Regaleinräumer "Hochstapler"

Michael Thomas, Andreas Glatz, Simon Weitzel, Felix Baral-Weber
Stand: 10. Juli 2016 21:04

Inhaltsverzeichnis

1 Einführung

1.1 Zweck des Dokuments

Dieses Dokument beschreibt die Anforderungen und Implementierungsdetails an eine Echtzeitsystemanwendung die das Abschlussprojekt des Moduls Echtzeitbetriebssysteme der EAH Jena. Dieses Dokument richtet sich dabei an den Vorgaben für ein Software Requirements Specification (SRS) nach dem IEEE Standard 830-1998. Der Modulverantwortliche ist Prof Dr. Oliver Jack aus dem Fachbereich Elektrotechnik und Informationstechnik.

1.2 Einstieg in die Projektphase

Die Arbeit an der Programmumsetzung wurde in die Teilbereiche Hochregal-Steuerung, Simulation, Integritätsprüfung und User-Input und Visualisierung aufgeteilt um diese Bereiche allein bzw. in Gruppen parallel abarbeiten zu können.
Dabei ergab sich folgende Aufteilung:

- Hochregal-Steuerung → Michael Thomas, Simon Weitzel
- Simulation → Felix Baral-Weber, Andreas Glatz

- Integritätsprüfung und User-Input → Simon Weitzel, Michael Thomas
- Visualisierung → Andreas Glatz, Felix Baral-Weber

Desweiteren wurden Hauptverantwortliche für alle Teilbereich des Projekt festgelegt.

Die da wären:

- Michael Thomas: Programmumsetzung
- Andreas Glatz: Tests
- Simon Weitzel: Strukturierte Analyse
- Felix Baral-Weber: Latex-Template und Funktionale Anforderung

2 Allgemeine Beschreibung

2.1 Aufgabenstellung und Umsetzungsansatz

Die Aufgabestellung des Semesterprojektes ist es eine Hochregallagersimulation unter zuhilfenahme eines Echtzeitbetriebssystems zu erstellen. Diese soll das gesamte System mit einer statisch gewählten Regaldimension sowohl steuern als auch simulieren. Das Simulationmodell umfasst ein Regallagersystem mit $5 * 10$ Regalfächern, einen Ein- und einen Ausgabeslot und einen Turm zum be- und entladen der Fächer. Dieser Turm ist auf einer Achse montiert auf welcher er sich in X-Richtung bewegen kann, desweiteren kann der Ausleger am Turm in Y-Richtung herauf und herab bewegen. An dem Ausleger ist widerum ein Schlitten befestigt welcher in Z-Richtung rein, also zum Hochregal hin, und raus, zu den Ein-/Ausgabe-Slots gefahren werden kann. Die Position dazwischen wird als neutrale Position bezeichnet. Sämtliche Bewegungen des Turms werden von Tastern auf den Schienen ermittelt um daraus die Position des Turmes bestimmen zu können. Auf der X-Achse und auf der Y-Achse befinden sich jeweils 10 Taster und auf der Z-Achse 3.
Um ein Klötzchen vom Eingabe-Slot oder aus einem Regalfach aufzunehmen wird der Schlitten am Ausleger unterhalb des Klötzchens aus der neutralen Position der Z-Achse in den Slot bzw. das Fach gefahren und dann erst angehoben. Umgekehrt wird beim Ablegen eines Klötzchens in den Ausgabe-Slot oder in ein Regalfach das Paket von oben auf den Slot bzw. das Regalfach abgesenkt bevor der Ausleger am Schlitten in die neutrale Position der Z-Achse zurückgefahren wird.

