Requierements and Design Documentation

zur „Werkstück-Sortieranlage“

Software Engineering 2 – Projekt

Team:

Rico Flaegel

Jan Quenzel

Tell Müller-Pettenpohl

Torsten Krane

Abgabe:

23.06.2011

Praktikumsbetreuer:

Prof. Dr. Zhen Ru Dai

Inhaltsverzeichnis

1. Requirements und Analysemodell

1.1 Anwendungsszenarien

Werkstücke werden in gewissen Zeitabständen aufs Band gelegt und Sensoren sortieren bestimmte Werkstücke aus.

Diese Werkstücke können auf das Band gelegt werden:

- mit richtiger Höhe
  - mit Metalleinsatz
    - Öffnung nach oben
    - Öffnung nach Unten
  - ohne Metalleinsatz
    - Öffnung nach oben
    - Öffnung nach Unten
- mit falscher Höhe

Folgende Sortierungen sollen erfolgen:

Höhe	Metall	Öffnung	Aussortieren
richtige Höhe	Mit Metalleinsatz	Öffnung nach oben	nein
		Öffnung nach Unten	nein
	Ohne Metalleinsatz	Öffnung nach oben	Ja (Band 2)
		Öffnung nach Unten	nein
mit falsche Höhe			Ja (Band 1)

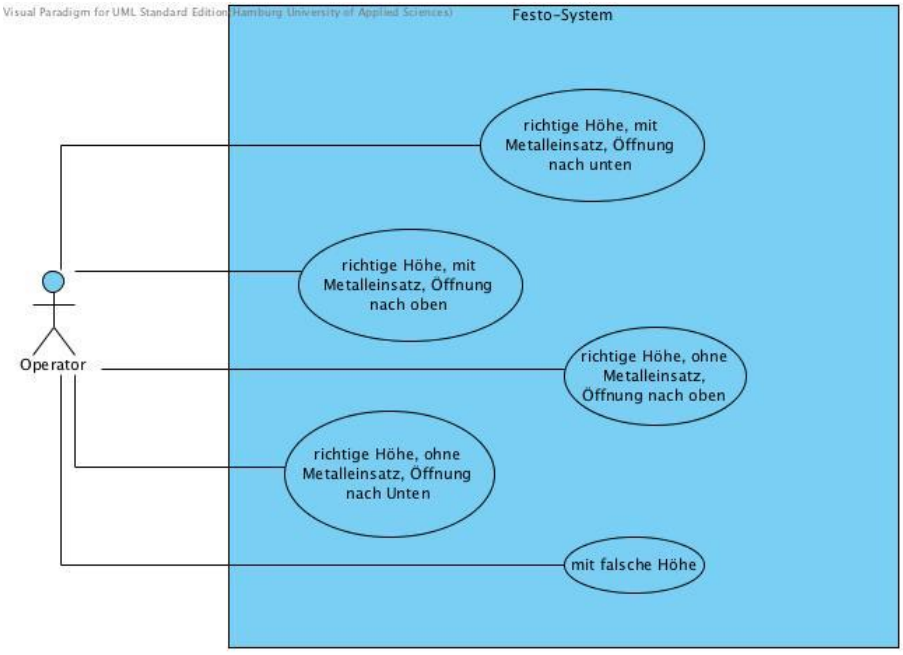

Bild 1.1.1 Sortierungen – UseCaseDiagram

Szenario 1:

Ein auszusortierendes Werkstück wird auf das Band gelegt.

Fall I: Werkstück mit falscher Höhe

Erste Lichtschranke B1(0) erkennt das Werkstück.
Band 1 läuft.
Zweite Lichtschranke B1(1) erkennt das Werkstück.
Sensor B1(2) erkennt Werkstück mit falscher Höhe.
Band 1 läuft.
Lichtschranke B1(6) erkennt das Werkstück.
Werkstück fällt in den Auswurfschacht 1.

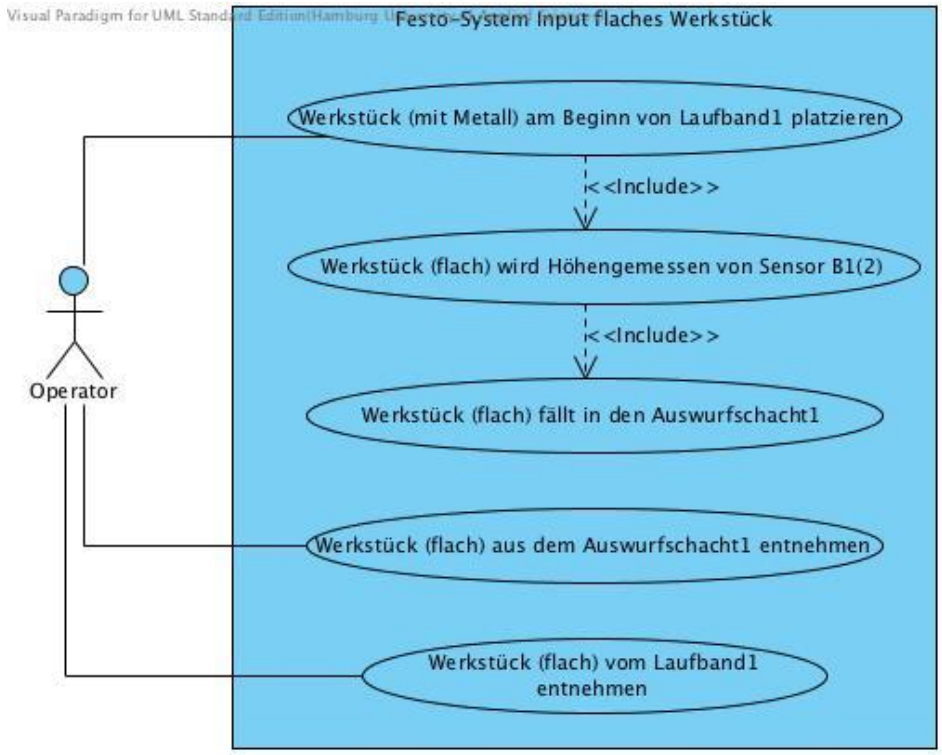

Bild 1.1.2 Szenario 1 – UseCaseDiagram

Lichtschranke B1(0) erkennt das Werkstück
Band 1 läuft
Lichtschranke B1(1) erkennt das Werkstück
Sensor B1(2) erkennt Werkstück mit richtiger Höhe und Bohrung nach oben
Band 1 läuft
Lichtschranke B1(3) erkennt Werkstück
Weiche A1(4) wird auf Durchgang geschaltet
Band 1 läuft
Lichtschranke B1(7) erkennt das Werkstück
Band 1 und 2 läuft
Lichtschranke B2(0) erkennt das Werkstück
Band 2 läuft
Lichtschranke B2(1) erkennt das Werkstück
Band 2 läuft
Lichtschranke B2(3) erkennt das Werkstück
Induktiver Sensor B2(4) erkennt kein Metall
Band 2 läuft
Lichtschranke B2(6) erkennt das Werkstück
Werkstück fällt in den Auswurfschacht 2

