



# Projektgruppe FAISE

Endbericht  
im Rahmen des Masterstudiums

Betreuer:  
Prof. Dr.-Ing. Jürgen Sauer  
Dipl.-Ing. (FH) Arne Stasch  
Dipl.-Inform. Jan-Hinrich Kämper  
Prof. Dr.-Ing. Axel Hahn

Vorgelegt von:  
Berthe Pulcherie Ongnomo  
Chancelle Merveille Tematio Yme  
Christopher Schwarz  
Jan Paul Vox  
Jan-Gerd Meß  
Jannik Fleßner  
Malte Falk  
Matthias Aden  
Michael Goldenstein  
Nagihan Aydin  
Raschid Alkhatib  
Simon Jakubowski

Abgabetermin: 30. September 2014

## Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Einsatzszenario . . . . .	2
1.4 Komponenten . . . . .	2
<b>2 Stand der Technik</b>	<b>4</b>
2.1 Fahrerlose Transportsysteme . . . . .	4
2.2 Fahrerlose Transportfahrzeuge . . . . .	5
2.2.1 Orientierungssystem bzw. Navigation . . . . .	6
2.2.2 Steuerungstechnik . . . . .	9
2.3 Materialflusssysteme . . . . .	11
2.3.0.1 Funktionen von Materialflusssystemen . . . . .	12
2.4 Multiagentensysteme . . . . .	15
2.5 Fallbeispiele . . . . .	16
2.5.1 FTS in der Gläsernen Manufaktur Dresden (Volkswagen) . . . . .	16
2.5.2 FTS beim Automobilhersteller BMW im Werk Leipzig . . . . .	16
<b>3 Projektorganisation</b>	<b>18</b>
3.1 Organisation der Teilgruppen (Materialfluss, Fahrzeuge, Simulation) . . . . .	18
3.2 Rollenverteilung . . . . .	18
3.3 Vorgehensmodell . . . . .	19
3.4 Werkzeuge . . . . .	21
3.4.1 Jira . . . . .	22
3.4.2 Confluence . . . . .	22
<b>4 Allgemeine Anforderungen</b>	<b>23</b>
4.1 Ablaufszenario . . . . .	23
<b>5 Teilbericht Simulation</b>	<b>25</b>
5.1 Lastenheft . . . . .	25
5.2 Grundlegende Designentscheidungen . . . . .	26
5.2.1 Eigenentwicklung . . . . .	26
5.2.2 Entwicklung einer Webanwendung . . . . .	27
5.2.3 Umsetzung durch GWT . . . . .	27
5.3 Konzeption der Systemkomponenten . . . . .	27
5.3.1 Gesamtarchitektur . . . . .	28
5.3.2 Konzeption der Benutzeroberfläche . . . . .	29

5.3.3	Generierung von Aufträgen . . . . .	29
5.3.4	Anforderungen an ein Multiagenten-Framework . . . . .	30
5.3.5	Benötigte Agententypen . . . . .	30
5.3.6	Kommunikation zwischen den Agenten . . . . .	31
5.3.7	Konzeption des Pathfindings . . . . .	32
5.3.8	Konzeption der Statistiken . . . . .	32
5.3.9	Interaktion der Komponenten . . . . .	33
5.4	Implementierung der Systemkomponenten . . . . .	34
5.4.1	Implementierte Gesamtarchitektur . . . . .	34
5.4.2	Benutzeroberfläche . . . . .	35
5.4.3	Generierung von Aufträgen . . . . .	35
5.4.4	Auswahl eines Multiagenten-Frameworks . . . . .	37
5.4.5	Implementierte Agententypen . . . . .	37
5.4.6	Kommunikation zwischen den Agenten . . . . .	37
5.4.6.1	Jobzuweisung an Ein- und Ausgänge . . . . .	38
5.4.6.2	Eingang fragt Ausgang und Zwischenlager . . . . .	40
5.4.6.3	Ausgang fragt Zwischenlager . . . . .	41
5.4.6.4	Auktion . . . . .	43
5.4.6.5	Paket von Startrampe abholen und zur Zielrampe bringen . .	43
5.4.7	Implementierung des Pathfindings . . . . .	45
5.4.8	Implementierung der Statistiken . . . . .	46
5.4.9	Interaktion der Komponenten . . . . .	47
5.4.9.1	Client-Server Kommunikation . . . . .	47
5.4.9.2	Start des Multiagenten-Systems . . . . .	48
5.4.9.3	Start des Jobtimers . . . . .	49
5.4.9.4	Visualisierung von Zustandsveränderungen . . . . .	51
5.5	Testen und Validieren . . . . .	53
<b>6</b>	<b>Teilbericht Materialfluss</b>	<b>55</b>
6.1	Anforderungen . . . . .	55
6.1.1	Funktionale Anforderungen . . . . .	55
6.1.2	Nicht-funktionale Anforderungen . . . . .	56
6.2	Beschreibung der Komponenten . . . . .	56
6.2.1	Rampen . . . . .	56
6.2.2	Mikrocontroller . . . . .	57
6.2.2.1	MICAZ-Modul . . . . .	59
6.2.2.2	MIB520 . . . . .	60

6.3	Werkzeuge . . . . .	60
6.3.1	Atmel Studio . . . . .	60
6.3.2	Atmel JTAGICE3 . . . . .	63
6.4	Systemarchitektur . . . . .	64
6.4.1	Contiki . . . . .	66
6.4.1.1	Build-Vorgang . . . . .	66
6.4.1.2	Prozesse . . . . .	67
6.4.1.3	Prozesskommunikation . . . . .	68
6.4.1.4	Scheduling und Timer . . . . .	69
6.4.1.5	Der Rime Kommunikationsstack . . . . .	69
6.4.2	Background Level: Treiber, Services und Interfaces . . . . .	71
6.4.2.1	Treiber Bolzen . . . . .	71
6.4.2.2	Interface Bolzen . . . . .	72
6.4.2.3	Treiber Externer Speicher . . . . .	73
6.4.2.4	Interface Externer Speicher . . . . .	73
6.4.2.5	Service Externer Speicher . . . . .	73
6.4.2.6	Treiber Lichtschranken . . . . .	73
6.4.2.7	Interface Lichtschranken . . . . .	73
6.4.2.8	Treiber Funkmodul . . . . .	74
6.4.2.9	Treiber UART-Schnittstelle . . . . .	74
6.4.2.10	Interface UART-Schnittstelle . . . . .	75
6.4.3	Agenten RTE . . . . .	76
6.4.3.1	Verwaltung der Agenten . . . . .	76
6.4.3.2	Austausch von Nachrichten . . . . .	77
6.5	Agenten . . . . .	78
6.5.1	Paket-Agent . . . . .	79
6.5.2	Plattform-Agent . . . . .	79
6.5.2.1	Plattform-Agent der Rampen . . . . .	80
6.5.2.2	Plattform-Agent der Volksbots . . . . .	80
6.5.3	Order-Agent . . . . .	80
6.5.4	Routing-Agenten . . . . .	81
6.6	Validierung . . . . .	83
6.6.1	Erreichte Funktionalität . . . . .	83
6.6.2	Probleme und Herausforderungen . . . . .	84
6.6.3	Ausblick . . . . .	85
<b>7</b>	<b>Teilbericht Fahrzeuge</b>	<b>87</b>
7.1	Anforderungen . . . . .	87

---

7.2	Beschreibung der Komponenten . . . . .	88
7.3	Werkzeuge . . . . .	89
7.3.1	Robot Operating System (ROS) . . . . .	90
7.3.2	Was ist ROS . . . . .	90
7.3.3	Was will ROS? . . . . .	90
7.3.4	Was kann ROS? . . . . .	90
7.3.5	Aufgaben des ROS . . . . .	91
7.3.6	ROS-Datensystem . . . . .	92
7.3.7	Vorteile und Nachteile des ROS . . . . .	93
7.3.7.1	Vorteile . . . . .	93
7.3.7.2	Nachteile . . . . .	94
7.4	Architektur . . . . .	94
7.5	Implementierung . . . . .	95
7.5.1	Inbetriebnahme der Hardware . . . . .	95
7.5.2	Implementierung der Navigation . . . . .	95
7.5.3	Implementierung der Paketübergabe . . . . .	99
7.5.4	Struktur der einzelnen Funktionalitäten . . . . .	99
7.6	Erreichte Funktionalitäten . . . . .	99
7.7	Herausforderungen . . . . .	100
7.8	Ausblick . . . . .	100
<b>8</b>	<b>Integration</b>	<b>102</b>
8.1	Ablauf in physischer Zelle . . . . .	102
<b>9</b>	<b>Ergebnisse</b>	<b>103</b>
9.1	Realisierte Ziele . . . . .	103
9.2	Fazit . . . . .	103
9.3	Vision . . . . .	104

## Abbildungsverzeichnis

1	Prinzipskizze zur induktiven und optischen Spurführung (Quelle: Günter Ullrich, 2011 S. 79) . . . . .	7
2	Prinzipskizze zur Koppelnavigation (links) und zur Magnet- bzw. Transpondernavigation (rechts) (Quelle: Günter Ullrich, 2011 S. 79) . . . . .	7
3	Prinzipskizze zur Koppelnavigation (links) und zur Magnet- bzw. Transpondernavigation (rechts) (Quelle: Günter Ullrich, 2011 S. 79) . . . . .	8
4	Prinzipskizze zur Koppelnavigation (links) und zur Magnet- bzw. Transpondernavigation (rechts) (Quelle: Günter Ullrich, 2011 S. 79) . . . . .	9
5	Die Systemarchitektur eines einfachen FTS (Quelle: Günter Ullrich, 2011 S. 93)	10
6	Allgemeine Darstellung einer FTF-Steuerung mit Datenschnittstellen (vgl. VDI 4451) . . . . .	11
7	Elemente einer Wertschöpfungskette (vgl. Wulz, J., 2008, S. 7) . . . . .	12
8	Beispiel eines Stetigförderers (entnommen aus Ten Hompel, Schmidt, Nagel, 2007, S. 131) . . . . .	13
9	Scrumplanung für einen Meilenstein . . . . .	20
10	Modell für die Darstellung der einzelnen Projektphasen . . . . .	21
11	Hersteller der Projektwerkzeuge . . . . .	21
12	Teil 1 Ablaufszenario . . . . .	23
13	Teil 2 Ablaufszenario . . . . .	24
14	Teil 3 Ablaufszenario . . . . .	24
15	Gesamtarchitektur . . . . .	28
16	Schematischer Aufbau der Benutzeroberfläche . . . . .	29
17	Interaktion der Systemkomponenten . . . . .	34
18	Implementierte Gesamtarchitektur . . . . .	35
19	Benutzeroberfläche . . . . .	36
20	Auftragsgenerierung . . . . .	36
21	Jobzuweisung an Ein- und Ausgänge . . . . .	39
22	Eingang fragt Ausgang und Zwischenlager . . . . .	41
23	Ausgang fragt Zwischenlager . . . . .	42
24	Auktion . . . . .	44
25	Paket abholen und zur Zielrampe bringen . . . . .	45
26	Starten einer Simulation . . . . .	48
27	Starten des MAS . . . . .	49
28	Start des Jobtimers . . . . .	50
29	Weiterleitung des Auftrags . . . . .	51
30	Feuern eines Events . . . . .	52

31	Clientseitige Verarbeitung des Events . . . . .	53
32	Beispiel einer eingesetzten Rampe . . . . .	57
33	Schematischer Aufbau eines Mikrocontrollers vgl. [Brinkschulte:2002:Mikrocontroller]	58
34	MICAZ-Modul [Memsic:2014:Online] . . . . .	60
35	Blockdiagramm der MICAZ-Module [Memsic:2014:Online] . . . . .	61
36	Entwicklungs-Ansicht des Atmel Studio . . . . .	62
37	Debugging-Ansicht des Atmel Studio . . . . .	62
38	Atmel JTAGICE3 [AtmelJTAGICE3:2014:Online] . . . . .	64
39	Architektur der Rampe [Stasch:Hahn] . . . . .	64
40	Architektur der Volksbots aus Sicht des Materialfluss [Stasch:Hahn] . . . . .	65
41	Der Rime Netzwerkstack [Dunkels:2007:Proc] . . . . .	70
42	Zustandsautomat des Routing Agenten . . . . .	82
43	Darsellung eines Volksbots . . . . .	89
44	Architektur des Volksbot . . . . .	94
45	Legende zu Abb. 46 und 47 . . . . .	95
46	eingebundene Funktionen in ROS . . . . .	96
47	Hauptsteuerung des Systems ( Epos2 Control ) . . . . .	97

## Tabellenverzeichnis

## 1 Einleitung

In diesem Kapitel wird ein einführender Überblick über die Projektgruppe Fully Autonomous Intralogistic Swarm Experiments gegeben, die im Rahmen der Masterstudiengänge Informatik und Wirtschaftsinformatik in der Abteilung Systemanalyse und -optimierung der Carl von Ossietzky Universität Oldenburg stattgefunden hat. Das Projekt lief über einen Zeitraum von zwei Semestern: Wintersemester 2013/2014 und Sommersemester 2014.

### 1.1 Motivation

Im Zeitalter der Globalisierung werden hohe Anforderungen an die Leistungsfähigkeit von modernen Intralogistiksystemen gestellt. Neben einem hohen Automatisierungsgrad wird gleichzeitig auch eine möglichst hohe Flexibilität gefordert, da sich Anforderungen im logistischen Umfeld häufig ändern (Vgl.[ieft]).

Stetigförderer bieten die Möglichkeit einen automatisierten Materialfluss einzurichten. Es handelt sich dabei um Transportsysteme, die Güter kontinuierlich und automatisiert entlang eines festgelegten Transportwegs befördern (Vgl.[stf]). Ein solches System könnte beispielsweise ein Netz von Schienen sein. Nachteile dieser Systeme sind insbesondere Unflexibilität und schlechte Skalierbarkeit. Ändern sich Anforderungen in einem Logistiksystem, dann stoßen Stetigförderer schnell an ihre Grenzen. Transportwege sind festgelegt und können nicht ohne einen gewissen Aufwand geändert werden. Auch kann die Anzahl an Gütern, die pro Zeiteinheit befördert werden kann, nicht ohne eine Änderung am Transportnetz maximiert werden.

Eine Alternative zu Stetigförderern sind Fahrerlose Transportsysteme (FTS). FTS sind ein Gesamtsystem aus Fahrerlosen Transportfahrzeugen, die Ware automatisiert befördern, und der Infrastruktur, die zum Betrieb der Transporteinheiten notwendig ist (Vgl.[fts]). Fahrerlose Transportsysteme sind wesentlich flexibler als Stetigförderer. Müssen mehr Güter befördert werden, so können zusätzliche Transporteinheiten aktiviert werden. Folglich sind FTS problemlos skalierbar und können schnell auf veränderte Anforderungen in einem Intralogistiksystem eingestellt werden. FTS bieten einen automatisierten Warenfluss bei gleichzeitig hoher Flexibilität und entsprechen somit den Anforderungen, die an moderne Intralogistiksysteme gestellt werden.

## 1.2 Zielsetzung

Im Rahmen der Projektgruppe FAISE soll ein System entwickelt werden, das den vollautomatisierten Warenfluss in einem Lager auf Basis von Fahrerlosen Transportsystemen simuliert. Dabei sollen die Transporteinheiten nicht zentral gesteuert werden, sondern dezentral agieren. Das Gesamtsystem besteht aus zwei Teilsystemen, einem physisch vorhandenem System und einer softwarebasierten Simulation.

Das physische System beinhaltet Rampen, auf denen Ware gelagert werden kann und Fahrerlose Transporteinheiten als Fördermittel. Die Akteure kommunizieren miteinander über ein Sensornetzwerk und werden auf Basis von Mikrocontrollern gesteuert. Ziel ist es den Materialfluss von den Transporteinheiten und Rampen vollständig autonom und ohne dezentrale Steuerung durchzuführen.

Das rein softwarebasierte System soll einen automatisierten Warenfluss, der durch Rampen und Fahrzeuge durchgeführt wird, simulieren. Die Realisierung der Akteure in der Software soll an die Eigenschaften der Akteure aus dem physischen System angelehnt sein. Beide Systeme sollen unabhängig voneinander laufen.

## 1.3 Einsatzszenario

Das Einsatzszenario besteht aus  $n$  fahrerlosen Transporteinheiten in einem Umschlagslager. Zusätzlich sind  $m$  Rampen verfügbar an denen Pakete zwischengelagert werden können. Im Gegensatz zu einem herkömmlichen Lager, werden Waren in einem Umschlagslager nur kurzfristig gelagert, um anschließend weitertransportiert zu werden. Es herrscht ein kontinuierlicher Materialfluss. Jedes Paket, das ins Lager gebracht wird, ist eindeutig identifizierbar und wird zu einem definierten Zeitpunkt ins Lager gebracht und wieder abgeholt. Die Rampen im Lager sollen drei unterschiedliche Zwecke erfüllen. Eingangsrampen dienen der Warenannahme, Zwischenrampen der Zwischenlagerung. Pakete werden zum Ausgangslager gebracht und zum Zwecke des Weitertransports dort abgeholt. Auf Basis des Einsatzszenarios wird im Rahmen der Anforderungen ein Ablaufszenario erstellt, das die Interaktionen zwischen den Akteuren beschreibt auf deren Basis eine automatisierte, dezentrale Abwicklung des Materialflusses erfolgen kann.

## 1.4 Komponenten

Das zu entwickelnde System lässt sich grob in die Komponenten physisches System und Simulation unterteilen. Das physische System wiederum lässt sich noch weiter in die Kom-

ponenten Materialfluss und Fahrzeuge aufteilen. Das Gesamtsystem lässt sich somit in drei Komponente aufteilen, die im Folgenden beschrieben werden:

- Materialfluss: Die Komponente Materialfluss beschreibt die Rampen auf denen Pakete gelagert werden. Die Rampen sollen mithilfe von Sensoren mit anderen Rampen und den fahrerlosen Transporteinheiten kommunizieren können. Durch Aktoren soll es möglich sein, Pakete zum Zwecke des Weitertransports an die Fahrzeuge zu übergeben.
- Fahrzeuge: Die Fahrzeuge übernehmen den Transport der Pakete und kommunizieren mit den Rampen, um den Transport automatisiert abzuwickeln. Die Fahrzeuge sollen anhand von Start- und Zielpunkten in der Lage sein, selbstständig Pfade zu erstellen und diese abzufahren.
- Simulation: Die Simulation soll als Softwaresystem realisiert werden, dass es erlaubt Szenarien mit einer bestimmten Anzahl an Rampen und Fahrzeugen zu erstellen. Wie im physischen System, sollen die Rampen und Fahrzeuge einen automatisierten Warenfluss durchführen und die Aktionen sollen visualisiert werden.

## 2 Stand der Technik

Die Fahrerlosen Transportsysteme und die Materialflusssysteme sind Prozesse der Logistik. In den vergangenen Jahren hat die Verbreitung Fahrerloser Transportsysteme (FTS) stark zugenommen. Beim Einsatz von FTS stellen sich vielfältige Konfigurierungs- und Planungsprobleme, so auch die Einsatzplanung für die einzelnen Fahrerlosen Transportfahrzeuge. [Guenther;Krueger:2000]. Der innerbetriebliche Materialfluss von Industrieunternehmen bietet fahrerlosen Transportsystemen (FTS) zahlreiche Einsatzgebiete: Sie verketten Produktionsprozesse, verknüpfen Fertigungsstationen oder ganze Betriebsbereiche und beschicken Montageplätze. Darüber hinaus dienen sie als mobile Werkbank oder versorgen und entsorgen Lager unterschiedlicher Art. Um die Systemvorteile von Fahrerlosen Transportsystemen und Materialflusssystemen zu optimieren, braucht man ein maßgerechtes Wissen auf Ihr spezifisches Anlagekonzept abzustimmen. Wichtige Kriterien sind allerdings z. B. die Einbindung der Fahrerlosen Transportsysteme in den gesamtbetrieblichen Materialfluss, die Anpassung an die vorhandenen Steuerungshierarchien und die optimale Auslegung der Technik in Bezug auf Fahrzeugbauart, Lastaufnahmemittel, Energiekonzept, Kommunikation und Leitsystem [Werner:2014:Online]. Ziele von Fahrerlosen Transportsystemen und Materialflusssystemen sind Kostensenkung durch Personaleinsparung, Verringerung von Transportschäden, hohe Zuverlässigkeit in Vorgängen und bessere Materialflussplanung. Dieses Kapitel wird in drei Teile gegliedert. Der erste Teil wird die Fahrerlosen Transportsysteme bzw die Orientierungs- und die Steuerungssysteme vorstellen und erklären; der zweite Teil ist eine Darstellung der Materialflusssysteme und ihrer verschiedenen Funktionen und der dritte Teil wird erklären, wie fahrerlose Transport- und Materialflusssysteme in großen Firmen wie Volkswagen und BMW Anwendung finden.