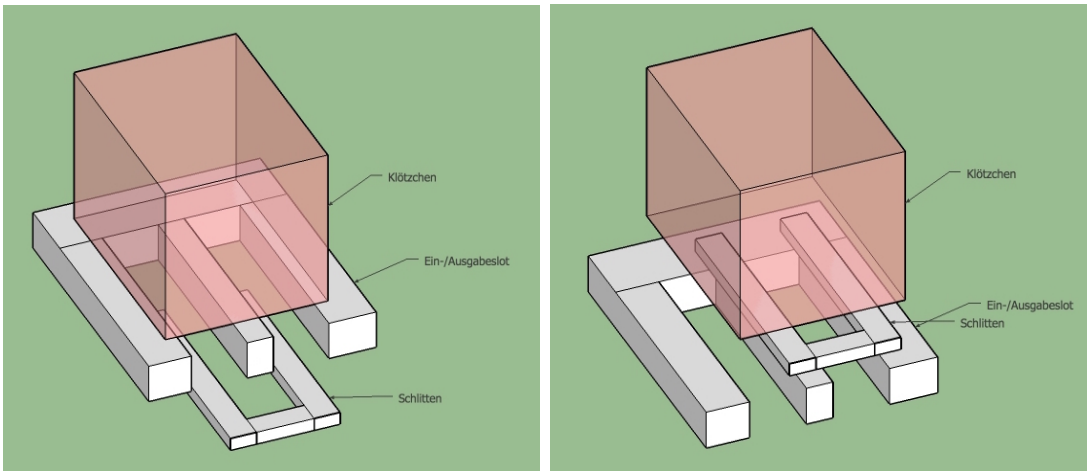

Abbildung 1: Aufnahme/Ablage an Ein-/Ausgabe-Slot

Zur Eingabe steht dem Bediener die Konsole zur Verfügung.

Folgende Befehle werden zur Verfügung gestellt:

- vsetspace[x][y] → Definiert einen Platz als schon belegt
- clearspace[x][y] → Definiert einen Platz als frei
- insert[x][y] → Holt ein Klötzchen vom Eingabe-Slot und legt es an gewünschter Position im Hochregallager ab
- remove[x][y] → Holt ein Klötuchen von der gewünschten Position ab und legt es an den Ausgabe-Slot

Sollte ein Befehl nicht möglich sein, da ein Hochregallagerplatz bereits belegt ist bzw. ein Fach aus dem ein Klötzchen geholt werden soll leer ist, wird der Anwender durch eine in der Konsole ausgegebene Fehlermeldung darauf aufmerksam gemacht und der Befehl nicht ausgeführt. Es können vor Abschluss eines Befehls weitere aufgegeben werden, welche sich in einer Warteschlange einreihen. Die Befehle "clearspace ünd "vsetspace"werden nicht an das Ende dieser gestellt, sondern nach Abschluss des letzten bereits angefangenen Auftrages ausgeführt, da für diese keine Bewegung des Turms nötig ist. Für den Fall, das zu diesem Zeitpunkt eine weitere Operation mit dem selben Regalfach bereits in der Warteschlange ist, wird diese dann nicht ausgeführt und der Bediener über das Aussetzen dieses Auftrages mittels Konsolenausgabe informiert. Sobald kein weiterer Auftrag in der Warteschlange ist wird der Turm an die Position vor dem Eingabe-Slot gefahren und verweilt dort bis ein neuer Auftrag aufgegeben wird.
Die Visualisierung des Hochregallagers erfolgt ebenfalls in der Konsole, in welcher sowohl das Hochregal und dessen Belegung als auch die Position des Turms und dessen Auslegers, durch ASCI-Zeichen stilisiert dargestellt und nach jeder Änderung aktualisiert werden. Dabei liegt der Ursprung der Koordinaten und somit der Regalplatz (0,0) in der

linken unteren Ecke. Die Darstellung des Ein- und Ausfahren des Auslegers wird unterhalb dargestellt. Dabei stellt die linke Position den zum Ein-/Ausgabe-Slot gefahrenen Arm da, die rechte Position die in das Regal hereingefahrene.

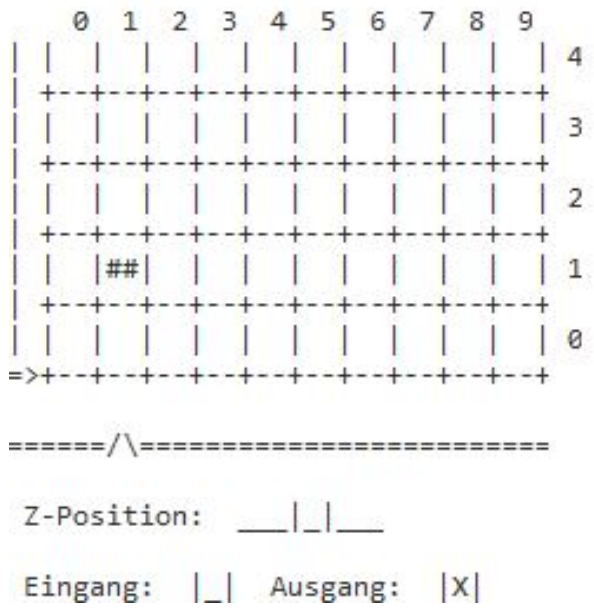

Abbildung 2: Ausgabevisualisierung in Konsole

2.2 Entwicklungsumgebung und Programmierung

Als Entwicklungs- und Testumgebung wurde Windriver Workbench 3.3 genutz und die Programmiersprache C verwendet. Es wurden für das Echtzeitbetriebssystem Vxworks Libaries inkludiert.