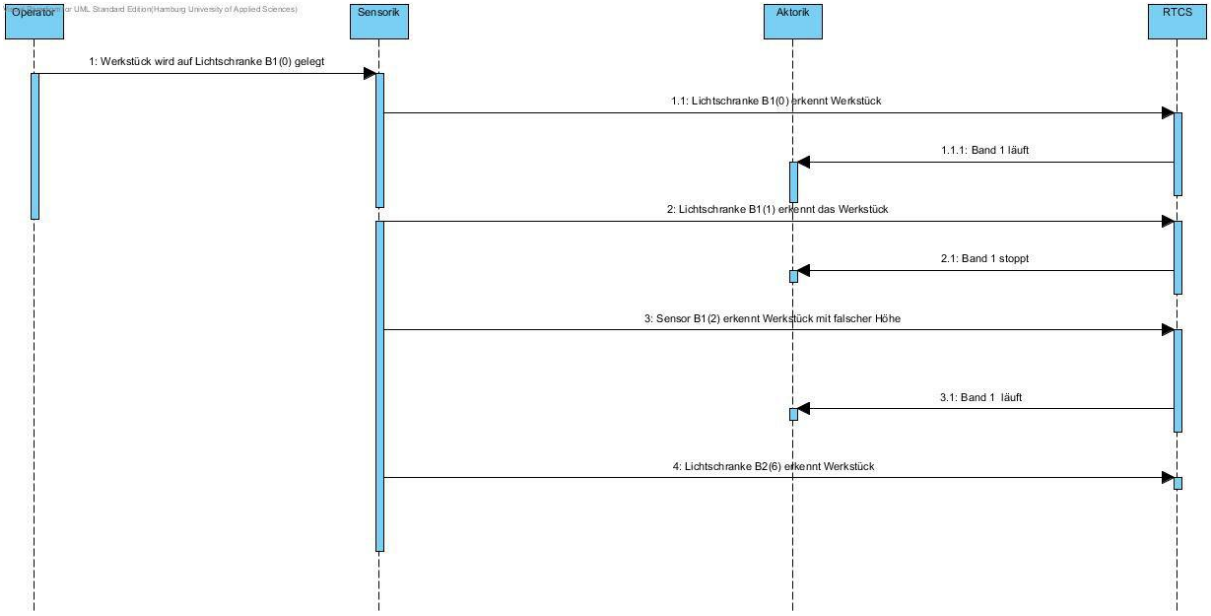

Bild 1.1.3 Szenario 1 – Sequenzdiagram

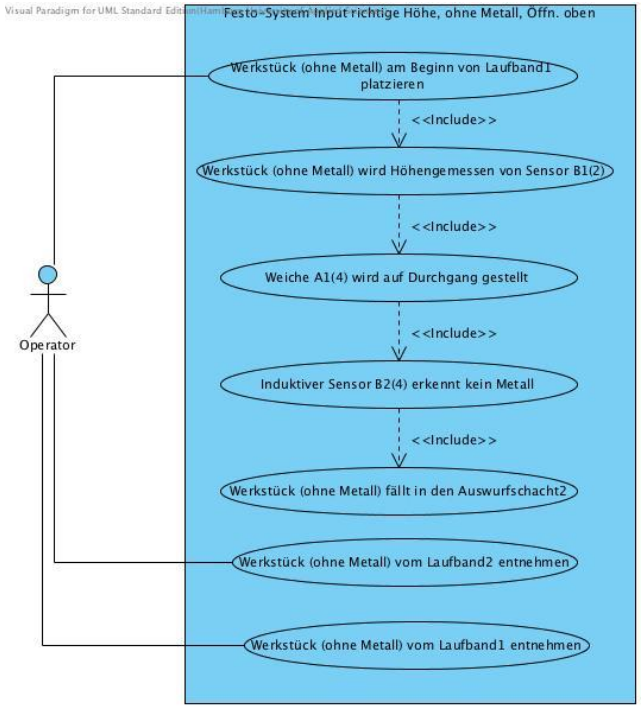

Bild 1.1.4 Szenario 1 – UseCaseDiagram

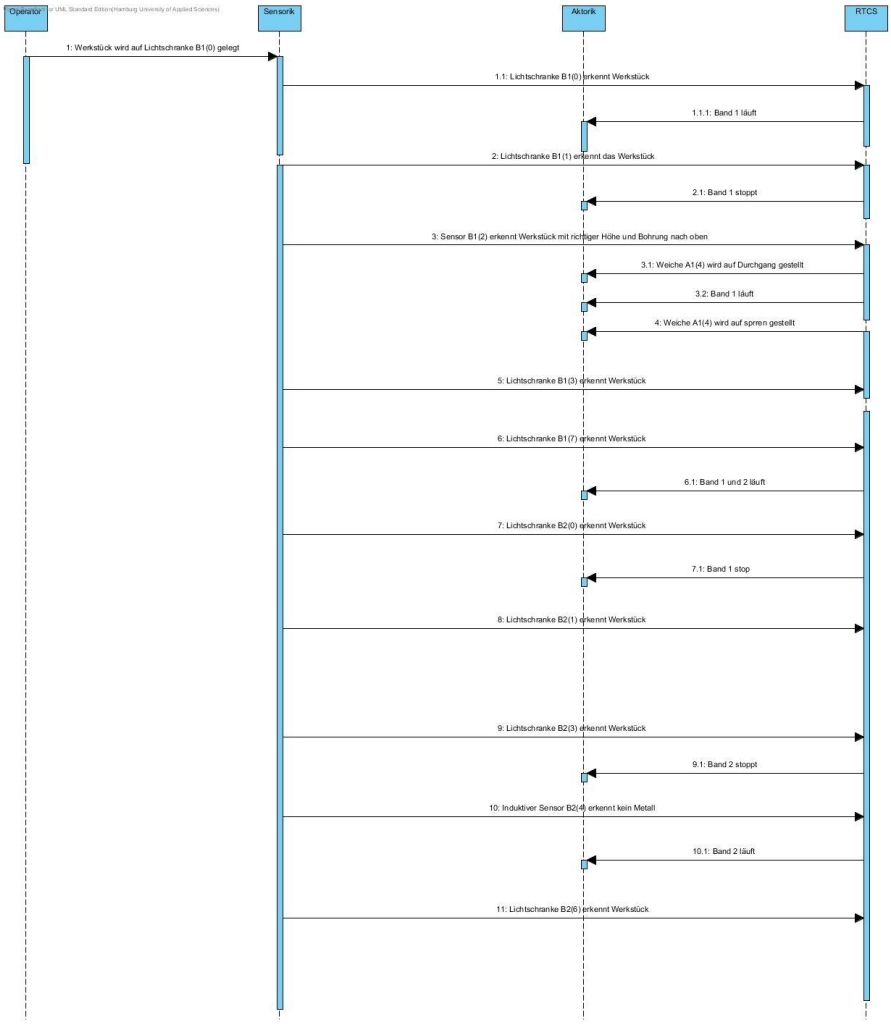

Bild 1.1.5 Szenario 1 – Sequenzdiagram

Szenario 2:

Ein nicht auszusortierendes Werkstück wird aufs Band gelegt

Fall I: Werkstück mit richtiger Höhe, mit Metall und Bohrung nach oben

Lichtschranke B1(0) erkennt Werkstück
Band 1 läuft
Lichtschranke B1(1) erkennt das Werkstück
Sensor B1(2) erkennt Werkstück mit richtiger Höhe und Bohrung nach oben
Band 1 läuft
Lichtschranke B1(3) erkennt das Werkstück
Weiche A1(4) wird auf Durchgang gestellt
Band 1 läuft
Lichtschranke B1(3) erkennt Werkstück
Band 1 läuft
Lichtschranke B1(7) erkennt Werkstück
Band 1 und 2 läuft
Lichtschranke B2(0) erkennt Werkstück
Band 2 läuft
Lichtschranke B2(1) erkennt Werkstück
Band 2 läuft
Lichtschranke B2(3) erkennt Werkstück
Induktiver Sensor B2(4) erkennt Metall
Weiche A2(4) wird auf Durchgang gestellt
Band 2 läuft
Lichtschranke B2(3) erkennt Werkstück
Band 2 läuft
Lichtschranke B2(7) erkennt Werkstück

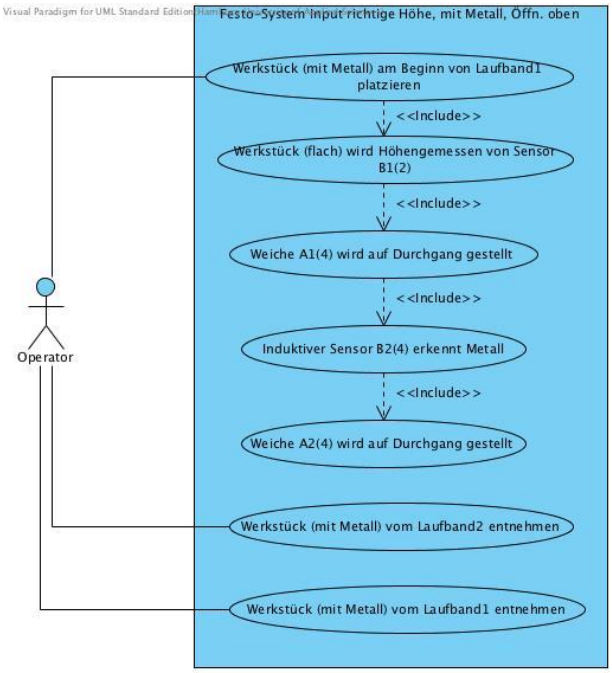

Bild 1.1.6 Szenario 2 – UseCaseDiagram

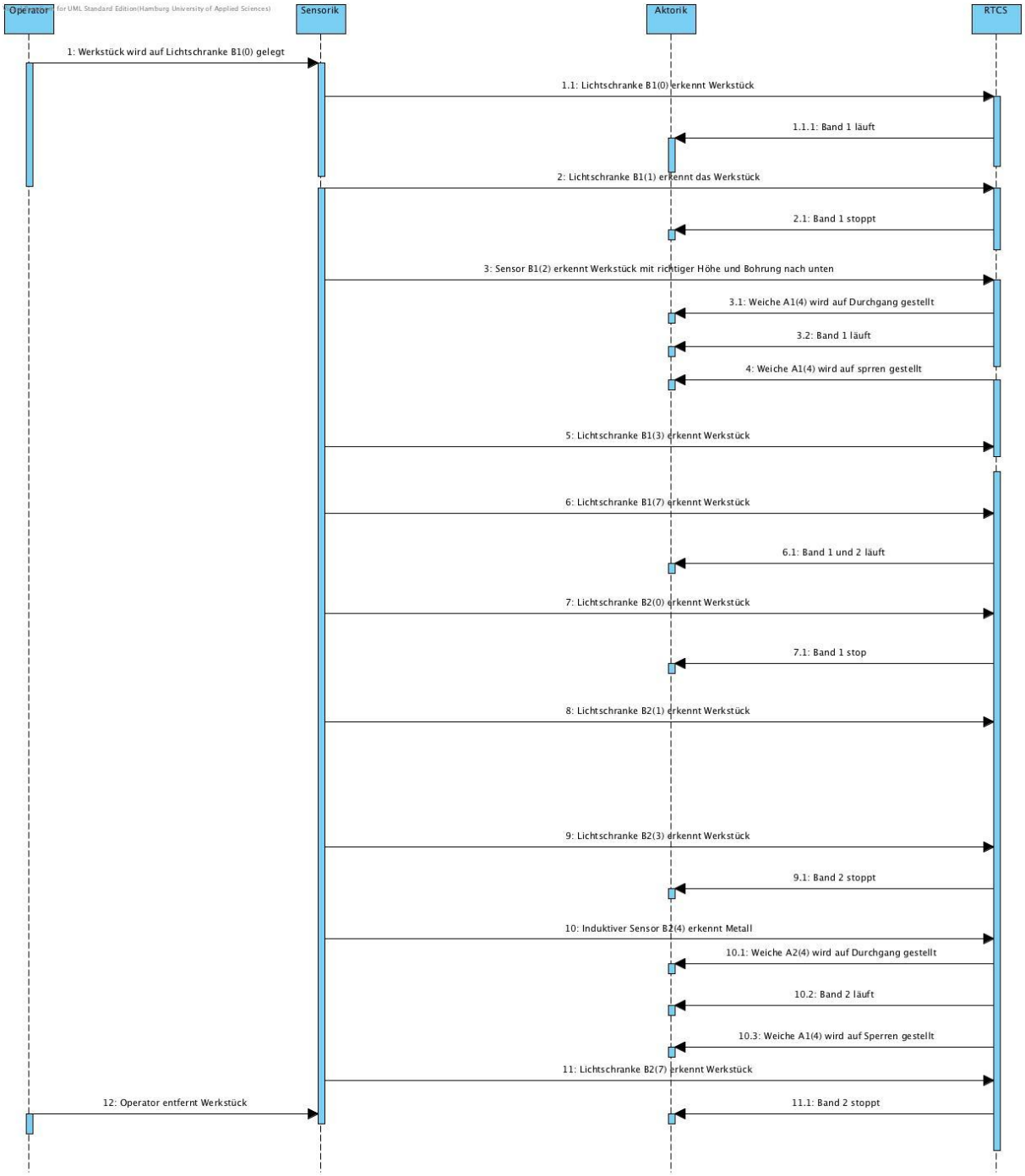

Bild 1.1.7 Szenario 2 – Sequenzdiagram

Fall II:

Werkstück mit richtiger Höhe, kein Metall und Bohrung nach unten

Lichtschranke B1(0) erkennt Werkstück
- Band 1 läuft
- Lichtschranke B1(1) erkennt das Werkstück
- Sensor B1(2) erkennt Werkstück mit richtiger Höhe und Bohrung nach unten
- Weiche A1(4) wird auf Durchgang gestellt
- Band 1 läuft
- Lichtschranke B1(3) erkennt Werkstück
- Band 1 läuft
- Lichtschranke B1(7) erkennt Werkstück
- Band 1 und 2 läuft
- Lichtschranke B2(0) erkennt Werkstück
- Lichtschranke B2(1) erkennt Werkstück
- Band 2 läuft
- Lichtschranke B2(3) erkennt Werkstück
- Induktiver Sensor B2(4) erkennt kein Metall
- Weiche A2(4) wird auf Durchgang gestellt
- Band 2 läuft
- Lichtschranke B2(3) erkennt Werkstück
- Band 2 läuft
- Lichtschranke B2(7) erkennt Werkstück