### 2.1 Fahrerlose Transportsysteme

Nach dem Verein Deutscher Ingenieure 2510 bestehen FTS im Wesentlichen aus „einem oder mehreren Fahrerlosen Transportfahrzeugen (FTF), einer Leitsteuerung, Einrichtung zur Standortbestimmung und Lager erfassung, Einrichtungen zur Datenübertragung sowie Infrastruktur und peripheren Einrichtungen“. In seinem Buch Transport und Lagerlogistik fasst Martin die Definition von VDI 2510 eines FTS zusammen. Er beschreibt ein FTS als mit FTF ausgestattete rechnergesteuerte Materialflussanlagen zum automatischen Transport von Gütern im innerbetrieblichen Materialfluss[Martin:2006]. Bei FTS handelt es sich um flurgebundene Fördersysteme mit automatisch geführten FTF. Die einzelnen FTF befördern Ladungsträger zwischen zwei oder mehrere Stationen innerhalb eines Gebietes. Die Fahrzeugsteuerung erfolgt automatisch und rechnergestützt. Der Einsatzbereich von FTS ist generell überwiegend innerbetrieblich ausgerichtet. In diesen Rahmen übernehmen FTS

sowohl reine Förderaufgaben, wie Verkettung von Fertigungs- und Montageeinrichtungen als auch Aufgaben der Lagerbedienung und Kommissionierung[Guenther;Krueger:2000]. Das FTS ist eine Technik, die im Vergleich gegenüber Stetigfördersystemen eine hohe Anpassungsfähigkeit an die sich ändernden Marktsituationen zum Vorteil hat. Daher konzentrieren die Forschungs- und Entwicklungsaktivitäten sich heutzutage auf die sog. „Zellulären Fördersysteme“, in welchen stetige Förderanlagen zur Verknüpfung von Logistischen Funktionen durch individuelle, autonom arbeitende FTF ersetzt werden (vgl. Ten Hompel; Heidenblut, 2008). Die Haupteinsatzgebiete des FTS liegen nun in der Intralogistik. Also bei der Organisation, der Steuerung, der Durchführung und der Optimierung des innerbetrieblichen Waren- und Materialflusses und Logistik, der Informationsströme sowie des Warenumschlags in Industrie, Handel und öffentlichen Einrichtungen. Z. B. Automobil- und Zuliefererindustrie, Papiererzeugung und –verarbeitung, Elektroindustrie, Getränke-, Lebensmittelindustrie, Baustoffe, Stahlindustrie, Kliniklogistik[Guenther:2011]. FTS bestehen im Wesentlichen aus drei Systemkomponenten: Die Fahrerlosen Transportfahrzeuge, das Orientierungssystem, das Steuerungssystem.

## 2.2 Fahrerlose Transportfahrzeuge

Die FTF sind flurgebundene Fördermittel mit eigenem Fahrantrieb, die automatisch geführt, gesteuert und berührungslos geführt werden. Sie dienen dem Materialtransport, und zwar zum Ziehen und/oder Tragen von Fördergut mit aktiven oder passiven (FTF mit passiver Lastaufnahme werden von anderen Fördermitteln gezogen oder manuell mit den Gütern bestückt) Lastaufnahmemittel [VDI:92](VDI 2510). Da das FTS mit fahrerlosen Aspekten systematisiert ist, ergeben sich dann auf der funktionalen Ebene Unterschiede zu fahrerbedienten Fahrzeugen, wie z. B. den klassischen Gabelstaplern und FTF. Im Rahmen dieser Arbeit wird nur auf eine Kategorie von FTF tiefer eingegangen: das Mini-FTF. Die Mini-FTF sind kleine, schnelle, intelligente und flexible Fahrzeuge, die extrem schnell Bedürfnisse befriedigen können. Heutzutage arbeiten viele Universitäten in der ganzen Welt im Bereich der Schwarm-Experimente. Hier sollen die kleinen FTF intelligent miteinander arbeiten. Die Fahrzeuge sollen sich ohne eine eigene separate FTS-Leitsteuerung untereinander verstständigen, Strategien entwickeln und gemeinsam Arbeiten ausführen. Die Forschungsgebiete heißen Agentensysteme und Schwarmtheorie. Die Mini-FTF können nur intralogistische Aufgaben ausführen. Dennoch sind viele unkonventionelle Einsatzfälle denkbar. Die Kommissionierung (eine ausführliche Begriffserklärung wird im Teil Materialfluss gegeben) ist die verbreitetste Anwendungsmöglichkeit von Mini-FTF[Guenther:2011]. Als Zusammenfassung kann man sagen, dass die Fahrzeugsteuerung die Systemsicherheit, das Energiemanagement, das Lastaufnahmemittel und die Lenkung eines FTF gewährleistet. Ein FTF kann ohne Energie nicht funktionieren. Damit ein FTF seine Aufgabe erfüllen kann, ist eine Ener-

gieversorgung notwendig. Die FTF können durch Akkus oder Traktionsbatterien oder mit Hilfe eines Induktionssystems oder einer Stromschiene mit Energie versorgt werden. Jedoch können die beiden Versorgungsarten gekoppelt werden, um ein Hybridsystem zu bekommen. Die Notwendigkeit der Existenz einer Ladestation in einem FTS ist unumstritten. Die FTF müssen immer mit Energie versorgt werden. Je nachdem wie die FTF programmiert sind, kann ein FTF bei Energiebedarf selber zur Ladestation fahren oder von einem Auftraggeber (Mensch) zur Ladestation geführt werden.

### 2.2.1 Orientierungssystem bzw. Navigation

Das Orientierungssystem bzw. die Navigation dient zur Lokalisierung des Fahrzeugs. Sie ist ein Hilfsmittel zur Berechnung des sichersten Wegs, um das Ziel zu erreichen. Außerdem dient die Navigation auch zur Vermeidung von eventuellen Kollisionen. Sie gilt sowohl für die Orientierung als auch für die Sicherheit des Fahrzeugs und seines Umfeldes. Während seiner Bewegung bzw. Orientierung folgt das FTF einer physischen oder virtuellen Linie (Spur), damit es sein Ziel gefahrlos erreichen kann. Aufgrund eines Sicherheitssystems sollte das FTF bei Kollisionsgefahr oder wenn Hindernisse vor ihm stehen, sofort anhalten. Unter Navigationshilfe versteht man nicht nur die Positionierung und Orientierung des Fahrzeugs sondern auch, wohin das Fahrzeug gelangen würde, wenn keine auf seine Bewegung verändernden Maßnahmen ergriffen würden. Die Steuerung sagt, was zu tun ist, und die Navigation bestimmt, auf welchem Weg das gewünschte Ziel sicher zu erreichen ist bzw. ob das FTF einen vorgegebenen Weg weiter verfolgen oder einen alternativen Weg nehmen soll. Die Steuerung von fahrerlosen Transportfahrzeugen, deren Grundfunktionen und der Umgang mit diesen werden in den VDI-Richtlinien [VDI92], [VDI94], [VDI04] vorgestellt. Für das Konstrukt der fahrerlosen Transportsysteme werden verschiedene Ansätze verfolgt, die abhängig vom System verschiedene Konstruktionsbemühungen auf das Fahrzeug oder auf der Strecke erfordern. Es gibt mehrere Navigationsverfahren: die physische Leitlinie, die Orientierung durch Magnetmarken, das Global Positioning System (GPS) und die Lasernavigation[Guenther:2011].

- **Die physische Leitlinie:** Fahrerlose Transportsysteme, die auf physischen Leitlinien navigieren bzw. fahren, benutzen Einrichtungen am oder im Fußboden. Die verschiedenen Varianten sind:
- **Orientierung durch optische Leitspur:** Bei dieser Methode wird ein Farbstrich mit deutlichem Farbkontrast zum umgebenden Boden entweder lackiert oder mit einem speziellen Gewebeband aufgebracht. Eine geeignete Kamerasensorik unter dem Fahrzeug nutzt Kantendetektions-Algorithmen und errechnet so die Ansteuerungssignale

für den Lenkmotor [Guenther:2011]. Optische Verfahren dienen durch eine ständige Kurskorrektur dazu, eine hohe Fahrgenauigkeit zu erreichen.

- **Orientierung durch induktive Leitspur:** Diese Methode der Navigation fahrerloser Transportfahrzeuge ist profitabel aufgrund der permanenten Kurskorrektur und außerdem besonders zuverlässig und fahrzeugseitig durch die Nutzung einfacher Komponente zu realisieren. Es ist möglich, die Stromversorgung der Fahrzeuge fahrbahnseitig zu realisieren, sodass die Nutzung schwerer Akkumulatoren entfällt. Jedoch sind Systeme mit Leitdrahtsteuerung nicht flexibel und in der Konstruktion sehr teuer.



Abbildung 1: Prinzipskizze zur induktiven und optischen Spurführung (Quelle: Günter Ullrich, 2011 S. 79)

- **Orientierung durch Magnetmarken:** Eine weitere Möglichkeit der Steuerung ist die Abtastung von Magnetstreifen oder magnetischen Markierungen auf der Straßenoberfläche. Dabei bedarf es zur Berechnung der Leitlinie entweder der Koppelnavigation, oder der Peilung von in regelmäßigen Abständen in den Boden eingelassenen Marken. Diese Marken können rein passive Dauermagnete oder aber quasi-aktive Transponder sein [Guenther:2011]. Das Bild 2 ist eine Repräsentation der Navigation durch Magnetstreifen.

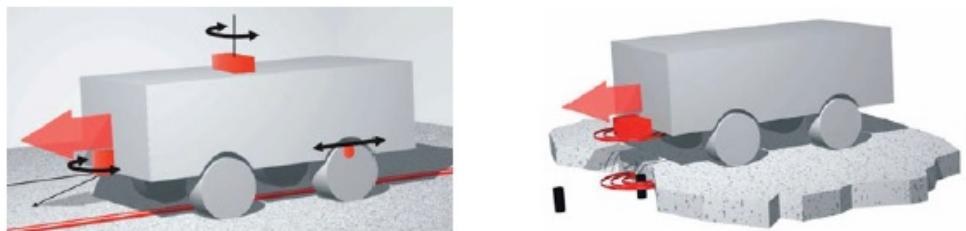


Abbildung 2: Prinzipskizze zur Koppelnavigation (links) und zur Magnet- bzw. Transpondernavigation (rechts) (Quelle: Günter Ullrich, 2011 S. 79)

- Bei der Lasernavigation bestimmt der Laserscanner die Position des FTF, dazu kommen noch optische Sensoren für die Erkennung von Hindernissen wie z. B. Menschen. Lasergeführte FTS bieten einen hohen Wert an Flexibilität, da sie ohne Bodeninstallation funktionieren. Nur bei engerem Raum kann die Lasernavigation nicht so effizient wie z. B. eine induktive Spurführung sein, wenn viele Fahrzeuge zum Einsatz kommen. Um die Systemvorteile einer Lasernavigation optimal zu nutzen, benötigt man allerdings ein passendes Anlagenkonzept. Die wichtigsten Kriterien sind: die Einbindung in das gesamtbetriebliche Materialflusssystem, die Anpassung an die vorhandenen Steuerungshierarchien und die optimale Auslegung der Technik in Bezug auf Fahrzeugbauart, Lastaufnahmemittel, Energiekonzept, Kommunikation und Leitsystem. Ein Aspekt, der für das Laser-geführte FTS spricht, ist die Wirtschaftlichkeit. Und dies trotz der Alternativen Elektro-, Low-Cost- sowie induktiv geführten FTS. Letztere lassen sich so einrichten, dass sie auch auf leitdrahtlosen, rein rechnergeführten Teilstrecken verkehren können. Keinerlei kostenintensive Bodeninstallation benötigt dagegen das über Lasersensor gesteuerte, völlig frei navigierende Laser-FTS. Die Fahrzeuge orientieren sich lediglich an im Raum verteilten Reflektoren und mit Hilfe der Kombination von Winkel- und Distanzmessung[Werner:2014:Online]. Das Bild 3 ist eine Visualisierung der Lasernavigation.



Abbildung 3: Prinzipskizze zur Koppelnavigation (links) und zur Magnet- bzw. Transpondernavigation (rechts) (Quelle: Günter Ullrich, 2011 S. 79)

- Orientierung durch GPS:** Seine Anwendung im Bereich der Fahrzeugsteuerung wird in Form des DGPS eingesetzt. DGPS bedeutet differential GPS und meint die Verwen-

dung eines zusätzlichen GPS-Empfängers, der nicht auf dem FTF, sondern stationär fest installiert ist. Mit Hilfe dieses ortsfesten GPS-Empfängers wird der sich zeitlich ändernde Fehler ermittelt, der dem GPS-System eigen ist. Mit Hilfe dieser Kenntnis können zeitgleich die fahrenden GPS-Empfänger auf den FTF exakte Positionen ermitteln[Guenther:2011]. Diese Navigationstechnik benötigt einen freien Sichtkegel von 15 Grad nach oben (siehe Bild 4), um zuverlässig arbeiten zu können. Die Schritte zur Erlangung der erforderlichen Fahr- und Positioniergenauigkeit sind:

- Prüfung der örtlichen Gegebenheiten, insb. der Empfangsstärken der Satelliten
- Einsatz des Differential-GPS
- Real Time Kinematic Differential GPS.



Abbildung 4: Prinzipskizze zur Koppelnavigation (links) und zur Magnet- bzw. Transpondernavigation (rechts) (Quelle: Günter Ullrich, 2011 S. 79)

Im Rahmen des Projekt FAISE wird die Navigation durch den Laser durchgeführt. Es kann hier kein Global Positioning System (GPS) verwendet werden, da das ganze Experiment in einem geschlossenen Raum gemacht wird. Weiterhin wird auch keine Navigation durch die physische Leitlinie oder durch die Stützpunkte im Boden erzielt, weil dazu der Boden aufgebrochen werden müsste.

## 2.2.2 Steuerungstechnik

Die interne Materialflusssteuerung ist eine Vorstufe der Transportauftragsabwicklung und wird nur dann benötigt, wenn die Transportaufträge nicht klar deziert übertragen, sondern aufbereitet werden müssen. Eine Anforderung wie z. B. benötige Ware A an Maschine B erfordert eine Umsetzung in einen oder mehrere Transportaufträge nach dem klassischen Muster. Hole von C und bringe nach D. Die FTS-interne Materialflusssteuerung kombiniert also Quelle und Senke über die in ihr hinterlegten Transportbeziehungen zu einem Transportauftrag und schickt diesen zur Durchführung an die Transportauftragsverwaltung. Diese ganze Transportauftragsverwaltung ist in der FTS-Leitsteuerung geregelt. Die FTS-Leitsteuerung ist die Kommandozentrale, um das FTS in das Umfeld zu integrieren.

Außerdem steuert es die FTF, die sich im System befinden. Damit ist das FTS dann in der Lage, die ihm übertragenen Aufträge zu erfüllen. „Eine FTS-Leitsteuerung besteht aus Hard- und Software. Kern ist ein Computerprogramm, das auf einem oder mehreren Rechnern abläuft. Sie dient der Koordination mehrerer Fahrerloser Transportfahrzeuge und/oder übernimmt die Integration des FTS in die innerbetrieblichen Abläufe.“ [VDI:1994](VDI 4451). Die Leitsteuerung bringt das FTS in seinem Umfeld zusammen, bietet seinen Bedienern vielfältige Service-Möglichkeiten und nimmt Transportaufträge entgegen. Weiterhin stellt sie den Aufgaben entsprechende Funktionsblöcke zur Verfügung. Die FTS-Leitsteuerung ist der Kern des FTS. In Rahmen des Projekt FAISE, wird auch eine Leiststeuerung benötigt. Eine Leiststeuerung ist nur mit Hilfe einer Systemarchitektur zu implementieren und zu verstehen. In seinem Buch Fahrerlose Transportsysteme, hat Günter Ulrich zwei verschiedene Systemarchitekturen dargestellt. Eine für ein einfaches FTS und eine andere für ein komplexes FTS. Da bei FAISE nur mit vier FTF gearbeitet wird, ist es sinnvoll mit einer einfachen Systemarchitektur zu arbeiten. Das Bild 3 ist eine Repräsentation einer einfachen Systemarchitektur.

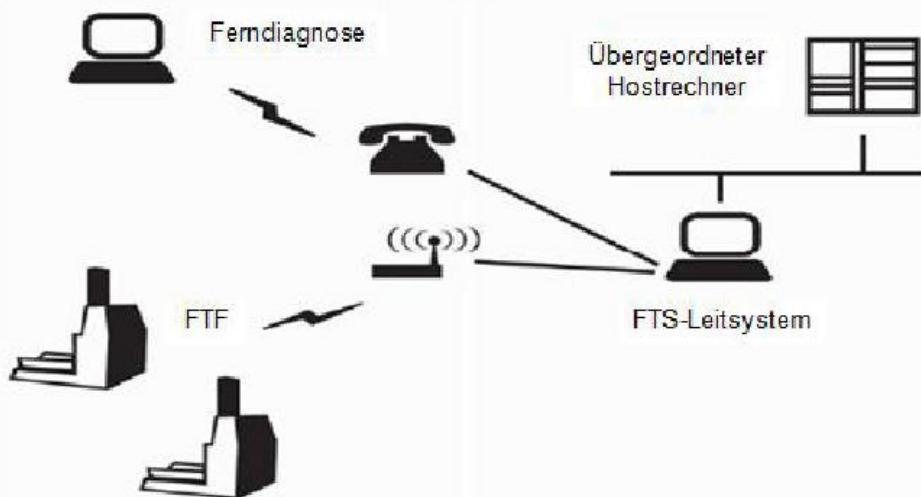


Abbildung 5: Die Systemarchitektur eines einfachen FTS (Quelle: Günter Ullrich, 2011 S. 93)

Es gibt eine geringe Anzahl von FTF, mit denen die Leitsteuerung per WLAN in Verbindung ist. Außerdem gibt es ein LAN, über das es eine direkte Verbindung mit einem übergeordneten Rechner gibt, von dem die Transportaufträge kommen. Über die angedeutete Telefonleitung ist eine VPN-Verbindung zur Ferndiagnose eingerichtet. Die Datenübertragung zu den übergeordneten Host-Rechnern erfolgt meist über lokale, Ethernet basierte Netzwerke mit dem Protokoll TCP/IP. Solche Host-Rechner können beispielweise Materialflussteuerungssysteme zur Produktionssteuerung (z. B. SAP) Produktionsplanungssysteme (PPS) Lagerverwaltungssysteme (LVS) sein[Guenther:2011]. Außerdem gehören nach der VDI

4451(Blatt 3) „zum internen Umfeld der FTF-Steuerung [...] das Lastaufnahmemittel (LAM), Sensoren und Aktoren, Bedienfeld am Fahrzeug und das Sicherheitssystem. Das externe Umfeld besteht aus der FTS-Leisteuerung, anderen FTF, automatischen Stationen und Gebäudeeinrichtungen“. Die Abbildung 1 stellt eine Darstellung einer FTF-Steuerung und ihr Steuerungsumfeld dar. Die administrative Ebene, die häufig über einen stationären Leitrech-

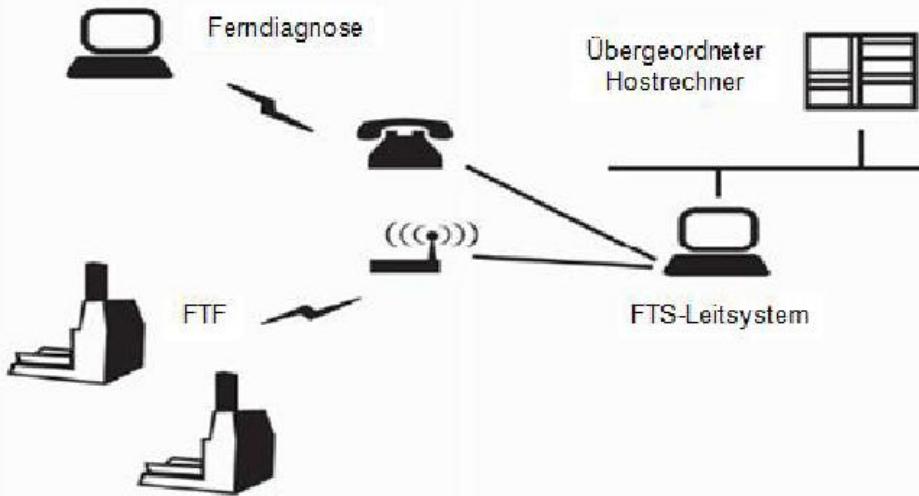


Abbildung 6: Allgemeine Darstellung einer FTF-Steuerung mit Datenschnittstellen (vgl. VDI 4451)

ner realisiert wird, verwaltet die Transportaufträge der ganzen Materialflussteuerung. Die operative Ebene, die auch als Fahrzeugsteuerung bezeichnet wird, erhält ihre Informationen über die Fahrzeugdisposition der administrativen Ebene. Der Funktionsblock Kommunikation leitet den stattgefundenen Datenaustausch zum Manager weiter. Dieser sorgt für die Koordination, indem er die Fahraufträge in einzelne Befehle aufteilt, sowie für ein reibungloses Zusammenwirken der einzelnen Funktionsblöcke. Neben dem Block Kommunikation sind weitere Blöcke vorhanden. Dazu gehört für die gesamte Lastübergabe inklusive der Lastlagererfassung verantwortliche Lastaufnahme, das Energiemanagement, welches den Lade- und Allgemeinzustand der Batterien überwacht, und der Block Überwachung/Sicherheitsschnittstelle, welcher zum Schutz der Personen und Sachgegenstände dient. Der Funktionsblock Fahren und die damit verbundene Sensorik bzw. Aktorik koordinieren die Ablaufsteuerung der Funktionen des Orientierungssystems [Langenbach:2012].

## 2.3 Materialflusssysteme

Damit ein Produkt auf den Markt kommen kann, muss man es durchdenken, erstellen und dann vermarkten. Die Produkterstellung und -vermarktung sind Prozesse des Wirtschaft-

tens. Vorprodukte oder Materialen werden von Beschaffungsmärkten in die Unternehmen geführt und dort durch besondere Produktionsprozesse transformiert. Am Ende der Produktion steht ein Endprodukt, das für den Konsum bereits ist. Die Produktion und Logistik von Gütern sind daher sehr wichtige Bereiche für den Unternehmenserfolg. Allerdings führen heute die unterschiedlichen Ausprägungen der Logistik z. B. in Produktions-, Handels- oder Verkehrsunternehmen zu einer terminologischen Differenzierung der Logistik. Der Materialflussbegriff leitet sich einfach von dem logistischen Konzept ab, in anderen Worten führt das Materialflusssystem in die Logistik zurück. Die Abbildung 2. dient zur Erläuterung einer konventionellen Wertschöpfungskette.