3 Spezifische Anforderungen

3.1 Strukturierte Analyse

3.1.1 Kontextdiagramm

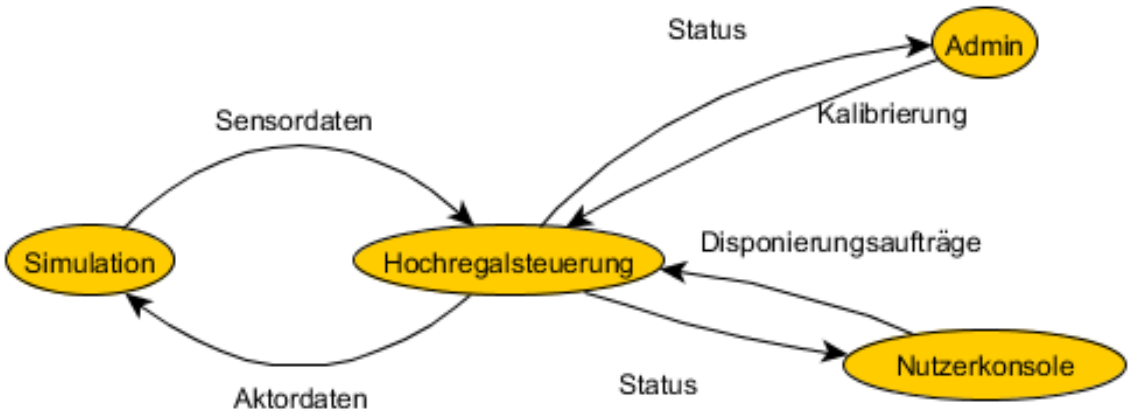

Abbildung 3: Kontextdiagramm

Aus dem Kontextdiagramm lässt sich leicht erkennen, dass das System aus zwei großen Teilen besteht. Auf der einen Seite ist sdie HRL-Steuerung, welche Eingaben erfasst, kontrolliert und die gewünschte Reaktion darauf berechnet. Auf der anderen Seite steht die Simulation, die für die physikalischen Abläufe nötige Zeit und das auslösen der diversen Sensoren simuliert. Verbunden sind beide durch diverse MessageQueues und eine globale Variable, welche in den weiteren Datenflussdiagrammen erleutert werden.