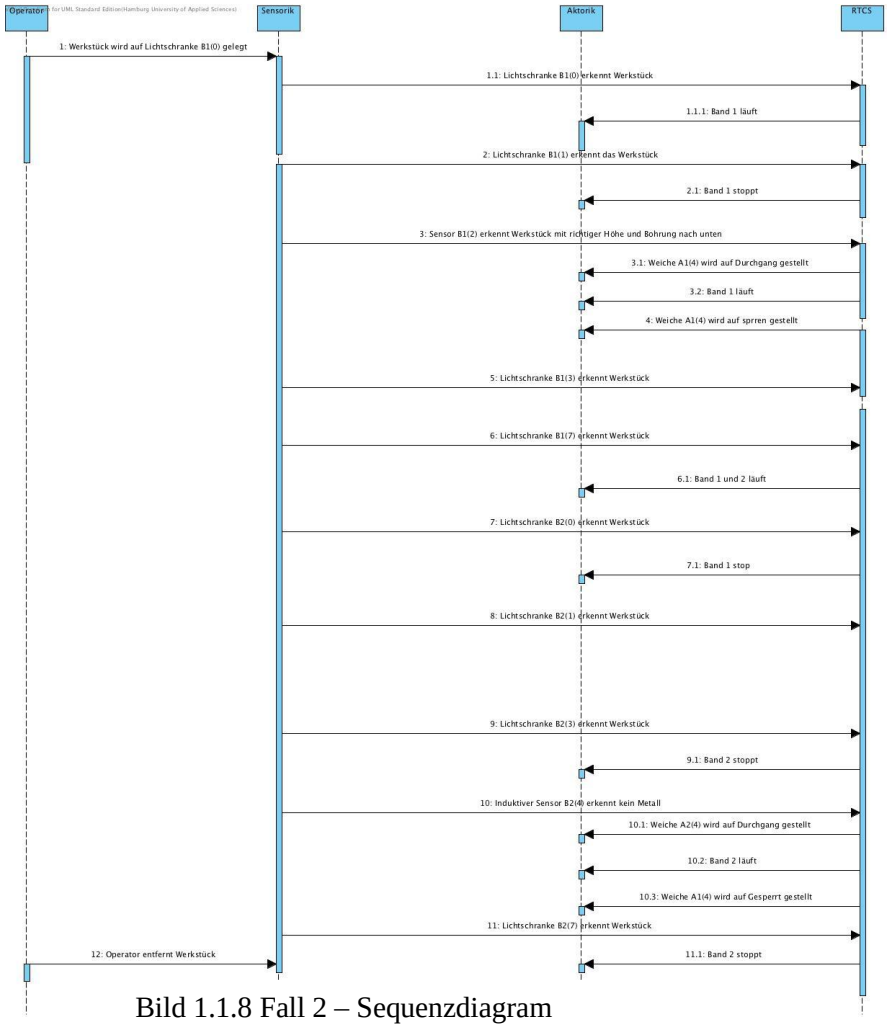

Bild 1.1.8 Fall 2 – Sequenzdiagram

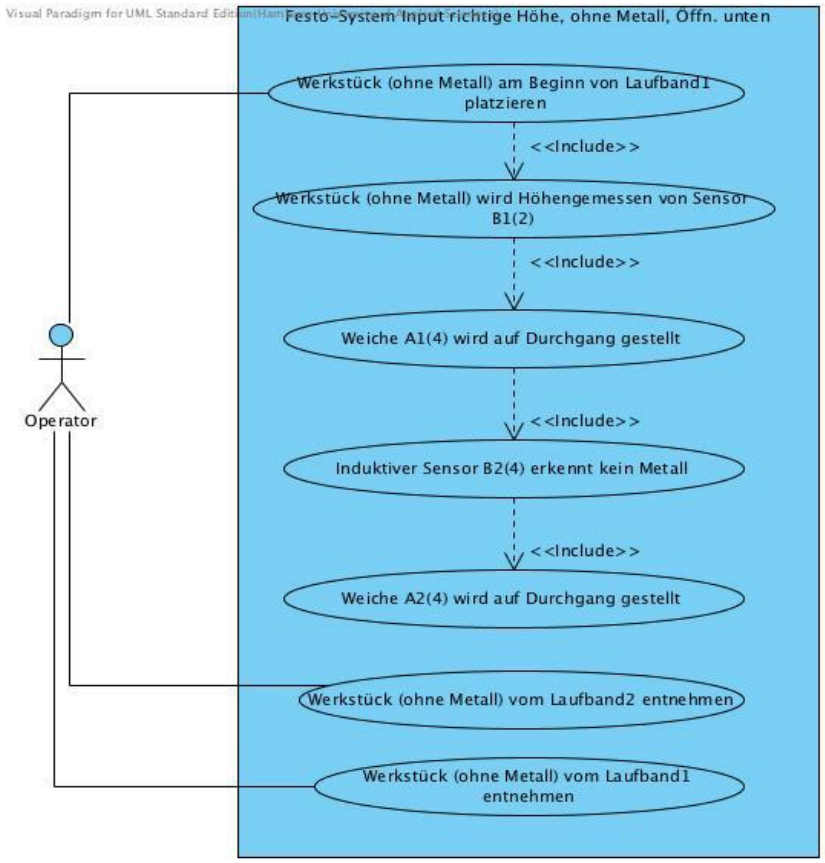

Bild 1.1.9 Fall 2 – UseCaseDiagram

1.2Systemarchitektur

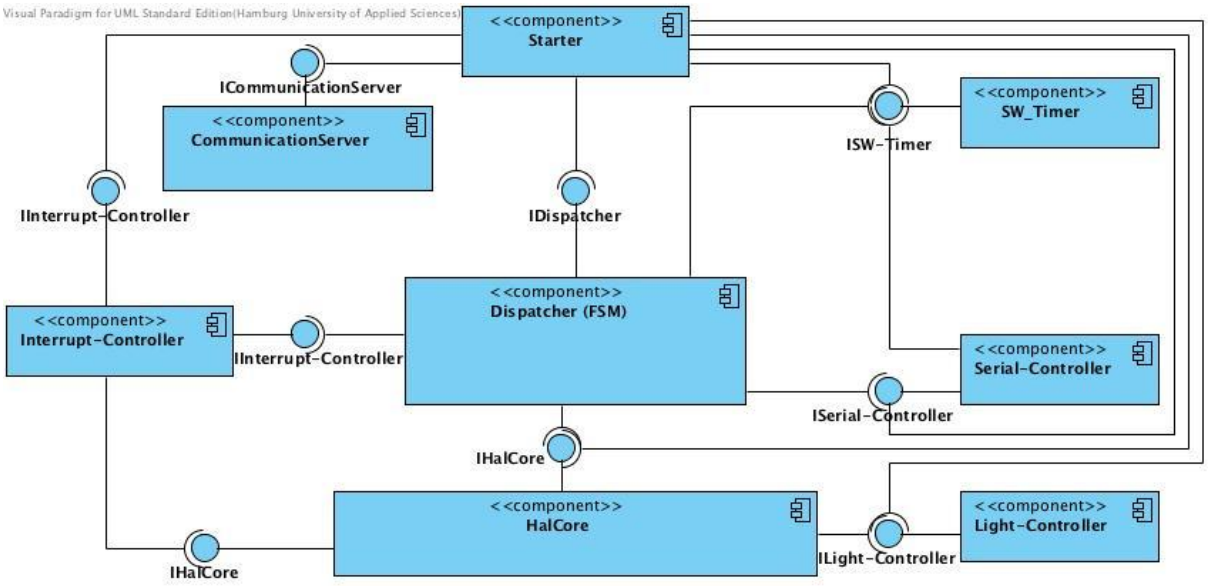

Bild 1.2.1 Systemarchitektur

Unser System besteht aus den folgenden Komponenten:

- HAL
  - HALCore
  - Lampensteuerung
  - Interrupt Controller
- Communication Server und Communication-Framework
- Anlagensteuerung (Sensor) bzw. Dispatcher
  - Band 1
  - Band 2
- Serial Controller
- Starter
- Exception-Logger

Das System wird durch den Starter in Betrieb genommen und startet alle nötigen Instanzen. Ein Menü hilft bei der Auswahl und Konfiguration. Zuerst werden Instanzen für die HAL und den Communication Server erzeugt und gestartet. Danach wird der Serial Controller gestartet und zum Abschluss die Anlagensteuerung. Nun ist das System betriebsbereit.

Die HAL besteht aus der HALCore, der Lampensteuerung und dem Interrupt Controller. Alle genannten Komponenten wurden als Singleton realisiert und thread-safe implementiert.

HALCore kapselt alle Zugriffe auf die Hardware und stellt eine umfangreiche Schnittstelle zur Manipulation der Hardware zur Verfügung. Alle Funktionen der Schnittstelle werden

entgegen genommen und in eine interne Queue geschrieben. Der Inhalt der Queue wird nacheinander abgearbeitet. Dies ermöglicht uns ein möglichst schnelles Anhalten aller Funktionen bei Betätigung des Notaus und stellt sicher, dass nur ein Schreibbefehl auf die Hardware zurzeit ausgeführt wird.

Der Interrupt Controller registriert beim Betriebssystem QNX eine ISR Funktion und nimmt von ihr verschickte Pulsmessages entgegen und schickt diese an die Anlagensteuerung.

In der Interrupt Service Routine wird analysiert, welcher der Hardware-Ports einen Interrupt ausgelöst hat und das Ergebnis mit samt dem Registerinhalt an den Interrupt Controller gesendet. Die Betätigung des Notaus wird gesondert geprüft um so schnell wie möglich darauf zu reagieren.

Die Lampensteuerung enthält ein Array mit je einem Eintrag pro Lampe. Dieser Eintrag enthält Informationen über die halbe Periodendauer, ob die Lampe momentan in Benutzung ist, ob sie leuchtet und wie viel Zeit noch vergeht bis zum nächsten Wechsel. Der Lampenthread legt sich solange schlafen bis der nächste Wechsel für eine der Lampen fällig wird und aktualisiert dann alle Lampen oder mindestens jede Sekunde einmal.

Das Communication Framework stellt Funktionalität rund um das Message Passing von QNX bereit. Genauere Informationen zur Anwendung und dessen Aufbau finden sich in der zugehörigen Dokumentation. Es wird innerhalb der Anwendung von allen Message Passing nutzenden Komponenten inkludiert. Dies sind vor allem der Communication Server, der Interrupt Controller, der Serial Controller, der Exception-Handler und die Anlagensteuerung.

Der Communication Server baut auf dem Communication Framework auf. Er dient dazu eine zentrale Instanz zu schaffen, die alle Channel-IDs der jeweiligen Komponenten beinhaltet. So brauch nur die Server-ID bekannt sein. Die Komponenten registrieren sich bei ihrer Initialisierung beim Communication Server und erfragen die von ihnen benötigten IDs. Mit Hilfe des Communication Frameworks können die Komponenten dann eine Verbindung zueinander aufbauen. Selbstverständlich nutzt der Communication Server das thread-safe Singleton Pattern.

Das thread-safe Singleton Pattern haben wir als Template implementiert, so dass es wieder verwendbar und leicht für andere Klassen zu übernehmen ist. Es bietet ebenfalls die Möglichkeit, die Singleton-Instanz zu beenden und zu löschen. Es wird von allen Komponenten der HAL, sowie dem Menü und dem Serial-Controller.

Der Serial Controller dient zur Übertragung von Daten von einem Band zum anderen Band der Anlagensteuerung. Über die serielle Schnittstelle wird die Weitergabe der Pucks verhandelt und Informationen bezüglich einer Bohrungsöffnung mit übergeben. Die Verbindung wird durch ein von uns geschriebenes Protokoll aufrechterhalten und auf Verbindungsabbrüche getestet. Nur die Anlagensteuerung nutzt den Serial Controller. Im Gegenzug informiert der Serial Controller per Pulse-Message die Anlagensteuerung darüber wenn Anfragen über die Serielle Schnittstelle ankommen. Auch Informationen über

auftretende Fehler werden weitergeleitet, ebenso die Anfrage ob Band 2 momentan bereit ist einen Puck aufzunehmen.

Unsere Anlagensteuerung hat zur Grundlage eine Liste mit Pucks. Dabei ist jeweils ein Puck ein Automat und zugehörig zu einem Werkstück auf dem Laufband. Die ankommenden Signale werden von der Anlage passend den Werkstücken zu geteilt. Je nach Konfiguration des Bandes wird entweder die Höhenmessung ausgelöst oder die Metallmessung vorgenommen. Die Fehlerbehandlung wird innerhalb unserer Puck-FSM durchgeführt. Der Exception-Logger hat daher nur noch die Aufgabe Informationen zu Exceptions zu speichern und mit zu protokollieren.

1.3Anforderungen

1 Werkstück hat das Laufband nach x Sekunden passiert.
2 Werkstück bewegt sich zwischen den Lichtschranken x Sekunden.
3 Auswurfschacht bietet genug Platz um Werkstücke aufzunehmen.

1.4Fehlerfälle

(1) Werkstück wird vor Lichtschranke B(1) aufgelegt.
(2) Werkstück wird vor Lichtschranke B(3) aufgelegt.
(3) Werkstück wird vor Lichtschranke B(7) aufgelegt.
(4) Werkstück wird vor Lichtschranke B(1) entfernt bzw. trifft nicht ein.
(5) Werkstück wird vor Lichtschranke B(3) entfernt bzw. trifft nicht ein.
(6) Werkstück wird vor Lichtschranke B(7) entfernt bzw. trifft nicht ein.
(7) Werkstück wird durch Verunreinigung fehlerhaft gemessen (z.B. Manipulation der Bohrtiefe).
(8) Weiche A(4) ist blockiert oder kann nicht geschlossen werden.
(9) Schacht ist voll.
(10) Werkstück befindet sich in Höhenmessung Lichtschranke B(1) aber nicht in Höhenbereich1 B(2) → Höhenmessung Fehlerhaft.