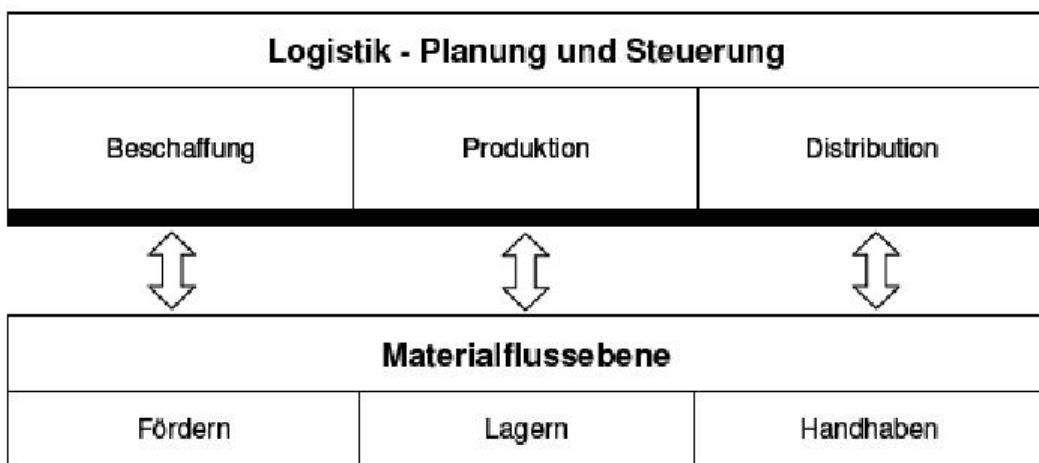


Abbildung 7: Elemente einer Wertschöpfungskette (vgl. Wulz, J., 2008, S. 7)

Der Begriff Materialfluss bedeutet die Verkettung aller Prozesse bei der Beschaffung, Bearbeitung, Verarbeitung sowie bei der Distribution von Gütern innerhalb festgelegter Bereiche. Deswegen lässt sich der Materialfluss in vier Stufen unterordnen: externer Transport, betriebsinterner Materialfluss, gebäudeinterner Materialfluss und Materialfluss am Arbeitsplatz. Nach dem Verein Deutscher Ingenieure bzw. VDI-241 beinhaltet die Logistik fünf Hauptfunktionen. Diese Funktionen sind Bearbeiten, Prüfen, Handhaben, Fördern, Lagern und Aufenthalten. Neben diesen Hauptfunktionen zählen auch Nebenfunktionen wie z. B. Montieren, Umschlagen, Kommissionieren, Palettieren und Verpacken [VDI:1978]. Jedoch sind auf der Ebene des Materialflusssystems nur drei Funktionen zu berücksichtigen: Fördern, Lagern, Handhaben. Die anderen Funktionen setzen sich normalerweise aus den erläuterten Funktionen zusammen. Dieser Arbeitsteil wird in zwei Teile gegliedert. Im ersten Teil werden die drei Funktionen der Materialflusssysteme vorgestellt. Im zweiten Teil wird eine Planung von Materialflusssystemen dargestellt.

### 2.3.0.1 Funktionen von Materialflusssystemen

- **Funktion Fördern**

Fördern bedeutet Transportieren und ist einer der wichtigsten Aspekte innerhalb des Materialflusssystems. Nach der VDI 2411 ist Fördern das Fortbewegen von Arbeitsgegenständen in einem System. „Die Fortbewegung oder Ortveränderung von Gütern oder Personen mit technischen Mitteln wird allgemein als Transport bezeichnet. Findet diese Ortsveränderung in einem räumlich begrenzten Gebiet wie beispielsweise innerhalb eines Betriebes oder Werkes statt, so wird dieser Vorgang durch den Begriff Fördern präzisiert. Das Fördern bzw. die Fördertechnik umfasst also das Bewegen von Gütern und Personen über relativ kurze Entferungen einschließlich der dazu notwendigen technischen organisatorischen und personellen Mittel“[Ten:2008]. Das Fördermittel (technisches Transportmittel zur Ortsveränderung von Gütern oder Personen) und das Förderelement bilden den physikalischen Bestandteil eines Fördervorgangs. Der Ablauf und die Steuerung werden durch den Fördervorgang dargestellt. In Punkt Fördermittel kann auf verschiedenste Elemente der Materialflusstechnik zurückgegriffen werden. Dies umfasst unter anderen Rollenbahnen und FTS. Neben der Möglichkeit auf automatisierte Fördermittel zurückzugreifen, kommen auch manuell mechanisierte bzw. rein manuelle Systeme zum Einsatz. In diesem Fall ist der Mensch oder der Bediener eines Fördermittels wesentlich für den Ablauf eines reibungsfreien Materialflusses in Zusammenspiel mit den physikalischen Elementen sowie dem Prozessablauf verantwortlich[Wulz:Johannes:2008](Wulz, J, 2008, S. 8). Das Bild 4 gilt als Beispiel eines Fördersystems.

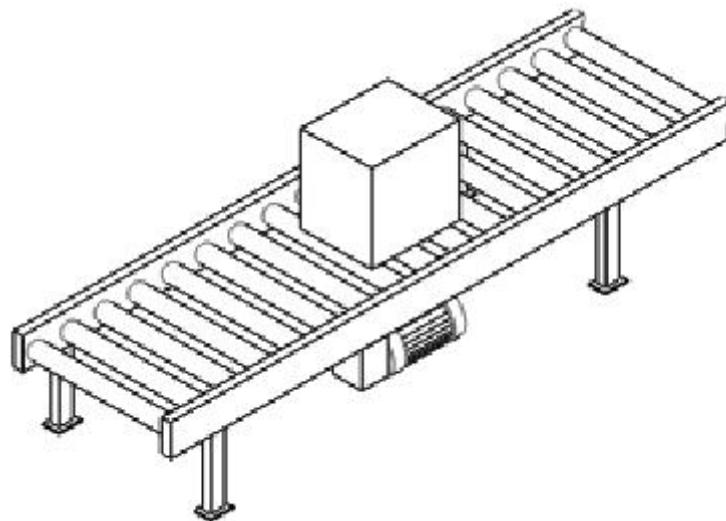


Abbildung 8: Beispiel eines Stetigförderers (entnommen aus Ten Hompel, Schmidt, Nagel, 2007, S. 131)

- **Funktion Lagern**

Das Lagern ist jedes geplante Liegen des Arbeitsgegenstandes im Materialfluss. Das Lager ist ein räumlich abgegrenzter Bereich bzw. eine Fläche zum Aufbewahren von Stück- und/oder Schüttgütern in Form von Rohmaterialien, Zwischenprodukten oder Endprodukten, das mengenmäßig erfasst wird[VDI:1978]. Die Einlagerung von Lagereinheiten, die Aufbewahrung und Bereithaltung von Lagereinheiten auf Lagerplätzen und die Auslagerung einer Lagereinheit, sind die grundlegenden Prozesse in einem Lager. Aufgrund der starken Veränderungen im Markt, müssen auch die unternehmerischen Abläufe an Lagersysteme schnell angepasst werden. In einem Lagersystem werden im Verlauf des Materialflusses Speicher- bzw. Lagerfunktionen sowie Förderfunktionen wahrgenommen. Aufgabe eines Lagers ist das Bevorraten, Puffern und Verteilen von Gütern. Während Vorratslager lang- und mittelfristige und Pufferlager kurzfristige Bedarfsschwankungen ausgleichen sollen, erfüllen Verteillager neben der Bevorratungs- noch eine Kommissionierfunktion. Daher können die Aufgaben eines Lagers anhand folgender Ausgleichsmaßnahmen beschrieben werden: Zeitausgleich, Mengenausgleich, Raumausgleich und Sortimentsausgleich[Stich:Bruckner:2002]. Ein Zeitausgleich ist immer dann erforderlich, wenn die Zeitfunktion der Nachfrage nicht der Zeitfunktion der Produktion entspricht. Beispielsweise steht eine losgrößenoptimierte Fertigung einer saisonalen Nachfrage gegenüber. Gerade in Bereichen mit Serienfertigung, in denen aus Kostengründen in der Regel größere Mengen als die Nachfragemengen produziert werden, muss Mengenausgleich vollzogen werden. Sobald der Produktionsort nicht mit dem des Produktabnehmers übereinstimmt, findet mit Hilfe von Verkehrsträgern ein Raumausgleich statt. Mit zunehmender Sortimentsbreite steigt die Wahrscheinlichkeit, dass die Anzahl der Produktionsstandorte steigt[Langenbach:2012].

- **Funktion Handhaben**

Der Begriff Handhaben wurde gedanklich von der menschlichen Hand abgeleitet, wird aber auch für automatisch ablaufende Vorgänge zur Manipulation von Objekten gebraucht. Handhaben bedeutet etwas greifen, bewegen und an einem bestimmten Ort ablegen. Das heißt, durch Handhaben wird die Lage oder Position von Objekten geändert. Im übertragenen Sinne bedeutet handhaben auch bewerkstelligen bzw. praktisch ausüben. Von Handhabungstechnik spricht man, wenn für die Handhabung Geräte eingesetzt werden. Die Richtlinie VDI 2860 definiert die Funktion Handhaben als „das Schaffen, definiertes Verändern oder vorübergehendes Aufrechterhalten einer vorgegebenen räumlichen Anordnung von geometrisch bestimmten Körpern.“ Die Teilfunktionen des Handhabens stellen das Speichern, das Bewegen, das Sichern, das Kontrollieren und das Verändern von Gütern dar. Das Handhaben kann sowohl als eine Funktion als auch eine Fertigung des Materialflusses betrachtet werden. Eine mögliche Handhabungsfunktion im Materialfluss ist z. B. das Palettieren, worunter

die Stapelung von Stückgütern zu einem Stückgutstapel nach einem gewissen Muster verstanden wird. Handhabungsfunktionen können entweder von Automaten z. B. Roboter oder von Menschen durchgeführt werden. Auf Grund der Greifflexibilität ist der Mensch jedoch meist unübertroffen in der Handhabung.

## 2.4 Multiagentensysteme

Zur Umsetzung der dezentralen Steuerung werden Softwareagenten eingesetzt. Bei Software-Agenten handelt es sich um Prozesse, die lose gekoppelt und leicht austauschbar sind [GH:2010]. Es existieren verschiedene Definitionen eines Agenten, von denen sich keine als Standard etablieren konnte. Die hier verwendete Definition stammt von Brenner, Zarnekow und Wittig. Sie definieren einen Agenten als „... ein Softwareprogramm, das für einen Benutzer bestimmte Aufgaben erledigen kann und dabei einen Grad an Intelligenz besitzt, der es befähigt, seine Aufgaben in Teilen autonom durchzuführen und mit seiner Umwelt auf sinnvolle Art und Weise zu interagieren“[BZW:1998]. Ein System, das aus mehreren kommunizierenden und interagierenden Agenten besteht heißt auch Multiagentensystem (MAS). Ein wesentlicher Vorteil von MAS bzw. von verteilten Steuerungssystemen ist die Fähigkeit, dynamisch auf Veränderungen zu reagieren. Ein Beispiel für eine solche Veränderung ist der Ausfall einer Steuerungseinheit beziehungsweise eines Agenten. Der Ausfall einer Einheit hat nicht unbedingt zur Folge, dass das gesamte System ausfällt. Die restlichen Einheiten können sich eigenständig auf eine solche Veränderung einstellen und diese beim weiteren Ablauf berücksichtigen[Roidl:2012]. Im Falle eines Materialflusssystems gilt gleiches auch für die Erweiterbarkeit des Systems: Reicht die Kapazität des Systems nicht mehr aus, können neue Module und somit auch Agenten mit erheblich geringerem Aufwand hinzugefügt werden, als dies bei statischen Systemen der Fall wäre.

Die Modellierung von agentenbasierten Systemen für industrielle Bereiche wird durch die Entwicklung von Standards festgelegt. Diese beschreiben Modelle für die Architektur sowie die Kommunikation zwischen Agenten. Die FIPA (Foundation of Intelligent Physical Agents) ist das Standardisierungsorgan für Agentensysteme. Seit der Gründung 1996 in der Schweiz wurden verschiedene Standardisierungen veröffentlicht, so zum Beispiel auch die Agentenkommunikation (Agent Communication), die als FIPA / ACL (Agent Communication Language, ACL) bekannt geworden ist. Jeder Agent ist mit einem eindeutigen Identifikationsnamen (Agent Identifier oder AID) versehen und wird im Agent Management System verwaltet[Roidl:2012].

## 2.5 Fallbeispiele

### 2.5.1 FTS in der Gläsernen Manufaktur Dresden (Volkswagen)

Volkswagen AG montiert das neue Modell der Luxusklasse „Phaeton“ in der „Gläsernen Manufaktur“ in Dresden. Die Materialversorgung übernimmt ein fahrerloses Transportsystem mit 56 frei navigierenden Fahrzeugen. Die gesamte Steuerungs- und Navigationstechnik stammt von FROG Navigation Systems, dem Projektpartner des Generalunternehmers AFT (Mechanik). Die Produktion ist auf drei Ebenen unterteilt: Die eigentliche Montage findet auf den beiden oberen Montageebenen statt: Die Rohkarosse befindet sich auf einer Montageplattform, die Teil des Schuppenbandes ist, das sich sicher in den Hallenboden einfügt und mit konstanter Geschwindigkeit durch die Montagezyklen bewegt. Danach erfolgt die Übergabe an eine schwere Elektrohängelift (EHB) zur Hängemontage. Während der Hängemontage erfolgt die Hochzeit, d. h. das Zusammenfügen von Karosse und Triebwerk, wobei der Triebwerk von einem Fahrerlosen Transportfahrzeug (FTF) herangebracht wird. Anschließend wird die Karosse wieder auf eine Schubplattform, die sogenannte Schuppe, zur Komplettierung und Qualitätskontrolle gestellt.

Im Untergeschoss, der Logistikebene, wird die verbauende Ausrüstung zur Verfügung gestellt und in Betrieb genommen. Die FTS übernimmt die Versorgungsleitungen der Materialien und damit eine erhebliche logistische Funktion. Um zwischen den Ebenen zu wechseln, nutzen die automatischen Fahrzeuge Hebebühnen. Das FTS hat die grundsätzliche Aufgabe, die Montagelinien (Schuppenband oder EHB) zu versorgen. Dabei wird allerdings zwischen folgenden sechs Gewerken unterschieden:

1. Anlieferung von Warenkörben auf die Schuppe
2. Anlieferung von Schalttafeln (Cockpits)
3. Anlieferung von Kabelsträngen
4. Anlieferung des Triebwerks mit Fahrwerk und Ausführung der Hochzeit
5. Anlieferung von Warenkörben zur Hängemontage
6. Anlieferung der Türen plus Warenkörbe

### 2.5.2 FTS beim Automobilhersteller BMW im Werk Leipzig

Das BMW-Werk in Leipzig hat im Jahre 2005 mit der Produktion der 3er Reihe (E90) gestartet. Im Bereich der Teileversorgung übernimmt erstmals in der Geschichte der Automobilind

dustrie ein Fahrerloses Transportsystem (FTS) umfangreiche Logistikfunktionen. Folgende Prozesse wurde für die Teilversorgung im Leipzig-Werk definiert:

- Direktanlieferung per LKW: Große Teile mit geringer Komplexität (z. B. Bodenmatte oder Kofferraumverkleidung) werden per LKW zeitnah und in unmittelbarer Nähe des Verbauortes angeliefert.
- Modulanlieferung per EHB8: Große und komplexe Baugruppen (z. B. Cockpit) werden direkt auf dem Werksgelände von externen Lieferanten oder BMW Mitarbeitern montiert.
- Lagerware per FTS: Die Mehrzahl der Teile wird in einem Versorgungszentrum gelagert, kommissioniert und mit Fahrerlosen Transportfahrzeugen (FTF) an die jeweiligen Verbauorte in der Montage gebracht [Guenther:2011].

Es sind 74 FTF im Einsatz, als Ladehilfsmittel werden mehr als 2.000 Rollwagen in zwei unterschiedlichen Ausführungen eingesetzt. Je FTF werden entweder zwei kleine Rollwagen, zur Aufnahme von Behältern bis DIN-Größe, oder ein sogenannter übergroßer Rollwagen zur Aufnahme von Großbehältern eingesetzt. Zusätzlich gibt es noch die Sequenziergestelle mit Sonderaufbauten [Guenther:2011]. Durch einen Laser-Scanner auf dem FTF werden der Personenschutz und die Hinderniserkennung übernommen.

Die Fahrerlosen Transportfahrzeuge finden ihren Weg mit Hilfe der so genannten freien Navigation. Damit ist gemeint, dass die Fahrzeuge ohne physikalische Leitspuren und nach einem kombinierten Prinzip aus Kopplung und Peilung arbeiten. Kopplung bedeutet die Auswertung von fahrzeuginternen Sensoren (Messräder und ein faseroptischer Kreisel), wodurch der zurückgelegte Weg samt Kurven bestimmt wird [Guenther:2011]. Bei jeder Peilung werden aufgetretene Fahrfehler, die durch Schlupf der Räder oder durch Veränderungen des Raddurchmessers auftreten können, korrigiert. Die Vorteile dieses, auch Magnet Navigation genannten, Verfahrens liegen in der Zuverlässigkeit und der Flexibilität bei zukünftigen Layoutanpassungen [Guenther:2011].

## 3 Projektorganisation

Die nachfolgenden Abschnitte beschreiben den organisatorischen Ablauf innerhalb der Projektgruppe. Dazu gehört die Aufteilung in Gruppen, sowie die zeitliche Planung und Rollenverteilung. Des weiteren werden hier das Vorgehen für das gesamte Projekt und die verwendeten Werkzeuge beschrieben.

### 3.1 Organisation der Teilgruppen (Materialfluss, Fahrzeuge, Simulation)

Die Projektgruppe ist in drei Teilgruppen unterteilt, da das Gesamtprojekt durch die drei Komponenten in eindeutig abgrenzbare Aufgabenfelder gegliedert ist und so Kompetenzen und Verantwortlichkeiten klar definiert werden können.

- **Materialfluss**

Die Teilgruppe Materialfluss befasst sich mit Programmierung der Sensorik und Aktorik für die Rampen, sowie dem Aufbau eines Sensornetzwerkes zur Kommunikation zwischen den verschiedenen Akteuren des physischen Systems.

- **Fahrzeuge**

Die Teilgruppe Fahrzeuge befasst sich mit allen Aspekten, die für das Funktionieren der Fahrzeuge verantwortlich sind. Dazu zählen unter anderem die Navigation, Odometrie und Lokalisierung. Auch ist die Einrichtung der Versorgungsinfrastruktur für die Fahrzeuge in Form von Ladestationen fällt in den Aufgabenbereich der Fahrzeuggruppe. Zusammen entwickeln die Teilgruppen Fahrzeuge und Materialfluss das physische System, so dass Kommunikation und Abstimmung zwischen diesen beiden Gruppen besonders wichtig sind.

- **Simulation**

Die Teilgruppe Simulation entwickelt die Software mit der eine virtuelle Simulation erstellt wird.

### 3.2 Rollenverteilung

Um organisatorische Aspekte innerhalb des Projektes besser umsetzen zu können, wurden unterschiedliche Rollen definiert, die jeweils einen Bereich des Projektes abdecken sollen. Dazu gehören folgende Aufgaben:

- **Administrator**

Der Administrator ist für die Einrichtung und Betreuung der Server und Tools zuständig, dazu zählt auch die Einrichtung der Webseite.

- **Aussendarstellung**

Um das Projekt vernünftig zu Repräsentieren, verwalten die zuständigen der Aussen-darstellung den Inhalt der Webseite. Ausserdem sind sie für die externen Kontakte und Events / Präsentationen verantwortlich.

- **Dokumentenbeauftragte**

Damit am Ende ein einheitliches Format für den Endbericht gilt, organisieren, sammeln und verwalten die Beauftragten jegliche Quellen und Berichte ( dazu zählt auch das Repository ). Des weiteren sind sie Ansprechpartner bei Fragen zur Literatur.

- **Gruppenleiter**

Da das Projekt aus drei Teilgruppen besteht, besitzt jede Gruppe einen eigenen Gruppenleiter, der die Prozesse innerhalb der Gruppe lenkt und gemeinsam mit den anderen Gruppenleitern das gesamte Projekt koordiniert.

- **Qualitätsmanagement**

Damit der Projektplan eingehalten wird, ist es Aufgabe des Qualitätsmanagements, dass die Prozesse ( Scrums ) eingehalten und korrekt ausgeführt werden. Zusätzlich sind sie für die System und Integrationstests verantwortlich.

- **Werkzeugbeauftragte**

Die Werkzeugbeauftragten verwalten die benötigten Geräte und Schlüssel für die gesamte Projektgruppe. Weiterhin regeln sie, mit Rücksprache mit den Betreuern, den Einkauf der Hardware und Software.

### 3.3 Vorgehensmodell

Für die Durchführung der Projektgruppe muss ein Vorgehensmodell, sowohl für die Gesamt- als auch für die Teilgruppen, festgelegt werden. Durch ein Vorgehensmodell wird die Arbeit im Team strukturiert und es wird festgelegt, wie bestimmte Aufgaben, wie z.B. Abgleich mit Kunden und Anwendern, umgesetzt werden sollen. Sowohl für die Teilgruppen als auch für die Gesamtgruppe wurde ein Scrummodell als Vorgehensmodell gewählt ( siehe Abbildung 9).