3.1.2 DFD1 Simulation

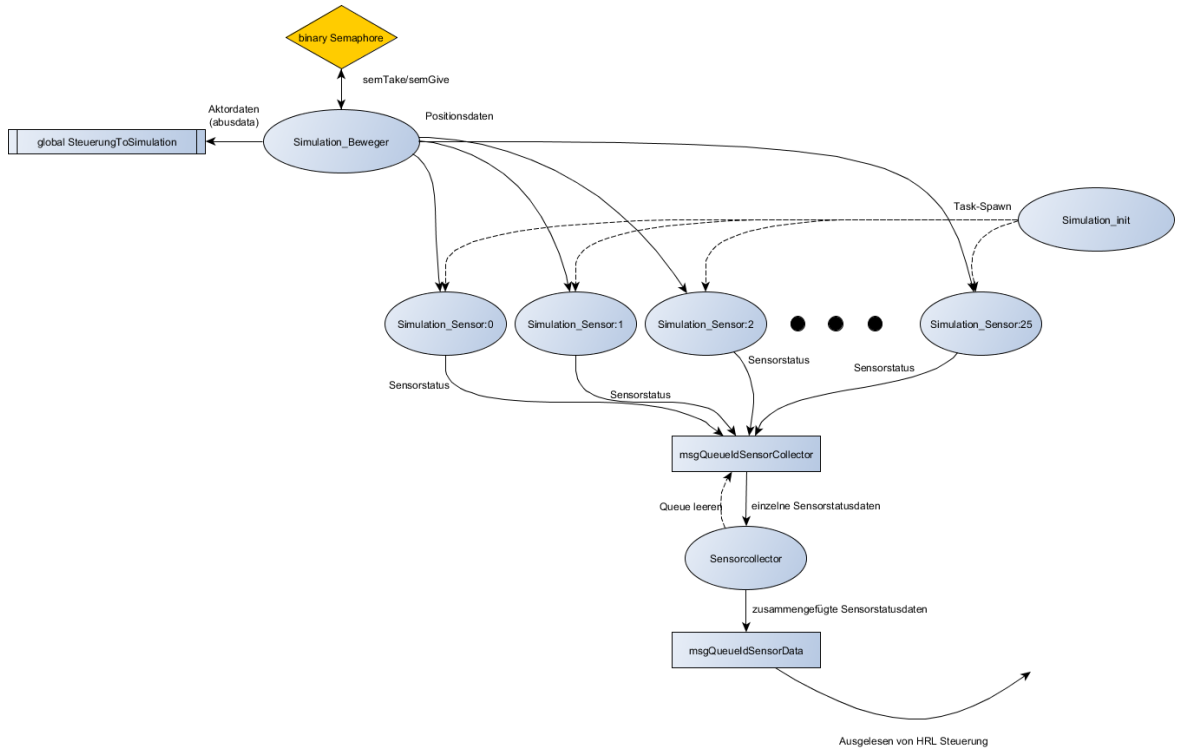

Abbildung 4: DFD1 Simulation

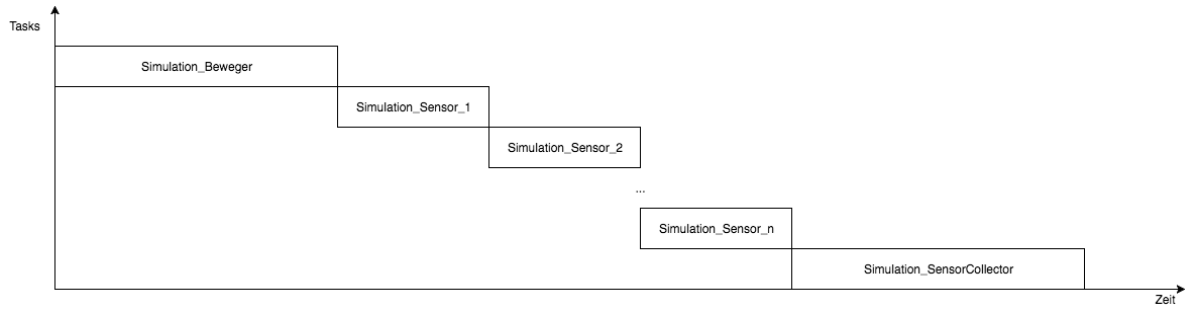

Abbildung 5: Gantt Diagramm der Task der Simulation

Alle Tasks in der Simulation haben eine Priorität von 200 und ändern diese nicht. Sie werden in einer festen Reihenfolge erzeugt und laufen dann in jedem Zyklus in dieser Sequenz.

Mini-Spec Simulation Beweger:
Wie in Abb.5 zu sehen ist läuft der **Beweger** als erster Task eines Simulationsschrittes. Der Beweger setzt virtuelle Position(Positionsdaten) nach Aktordaten (abusdata),

welche durch einen Semaphore geschützt werden. Da die Simulation minimal blockiert werden soll implementiert die Semaphore eine Prioritätsvererbung.

Mini-Spec Simulation Sensor:
Jeder Sensor bekommt eine ID die der bitstelle in Sensorbits entspricht. Zur initialisierung errechnet er so seine Position (*triggeroffset*).

Mini-Spec Simulation Sensor 0-22:
Sensor Task berechnet eigenes *triggeroffset* und gleicht dieses mit virtueller Position ab.
if(virtuelle Position-X == triggeroffset-X) true und eigene ID in *msgQueueIdSensorCollector*
else false und eigene ID in *msgQueueIdSensorCollector*

analog für Y- und Z-Sensoren

Mini-Spec Simulation Sensor 23:
Der Lichttaster wird gesetzt wenn er eine Auflade- bzw. Abladeoperation detektiert. Diese besteht (wie in Abb. 6 zu sehen) aus unter den Ablagepunkt fahren, den Arm nach außen bzw. innen fahren und dann anheben, bzw. über den Ausgabepunkt fahren, den Arm nach außen bzw innen fahren und danach absenken.