(1)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(2)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(3)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(4)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(5)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(6)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(7)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(8)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(9)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

(10)Lampe A(7) (rot) blinkt schnell und Operator muss die Quittierungstaste betätigen.

Operator muss das Festo-System prüfen.

2. Design und Dokumentation

2.1Schnittstellenkapselungen

Für detaillierte Informationen ist die DoxyGen-Doku mit Link im Anhang aufzusuchen.

2.1.1 Communication Framework:

Dokumentation Communication Framework:

Aufbau:

Die Basis für dieses Framework liefert eine Liste in der Teilnehmer gespeichert werden können. Gespeichert werden dabei Informationen um wen es sich handelt, dessen Channel ID und die eigene Connection ID um über diesen Channel Daten zu senden. Gleichzeitig sind bereits Variablen enthalten um die eigene Channel-ID und die Connection ID auf eben diesen Channel zu speichern. Zusätzlich sind noch 2 Strukturen für das Empfangen und das Verschicken von Nachrichten enthalten. Diese Nachrichten können entweder Pulse-Messages, Events oder ein selbstdefinierter Datentyp sein.

Unser selbstdefinierter Datentyp beinhaltet die Connection-ID, Channel-ID, einen Befehl und den Typ des Absenders.

Jede Communication inkludierende Klasse wird als Communicator-Device bezeichnet.

Es gibt folgende Befehle:

- addToServer: fügt das anfragende Communicator-Device zum Communication Server hinzu.
- removeFromServer: löscht das anfragende Communicator-Device vom Communication Server.
- closeConnection: teilt einem Communicator-Device mit, dass der Sender die Verbindung beendet.
- startConnection: teilt einem Communicator-Device mit, dass der Absender eine Verbindung aufbauen möchte.
- getIDforCom: das Communicator-Device fragt beim Communication Server nach der ID für ein bestimmtes Communicator-Device. Achtung: hier wird der CommunicatorType nicht dem Absender entsprechen sondern dem Gesuchten!
- react: Nachricht, dass auf einen aufgetretenen Interrupt auf Port B reagiert werden soll.
- reactC: Nachricht, dass auf einen aufgetretenen Interrupt auf Port C reagiert werden soll.
- reactSerial: Nachricht, dass auf eine aufgetretene Nachricht von der Seriellen Schnittstelle reagiert werden soll.
- information: Information, dass etwas passiert ist. Momentan nicht verwendet, für zukünftiges reserviert.
- OK: wird nur für Rückantworten(Replys) zur Bestätigung verwendet. Falls ein Fehler auftrat wird error zurückgesendet.
- notAvailable: für den Communication Server reservierte Nachricht. Kann auf die Anfrage mittels getIDforCom folgen, sollte das gewünschte Communicator-Device nicht verfügbar sein.

- sendID: Für zukünftiges reservierter Befehl.
- Error: Nachricht für Fehler, vor allem genutzt bei Replys um Anzuzeigen, dass die angeforderte Aufgabe nicht ausgeführt werden konnte.

Anwendung:

Die normale Anwendung erfolgt in dem die Klasse welche das Communication Framework nutzen soll von Communication.h erbt. Nun sind für den standardmäßigen Gebrauch nur noch einige Funktionen anzupassen bzw. einzufügen.

Zuerst einmal muss die Verbindung aufgebaut werden und die lauschende Schleife eingefügt werden. An „settingUpCommunicatorDevice“ muss der passende CommunicatorType übergeben werden. Achtung: *COMMUNICATIONCONTROLLER* ist nicht zu verwenden!
Der dazugehörige Code-Schnippsel sieht wie folgt aus:

```
if (settingUpCommunicatorDevice(NONE)) {
	/* do your own setup... */

	/* listen loop */
	while (!isStopped()) {
		rcvid = MsgReceive(chid, r_msg, sizeof(Message), NULL);
		/* reaction on that message should be             */
		/* done in handlePulsMessage or handleNormalMessage*/
		handleMessage();
		/* possibility to add more own code */
	}
	/* close all connections and stuff you opened*/

	endCommunication();
}
```

handleMessage wird prüfen ob die erhaltene Nachricht ein Puls oder eine normale Nachricht ist. Bei einem Puls wird handlePulsMessage() aufgerufen, ansonsten handleNormalMessage().

handlePulsMessage() muss implementiert werden! Sie könnte beispielsweise so aussehen:

```
void ExampleClass::handlePulsMessage() {
	/* here comes the code that handles a received Pulse Message */
	/* could be empty or give you an error message or whatever   */
}
```

handleNormalMessage() muss ebenfalls implementiert werden und kann so aussehen:

```
void ExampleClass::handleNormalMessage() {
	/* here will be the standard reactions done,        */
	/* like building up a connection, usage recommended */
	if(!handleConnectionMessage()){
		/* here can you build your own code or example errors */
		printf("InterruptController: unknown command in message
encountered\n");
		/* this can also be empty
	  */
	}
	/* message got handled */
}
```

handleConnectionMessage() überprüft die Nachricht auf abzuarbeitende Befehle. Er baut automatisch die Verbindung auf oder beendet sie je nach Inhalt der Nachricht.

Auf Message kann man wie folgt zugreifen:

Message ist entweder eine sigevent structure, eine _pulse strucutre oder eine Msg structure.

Mittels getValueFromReceivePulse oder getCodeFromReceivePulse kann der Value oder Code eines erhaltenen Pulses einfach ausgelesen werden. Ansonsten kann sowohl auf Message *r_msg als auch auf Message *msg einfach zugegriffen werden. R_msg dient zum Empfangen, während msg zum Senden verwendet wird.

Zum Senden eines Pulses wird die Funktion `bool sendPulses(CommunicatorType target, int code, int value)` oder `bool sendPulses(CommunicatorType target, int number, int code, int value)` verwendet.

Um eine normale Nachricht zu versenden muss zunächst einmal eine Nachricht zusammengestellt werden. Die Objekte dafür sind bereits vorhanden falls die Nachricht innerhalb der Lauschenden Schleife versendet werden soll. `bool sendMessage(Message *msg, Message * rec)` und `bool sendMessage()` stehen dafür zur Verfügung. Falls parallel zur lauschenden Schleife eine Nachricht verschickt werden soll, muss hierfür Speicher für die Nachrichten reserviert werden.
Bevor die Nachricht versendet werden kann muss sie mit Inhalt befüllt werden, dies ist etwas aufwendiger:

```
void Communication::buildMessage(void *s, int chid, int coid, MsgType activity,CommunicatorType c);
void Communication::buildMessage(void *s, int chid, int coid, MsgType activity,CommunicatorType c, int val);
```

Der Pointer "s" ist ein Pointer auf die zu beschreibende Message. Die "chid" ist die ID des zunutzenden Channels. Die "coid" ist die Connection ID des Senders. "C" ist der Typ des Kommunizierenden. "activity" ist eines von reservierten Codewörtern, die eine Aktion hervorrufen soll. Val ist ein Wert der versendet werden soll.

Für weitere Informationen zu den einzelnen Funktionen schauen Sie bitte in der DoxyGen-Dokumentation nach.