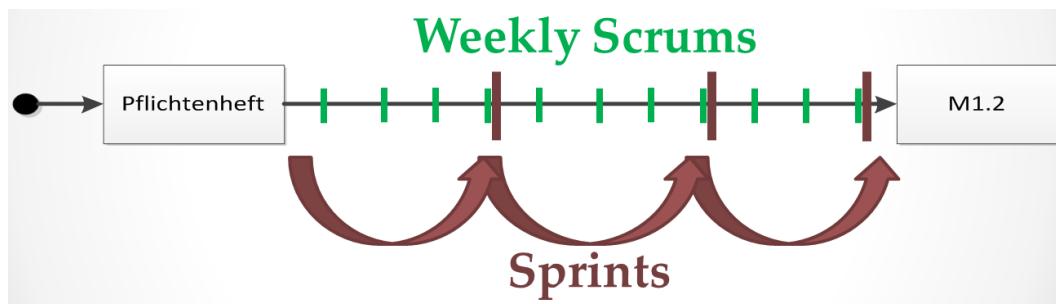


Abbildung 9: Scrumplanung für einen Meilenstein

Da es für ein sehr komplexes Projekt, wie das vorliegende, schwierig ist nur von einer groben Vision, sowie von User Stories auszugehen, wurden zunächst auf Basis des vorliegenden Lastenhefts in jeder Teilgruppe Pflichtenhefte erstellt. Anschließend wurden die dazugehörigen User Stories definiert, die die Anforderungen aus dem Pflichtenheft berücksichtigen und aus Anwendersicht darstellen. Die Sprints in den Teilgruppen sind mit einem Monat bemessen und werden zur Durchführung der User Stories genutzt. Die Rolle des Product Owners wird von den beiden Betreuern eingenommen, die sowohl für die Teil- als auch für die Gesamtgruppen zur Verfügung stehen, um die entwickelten Funktionalitäten abzugleichen. Die Durchführung von Daily Scrums ist zeitlich nicht möglich, da es sich um eine studentische Projektgruppe handelt, deren Stundenplan keine täglichen Treffen ermöglicht. Deshalb wurde das Scrum Vorgehensmodell dahingehend angepasst, dass die Daily Scrums in Weekly Scrums abgewandelt wurden. Die Weekly Scrums finden sowohl in den Teilgruppen als auch in der Gesamtgruppe statt. In den Weekly Scrums der Gesamtgruppe wird zunächst von jeder Person berichtet, welche Aufgaben in der vorherigen Woche erledigt wurden, damit entstandene Probleme und Hindernisse direkt in der Gruppe besprochen und eventuell beseitigt werden können. Die Ergebnisse aus den Teilgruppen werden ebenfalls vorgestellt und mit den Product Ownern abgeglichen. Das Scrum Vorgehensmodell wird mit Prototyping kombiniert (siehe Abbildung 10).

Durch das Prototyping sollen zu bestimmten Meilensteinen die kombinierten Ergebnisse aus den Teilgruppen vorgestellt werden, um den Stand des Gesamtsystems begutachten zu können. Betrachtet man das gesamte Projekt, so besteht es aus 2 Prototyping Phasen. Diese trennen sich im zweiten Meilenstein. Bis zu diesem Zeitpunkt wird mit horizontalen Prototyping die Basis des Projektes geschaffen. Das bedeutet, dass alle Grundfunktionalitäten implementiert und umgesetzt werden. Danach folgt das vertikale Prototyping in dem die Funktionalitäten um weitere Aspekte und Feinheiten erweitert werden. Innerhalb dieser beiden Phasen kommt es immer wieder zu Aufgabenbereichen, die jede Teilgruppe für sich umsetzt. Ebenfalls sind Phasen vorhanden, in denen die Ergebnisse der einzelnen Gruppen zusammengeführt werden müssen, wodurch das Zusammenspiel zwischen dem Material-

fluss und der Volksbots, sowie die Darstellung in der Simulation, entsteht.



Abbildung 10: Modell für die Darstellung der einzelnen Projektphasen

### 3.4 Werkzeuge

Zur Unterstützung der Zusammenarbeit in der Projektgruppe wurden verschiedene Werkzeuge ausgewählt um die Kollaboration zu vereinfachen. Um eine gute Verknüpfung der Werkzeuge zu gewährleisten, wurde versucht die Produkte aus einer Hand zu beziehen. Nach einiger Recherche, stachen zwei Möglichkeiten heraus. Entweder die Open-Source Software Redmine oder eine kommerzielle Lösungen von der Firma Atlassian.

Die beiden Lösungen wurden daraufhin auf Basis verschiedener Kriterien verglichen. Zu diesen Kriterien zählten mitunter Usability, Verbreitung am Markt, Stabilität und Wartbarkeit.

Die Evaluation ergab Atlassian als klaren Sieger und die Projektgruppe einigte sich im speziellen auf die Werkzeuge **Jira** und **Confluence**.



Abbildung 11: Hersteller der Projektwerkzeuge

### 3.4.1 Jira

Jira ist ein Projektmanagement-Tool. Neben dem Verwalten von Vorgängen und Fehlern, lässt sich Jira mit einer *Agile* Erweiterung sehr gut zum planen, steuern und verwalten von Sprints. Eine einfache Übersicht über die verschiedenen, anstehenden Aufgaben können die Projektmitglieder über die sogenannten Boards erhalten. Hier wird genau für die verschiedenen User Stories deren Fortschritt angezeigt und welches Projektmitglied gerade welche Aufgabe bearbeitet.

### 3.4.2 Confluence

Confluence ist ein Wiki welches sich sehr gut in Jira integriert. So muss die Benutzerverwaltung nur einfach in Jira eingerichtet werden. Confluence greift dann auf das Benutzerverzeichnis von Jira zurück.

## 4 Allgemeine Anforderungen

In diesem Abschnitt sollen allgemeine Anforderungen beschrieben werden, die für das Gesamtsystem gelten. Dazu wird ein Ablaufszenario beschrieben, dass beschreibt, wie die Akteure (Rampen und Fahrzeuge) miteinander interagieren, damit ein vollautomatisierter Warenfluss entsteht.

### 4.1 Ablaufszenario

Das Ablaufszenario beschreibt die Schritte, die von den Akteuren ausgeführt werden, um das Ziel zu erreichen, aus logischer Sicht ohne dabei auf die technischen Implementierungsdetails einzugehen. Es dient als Basis für die Implementierung sowohl des physischen Systems als auch der Simulationssoftware. Abbildung 12 zeigt den schemenhaften Aufbau des Umschlagslagers. Die Rampen unterteilen sich in Eingangs-, Zwischenlager- und Ausgangsrampen abhängig von ihrer Position. Pakete treffen am Eingang ein und werden dort aufgenommen.

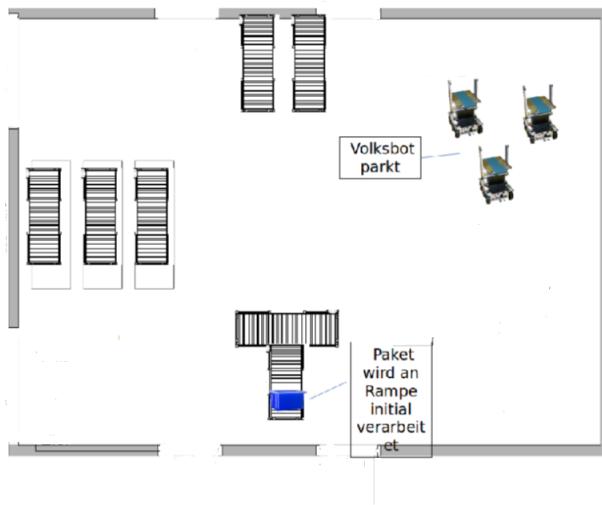


Abbildung 12: Teil 1 Ablaufszenario

Die Eingangsrampen suchen für das Paket ein Ziel, indem das Zwischenlager gefragt wird, ob Platz vorhanden ist oder ob am Ausgang Bedarf besteht. Sobald ein Ziel gefunden ist, sucht die Rampe über eine Auktion ein Transportmittel. Die Bots berechnen eine Aufwandsabschätzung und schicken diese an die Rampe, die einen Bot auswählt (Vgl.13).

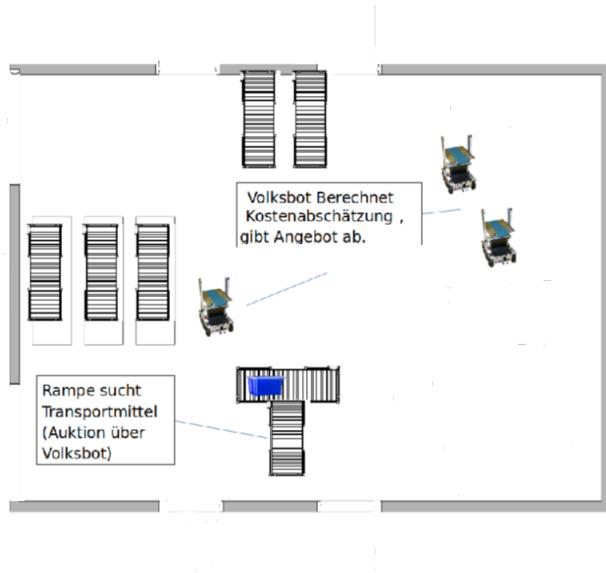


Abbildung 13: Teil 2 Ablaufszenario

Die Bots fahren zu den Eingangsrampen, laden die Pakete auf und bringen diese zu den Zwischenrampen oder Ausgangsrampen (Vgl.14). Pakete, die im Zwischenlager ankommen, werden analog dem beschriebenen Ablaufmuster von den Bots zu den Rampen gebracht. Der einzige Unterschied besteht darin, dass die Ausgangsrampen anfragen, ob ein gewünschtes Paket im Zwischenlager vorhanden ist.

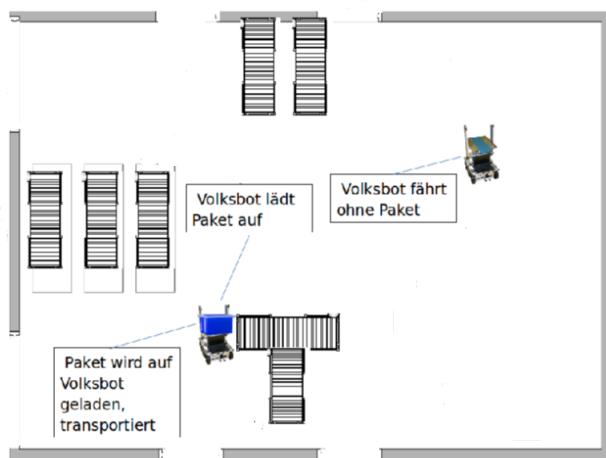


Abbildung 14: Teil 3 Ablaufszenario

## 5 Teilbericht Simulation

Der Teilbericht der Simulationsgruppe beschreibt die entwickelte Simulationssoftware von der Anforderungsanalyse über die Implementierung hin zum Testen und Validieren. Im Lastenheft werden, die für die Simulation vorgegebenen Anforderungen beschrieben. Grundlegende Designentscheidungen, die vor dem Erstellen der Anforderungen getroffen werden müssen, sind ebenfalls Teil der nachfolgenden Kapitel. In dem darauffolgenden Kapitel werden die jeweiligen Systemkomponenten konzipiert. Anschließend soll beschrieben werden, wie die Komponenten implementiert wurden. Das letzte Kapitel dieses Teilberichts beinhaltet Testen und Validieren des entwickelten Systems.

### 5.1 Lastenheft

In der Teilgruppe Simulation soll auf Basis des Ablaufkonzepts (Vgl. Abschnitt 4.1) ein Softwaretool entwickelt werden, das es erlaubt, einen automatisierten Materialfluss auf Basis von FTS zu simulieren, ohne dabei an die zahlenmäßigen Beschränkungen des physischen Systems gebunden zu sein. Vonseiten des Auftraggebers wurde ein Lastenheft vorgegeben, das die gewünschten Kernfunktionalitäten der Software beschreibt. Es enthält folgende Anforderungen:

1. **Akteure:** Die virtuellen Akteure ähneln in ihrem Verhalten und Eigenschaften (Geschwindigkeit, Dauer einer Paketübergabe etc.) den echten Objekten aus dem physischen System (Volksbots und Rampen).
2. **Ablauf:** Der in Abschnitt 4.1 beschriebene Ablauf, wird durch die Software simuliert.
3. **Visualisierung:** Die Zustände der Akteure werden dynamisch visualisiert. Wird beispielsweise die Anzahl der Pakete auf einer Rampe um eins erhöht, dann soll dies unmittelbar in der Anzeige visualisiert werden.
4. **Generierung von Aufträgen:** Eingehende und ausgehende Transportaufträge können erstellt und ausgeführt werden.
5. **Einstellungen:** Parameter der Simulation (Anzahl und Art der Akteure, Anzahl der Aufträge etc.) können vom Nutzer vor dem Starten der Simulation angepasst werden.
6. **Statistiken:** Es werden wichtige Daten geloggt, um am Ende eines Simulationslaufs aussagekräftige Analysen machen zu können.

## 5.2 Grundlegende Designentscheidungen

Vor der Entwicklung der Simulationssoftware mussten grundlegende Designentscheidung getroffen werden. Zum einen musste entschieden werden, ob die Software auf Basis eines vorhanden Tools oder komplett neu entwickelt werden sollte. Auch musste zwischen Desktop- und Webanwendung entschieden werden und ob die jeweilige Alternative mit oder ohne Zuhilfenahme eines Frameworks implementiert wird. In den nachfolgenden Abschnitten werden die getroffenen Designentscheidungen begründet.

### 5.2.1 Eigenentwicklung

Im Vorfeld der Entwicklung wurde der Teilgruppe Simulation das Player/Stage Tool als Alternative zu einer kompletten Neuentwicklung einer Software vorgeschlagen. Das Tool beinhaltet zum einen die Komponente Player, die eine Hardware Abstraktionsschicht darstellt. Mit dieser Komponente kann mit Robotern, wie beispielsweise einem Volksbot, interagiert werden, ohne dass technische Details der Komponenten (Laserscanner, Motor etc.) bekannt sein müssen. Auf Basis von selbstgeschriebenem Code können Roboter gesteuert werden. Die Komponente Stage horcht auf die Befehle, die Player ausführt und visualisiert diese in einem eigenen Graphical User Interface. Jedoch kann Stage auch ohne Hardware benutzt werden, indem man über Konfigurationsdateien ein eigenes Szenario erstellt und die virtuellen Roboter über den eigenen Code steuert. Somit bietet das Tool die Möglichkeit, eine Simulation mit Robotern zu erstellen und das gewünschte Verhalten der Roboter über eigenen Code abzubilden. Auch muss die Visualisierung nicht selbst entwickelt werden (Vgl.[**plstg**]).

Dennoch wurde eine Eigenentwicklung der Nutzung des Tools vorgezogen. Die Benutzeroberfläche von Stage bietet die Möglichkeit, die Anzahl der Roboter und das Layout eines Szenarios über die entsprechenden Konfigurationsdateien einzustellen (Vgl.[**plstg**]). Jedoch gibt es beispielsweise keine Möglichkeit Aufträge zu erstellen bzw. zu simulieren oder Statistiken anzuzeigen. Somit ist die Entwicklung einer eigenen Benutzeroberfläche unumgänglich. Das bedeutet, dass die Stage Oberfläche über eine eigene Benutzeroberfläche gesteuert werden muss. Somit hätte man eine Trennung zwischen Visualisierung und Konfiguration eines Szenarios, was die Benutzerfreundlichkeit erheblich beeinträchtigt, da ein Nutzer den Durchlauf einer Simulation über zwei Benutzeroberflächen hinweg verfolgen müsste. Die Entwicklung eines eigenen Systems bietet somit erheblich mehr Benutzerfreundlichkeit und ermöglicht es, alle Anforderungen an das Interface in einer Benutzeroberfläche zu integrieren.

### 5.2.2 Entwicklung einer Webanwendung

Die Software soll als Webanwendung implementiert werden. Gegenüber einer Desktopanwendung bietet eine Webapplikation folgende Vorteile:

- Das System ist plattformunabhängig und kann somit auf jedem Rechner, der über einen Webbrowsert verfügt, ausgeführt werden.
- Die Software muss nicht lokal installiert werden und kann direkt genutzt werden.
- Werden Änderungen an der Software vorgenommen, sind diese direkt verfügbar, da Updates über den Webserver eingespeist werden. Die Software ist somit immer auf dem aktuellsten Stand.

### 5.2.3 Umsetzung durch GWT

Die Entwicklung der Webanwendung sollte mithilfe eines Frameworks erfolgen, das es erlaubt, den Code sowohl für die Client- als auch für die Serverseite in einer Programmiersprache zu entwickeln. Außerdem sollte das Framework Schnittstellen bieten, um asynchrone Kommunikation und Push-Dienste zu nutzen, ohne sich um die exakten Details kümmern zu müssen. Ausgewählt wurde das Google Web Toolkit (GWT). GWT ist ein von Google entwickeltes Framework zur Erstellung von Webanwendungen. Der Java-Code für den Client wird von dem GWT Compiler in den entsprechenden Javascript- und HTML-Code übersetzt. Somit kann die Entwicklung sowohl für Client als auch für den Server auf Basis von Java erfolgen. Zudem entfällt die Anpassung des Javascript-Codes für die verschiedenen Browser, da GWT beim Kompilieren automatisch für jeden Browser eine lauffähige Version erzeugt. Weiterhin besitzt GWT eine RCP-Schnittstelle für die Kommunikation zwischen Client und Server und lässt sich um Komponenten erweitern, um Daten vom Server zum Client zu pushen (Vgl.[[gwt](#)]). Somit erfüllt GWT sämtliche an ein Framework gestellte Anforderungen. Weitere Alternativen wurden nicht in Betracht gezogen, da drei von fünf Mitgliedern der Teilgruppe Simulation bereits positive Erfahrungen mit GWT gemacht haben und die anderen Mitglieder somit schnell einarbeiten konnten.

## 5.3 Konzeption der Systemkomponenten

In diesem Abschnitt wird die Konzeption der Gesamtarchitektur als auch der einzelnen Systemkomponenten beschrieben, die im Rahmen der Sprints erarbeitet wurde. Die Konzeption beinhaltet zum einen die Anforderungen, als auch die daraus abgeleiteten Implementierungsvorgaben.

### 5.3.1 Gesamtarchitektur

Wie in Kapitel 5.2.2 beschrieben, soll die Software als Webanwendung realisiert werden. Eine Webanwendung erfordert eine Client-Server Architektur. Abbildung 15 beschreibt die wesentlichen Komponenten des Systems und wie diese sich auf die Client- und Serverseite verteilen. Basis der Anwendung ist der Webserver, der die Basiskomponenten des Systems hosted. Zum einen stellt er die Laufzeitumgebung für Webanwendung und Datenbank bereit. Die Datenbank wird benötigt, um Daten, wie beispielsweise erstellte Szenarien, persistent zu speichern. Aus der Webanwendung heraus kann auf die Datenbank lesend und schreibend zugegriffen werden. In die Webanwendung soll ein Multiagentensystem (MAS) eingebettet werden. Ein MAS ist ein Netzwerk aus Softwareagenten. Softwareagenten sind Softwareeinheiten, die in der Lage sind, Aufgaben selbstständig durchzuführen (Vgl. [mas]). Mithilfe des MAS können die im Ablaufszenario beschriebenen Aktionen softwareseitig auf die Akteure abgebildet werden.

Der Webserver beinhaltet die Logik des Systems. Auf dem Client soll die Visualisierung erfolgen und der Nutzer soll das Starten einer Simulation initiieren können. Das System soll von einem Webbrowser aus aufrufbar sein, in dem die Ergebnisse der serverseitigen Prozesse dargestellt werden.



Abbildung 15: Gesamtarchitektur

### 5.3.2 Konzeption der Benutzeroberfläche

Die Benutzeroberfläche soll einem Nutzer die Möglichkeit bieten, auf sämtliche Funktionen, die für die Durchführung eines Simulationsdurchlaufs relevant sind, zuzugreifen. Abbildung 16 zeigt den schematischen Aufbau der Gui anhand eines Mockups. Die Menüleiste beinhaltet drei Menu Items: Simulation, Auftragsliste und Statistiken. Über die Items Simulation und Auftragsliste sollen erstellte Szenarien und Auftragslisten geladen und gespeichert werden können. Außerdem soll eine Simulation gestartet werden können. Das Statistik Item erlaubt den Zugriff auf Statistiken, die für einen Durchlauf generiert wurden. Links unter der Menüleiste befindet sich die Auftragsliste, über die Aufträge generiert und angezeigt werden können. Darunter befindet sich der Bereich, der für die Modellierung eines Szenarios relevant ist. Es sollen Rampen, Fahrzeuge und Wände als Modellelemente auswählbar sein und in der Zeichenfläche platziert werden können. Die Zeichenfläche selber befindet sich rechts unter der Menüleiste. Dort werden die Aktionen der Akteure, wie z. B. Aufladen eines Pakets, visualisiert. Das unterste Element enthält eine Debug-Konsole, in der serverseitige Aktionen dargestellt werden können, die nicht in der Zeichenfläche dargestellt werden sollen.



Abbildung 16: Schematischer Aufbau der Benutzeroberfläche

### 5.3.3 Generierung von Aufträgen

Die Simulationssoftware soll ein Umschlagslager simulieren. Das bedeutet, dass Pakete in das Lager geliefert, zwischengelagert und an den Ausgangsrampen wieder abgeholt werden, wenn ein bestimmtes Paket angefragt wird (Vgl.1.3). Das bedeutet, dass eine Unterscheidung getroffen werden muss zwischen einem physischen Paket und einer Nachfrage

nach einem bestimmten Paket, die ein Ausgang stellt. Deshalb soll zwischen eingehenden und ausgehenden Aufträgen unterschieden werden. Es soll möglich sein, eine festgelegte Anzahl an Aufträgen zufällig über die Gui zu generieren. Es muss sichergestellt werden, dass die Menge an generierten eingehenden und ausgehenden Aufträgen in einem bestimmten Verhältnis zueinander stehen und ausgehende Aufträge für vorhandene Pakete generiert werden. So wird sichergestellt, dass Pakete nicht nur im Zwischenlager landen, sondern auch am Ausgang nachgefragt werden.

#### 5.3.4 Anforderungen an ein Multiagenten-Framework

Um ein Umschlagslager und die darin enthaltenden Akteure (Rampen und Fahrzeuge) zu simulieren, soll ein Multiagentensystem eingesetzt werden (Vgl.5.3.1). Zur Erstellung eines MAS soll ein Framework verwendet werden, dass die nachfolgenden Anforderungen erfüllt: Das Framework muss das Erstellen von verschiedenen Agententypen ermöglichen, um die Akteure mit ihren spezifischen Aufgabenstellungen simulieren. Agenten müssen in der Lage sein, untereinander Nachrichten auszutauschen und auf bestimmte Nachrichten oder Ereignisse mit definierten Verhaltensweisen zu reagieren. Weiterhin sollen die Aktionen der Agenten denen der realen Akteure hinsichtlich der Dauer ähneln. Außerdem muss das Framework Aktionen von verschiedenen Agenten parallel ausführen können, damit beispielsweise das gleichzeitige Fahren mehrerer Fahrzeuge möglich ist.