Mini-Spec Simulation Sensor 24-25:
Sensoren für Lichtschranken. Berechnung nicht über *triggeroffset*, sondern die Lichtschranken werden über Zeitkonstanten gesteuert, welche hoch bzw. runter gezählt werden.
Setzen bzw. Entfernen eines Klötzchens aus bzw. in einen Lichttaster:

if(Z = in Regal und Y= hoch)
Auslagerungsvorgang in Regal erkannt -> Lichttaster anschließend bedeckt und Belegungsmatrix[X][Y]= nicht belegt

if(Z = in Regal und Y= runter)
Einlagerungsvorgang in Regal erkannt -> Lichttaster anschließend nicht bedeckt und Belegungsmatrix[X][Y] = belegt

if(Z = aus Regal heraus und Y= hoch)
Aufnahmevorgang in Eingabe-Slot erkannt -> Lichttaster anschließend bedeckt und Zeitkonstante von Lichtschrank in Slot wird zurückgesetzt

if(Z = aus Regal heraus und Y= runter)
Auslagerungsvorgang in Ausgabe-Slot erkannt -> Lichttaster anschließend nicht bedeckt Zeitkonstante von Lichtschrank in Slot wird zurückgesetzt

«««< Updated upstream Mini-Spec Sensorcollector:
Sensorcollector greift Datensatz aus msgQueueIdSensorCollector ab(leeren der Queue), fügt sie zu sbusdata zusammen und gibt diese in msgQueueIdSensorData

3.1.3 DFD1 Steuerung

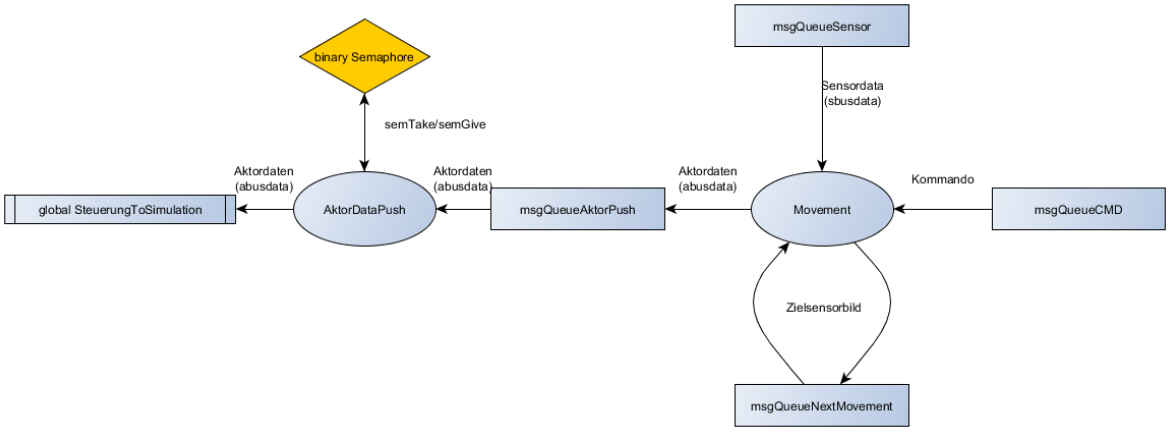

Abbildung 6: DFD1 Steuerung

Mini-Spec Movement:
Der Movement-Task fragt kontinuierlich nach neuem Auftrag aus msgQueueCMD, ist keiner da wird Timeout ausgelöst und System geht in Pause-Modus bzw. Zusatzaufgabe über. Wenn Auftrag bereitsteht wird Belegung geändert(für Befehle vsetspace und clearspace) bzw. es werden die 8 Teilsensorbilder generiert(Befehle insert und remove). Diese werden in msgQueueNextMovement weitergeleitet und werden anschließend nacheinander abgearbeitet (aus der Queue). Für jeden Teilauftrag wird auf neue Sensordaten aus msgQueueSensor gewartet und die daraus entstehenden aktuellen Sensordaten mit den Soll-Sensordaten verglichen, dabei werden alle neuen Sensordaten auf ihre Legitimität überprüft und im Fehlerfall (z.B. blockierte Schiene) werden alle Motoren gestoppt und das System angehalten. Die Aktordoten werden in die msgQueueAktorPush geleitet.

Mini-Spec AktorDataPush: AktorDataPush dient zur Weitervermittlung der Aktordaten, welche geschützt weitergeleitet werden müssen(globale Variable). Wartet auf Aktordaten in msgQueueAktorPush, sind welche vorhanden wartet er auf die Semaphore um anschließend in die globale Variable zu schreiben aus welcher die Simulation liest. Für diese Semaphore wurde eine Prioritätsvererbung implementiert um den Schreibvorgang

auf die globale Variable möglichst nicht aus zu bremsen.