2.2 Rechnerkopplung

Zur Kommunikation wird die serielle Schnittstelle verwendet. Ein Thread wird für die serielle Schnittstelle erstellt, der auf einkommende Daten über die serielle Schnittstelle lauscht. So bald ein Datenblock eintrifft, wird dieser analysiert und anschließend eine Message an die zuständige Anlagensteuerung geschickt. Zum Senden über die serielle Schnittstelle wird eine eigene Methode erstellt, die den Befehl bzw. die Information aufbereitet und die Daten seriell verschickt.

Um sicherzustellen, dass die Serielle Schnittstelle nicht unterbrochen ist, wird in einem definierten Zeitabstand eine SYNC Message an das jeweils andere Band geschickt und ein sync_error Timer wird gestartet.
Sollte diese SYNC Message nicht ankommen und in einer definierten Zeit wieder eine SYNC Message ankommen, läuft der sync_timer ab und die syncError() Funktion wird aufgerufen. Wird die Verbindung anschließend wiederhergestellt, synchronisiert sich die Schnittstelle wieder und die Anlage läuft normal weiter.

2.3Verhalten der Anlage

Zur Vorbereitung wurde eine FSM erstellt. Diese wird in Meilenstein 4 implementiert. Als Pattern wird das State-Pattern verwendet.

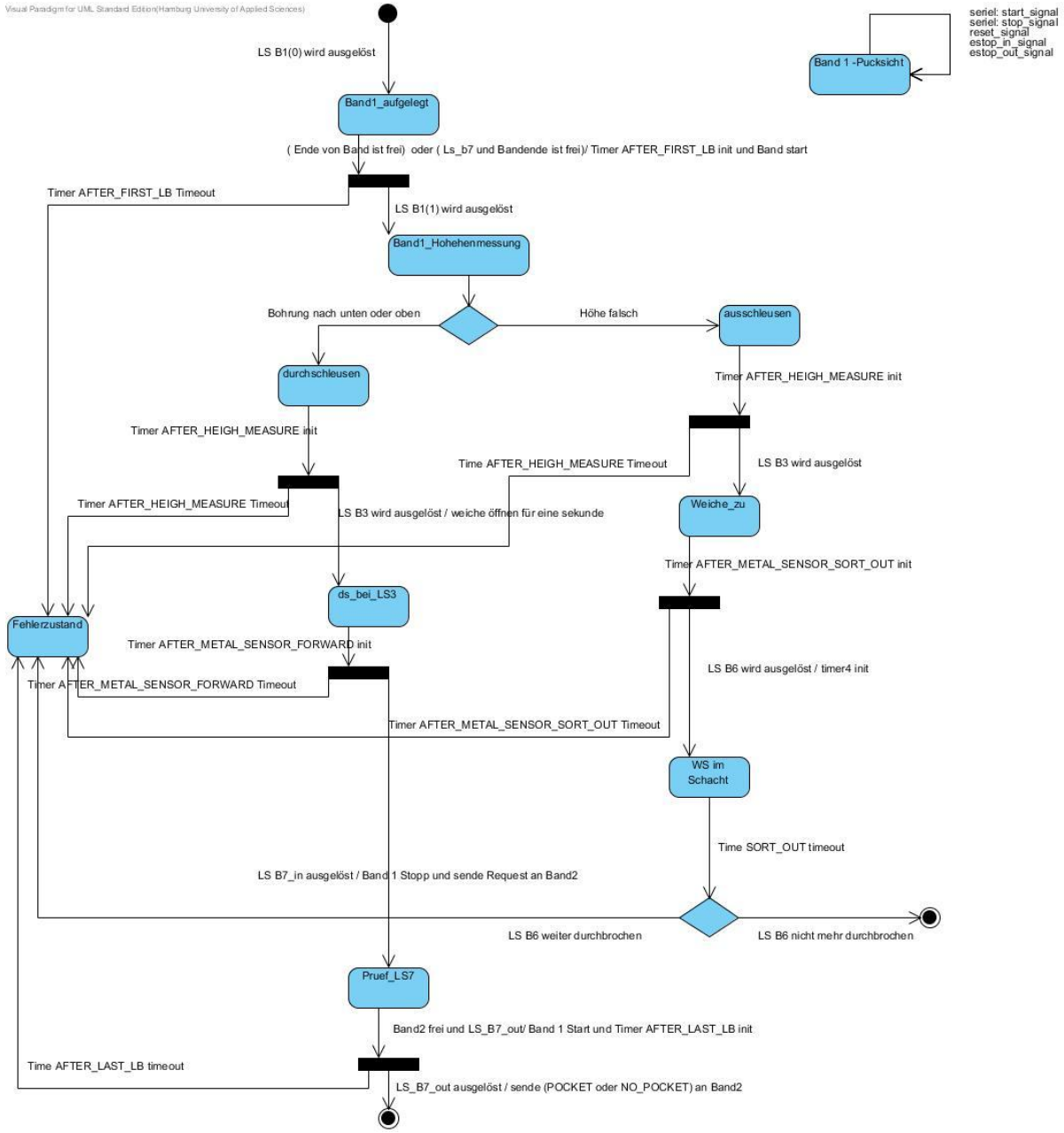

Bild 2.3.1 zeigt das Verhalten von Band 1.