#### 5.3.5 Benötigte Agententypen

Sowohl Rampen als auch Fahrzeuge müssen verschiedene Aktionen durchführen. Dazu gehören u. a. das Befördern von Paketen, die Vergabe von Aufträge, Durchführung von Auktionen usw. Würde man alle Aufgaben, die ein Akteur durchführen muss, in einem Agenten bündeln, so wäre ein solcher Agent nur schwer wartbar und es könnten abhängig vom Agenten-Framework Probleme bei der Parallelisierung von Aktionen auftreten. Es bietet sich an, die erforderlichen Aufgaben eines Akteurs auf mehreren Agenten zu verteilen. Durch die Modularisierung kann die Entwicklung des Systems parallelisiert werden und Änderungen an einem Agenten haben geringere Auswirkungen auf das Gesamtsystem. Die Wartbarkeit des Systems erhöht sich. Für das zu entwickelnde System wurden die folgenden vier Agententypen konzipiert:

- Paketagent: Verwaltung der Paketdaten
- Orderagent: Ermittlung von Zielrampen und Zuweisung von Zielen (Wird nur bei Rampen benötigt)

- Routingagent: Durchführung von Auktionen und Berechnung von möglichen Pfaden
- Plattformagent: Durchführung physischer Aktionen (Fahren, Aufladen von Paketen etc.)

### 5.3.6 Kommunikation zwischen den Agenten

Die zu entwickelnde Software soll die in Abschnitt 4.1 beschriebenen Abläufe umsetzen. Die Aufgaben der verschiedenen Akteure müssen auf die Agenten verteilt werden. Die entworfenen Kommunikationsschritte der Agenten sollen für den Fall beschrieben werden, dass ein Eingang Ausgänge und Zwischenlager fragt, ob ein Paket entgegengenommen werden kann. Anhand des Beispiels soll die Verteilung der Aufgaben auf die unterschiedlichen Agenten skizziert werden.

1. Trifft ein Paket ein, verlangt der Paketagent vom Orderagenten eine Destination für das entsprechende Paket.
2. Der Orderagent einer Eingangsrampe fragt die Orderagenten der Ausgangs- und Zwischenrampen.
3. Die Orderagenten prüfen im Abgleich mit ihren Paketagenten, ob Platz frei ist (Zwischenrampe) oder die Paket-ID benötigt wird (Ausgang). Anschließend antworten sie dem Orderagenten am Eingang.
4. Wurde ein Ziel für das Paket gefunden, soll der Orderagent den Start einer Auktion initiieren, indem er den Routingagenten benachrichtigt.
5. Der Routingagent verlangt eine Aufwandsschätzung von allen Routingagenten der Fahrzeuge.
6. Der Routingagent eines Fahrzeugs berechnet eine Aufwandsschätzung anhand seiner Position und antwortet dem Routingagenten der Rampe. Fährt der Volksbot oder nimmt er an einer anderen Auktion teil, so wird -1 als Aufwandsschätzung zurückgeschickt.
7. Der Routingagent einer Rampe wählt, sofern vorhanden, den Bot aus, der den geringsten Aufwand benötigt, um ein Paket abzuholen und weist ihm den Auftrag zu.
8. Der Plattformagent fährt zu der jeweiligen Eingangsrampe und lädt das Paket auf. Dies geschieht durch einen Nachrichtenaustausch mit dem jeweiligen Plattformagenten der Rampe, der die Paketdaten übergibt. Das Fahren zur Zielrampe und das Abladen des Pakets erfolgt analog.

### 5.3.7 Konzeption des Pathfindings

Die Fahrzeuge eines Szenarios holen die Pakete von den Eingangsrampen ab und liefern sie zu den jeweiligen Zielrampen. Da der Anwender nicht selbst in den Simulationsablauf eingreifen möchte, um mögliche Pfade von Hand anzugeben (für den Transport der Pakete), wurde ein Pathfinding-Konzept entwickelt. Zunächst wird die Startposition in der Karte ermittelt. Anschließend wird im Uhrzeigersinn jede benachbarte Kachel ermittelt, bei der es sich um kein Hindernis handelt. Sollte eine gefundene Kachel noch nicht bearbeitet worden sein, wird ihr ein Wert zugewiesen, der Aufschluss darüber gibt, wie weit die anliegende Kachel von der aktuellen entfernt ist (notwendig u.a. für diagonales Fahren). Zudem wird der aktuelle Gridwert erhöht und der neuen Kachel zugewiesen. Sollte eine Kachel mehrmals bei der Zuweisung ermittelt werden, wird der kleinste gefundene Wert in der Kachel beibehalten, da ein kleinerer Wert einen kürzeren Pfad repräsentiert.

Nachdem alle Kacheln auf der Karte mit Werten versehen wurden, wird vom Endpunkt aus ein möglicher Weg zum Startpunkt zurück berechnet. Dabei wird der Wert in der Endkachel genommen und mit den benachbarten Kacheln verglichen. Sollte mindestens eine Kachel eine kleinere Wertigkeit aufweisen, wird somit erkannt, dass es sich um einen kürzeren Weg zur Startposition handelt. Sollten mehrere angrenzende Kacheln diese Eigenschaft aufweisen, wird zufällig eine von ihnen ausgewählt. Während dieser Rückwärtssuche (von der Endposition zur Startposition) wird bis die Startposition wieder gefunden wurde, jede Position der ausgewählten Kacheln gespeichert. Nachdem der Startpunkt gefunden wurde, müssen sämtliche gefundenen Positionen in der Liste getauscht werden (Punkte an Ende der Liste starten nun am Anfang der Liste und umgekehrt). Dies ist notwendig, da das Fahrzeug diese Positionen abfahren soll, jedoch liegen die Punkte in umgekehrter Reihenfolge vor, weil der Pfad von der Endposition zur Startposition ermittelt wird.

### 5.3.8 Konzeption der Statistiken

Um eine Vergleichbarkeit zwischen verschiedenen Ausführungen einer Simulation zu schaffen, müssen Statistiken berechnet werden, die einfache Kennzahlen berechnet, mithilfe dessen man die Simulationen bewerten und vergleichen kann. Als wichtigen Kennzahlen wurde hier die Auslastung der Volksbots und die Durchlaufzeit der Pakete in der Simulation gewählt.

- Auslastung der Volksbots:** Die Statistik erhält immer dann Daten, wenn ein Bot einen seinen Status wechselt („Arbeitend“-> „Wartend“). Daraufhin muss das Intervall berechnet werden, welches zwischen dem jetzigen und dem letzten Statuswechsel lag.

Dieses sagt dann aus, wie lange der Volksbot sich in dem jeweiligen Status befand. Diese Daten werden dann über alle Bots aggregiert und ins Verhältnis gesetzt und visuell aufbereitet abrufbar.

2. **Paketdurchlaufzeit:** Die Statistik der Paketdurchlaufzeit muss immer dann benachrichtet werden, sobald ein neuer Auftrag in die Simulation eintritt. Dieser Zeitpunkt wird im dann festgehalten und sobald dieses Paket dann die Simulation verlässt (Abtransport eines Pakets) wird das Intervall zwischen Eintritt und Austritt des Pakets berechnet. Der neue Wert wird dann in einem Liniendiagramm vermerkt. Das Liniendiagramm bietet die Möglichkeit, zu erkennen wie sich die Paketdurchlaufzeit über die Ablaufzeit der Simulation verändert.

Für die Kapselung der Aufgaben, die zur Datenhaltung und Berechnung der Statistiken nötig sind, wird eigens ein Statistik-Agent in der Simulation eingeführt. Dieser soll zu bestimmten Ereignissen informiert werden. Er sorgt dafür, die Daten zu aggregieren und in aufbereiteter Form der UI bereitzustellen. Des weiteren informiert er die UI, sobald neue Daten bereitstehen.

### 5.3.9 Interaktion der Komponenten

Durch das Zusammenspiel der Systemkomponenten soll eine Simulation durchgeführt werden. Die Aufträge müssen entsprechend ihrer Zeiten an den Server geschickt werden. Dies soll clientseitig durch einen Timer durchgeführt werden. Abbildung 17 zeigt den Ablauf und die Komponenten, die für das Starten einer Simulation erforderlich sind:

1. Ein potenzieller Nutzer startet eine Simulation über die Benutzeroberfläche.
2. Der Server wird durch die GUI informiert, dass die Simulation gestartet werden soll.
3. Der Server startet das Multiagentensystem.
4. Der Server meldet dem Client den Start des MAS.
5. Der Client weiß nun, dass die Agenten bereit sind, Aufträge entgegenzunehmen und startet den Timer für die Jobliste.
6. Die Aufträge werden gemäß ihrer Startzeit an den Server geschickt.
7. Der Server leitet die Aufträge an das MAS weiter
8. Die Daten über sichtbare Zustandsveränderungen werden an den Client geschickt und dort in der GUI visualisiert.

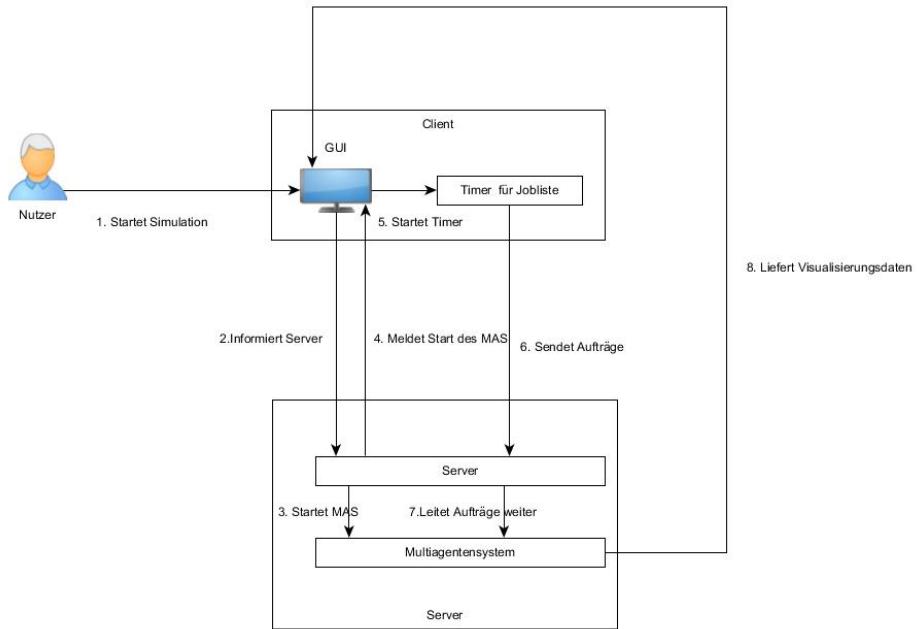


Abbildung 17: Interaktion der Systemkomponenten

## 5.4 Implementierung der Systemkomponenten

Inhalt diesen Abschnitts ist die Implementierung der zuvor konzipierten Systemkomponenten. Zum einen soll die Auswahl von Technologien und Frameworks zur Umsetzung beschrieben sowie die Funktionalität des Systems auf technischer Ebene dargestellt werden.

### 5.4.1 Implementierte Gesamtarchitektur

Abbildung 18 zeigt die konkrete Gesamtarchitektur des Systems, die durch die Auswahl von Umsetzungstechnologien entstanden ist. Die Webanwendung soll, wie bereits in Abschnitt 5.2.3 beschrieben, durch das GWT Framework implementiert werden. Der Code für Server und Client, der für Visualisierung, Client-Server Kommunikation u. ä., benötigt wird, wird durch GWT-Bibliotheken bereitgestellt. Eingebettet in den serverseitigen Code wird das Multiagentensystem, das mithilfe des Java Agent Development Framework (JADE)<sup>1</sup> implementiert wurde. Als Datenbankmanagementsystem wurde PostgreSQL ausgewählt. Die Applikation wird durch den internen Jetty Server von GWT gehostet, der die Java Laufzeitumgebung und das GWT Software Development Kit ebenfalls bereitstellt. Die Anwendung

<sup>1</sup>[jade]

kann im Produktivmodus prinzipiell auf jedem Java-fähigen Webserver aufgesetzt werden.



Abbildung 18: Implementierte Gesamtarchitektur

#### 5.4.2 Benutzeroberfläche

Die Implementierung der Benutzeroberfläche erfolgt durch die Klassen Mainframeview und Mainframepresenter. Die View beinhaltet die graphischen Elemente und der Presenter ist für die Programmlogik, wie das Hinzufügen von Listenern oder das Senden von Daten an den Server verantwortlich. Die Benutzeroberfläche ist in die fünf Bereiche Menüleiste, Auftragsliste, Modellelemente, Zeichenfläche und Debugkonsole unterteilt. In der Auftragsliste werden eingehende und ausgehende Aufträge mit ihrer Startzeit, ihrer Paket-ID und ihrem jeweiligen Ziel gelistet. Die Auflistung ist nicht chronologisch, sondern anhand der ID geordnet. Einzelne Aufträge können über einen Button in der vierten Spalte entfernt werden. Eine beliebige Anzahl an Aufträgen kann generiert werden. Der Bereich Modellelemente enthält Rampen, Fahrzeuge und Wände, die per Drag an Drop auf der Zeichenoberfläche platziert werden können. Im untersten Bereich befindet sich die Debugkonsole.

#### 5.4.3 Generierung von Aufträgen

Abbildung 20 zeigt die Methode addRandomJobs mit deren Hilfe zufällig eingehende und ausgehende Aufträge generiert werden. Für jeden zu generierenden Auftrag wird in der Variable destinationID zufällig festgelegt, ob es sich um einen eingehenden oder ausgehenden



Abbildung 19: Benutzeroberfläche

Auftrag handelt. Eingehende und Ausgehende Aufträge treten mit gleicher Wahrscheinlichkeit auf. Der Wert Null steht für einen eingehenden und minus Eins für einen ausgehenden Auftrag. In der Variable Timestamp wird die Zeit für den Auftrag festgelegt. Jeder Auftrag beginnt eine Sekunde später als der vorhergehende. Die Zeiten können prinzipiell beliebig angepasst werden. Die ID eines eingehenden Auftrags wird durch eine Variable der Klasse Joblist generiert, die um eins inkrementiert wird. Die IDs der ausgehenden Aufträge werden ebenfalls zufällig generiert, jedoch wird sichergestellt, dass ein ausgehender Auftrag für ein schon vorhandenes Paket generiert wird (Vgl. Abschnitt 5.3.3).

```
public void addRandomJobs(int numberOfJobs) {
    for (int i = 0; i < numberOfJobs; i++) {
        int destinationId = (Math.random() < 0.5) ? 0 : -1;

        int timestamp = (lastTimestamp) + 1;
        lastTimestamp = timestamp;

        addJob(new Job(timestamp, destinationId, this));
    }
}
```

Abbildung 20: Auftragsgenerierung

#### 5.4.4 Auswahl eines Multiagenten-Frameworks

Das Java Agent Development Framework wurde genutzt, um ein Netzwerk aus verschiedenen Agenten zu implementieren. Jade erlaubt das Erstellen von verschiedenen Agenten. Agenten werden als Java-Klasse implementiert, die von der vordefinierten Klasse Agent erbt. Die Aktionen, die ein Agent ausführen soll, werden durch Behaviours implementiert, die ebenfalls Klassen sind. Um die Anzahl an Klassen zu reduzieren, wurden die Behaviours als interne Klassen in die Agenten eingefügt. Jade ermöglicht die Kommunikation zwischen den Agenten durch ein vordefiniertes Nachrichtensystem. Nachrichten können sowohl gezielt an einen Agenten adressiert oder an alle Agenten verschickt werden. Die Agenten können in Echtzeit miteinander kommunizieren und mehrere Agenten können parallel Aktionen ausführen, da jeder Agent in einem eigenen Thread läuft (Vgl.[**jadetwo**]).

#### 5.4.5 Implementierte Agententypen

Implementiert wurden sieben verschiedene Agententypen. Jede Rampe besitzt jeweils einen Paket-, Order-, Routing- und Plattformagenten. Ein Fahrzeug besitzt einen Paket-, Routing- und Plattformagenten. Die Routing- und Plattformagenten wurden für Fahrzeuge und Rampen durch unterschiedliche Agenten implementiert, um die Anzahl an Behaviours pro Agent zu reduzieren und den Code übersichtlicher zu gestalten und unnötigen Speicherverbrauch zu vermeiden. Damit ein Agent weiß, welchem Szenario und welchem Akteur er zugeordnet ist, werden Szenario und Conveyor als Referenz übergeben <sup>2</sup>. Neben den genannten Agenten war es notwendig, einen Jobagent zu implementieren, der die Übergabe eines Pakets zu simulieren, die im physischen System durch einen Menschen geschieht. Außerdem werden ausgehende Aufträge durch den Jobagent an die Ausgangsrampen übermittelt. Es wurde anders als im physischen System darauf verzichtet für jedes Paket einen Paketagenten zu erzeugen, da die Synchronisation der Paketdaten ansonsten einen zu großen Synchronisationsaufwand erfordert hätte, was wiederum die Anzahl der Nachrichten zwischen den Agenten erhöht und fehleranfälliger ist.

#### 5.4.6 Kommunikation zwischen den Agenten

In den nachfolgenden Abschnitten soll die Kommunikation zwischen den Agenten, die implementiert wurde, anhand von Sequenzdiagrammen beschrieben werden. Dabei werden folgende Anwendungsszenarien durchlaufen:

- Jobzuweisung an Ein- und Ausgänge.

---

<sup>2</sup>Der genaue Ablauf der Initialisierung wird in Abschnitt 5.3.9 beschrieben

**5.4.6.1 Jobzuweisung an Ein- und Ausgänge** Abbildung 21 zeigt die Jobzuweisung an die Ein- und Ausgangsrampen durch den Jobagent. Bevor die Jobzuweisung beginnt, wird die Simulation durch den Client gestartet. Neben dem Jobagent, den Plattformagenten der Rampen und dem Paketagenten ist das Servlet AgentPlattformServiceImpl an der Kommunikation beteiligt. Auf die Aspekte der Client-Server Kommunikation wird in Abschnitt 5.3.9 genauer eingegangen. Das Diagramm zeigt folgenden Ablauf:

- Die Simulation wird aus dem Browser heraus gestartet und das Servlet wird benachrichtigt.
- Das Servlet startet die Agentenplattform und initialisiert die Agenten, einschließlich dem Jobagent.
- Der Jobagent startet einmalig eine Anfrage an die Plattformagenten der Rampen, um die IDs und die Anzahl der Ein- und Ausgänge zu erfragen, die für die Jobzuweisung nötig sind.
- Die Plattformagenten senden ihren Rampentyp an den Jobagent.
- Der Jobagent besitzt zwei Listen in denen er die IDs, der Ein- und Ausgänge speichert. Anhand des empfangenen Rampentyps, speichert er die ID des Senders in der entsprechenden Liste. Anschließend wird der Client benachrichtigt, dass die Simulation gestartet wurde.
- Der Client startet den Jobtimer und sendet die Aufträge entsprechend ihrer zeitlichen Reihenfolge an das Servlet.
- Das Servlet leitet die Aufträge an den Jobagent weiter.
- Der Jobagent empfängt den Auftrag und sendet eine Nachricht an sich selbst.
- Je nach Art des Auftrags werden entweder die Eingangs- oder Ausgangsrampen gefragt, ob der Auftrag entgegengenommen werden kann.
- Die Plattformagenten senden eine Nachricht an ihre Paketagenten, um zu prüfen, ob der Auftrag aufgenommen werden kann. Der Plattformagent wartet auf die Antwort des Paketagenten.
- Der Paketagent prüft, ob der Auftrag entgegengenommen werden kann und antwortet dem Plattformagent.
- Der Plattformagent verarbeitet die Anfrage und antwortet dem Jobagent. Läuft der Plattformagent auf einer Ausgangsrampen, so wird automatisch geantwortet, dass Platz verfügbar ist, da ausgehende Aufträge nur IDs repräsentieren, die der Ausgang anfragt. Ein Ausgang kann unbegrenzt IDs anfragen.

- Der Jobagent wählt unter den Rampen, die einen Auftrag entgegennehmen können, zufällig eine aus und sendet die Auftragsdaten an den Plattformagenten.
- Der Plattformagent initialisiert mit den Auftragsdaten die Paketdaten und schickt diese an den Paketagenten. Zu beachten ist, dass die Klasse PackageDate sowohl eingehende als auch ausgehende Aufträge repräsentiert.
- Der Paketagent fügt das Paket einer Liste hinzu.