3.2 Funktionale Anforderungen

Beweger

Wie in Abb.5 zu sehen ist läuft der **Beweger** als erster Task eines Simulationsschrittes. Zuerst aktualisiert er die Aktorwerte aus der gloablen Variable **AktorBusData**. Diese entspricht den Busdaten und ist durch eine Semaphore geschützt. Da die Simulation minimal blockiert werden soll implementiert die Semaphore eine Prioritätsvererbung. Daraufhin „verschiebt“er den virtuellen Turm entsprechend der Aktordaten.

Sensorcollector

Der **Sensorcollector** läuft nach allen Sensoren und sammelt aus einer Message Queue die Einträge aller Sensoren. Wenn ein Sensor ausfällt bleibt der Wert des Sensors auf 1. Daraufhin schiebt er alle Sensorwerte, in einem Vektor, in eine Messega Queue auf die die Steuerung Zugriff hat.

Simulation Sensor x

Jeder Sensor bekommt eine id übergeben die seine Funktion und ihn eindeutig definiert.

1. **Tastsensoren** überprüfen ob die Turmposition mit der Sensorposition übereinstimmt.

2. **Lichtschranken** Simulieren entweder das eintreffen eines Klötzchens am Eingabeslot oder das Entfernen eines Klötzchens am Ausgabeslot, jeweils mit einem vordefinierten Delay.

3. Der **Lichttaster** im Turm wird dann ausgelöst wenn die vorige y-Position des Turms im vorigen Schritt über (beim Ablegen eines Klötzchens) bzw. unter (beim Aufnehmen) der jetzigen Position ist und (beim Ablegen) ein Klötzchen im Turm ist bzw. (beim Aufnehmen) ein Klötzchen an der Aufnahmeposition ist.

Die jeweiligen Sensordaten schiebt der Task daraufhin in eine Message Queue für den Sensorcollector.

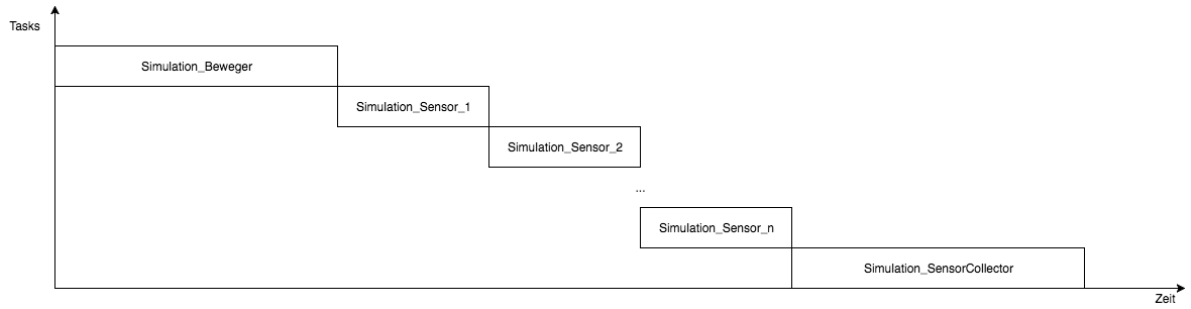

Abbildung 7: Gantt Diagramm der Task der Simulation

Alle Tasks in der Simulation haben eine Priorität von 200 und ändern diese nicht. Sie werden in einer festen Reihenfolge erzeugt und laufen dann in jedem Zyklus in dieser Sequenz.

Taster X[0-9]	[gedrückt|gehalten]
Taster Y[0-9]	[gedrückt|gehalten]
Taster Z[0-3]	[gedrückt|gehalten]
Lichtschranke Eingabe	[unterbrochen|nicht unterbrochen]
Lichtschranke Ausgabe	[unterbrochen|nicht unterbrochen]
Turmlichtschalter	[bedeckt|nicht bedeckt]

Tabelle 1: Requirements Dictionary: Die Stati der Sensoren

Mini-Spec Sensorcollector:
Der Sensorcollector läuft nach allen Sensoren. Er greift die Messwerte der einzelnen Sensoren aus *msgQueueIdSensorCollector* ab (leeren der Queue) und fügt sie zu *sbusdata* zusammen und schiebt diese in *msgQueueIdSensorData*.

