodi

Die Betriebsmodi sind zuerst durch Betätigung der Tasten zu testen

- Anlage An
  Ausgelöst durch Betätigung des Start-Button
  Sigalisiert durch Grüne Ampel und Start-Tastenbeleuchtung
- Anlage Aus
  Ausgelöst durch Betätigung Stop-Button oder Ausgangszustand bei Inbetriebnahme/Initialisierung
  Band steht, Schranken geschlossen
- Anlage-Stop (e-stop)
  Ausgelöst durch Betätigung e-Stop oder
  Ausgelöst bei Fehlerfall im Betriebsablauf
  Auslösendes Ereignis wird auf der Konsole ausgegeben
  Band steht, Schranken geschlossen, Ampel blinkt Rot
  Zurücksetzen des Stop-Modus durch Reset-Button

Puk-Typen

Die Puktypen werden nach sensorisch eindeutiger Identifizierung d.h. spätestens bei der Metall-Erkennung auf der Konsole ausgegeben.
Ohne vorgegebene Sequenz sind die Prioritäten der Ziele der Puk-Typen in der folgenden Tabelle dargestellt.

Typ	**Resultat regulär**	**Gewünschte Rutsche voll**
Bohrung unten	Band 1 / Band 2	Band 2
Flach	Band 1 (und gelb blinkt)	Band 2 (und gelb blinkt)
Bohrung	Falls Sequenz falsch Band 2	-
Bohrung Metall	Falls Sequenz falsch Band 2	-
Typ 1 / 5	Band 1	Band 2
Typ 2 / 4	Band 2	Band 1
Unbekanntes Objekt	Band 2 / Band 1	Band 1 / Band 2

Vordefinierte Sequenz (vorläufig)

1. Bohrung Metall

2. Flach

3. Typ 1

In der Reihenfolge sollen die Werkstücke auf Band 1 aussortiert werden, andere Typen werden an das folgende Band gesendet, wenn dieses eine Freigabe erteilt.

Zu testen ist hier die Einhaltung der Sequenz durch verschiedene Reihenfolgen der Puk-Typen nach jedem Durchlauf.

Regulärer Betriebsablauf

Ausgangszustand: Laufbänder leer, Rutschen Leer

Typerkennung

> In Abstand von kompletten Druchläufen (Band steht ohne Fehler) alle Puk-Varianten in den Einlauf geben und Konsolen-Ausgabe prüfen

Korrekte Reihenfolge erkennen

> Abwechselnd einen Sequenz-Puk und einen anderen Puk in Stationsabständen in den Einlauf geben.
>
> Erwartung: Korrektes Aussortieren der Puks in Rutsche1 und Weiterleitung an Band 2 der anderen Puks

Fehlerfälle

Zeitverletzungen

Falscher Gegenstand oder Puk erreicht verfrüht seine nächste Station.

> Auslösen des nächsten In-Position-Sensors
>
> Laufbänder stoppen, Fehlermeldung

Blockade oder vermisster Puk

> Auslösen durch Verzögerung / Wegnahme eines Puks
>
> Laufbänder stoppen, Fehlermeldung

Rutsche voll, obwohl Sequenz nicht komplett

> Auslösen durch manuelles Einlegen von Puks in Rutsche 1 bis Rutsche voll

Beide Rutschen sind voll

> Durch normalen Betrieb vollgelaufene Rutschen
>
> Laufbänder stoppen, Fehlermeldung