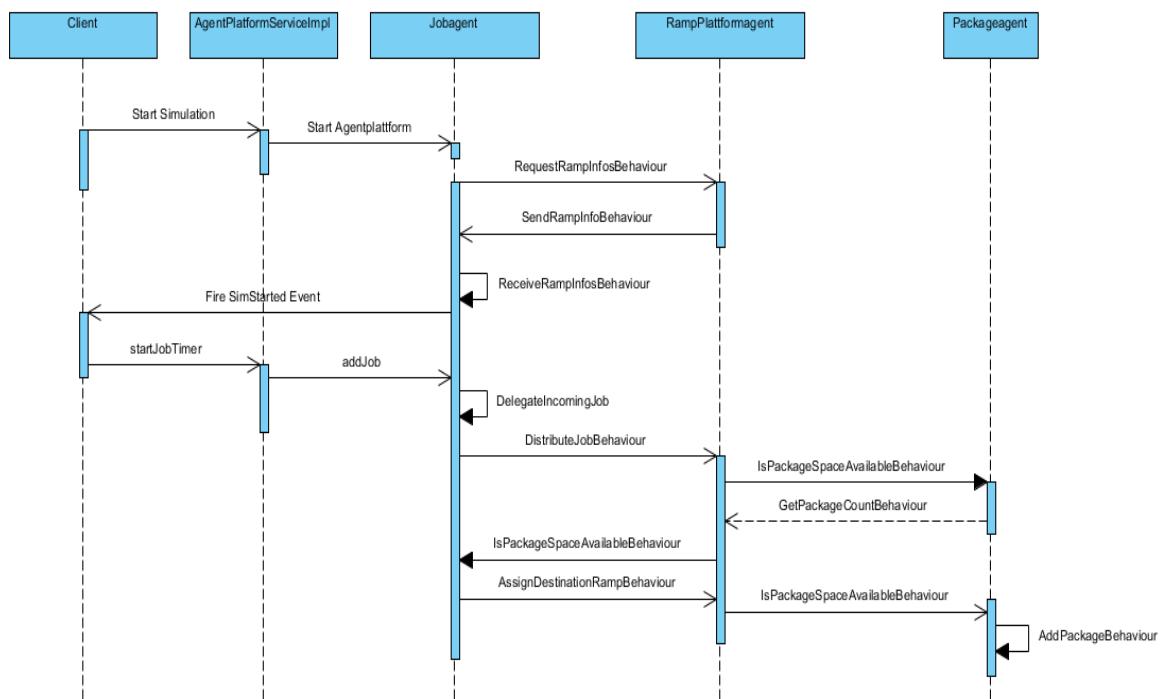


Abbildung 21: Jobzuweisung an Ein- und Ausgänge

**5.4.6.2 Eingang fragt Ausgang und Zwischenlager** Pakete, die am Eingang eintreffen, werden weitergeleitet. Besteht ein Bedarf am Ausgang, so soll das Paket zum Ausgang befördert werden. Ansonsten wird es ins Zwischenlager gebracht. Abbildung 22 zeigt den Nachrichtenaustausch der Agenten, der zur Zielfindung nötig ist:

- Der Paketagent stellt mithilfe einer Tickerbehaviour zyklisch Anfragen an seinen Orderagenten, um ein Ziel für das vorderste Paket zu bekommen. Bevor der Orderagent benachrichtigt wird, prüft der Paketagent mithilfe der Variable OutgoingJobFlag, ob für das Paket bereits ein Ziel gefunden wurde. Falls nicht, sendet er die Anfrage an den Orderagenten.
- Der Orderagent sendet eine Nachricht an alle Orderagenten der Ausgangs- und Zwischenrampen.
- Der jeweilige Orderagent fragt seinen Paketagenten, ob bereits ein Paket erwartet wird. Dies ist nötig, damit sich nicht mehrere Bots vor einer Rampe blockieren.
- Der Paketagent prüft anhand der Variable IncomingJobFlag, ob ein Paket erwartet wird und antwortet seinem Orderagenten.
- Sofern kein Paket erwartet wird, fährt der Orderagent fort und stellt eine erneute Anfrage an seinen Paketagenten, um zu fragen, ob das entsprechende Paket benötigt wird (Ausgang) oder ob Platz frei ist (Zwischenlager).
- Der jeweilige Orderagent antwortet dem Eingang, ob das Paket entgegengenommen werden kann oder nicht.
- Der Orderagent des Eingangs speichert die IDs der Rampen, die ein Paket aufnehmen können, wählt unter diesen zufällig eine aus und übermittelt die Daten für Start- und Zielrampe an den Routingagent, damit dieser die Auktion starten kann. Der Orderagent wartet bis die Auktion beendet ist.
- Wurde die Auktion erfolgreich durchgeführt, teilt der Orderagent des Eingangs dem Paketagenten der Zielrampe mit, dass sich ein Paket im Anmarsch befindet, damit dieser das IncomingJobFlag setzt.
- Paketagent antwortet dem Orderagenten, dass das Flag gesetzt wurde.
- Der Orderagent informiert seinen Paketagenten, damit dieser das OutgoingJobFlag setzt.

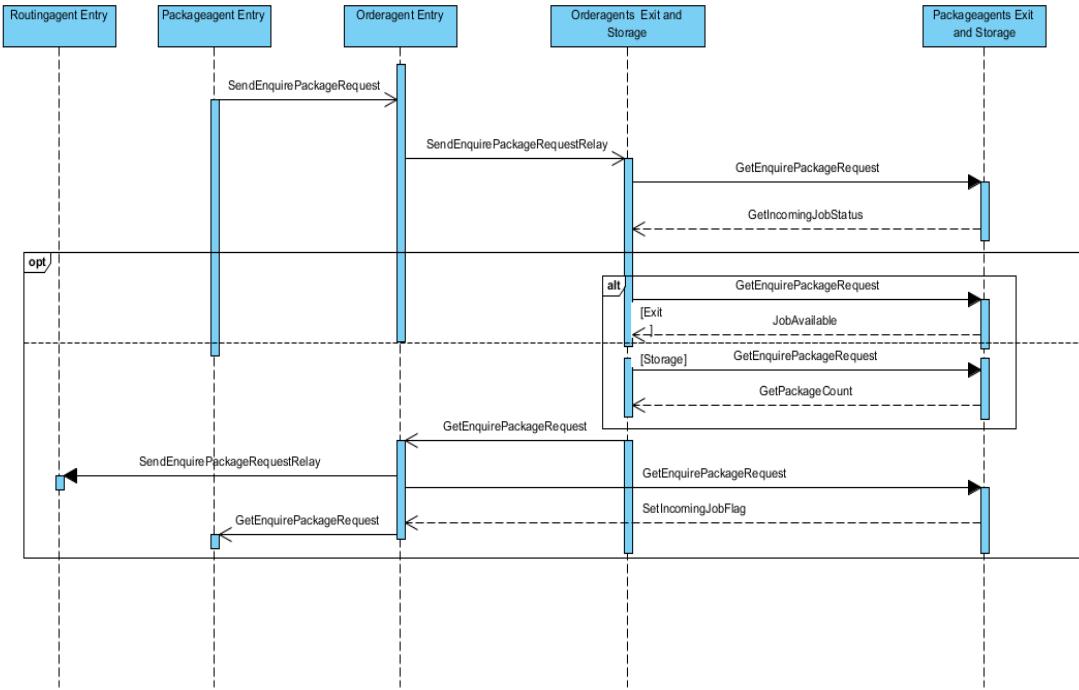


Abbildung 22: Eingang fragt Ausgang und Zwischenlager

**5.4.6.3 Ausgang fragt Zwischenlager** Pakete, die am Eingang eintreffen und für die noch keine Nachfrage an den Ausgangsrampen besteht, werden ins Zwischenlager gebracht. Damit auch diese Pakete zum Ausgang gelangen, fragt der Ausgang zyklisch im Zwischenlager für die entsprechenden Pakete nach. Abbildung 23 zeigt den Ablauf im Detail:

- Der Paketagent des Ausgangs stellt mithilfe einer Tickerbehaviour zyklisch Anfragen an seinen Orderagenten, damit im Zwischenlager geprüft wird, ob ein Paket vorhanden ist, für das eine Nachfrage besteht. Bevor der Orderagent benachrichtigt wird, prüft der Paketagent mithilfe der Variable IncomingJobFlag, ob bereits ein Paket erwartet wird. Falls nicht, sendet er die Anfrage an den Orderagenten.
- Der Orderagent sendet eine Nachricht an alle Orderagenten der Zwischenrampen.
- Der jeweilige Orderagent fragt seinen Paketagenten, ob bereits ein Paket abgeholt wird. Dies ist nötig, damit sich nicht mehrere Bots vor einer Rampe blockieren und damit ein Bot nicht das falsche Paket abholt.
- Der Paketagent prüft anhand der Variable OutgoingJobFlag, ob ein Paket erwartet wird und antwortet seinem Orderagenten.

- Sofern kein Paket erwartet wird, fährt der Orderagent fort und stellt eine erneute Anfrage an seinen Paketagenten, um zu fragen, ob das Paket, das der Ausgang verlangt, an vorderster Stelle der Rampe ist.
- Der jeweilige Orderagent antwortet dem Ausgang, ob das Paket vorhanden ist oder nicht.
- Sofern das Paket in einem der Zwischenlager ist, benachrichtigt der Orderagent des Ausgangs den Routingagenten des Zwischenlagers, damit das Paket von einem Bot abgeholt wird.

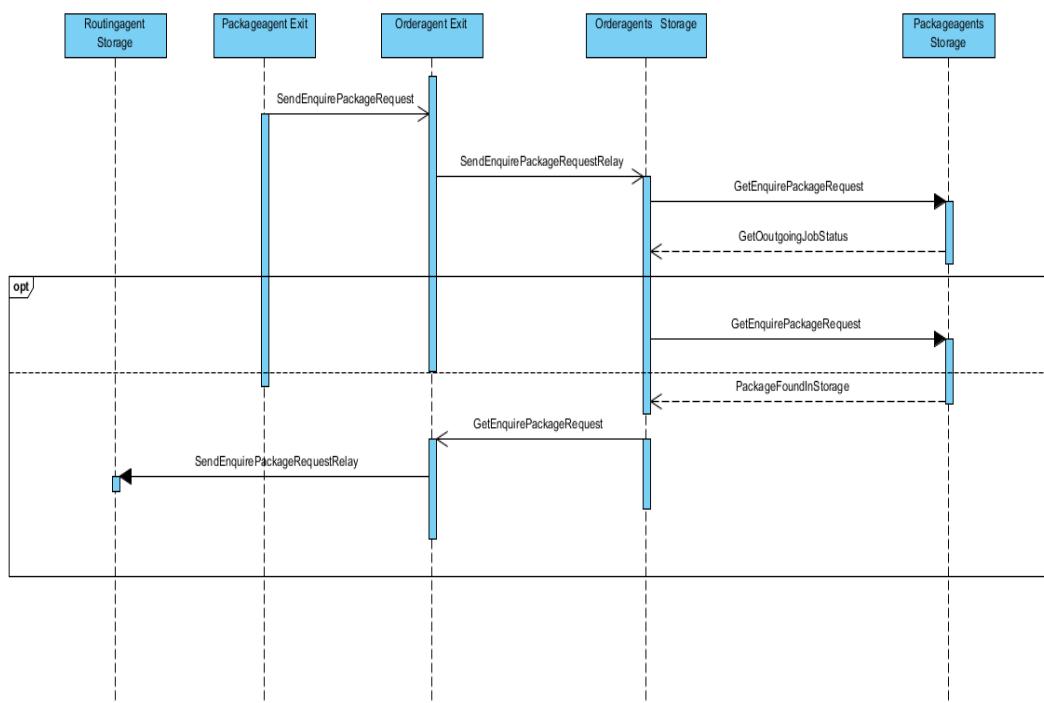


Abbildung 23: Ausgang fragt Zwischenlager

**5.4.6.4 Auktion** Nachdem ein Paket ein Ziel zugewiesen bekommen hat, startet der Routingagent der jeweiligen Eingangs- oder Zwischenrampe eine Auktion, um ein Fahrzeug zu finden, das das Paket befördert. Abbildung 24 zeigt den Ablauf der Auktion:

- Der Routingagent schickt eine Nachricht an alle Routingagenten der Fahrzeuge.
- Die Routingagenten der Fahrzeuge fragen ihre Paketagenten, ob bereits ein Auftrag durchgeführt wird. Der Routingagent wartet bis der Paketagent geantwortet hat.
- Der Paketagent antwortet dem Routingagenten.
- Falls das Fahrzeug nicht belegt ist und der Bot auch an keiner anderen Auktion teilnimmt, erfragt der Routingagent seine aktuelle Position von seinem Plattformagenten.
- Der Plattformagent schickt die aktuelle Position an den Routingagenten.
- Der Routingagent berechnet mithilfe des Pathfindings eine Aufwandsabschätzung und schickt Sie dem Routingagenten der Rampe. Falls er nicht in der Lage ist, ein Paket zu befördern, teilt er dies der Rampe über die Nachricht mit.
- Der Routingagent der Rampe speichert alle Estimations nacheinander ab. Haben alle Fahrzeuge geantwortet oder ist der Timeout für die Auktion abgelaufen, wird, sofern vorhanden, der Bot mit der besten Estimation ausgewählt. Falls es einen Bot gibt, der das Paket abholen kann, wird der Paketagent des Zwischenlagers oder Ausgangs benachrichtigt, dass ein Paket in Kürze geliefert wird, damit das IncomingJobFlag gesetzt wird. Dadurch wird verhindert, dass die Bots sich gegenseitig vor einer Rampe blockieren.
- Der Paketagent antwortet dem Routingagenten nach dem Setzen des Flags.
- Der Routingagent teilt dem Routingagenten des ausgewählten Fahrzeugs mit, dass es ausgewählt wurde.
- Der Routinagent des Fahrzeugs sendet eine Nachricht an seinen Paketagenten, um Platz für das Paket zu reservieren.
- Der Fahrzeug Routingagent leitet die Information an seinen Plattformagenten weiter, damit die Fahrt beginnen kann.

**5.4.6.5 Paket von Startrampe abholen und zur Zielrampe bringen** Nachdem das Fahren initiiert wurde, führt der Plattformagent des entsprechenden Fahrzeugs den Transport durch. Abbildung 25 zeigt den Ablauf auf Agentenebene:

- Das Fahrzeug fährt zur Startrampe und benachrichtigt den Plattformagenten der jeweiligen Rampe.

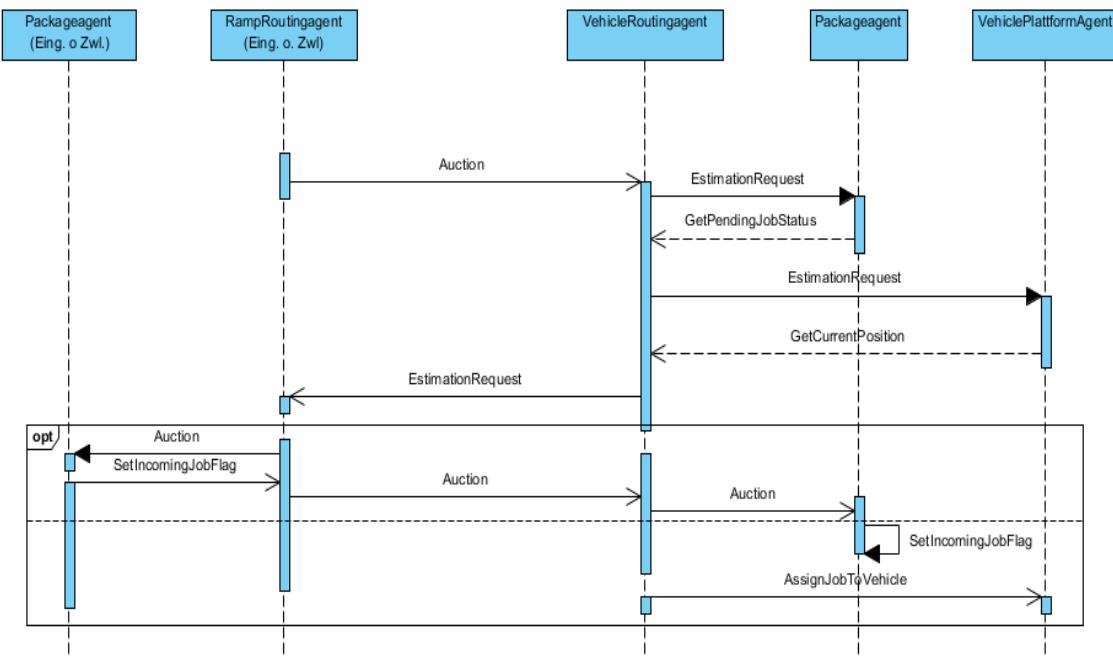


Abbildung 24: Auktion

- Dieser benachrichtigt seinen Paketagenten, dass das Paket transferiert werden kann.
- Der Paketagent sendet eine Nachricht an den Paketagenten des Fahrzeugs, um das Paket zu übergeben und entfernt es von der Rampe.
- Der Paketagent des Fahrzeugs fügt das Paket hinzu und antwortet dem anderen Paketagenten.
- Der Paketagent der Startrampe benachrichtigt den Plattformagenten, dass das Paket übergeben wurde.
- Dieser benachrichtigt wiederum den Plattformagenten des Fahrzeugs, damit dieser weiß, dass er weiterfahren kann.
- Der Plattformagent des Fahrzeugs fährt zur Zielrampe und benachrichtigt seinen Paketagenten, dass das Paket übergeben werden kann.
- Der Paketagent entfernt das Paket und sendet eine Nachricht an den Paketagenten der Rampe, dass das Paket aufgenommen werden soll.
- Dieser fügt das Paket hinzu und gibt dem Paketagenten des Fahrzeugs Bescheid, dass das Aufladen beendet ist.

- Der Fahrzeug Paketagent informiert seinen Plattformagent, dass der Auftrag erledigt ist.



Abbildung 25: Paket abholen und zur Zielrampe bringen

#### 5.4.7 Implementierung des Pathfindings

Der entwickelte Pathfindingalgorithmus verwendet eine Karte, die in Kacheln unterteilt ist. Eine Kachel kann dabei entweder als einfache, befahrbare Fläche, als Startposition, als Endposition oder als Hindernis definiert werden. Damit die Simulation um alternative Pathfindingalgorithmen erweitert werden kann, wurde eine einheitliche Schnittstelle definiert, um schnell und einfach (ohne Umbauten von Programmelementen in denen das Pathfinding benutzt wird) mögliche zukünftige Algorithmen einbinden zu können. Diese Schnittstelle ist in Form eines Interfaces definiert. Im Interface sind Funktionen zu finden, die von jeder alternativen Pathfindingroutine implementiert werden müssen. Sollten dabei einige keine Verwendung finden, sollte schlichtweg ein Standardwert zurückgeliefert werden, der sich nicht verändert lässt. Folgende Funktionen sind im Interface vorgesehen:

- Statusmeldung, ob ein Pfad zur Zeit berechnet wird
- Anzahl der Zeilen und Spalten in der Liste
- Position des Start-/Stoppunktes

- Angabe, ob man diagonales Fahren (de)aktivieren möchte
- Vermeidung von (möglicherweise unnötigen) Rotationen
- Initialisierung des Pathfindingalgorithmus
- Aktuelle Status-(bzw. Fehler-)Meldung
- Funktion zum Suchen eines Pfades unter andere einer Start- und Stopposition

Jede Kachel wird im System als GridItem definiert. Dabei wird in der Kachel die Größe hinterlegt (ist bei allen Kacheln in einer Karte die selbe Größe). Jedes GridItem besitzt einen Typ. Darüber hinaus gibt es noch GridValue und StepValue. GridValue ist dabei der Wert einer Kachel, die bei der Vorwärtssuche ermittelt wurde, wo eine aktuelle Kachel ihre benachbarten freien Kachel einen Wert für die Pfadfindung zugewiesen hat. StepValue repräsentiert hingegen die Anzahl an Schritten im System, um diese Kachel zu erreichen.

Damit bei der Rückwärtsberechnung die Punkte eines Pfades gespeichert werden können, wurde die Klasse PathPoint implementiert. Diese kann die Position (x/y-Koordinate) speichern. Sie speichert unter anderem aber auch den Erwartungswert für die Kachel an dieses Position (EstimationValue), welche Aufschluss darüber gibt wie hoch die Kosten (Aufwand) sind um diese Position zu erreichen. Sollte (durch einen zusätzliche Eigenschaft) sich die Richtung der aktuellen Kachel von der vorherigen nicht unterscheiden, hat diese einen kleineren Erwartungswert.

Der implementierte Pathfindingalgorithmus wurde hingehen in zwei Klassen implementiert. Bei der einen Klasse handelt sich um eine abstrakte Klasse, welches alle Funktionen, Eigenschaften und Statusmeldungen vom eigentlichen Pathfinding beherbergt, bis auf die Implementation der eigentlichen Pfadsuche. Die Funktion für die Pfadsuche wurde in einer separaten Klasse implementiert, welche von der abstrakten Klasse erbt (Polymorphie), so dass diese sämliche zu Grunde legende Funktionen verwenden kann. Diese Struktur wurde dahingehend entwickelt, weil der verwendete Suchalgorithmus lediglich einen Pfad ermittelt und zurück liefert. Jedoch wäre es auch möglich den Suchalgorithmus zu modifizieren, um so mehrere alternative Routen auf einer Karte zu ermitteln, die mit bestimmten Kriterien versehen sind. Da diese Suche auch die grundlegenden Funktionen verwenden würde, wurde nur die eigentliche Funktion für die Pfadsuche gekapselt (die Rückwärtssuche).

#### 5.4.8 Implementierung der Statistiken

Nachdem innerhalb des Kapitels 5.3.8 das Konzept der Statistiken erarbeitet wurde, geht es in diesem Kapitel darum, die Statistiken umzusetzen. Um Statistiken anbieten zu kön-

nen, musste zunächst ein geeignetes Framework gefunden werden, mit dessen Hilfe man leich Statistiken erzeugen kann. Hierzu wurde das Framework **Sencha GXT** gewählt (<http://www.sencha.com/products/gxt/>). Diese bietet eine Vielzahl von Möglichkeiten, um in einfacher Art und Weise Diagramme zu generieren.

Die Einbindung der Diagramme erfolgt über Popups. So hat der Benutzer die Möglichkeit, nach belieben die für ihn interessante Statistik ein- bzw. auszublenden.

Die Berechnung der Statistiken erfolgt mithilfe des Statistik-Agenten, der eigens dafür in die Simulation eingebracht wird. Dieser wird an definierten Stellen von den unterschiedlichen Agenten aufgerufen, wenn ein bestimmtes Ereignis eingetreten ist (z.B. Ein Auftrag verlässt die Simulation, ein Bot wechselt seinen Status von Arbeitend auf Wartend, ...). Diese Daten werden dann vom Statistik-Agenten aufgefangen und zu seiner Datenhaltung hinzugefügt. Sobald ein Ereignis eine Neuberechnung einer Statistik erfordert, informiert der Statistik-Agent die UI und fordert eine Neuzeichnung dieser. So erhält der Benutzer sofort Feedback über das soeben eingetretene Ereignis. Die Kommunikation zwischen Server und Client erfolgt hier mithilfe von AJAX. Die Ergebnisse werden dem Benutzer also nahezu Live präsentiert.

Die Statistiken sind in der Simulation im Reiter SStatistiken zu finden.