3.2.1 DFD1 Steuerung

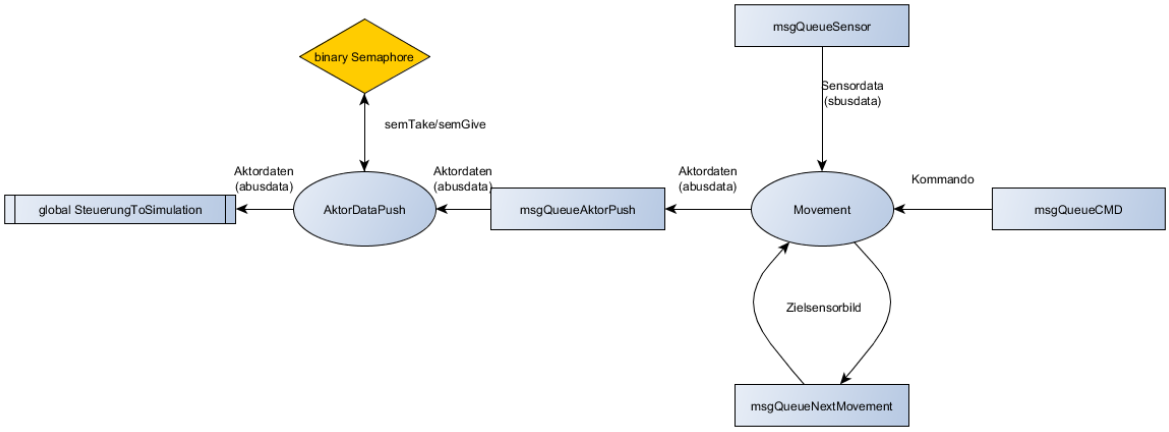

Abbildung 8: DFD1 Steuerung

Mini-Spec Movement:
Der Movement-Task fragt kontinuierlich nach neuem Auftrag aus msgQueueCMD, ist keiner da wird Timeout ausgelöst und System geht in Pause-Modus bzw. Zusatzaufgabe über. Wenn Auftrag bereitsteht wird Belegung geändert(für Befehle vsetspace und clearspace) bzw. es werden die 8 Teilsensorbilder generiert(Befehle insert und remove). Diese werden in msgQueueNextMovement weitergeleitet und werden anschließend nacheinander abgearbeitet (aus der Queue). Für jeden Teilauftrag wird auf neue Sensordaten aus msgQueueSensor gewartet und die daraus entstehenden aktuellen Sensordaten mit den Soll-Sensordaten verglichen und die Aktoren per msgQueueAktorPush angesteuert.

Mini-Spec AktorDataPush:
AktorDataPush dient zur Weitervermittlung der Aktordaten, welche geschützt weitergeleitet werden müssen(globale Variable). Wartet auf Aktordaten in msgQueueAktorPush, sind welche vorhanden wartet er auf die Semaphore um anschließend in die globale Variable zu schreiben aus welcher die Simulation liest. Für diese Semaphore wurde eine Prioritätsvererbung implementiert um den Schreibvorgang auf die globale Variable möglichst nicht aus zu bremsen.

4 Tests

4.1 Zeitliche Analyse

4.1.1 Test Simulation mit System Viewer

In Abb: **??** wird eine Sequenz der Simulation während sich das Programm in einer Pause befindet, also keine Aufträge auszuführen sind, gezeigt. Diese Sequenz beginnt mit dem

Beweger-Task (1.) welcher die virtuelle Position des Turms aktualisiert. Anschließend werden alle Sensor-Tasks nacheinander aktiv und überprüfen ob sich an ihrer Stelle gerade der Turm befindet und schreiben ihre Antwort darauf in die MessageQueue. Diese wird von dem Sensor-Collector, nachdem alle Sensoren durchgelaufen sind, ausgelesen und von diesem zu einem Gesamt-Konstrukt zusammengefügt und in die MessageQueue an die HRL-Steuerung geschrieben.
Das Gantt-Diagramm Abb: 5 wird damit bestätigt.