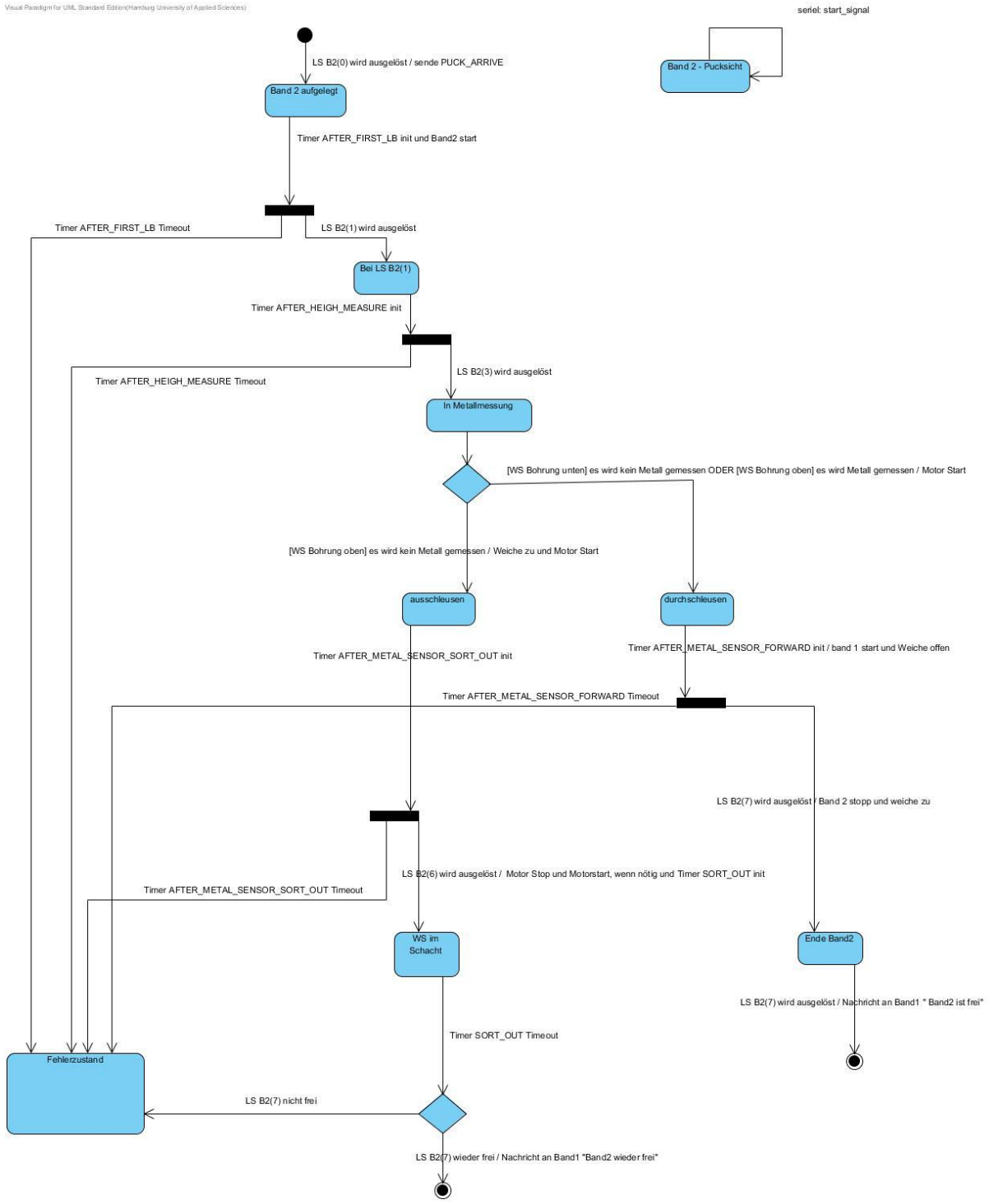

Bild 2.3.2 zeigt das Verhalten von Band 2.

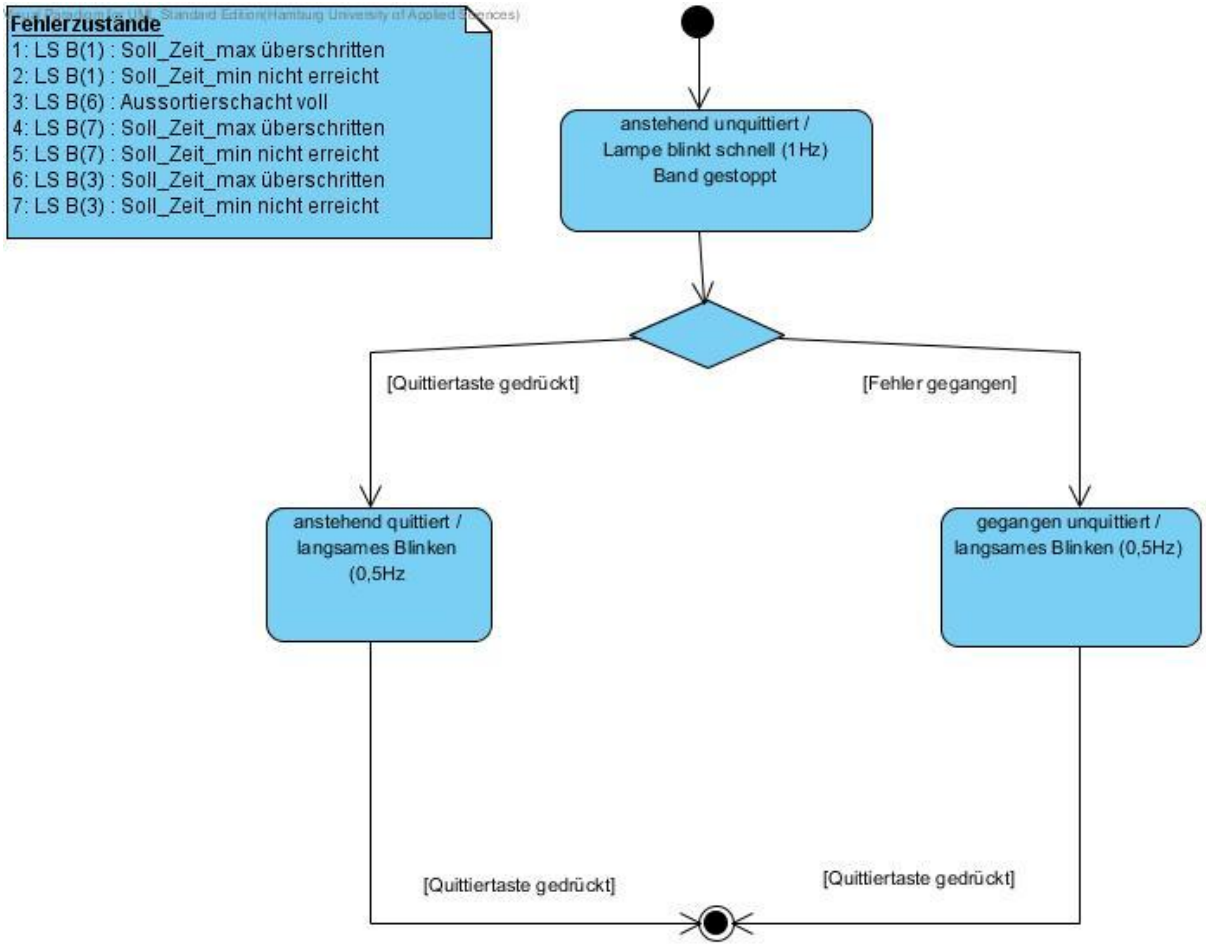

Bild 2.3.3 zeigt den Fehlerzustand

3. Tests

3.1Testfälle

1 Werkstück mit falscher Höhe: wird in Auswurfschacht1 aussortiert
2 Werkstück mit richtiger Höhe, ohne Metalleinsatz, Öffnung nach oben
: wird in Auswurfschacht2 aussortiert.
3 Werkstück mit richtige Höhe, mit Metalleinsatz und Öffnung nach oben
: trifft am Ende von Laufband2 ein.
4 Werkstück mit richtiger Höhe, mit Metalleinsatz, Öffnung nach unten
: trifft am Ende von Laufband2 ein.
5 Werkstück mit richtiger Höhe, ohne Metalleinsatz und Öffnung nach unten
: trifft am Ende von Laufband2 ein.

3.2Testszenarien

Die Testszenarien befinden sich im Ordner Anhang/Tests.

4 Anhang

4.1 Entscheidung der Timer Art

Für das Projekt wurde ein Betriebssystem-Timer gewählt, da dieser eine Auflösung im Millisekunden Bereich hat. Der ansonsten zur Verfügung stehende Hardware-Timer besitzt eine Auflösung im Nanosekunden Bereich. Da wir jedoch damit mechanische Bauteile ansteuern, ist der Betriebssystem-Timer völlig ausreichend und wir benötigen nicht die teure Timer-Hardware. Dafür verwenden wir von QNX Funktionen, die nach einem periodischen bzw. einmaligen Zeitintervall eine Pulse Message verschickt oder einen Callback durchführt.

4.2 Projekt- und Zeitplan

Zu finden im Unterordner Pläne.

4.3 Doxygen-Dokumentation

Zu finden im Unterordner Dokumentation.