#### 5.4.9 Interaktion der Komponenten

In Abschnitt 5.3.9 wurden die Interaktionen der Systemkomponenten auf logischer Ebene beschrieben, die für das Starten einer Simulation bis hin zur Visualisierung notwendig sind. Im Folgenden Abschnitt soll beschrieben werden, wie und mit welchen Mitteln die einzelnen Schritte implementiert wurden.

**5.4.9.1 Client-Server Kommunikation** Das Starten einer Simulation wird über das Menüitem „Starten/Anhalten“ ausgelöst (s. Abbildung26). Wird das Menüitem gedrückt, dann wird die Methode execute ausgeführt. In der Methode wird geprüft, ob die Simulation bereits läuft, um festzustellen, ob die Simulation gestartet oder angehalten werden soll. Wenn die Simulation noch nicht gestartet wurde, wird zunächst geprüft, ob das erstellte Szenario konsistent ist (Mindestens eine Eingangs- und Ausgangsrampe). Wenn dies der Fall ist, wird die Simulation gestartet. Dies erfordert eine Client-Server Kommunikation. Die Implementierung erfolgt durch den von GWT bereitgestellten Remote Procedure Call (RPC) Mechanismus (Vgl. [gwtrpc]).

Die Variable agentPlatformService referenziert das Interface AgentPlatformServiceAsync.

Serverseitig gibt es ein Servlet „AgentPlatformService“, das verschiedene Methoden bereitstellt, die über das Interface AgentPlatformServiceAsync aufgerufen werden können. Die Variable ist als AgentPlatformServiceAsync Interface deklariert, wird aber mit einer automatisch generierten Proxy-Klasse instantiiert (s. Quellcode/Konstruktor der Klasse MainframePresenter). In der Methode execute wird über den Aufruf der Methode startSimulation das Szenario übergeben und zum Server geschickt. Für den Aufruf ist ein Objekt notwendig, das das Interface AsyncCallback implementiert, welches wiederum der Methode als interne Klasse übergeben wird. Der Typparameter, in diesem Fall ein Integer, definiert, welcher Rückgabewert vom Server erwartet wird. Außerdem wird in den Methoden onFailure und onSuccess definiert, was passieren soll, wenn der RPC fehlschlägt bzw. erfolgreich war. Wenn die Simulation serverseitig erfolgreich gestartet wurde, dann soll die auf dem Server generierte ID dem aktuellen Szenario zugewiesen werden und der Status der Simulation auf gestartet gesetzt werden.

```

/*
 * start / stop simulation
 *
 * @author Matthias, Nagi
 */
menuName = "Starten/Anhalten";
mapSimMenuItem.put(menuName, this.display.getSimulationMenuBar().addItem(menuName, new Command() {
    public void execute() {
        // doesn't run yet?
        if (!isSimulationRunning()) {
            //Only start simulation if szenario is consisten
            if(checkIfSzenarioIsConsistent()){
                // start simulation
                agentPlatformService.startSimulation(MainFramePresenter.this.currentSzenario, new AsyncCallback<Integer>() {
                    public void onFailure(Throwable caught) {
                        Window.alert("Simulation error: An error occurred during start of simulation!");
                    }

                    public void onSuccess(Integer id) {
                        MainFramePresenter.this.currentSzenario.setID(id);
                        MainFramePresenter.this.setSimulationState(true);
                    }
                });
            }else {
                Window.alert("Szenario cannot be started, because it is inconsistent. You need at least from each Ramp type one exemplar");
            }
        }else {
            // stop simulation
            agentPlatformService.stopSimulation(new Empty AsyncCallback());
            MainFramePresenter.this.setSimulationState(false);
        }
    }
});
```

Abbildung 26: Starten einer Simulation

**5.4.9.2 Start des Multiagenten-Systems** Nachdem die Daten zum Server geschickt wurden, wird die Methode startSimulation des Servlets AgentPlatformServiceImpl aufgerufen. Für das Szenario wird eine ID generiert und über das Object Array wird das Szenario allen Agenten als Parameter zur Verfügung gestellt. Danach wird eine Agentenplattform für das jeweilige Szenario gestartet. Die Liste der Conveyor (Fahrzeuge und Rampen) aus dem Sze-

nario wird durchlaufen und für jeden Conveyor werden die benötigten Agenten erzeugt. Die Methode addAgentToSimulation erzeugt anhand der Conveyor- und Szenario-ID einen eindeutigen Namen für jeden Agenten, übergibt die Parameter und fügt ihn der Simulation hinzu. Anschließend wird der Jobagent initialisiert und die Agenten werden gestartet. Die letzte Anweisung schickt die Szenario ID zum Client zurück.

```

public int startSimulation(Szenario szenario) {
    // async function call and static variable, so remember,
    // in case someone else starts before function is finished
    szenario.setID(++szenarioID);

    Object[] argsJobAgent = new Object[1];
    argsJobAgent[0] = szenario;

    List<Conveyor> lstConveyor = szenario.getConveyorList();

    if (lstConveyor.size() < 1)
        return -1;

    startAgentPlatform(szenario);

    for (Conveyor myConveyor : lstConveyor) {
        Object[] argsAgent = new Object[2];
        argsAgent[0] = szenario;
        argsAgent[1] = myConveyor;

        int id = myConveyor.getID();

        if (myConveyor instanceof ConveyorRamp) {
            addAgentToSimulation(id, szenario.getId(), argsAgent, PackageAgent.NAME , new PackageAgent());
            addAgentToSimulation(id, szenario.getId(), argsAgent, RampOrderAgent.NAME , new RampOrderAgent());
            addAgentToSimulation(id, szenario.getId(), argsAgent, RampPlattformAgent.NAME , new RampPlattformAgent());
            addAgentToSimulation(id, szenario.getId(), argsAgent, RampRoutingAgent.NAME , new RampRoutingAgent());
        } else if (myConveyor instanceof ConveyorVehicle) {
            addAgentToSimulation(id, szenario.getId(), argsAgent, PackageAgent.NAME , new PackageAgent());
            addAgentToSimulation(id, szenario.getId(), argsAgent, VehiclePlattformAgent.NAME , new VehiclePlattformAgent());
            addAgentToSimulation(id, szenario.getId(), argsAgent, VehicleRoutingAgent.NAME , new VehicleRoutingAgent());
        }
    }

    addAgentToSimulation(0, szenario.getId(), argsJobAgent, JobAgent.NAME, new JobAgent());

    startAgents();

    return szenario.getId();
}

```

Abbildung 27: Starten des MAS

**5.4.9.3 Start des Jobtimers** Nachdem der Jobagent alle Informationen erhalten hat, die er für die Zuweisung von Aufträgen benötigt, schickt er eine Nachricht an den Client, dass die Aufträge zum Server geschickt werden können (Vgl.Abschnitt 5.4.6.1). Der Client startet den Jobtimer mit der Methode startJobTimer des MainframePresenters (s. Abbildung 28). Die Integer Variable elapsedSec repräsentiert die Zeit. Der Timer wird gestartet und die run-Methode wird kontinuierlich ausgeführt, bis der Jobtimer gestoppt wird. Die Zeitvariable wird bei jedem Durchlauf um eins hochgezählt, was bedeutet, dass eine Sekunde vergangen ist. Die einzelnen Jobs aus der Liste werden in der for-Schleife abgefragt und ihr Zeitstempel wird mit der elapsedSec Variable abgeglichen. Ist die vergangene Zeit

größer oder gleich der Zeit zu der ein Job ausgeführt werden soll, dann wird der Job mit der Methode addJob zum Server geschickt. Mit der vorletzten Anweisung wird festgelegt, in welchen Zeitabständen der Timer ausgeführt werden soll. Die letzte Anweisung dient dazu, den Timer initial zu starten.

```

public void startJobTimer() {
    if (tmrJobStarter != null)
        return;

    elapsedTimeSec = 0;

    tmrJobStarter = new Timer() {
        public void run() {
            elapsedTimeSec += 1;

            if (lstJobs.size() < 1)
                return;

            //Job pendingJob = lstJobs.getJob(0);

            for (Job pendingJob : lstJobs.getJoblist()) {
                if (elapsedTimeSec >= pendingJob.getTimestamp()) {
                    agentPlatformService.addJob(currentSzenario.getId(), pendingJob, new Empty AsyncCallback());
                    break;
                }
            }
        }
    };
    tmrJobStarter.scheduleRepeating(1000);
    tmrJobStarter.run();
}

```

Abbildung 28: Start des Jobtimers

Abbildung 29 zeigt die serverseitige Weiterleitung des empfangenen Auftrags. In der Methode addJob des AgentPlatformServiceImpl Servlets wird der jeweilige Jobagent anhand der Szenario ID aus einer Hashmap geholt. Es wird ein ACLMessageObjekt erzeugt und der Nachrichtentyp wird im Konstruktor festgelegt, um gezielt die notwendige Behaviour zu adressieren. Als Empfänger wird der Jobagent selber festgelegt. Anschließend wird der Job der Nachricht als ContentObject hinzugefügt und versendet.

```

public void addJob(int szenarioID, Job myJob) {
    Agent myAgent = mapJobAgent.get(szenarioID);

    ACLMessage msg = new ACLMessage(MessageType.ASSIGN_JOB);
    msg.addReceiver(myAgent.getAID());

    try {
        msg.setContentObject(myJob);
        myAgent.send(msg);
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}

```

Abbildung 29: Weiterleitung des Auftrags

**5.4.9.4 Visualisierung von Zustandsveränderungen** Die einzelnen Agenten führen Aktionen durch. Werden dadurch Zustände verändert (z. B. Hinzufügen eines Pakets etc.), müssen diese visualisiert werden. Das MAS läuft auf dem Server. Damit ein Client die Zustandsveränderungen visualisieren kann, muss er vom Server über die Art der Zustandsveränderung informiert werden. Das Schicken von Nachrichten vom Server zum Client wurde mithilfe des GWT Eventservices implementiert. Der Eventservice ist ein event-basiertes Kommunikationsframework, dass auf dem GWT-RPC Mechanismus und der Comet Server-Push Technologie basiert (Vgl. [gwteventservice]). Abbildung 30 zeigt die Methode Add-Package des Packageagents, die für das Hinzufügen eines Pakets benötigt wird. Nachdem das Paket in die Liste eingefügt wurde, wird mithilfe der Klasse EventHelper ein Event zum Client gefeuert. Der EventHelper erbt von dem Servlet RemoteEventServiceServlet, das vom EventService Framework bereitgestellt wird und das GWT-Servlet, um die Funktionalität zum Versenden von Events erweitert (Vgl.[gwteventservice]).

Die Klasse PackageAddedEvent ist eine selbst definierte Event-Klasse, die das Interface Event des Eventservices implementiert (Vgl.[gwteventservice]). Mithilfe dieser Klasse können Informationen für den Client transportiert werden. Die Informationen werden im Konstruktor der Klasse PackageAddedEvent übergeben. In diesem Fall werden die Conveyor- und Package-ID übergeben, damit der Client weiß, welchen Conveyor er neu zeichnen soll und welches Paket mit welcher ID hinzugefügt wurde.

Abbildung 31 zeigt die clientseitige Verarbeitung der Events, die vom Server empfangen werden. In der Methode HandleEvents des Mainframepresenters , wird zunächst

```
private class AddPackage extends CyclicReceiverBehaviour {
    protected AddPackage(int msgType) {
        super(MessageTemplate.MatchPerformative(msgType));
    }

    public void onMessage(ACLMessage msg) throws UnreadableException {
        PackageAgent currentAgent = (PackageAgent) myAgent;
        PackageData myPackage = (PackageData) msg.getContentObject();

        currentAgent.lstPackage.add(myPackage);

        EventHelper.addEvent(new PackageAddedEvent(myConveyor.getID(), myPackage.getPackageID()));
    }
}
```

Abbildung 30: Feuern eines Events

eine Instanz der Klasse RemoteEventService über die RemoteEventServiceFactory geliefert. Alle vorhandenen Listener werden entfernt und es wird ein neuer Listener hinzugefügt. Beim Hinzufügen des Listeners wird eine Domain mit einer Zeichenkette übergeben. Durch die Domain kann gezielt festgelegt werden, für wen eine Nachricht bestimmt ist (Vgl.[**gwteventservice**]). Die Zuordnung einer Nachricht zu einer Domain, wird beim Versenden der Nachricht festgelegt (S. Quellcode Klasse EventHelper). In der Methode apply des Listeners wird für alle definierten Events festgelegt, welche Aktion beim Empfang auf dem Client ausgeführt werden soll. Beispielsweise wird durch das SimStartedEvent das Starten des Jobtimers durch die Methode startJobTimer initialisiert.

```

private void handleEvents() {
    myRES = RemoteEventServiceFactory.getInstance().getRemoteEventService();
    myRES.removeListeners();

    myRES.addListener(DomainFactory.getDomain(DOMAIN_NAME), new RemoteEventListener() {
        public void apply(Event anEvent) {
            if (anEvent instanceof SimStartedEvent) {
                bSimulationRunning = true;
                display.log("Simulation started!");

                startJobTimer();

                return;
            }

            if (anEvent instanceof SimStoppedEvent) {
                bSimulationRunning = false;
                display.log("Simulation stopped!");

                stopJobTimer();

                return;
            }

            if (anEvent instanceof JobAssignedEvent) {
                JobAssignedEvent myEvent = (JobAssignedEvent)anEvent;

                lstJobs.removeJob(myEvent.getJob().getPackageId(), myEvent.getJob().getType());

                setupJobTable();

                return;
            }

            if (anEvent instanceof JobUnassignableEvent) {
                JobUnassignableEvent myEvent = (JobUnassignableEvent)anEvent;
            }
        }
    });
}

```

Abbildung 31: Clientseitige Verarbeitung des Events

## 5.5 Testen und Validieren

Das Artefakt, welches zuletzt ausgebracht werden soll, wird mithilfe verschiedener Werkzeuge validiert. Dazu gehören mitunter Unit-Tests und manuelle UI-Tests. Mithilfe der Unit-Tests werden verschiedene Komponenten innerhalb der Software getestet und dessen Funktionsweise mithilfe dessen garantiert. Unit-Tests sind aber nur dann ein probates Mittel, wenn man abgeschlossene Funktionen innerhalb der Software testen möchte. Sobald es darum geht, Anordnung und Funktionsweise von UI-Komponenten zu testen (wie z.B. die Bewegung der Bots), reichen die Möglichkeiten von Unit-Tests leider nicht mehr aus. Zu diesem Zwecke könnte man automatisierte UI-Tests implementieren. Diese sind aber nur schwer zu realisieren und bedürfen einem hohen Arbeitsaufwand bei der initialen Realisierung. Zunächst müsste die Infrastruktur für solche Tests bereitgestellt werden und in der Praxis hat sich gezeigt, dass diese einen sehr hohen Wartungsaufwand benötigen. Aus diesem Grund wurde hier auf automatisierte UI-Tests verzichtet und wurden durch manuelle Tests ersetzt. Nachdem die Simulationsmitglieder Features ausgebracht haben, wurden jeweils eine handvoll UI-Tests durchgeführt, die die Korrektheit UI versichert haben. Zu diesem Testset gehören folgende Tests mit den jeweiligen Akzeptanzkriterien:

- Laden eines Szenarios.
  - Sind die Objekte korrekt auf der UI angeordnet und charakterisiert (wird der Bot

als Bot erkannt oder fälschlicherweise als Rampe / Wand)?

- Aktuere können per Drag and Drop in dem Szenario anders platziert werden
  - Die Aktuere lassen sich mit der linken Maustaste aufheben
  - Ein Aktuere der soeben aufgehoben wurde, kann durch erneutes linksklicken, neu platziert werden
- Speichern eines geänderten Szenarios.
  - Das Szenario wird nach dem speichern korrekt, mit den vorgenommenen Änderungen geöffnet.
  - Das alte Szenario wurde nicht überschrieben
- Ausführen der Simulation
  - Die Simulation läuft über 5 Minuten stabil
  - Es wird keine Exception im in der Logausgabe erzeugt
  - Die UI wird über Bewegungen der Aktuere informiert und zeichnet diese wie erwartet

## 6 Teilbericht Materialfluss

Das Ziel der Materialflussgruppe ist die Umsetzung einer physischen Zelle des Umschlagslagers. Dabei sollen vier Rampen für die (Zwischen-)Lagerung und vier Volksbots für die Distribution der Pakete eingesetzt werden. Aufgabe des Materialflusssystems ist die verteilte Steuerung und Überwachung dieser Ressourcen mittels eines Multi-Agenten-Systems (MAS).

Das folgende Kapitel beschreibt die Ergebnisse der Materialflussgruppe. Dazu werden zunächst die gestellten Anforderungen zusammengetragen. Anschließend werden die eingesetzten Komponenten und Werkzeuge näher beschrieben, bevor auf die entwickelte Systemarchitektur eingegangen wird. Im letzten Schritt erfolgen die Validierung der Ergebnisse im Vergleich mit den Anforderungen und eine Analyse der aufgetretenen Herausforderungen und Probleme.

### 6.1 Anforderungen

In diesem Abschnitt werden die gestellten Anforderungen zusammengetragen. Wir unterscheiden dabei zwischen funktionalen Anforderungen, die die direkte Funktionalität des fertigen Systems beschreiben, und nicht-funktionalen Anforderungen, die die qualitativen Eigenschaften des Systems widerspiegeln.

#### 6.1.1 Funktionale Anforderungen

1. **Plattform:** Die physische Zelle wird als Netzwerk von Knoten in einem drahtlosen Sensornetzwerk implementiert. Als Plattform dienen MICAz-Module mit Atmel AT-Mega 128 Mikrocontroller und CC2420 Funkchip (siehe Abschnitt ??).
2. **Aktorik/Sensorik:** Die Rampen verfügen über Magnetstifte zum Vereinzeln der Pakete und Lichtschranken zum Erkennen von Paketen. Sie werden von den MICAz-Modulen angesteuert beziehungsweise ausgelesen.
3. **Kommunikation:** Die MICAz-Module auf Rampen und Volksbots kommunizieren drahtlos untereinander auf Basis von Agenten-Nachrichten.
4. **Synchronisation:** Die Simulation wird über ein Micaz-Modul, das als Gateway fungiert, an die drahtlose Kommunikation angebunden. Die Synchronisation der Zustände erfolgt über eine serielle Schnittstelle.

5. **Disposition:** Die Controller kennen den Belegungszustand der Rampe und generieren nach dem FIFO-Prinzip Aufträge, die sie an die Volksbots vergeben.
6. **Übergabe:** Wenn eine Ein- oder Auslagerung an einer Rampe ausgeführt werden soll, so übernimmt der Controller der Rampe die Kontrolle über die Fördereinheit des Fahrzeugs und sorgt dafür, dass das Paket verladen wird.
7. **Kooperation:** Einsatz kooperativer Lösungsstrategien für die Materialflussteuerung, Überwachung und Steuerung mittels Multi-Agentensystem.

### 6.1.2 Nicht-funktionale Anforderungen

1. **Ressourcen:** Bei der Entwicklung muss auf den sparsamen Umgang mit Hardwareressourcen (Rechenzeit, Kommunikationsbandbreite, Speicher) geachtet werden. Insbesondere der vorhandene Arbeitsspeicher und das Kommunikationsmedium dürfen nicht überlastet werden, um einen stabilen Betrieb zu garantieren.
2. **Stabilität:** Es müssen Maßnahmen getroffen werden, um ein stabiles System zu schaffen. Dies gilt insbesondere für die möglichst verlustfreie Übertragung von drahtlosen Nachrichten.
3. **Volksbots:** Die Module des Materialfluss über eine definierte Schnittstelle mit den Fahrzeugen kommunizieren, um auf die Aktorik und Sensorik der Volksbots zugreifen und schließlich einen Transport der Pakete gewährleisten zu können.

## 6.2 Beschreibung der Komponenten

Dieser Abschnitt beschreibt die physikalischen Komponenten, die von der Teilgruppe Materialfluss verwendet wurden. Zu den diesen Komponenten zählen die Rampen, sowie die MICAZ-Module mit ihren Mikrocontrollern.

### 6.2.1 Rampen

Rampen stellen Ein- und Ausgänge, sowie Zwischenlager im physischen System dar. Auf einer Rampe finden bis zu vier Pakete Platz. Bolzen hinter dem ersten Paket, separiert dieses von den anderen Dreien. Damit das vorderste Paket nicht vorne von der Rampe herunterfällt, sind an der Vorderseite zwei weitere Bolzen angebracht.

Durch vier Lichtschranken, wird eine Überwachung der Rampe ermöglicht. Diese beinhaltet zum einen das Abfragen, wie viele Pakete auf einer Rampe liegen. Zum anderen kann durch die Überwachung überprüft werden, an welcher Stelle Pakete liegen.

Alle vier Bolzen sind seitlich der Rampe befestigt. Eine autonome Steuerung der Rampen, wird durch ein angebrachtes MICAZ-Modul ermöglicht. Abbildung 32 zeigt ein Beispiel solch einer Rampe.



Abbildung 32: Beispiel einer eingesetzten Rampe

### 6.2.2 Mikrocontroller

Ein Mikrocontroller ist ein vollständiger Kleinstrechner auf einem einzigen Chip, dessen Zentraleinheit aus einem oder mehreren Mikroprozessen besteht. Zusätzlich enthält ein Mi-

krocontroller Speicher und Ein- bzw. Ausgabeschnittstellen zur Außenwelt. Dazu können neben einfach Ausgangspins auch komplexere Busprotokolle wie etwa USART, SPI oder CAN gehören.

Mikrocontroller werden eingesetzt, wenn eine Kommunikations- oder Steuerungsaufgabe mit möglichst geringen Ressourcen (Baugröße, Energie, Kosten) gelöst werden müssen. Die in einem Mikrocontroller verbauten Prozessorkern, Speicher und die Aus- und Eingabeschnittstellen, sind auf die Lösung derartiger Aufgaben zugeschnitten. Die große Anzahl an potenziellen Aufgabenstellungen hat zur Folge, dass es eine Vielfalt von Mikrocontrollern gibt. Meist sind die Mikrocontroller deshalb in Mikrocontrollerfamilien aufgeteilt. Innerhalb einer Familie unterscheiden sich die Controller nicht im Prozessorkern, sondern im verfügbaren Speicher und in den Ein- und Ausgabeschnittstellen [ECHT2005]. In Abbildung 33 ist der schematische Aufbau eines MCs dargestellt.