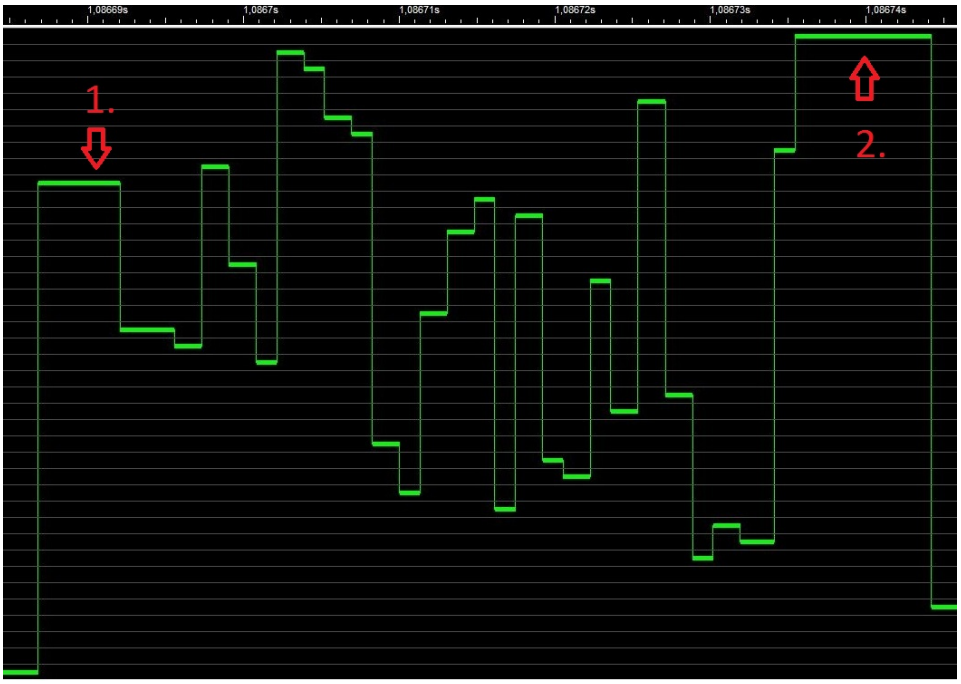

Abbildung 9: Simulation mit System Viewer

4.1.2 Test Hochregal-Steuerung mit System Viewer

Die Hochregal-Steuerung besteht aus lediglich zwei Tasks, welche sich nicht unterbrechen. Der Movement-Task (1.) beginnt mit der Berechnung der Aktor-Befehle, welche anschließend von dem AktorPushData-Task(2.) an die Simulation weitergeleitet werden. Auch dieses ist somit Bestätigt.

Abbildung 10: HRL-Steuerung mit System Viewer

4.2 Systemtests

4.2.1 Jobannahmetest

aktuell | neu –>	vsetspace x y	clearspace x y	insert x y	remove x y
vsetspace x y	pass	pass	pass	pass
clearspace x y	pass	pass	pass	pass
insert x y	pass	pass	pass	pass
remove x y	pass	pass	pass	pass

Nach diesem Test können wir bestätigen, dass das Programm aus jedem Job in den Folgejob gelangt.

4.2.2 Vollständigkeit / Korrektheit

	Änderung der Belegungsmatrix
vsetspace	pass

Dieser Test bestätigt die Funktion des vsetspace-Befehls innerhalb der Spezifikationen.

	Änderung der Belegungsmatrix
clearspace	pass

Dieser Test bestätigt die Funktion des clearspace-Befehls innerhalb der Spezifikationen.

	fahre zur Eingabe	Paket annehmen	fahre zur Ablage	Paket ablegen
insert x y	pass	pass	pass	pass

Dieser Test bestätigt die Funktion des insert-Befehls innerhalb der Spezifikationen.

	fahre zur Ablage	Paket annehmen	fahre zur Augabe	Paket ablegen
remove x y	pass	pass	pass	pass

Dieser Test bestätigt die Funktion des remove-Befehls innerhalb der Spezifikationen.

Zudem wurde das Verhalten der automatisches Einlagerung überprüft falls das Regal voll ist. In diesem fall wird der Job nicht mehr angenommen und der zustand am Terminal ausgegeben.

Falls mehrere Befehle in der Warteschlange den selben Platz setzen oder leeren wollen wurde dies nicht erkannt, dieses fehlverhalten wurde verbessert und am Terminal quittiert.

4.2.3 Test außerhalb der Spezifikationen

In diesem Test wird das verhalten des Programmes bei Fehlbedienung getestet, ein pass bedeutet das der Job nicht angenommen wird.

	x<0 | y<0	x > 9 | y > 4	ohne Param.	ende von Pause oder Zusatz
vsetspace	pass	pass	pass	pass
clearspace	pass	pass	pass	pass
insert	pass	pass	pass	pass
remove	pass	pass	pass	pass

Dieser Test bestätigt das bei jeder fehlerhaften Eingabe der Job nicht angenommen wird und dieses auch am Terminal anzeigt.