Abbildung 33: Schematischer Aufbau eines Mikrocontrollers vgl. [Brinkschulte:2002:Mikrocontroller]

Die zentrale Steuereinheit eines MCs ist der **Prozessor** (engl.: Central Processing Unit (CPU)). Sie ist die wichtigste Funktionseinheit und für die Verarbeitung von Befehlen und arithmetischen Berechnungen verantwortlich. Über den internen Bus kann die CPU mit

weiteren Grundbausteinen kommunizieren und beispielsweise auf Daten innerhalb des Speichers zugreifen.

Der **Speicher** besteht in der Regel aus dem Arbeitsspeicher (RAM, kurz für: Random Access Memory) und dem Programmspeicher bzw. Flash-Speicher. Normalerweise werden diese zwei Speichertypen logisch voneinander getrennt. Programme werden im nichtflüchtigen Flash-Speicher gesichert. Dieser kann mehrere Kilobyte (KB) bis Megabyte (MB) umfassen. Bei speziellen Systemen ist es möglich den Programmspeicher durch externe Flash-Komponenten zu erweitern um zusätzlichen Speicherplatz zu gewinnen.

Zwischenergebnisse, Messwerte von Sensoren, Steuergrößen usw. werden auf dem RAM abgelegt. Dieser ist deutlich schneller als der Flash-Speicher, verfügt aber in der Regel über deutlich weniger Speicherplatz. Alle Werte, welche zur Laufzeit im RAM abgelegt werden, sind im Gegensatz zum Flash-Speicher flüchtig. Das bedeutet, dass Daten bei einem Neustart des Mikrocontrollers nicht erhalten bleiben.

Durch die **Peripheriekomponenten** wird die Verbindung und Kommunikation zwischen Controller und Außenwelt ermöglicht. Über die digitalen Ein- und Ausgänge (GPIO, kurz für: General Purpose Input/Output) können Sensoren, Aktoren oder andere Systeme mit dem Mikrocontroller verbunden werden. Die meisten Mikrocontroller bieten eine Vielzahl von Ein- und Ausgängen [SOM2012].

Bei der Umsetzung des Projekts wurden MICAZ-Module eingesetzt. Im Folgenden werden kurz die Eigenheiten dieser Module erläutert.

**6.2.2.1 MICAZ-Modul** Ein MICAZ-Modul ist drahtloser Sensornetzwerknoten von der Firma Memsic. Mehrere dieser Module übernehmen in der physischen Zelle die Berechnung Geschäftslogik und die Steuerung der Rampen. Abbildung 34 zeigt ein solches Modul, während Abbildung 35 ein Blockdiagramm von dessen Struktur darstellt. Herzstück der Module ist ein ATmega128L-Mikrocontroller. Bei diesem handelt es sich um einen Low-Power-Mikrocontroller von der Firma Atmel. Darüber hinaus verfügt ein MICAZ-Modul über einen CC2420-Funkchip der Firma Texas Instruments. Dieser ermöglicht die drahtlose Kommunikation mit anderen Modulen auf einer Frequenz von 2.4 GHz ermöglicht. Es wird dabei der IEEE 802.15.4 Standard verwendet. Eine Antenne kann über eine MMCX-Schnittstelle mit dem Modul verbunden werden, um Signalstärke und -reichweite zu erhöhen. Weiter verfügen die Module über einen 128 KB großen Flash-Speicher. Zugang zu einer Vielzahl der Leitungen des Moduls gewährt ein 51-poliger Steckverbinder. Über eine Erweiterungsplatine werden so etwa die Lichtschranken und Magnetbolzen der Rampe angeschlossen. Denkbar wäre auch ein größerer Arbeitsspeicher, alle nötigen Pins des Mikrocontrollers sind über den Steckverbinder erreichbar. Im Projekt wurde die

Steckverbindung weiterhin dafür genutzt, die MICAZ-Module über ein MIB520 an einen PC anzuschließen, um sie über eine UART-Schnittstelle auszulesen und per JTAG (siehe Unterunterabschnitt 6.3.2) zu programmieren [**MICSHEET**, **C2420SHEET**].



Abbildung 34: MICAZ-Modul [**Memsic:2014:Online**]

**6.2.2.2 MIB520** Ein MIB520 stellt eine Schnittstelle für MICAZ-Module dar. Es erlaubt die Verbindung eines MICAZ-Moduls mit einem Computer per USB-Schnittstelle. So kann über die USART-Schnittstelle des Moduls mit dem PC kommuniziert werden. Über diese Verbindung wurde die Schnittstelle zu den Volksbots realisiert. Weiterhin verfügt ein MIB520-Gateway über eine **JTAG-Schnittstelle**.

### 6.3 Werkzeuge

Die Entwicklung von Anwendungen für Mikrocontroller bietet eine Reihe von Besonderheiten, insbesondere durch die begrenzte Beobachtbarkeit und Kontrolle zur Laufzeit der Ausführung. Entsprechend müssen auch die genutzten Werkzeuge über spezielle Funktionalitäten verfügen, um die Entwicklung trotzdem effizient zu gestalten. Die in dieser Projektgruppe eingesetzten Tools sollen nun im Folgenden kurz vorgestellt werden.

#### 6.3.1 Atmel Studio

Atmel Studio (vormals AVR Studio) ist eine integrierte Entwicklungsumgebung für eingebettete Systeme, insbesondere für die Atmel-eigenen Mikrocontrollerfamilien. Es basiert auf Microsoft Visual Studio, eignet sich aber insbesondere für die Entwicklung von Software für eingebettete 8- und 32-Bit Controller mit C, C++ und Assembler. Kern der Anwendung ist

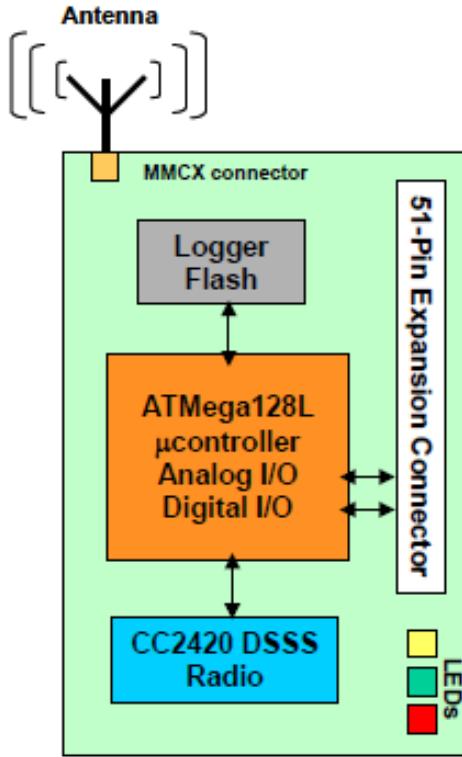


Abbildung 35: Blockdiagramm der MICAZ-Module [Memsic:2014:Online]

ein Cross-Compiler, der es erlaubt, ausführbare Binärdateien direkt für die Zielhardware zu kompilieren.

Abbildung 36 zeigt einen Screenshot vom Atmel Studio. Das Hauptfenster links zeigt die Code-Ansicht mit farblich markierter Code-Struktur, während rechts die Dateistruktur des Projekts abgebildet ist.

Neben dem Cross-Compiler beinhaltet das Atmel Studio auch einen Simulator, der es erlaubt, die Ausführung der entwickelten Software auf der Zielhardware zu simulieren. Dabei können Register, Hauptspeicher und Pins des Zielcontrollers beobachtet und Haltepunkte im Code gesetzt werden. Die Simulation ist um ein Vielfaches langsamer als die Ausführung auf echter Hardware, sodass sie etwa für das Testen von Zeitschranken kaum in Frage kommt, allerdings gibt sie einen ersten Eindruck von der Funktionalität der eigenen Anwendung.

Abbildung 37 zeigt einen Screenshot der Debugging-Ansicht des Atmel Studios. Links oben ist das Code-Fenster zu sehen. Die Ausführung ist gerade angehalten, sodass die nächste auszuführende Anweisung gelb markiert ist. Der rote Punkt links stellt einen Haltpunkt dar: Vor Abarbeitung der Anweisungen in dieser Zeile wird die Ausführung pausiert. Rechts oben befindet sich der *Processor View*, der über Registerinhalte, Taktzyklen, Tak-

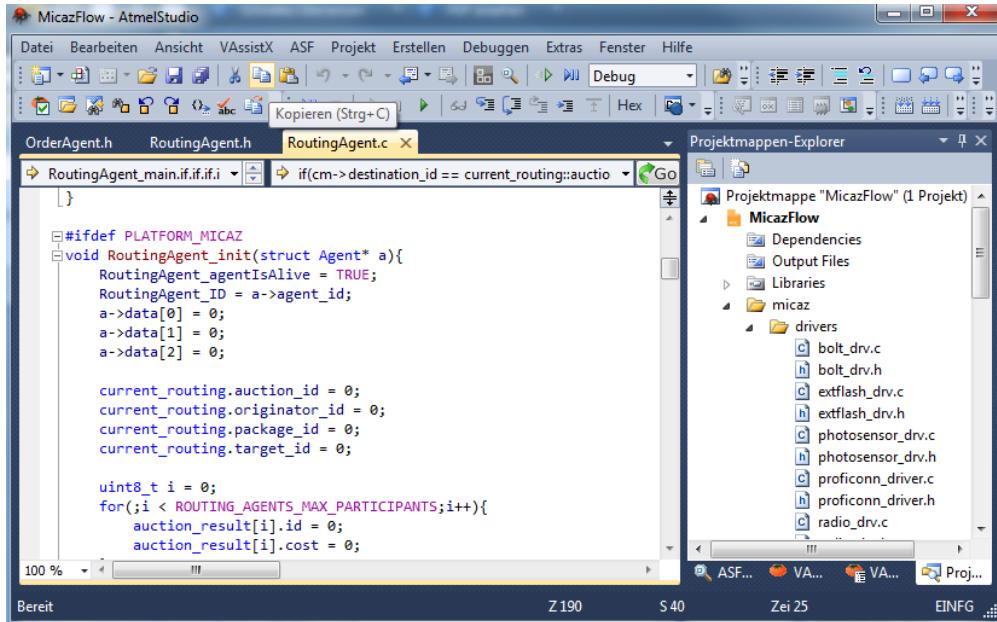


Abbildung 36: Entwicklungs-Ansicht des Atmel Studio

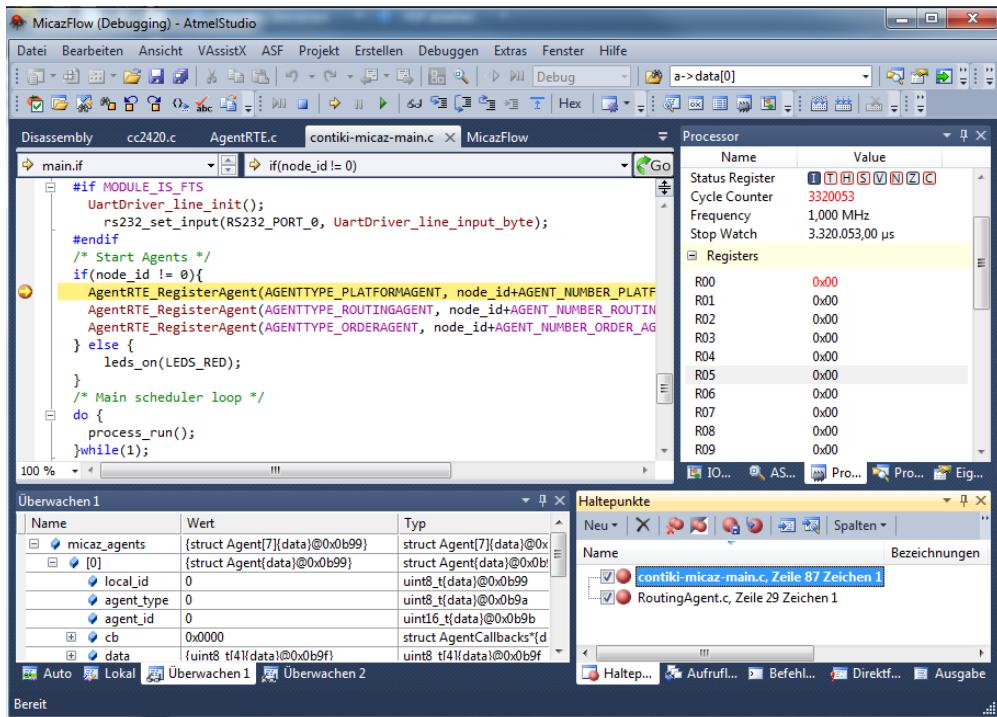


Abbildung 37: Debugging-Ansicht des Atmel Studio

rate und weitere prozessorspezifische Details informiert. Unten links befindet sich das Überwachungsfenster, in dem die Werte von ein oder mehreren Variablen oder ganzen Strukturen beobachtet und verändert werden können. Rechts daneben sind schließlich alle

Haltepunkte aufgeführt, die zur Zeit im Code definiert sind. Man unterscheidet dabei zwischen Programm- und Datenhaltepunkten. Programmhaltepunkte lösen immer dann aus, wenn bei der Ausführung die entsprechende Zeile im Code erreicht wird. Datenhaltepunkte hingegen lösen immer dann aus, wenn sich der Wert der angegebenen Speicherstelle beziehungsweise Variablen ändert.

### 6.3.2 Atmel JTAGICE3

Die Probleme der simulationsbasierten Ausführung der entwickelten Anwendung sind vor allem die üblicherweise deutlich niedrigere Geschwindigkeit im Vergleich mit der Ausführung auf der Zielhardware und zum Anderen die fehlende Peripherie, sodass angeschlossene Sensoren, Aktoren oder andere Hardwarebausteine, wie der CC2420-Funkchip der MiCAz-Module, nicht getestet werden können.

Eine Lösung bietet der Atmel JTAGICE3 [[AtmelJTAGICE3:2014:Online](#)], der über eine JTAG-Schnittstelle direkt auf die Zielhardware zugreifen und ihren Zustand auslesen und verändern kann. JTAG steht für *Joint Test Action Group* und ist eine standardisierte (IEEE Standard 1149.1, siehe [[IEEE1149:2014:Online](#)]) Schnittstelle für das Programmieren und das Debugging von integrierten Schaltungen. Es erlaubt - ähnlich wie das simulationsbasierte Debugging - den direkten Zugriff auf Register- und Speicherwerte, allerdings erfolgt der Zugriff direkt auf der Hardware. Das bedeutet insbesondere, dass auch eingehende Drahtlos- beziehungsweise UART-Nachrichten, die über die externen Ports des Mikrocontrollers eingehen, direkt überwacht und ausgewertet werden können.

Dafür verfügt das MIB520 über eine Steckerleiste, den sogenannten Test Access Port (TAP). Dieser besteht aus fünf Steuerleitungen, die die Testausführung auf dem Mikrocontroller regeln:

- Test Clock (TCK)
- Test Reset (TRST)
- Test Mode Select (TMS)
- Test Data In (TDI)
- Test Data Out (TDO)

Darüber ist es möglich, bei leicht verminderter Taktfrequenz alle relevanten Informationen zur Laufzeit aus dem Mikrocontroller zu gewinnen oder gar Veränderungen an Register- und Speicherwerten vorzunehmen, sowie Haltepunkte zu setzen. Außerdem kann ein Mikrocontroller über den JTAGICE3 auch programmiert und sein Festspeicher (meist EEPROM oder Flash) beschrieben werden.

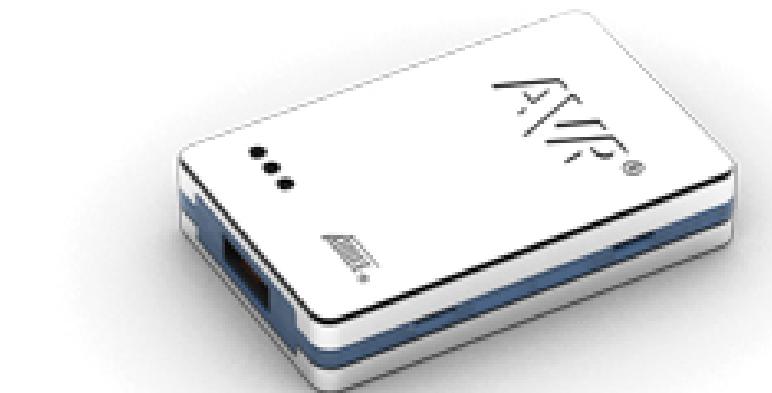


Abbildung 38: Atmel JTAGICE3 [AtmelJTAGICE3:2014:Online]

Abbildung 38 zeigt einen solches Gerät mit drei Kontrolllämpchen für Stromversorgung und Status, sowie dem zehn-poligen TAP-Konnektor (einige Pins werden für JTAG nicht genutzt, siehe oben).

## 6.4 Systemarchitektur

Das Materialflusssystem auf den MICAZ-Modulen ist in einer Schichtenarchitektur aufgebaut. Diese ist an den Aufbau von AUTOSAR [AUTOSAR:2014:Online] angelehnt. Abbildungen 39 und 40 zeigen den Aufbau des Systems auf den Rampen beziehungsweise Volksbots.

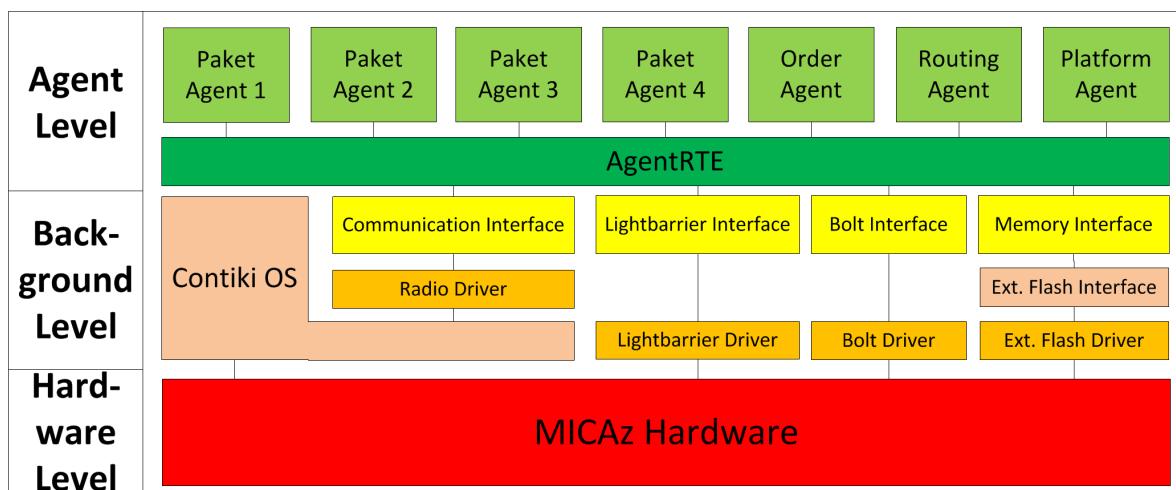


Abbildung 39: Architektur der Rampe [Stasch:Hahn]

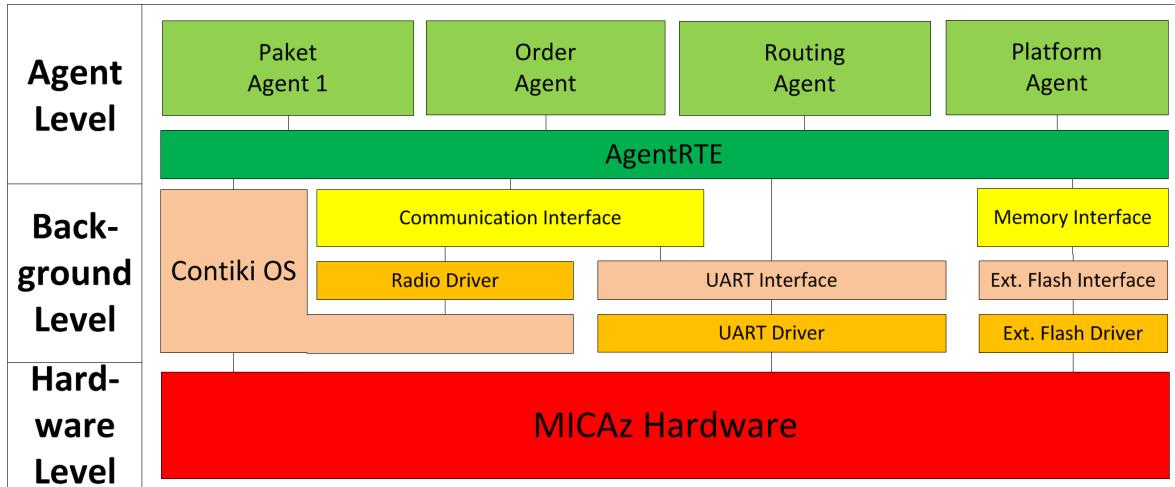


Abbildung 40: Architektur der Volksbots aus Sicht des Materialfluss [Stasch:Hahn]

Ganz unten in der Hierarchie befindet sich die eigentliche MICAZ Hardware (Hardware Level) mit allen Peripherie-Komponenten. Diese wird vom den darüber liegenden Background Level angesteuert. Im Background Level befinden sich im je nach Modul hardwareabhängige Treiber für drahtlose und serielle Kommunikation, Lichtschranken, Bolzen und einen externen Flash-Speicher. Diese agieren meist auf Pin-Ebene, steuern also die einzelnen GPIOs des Mikrocontrollers. Eine Besonderheit stellt hier der Radio-Driver dar, der nicht direkt auf die Hardware, sondern auf den Kommunikationsstack des Echtzeitbetriebssystems Contiki zugreift.

Darüber finden sich Interfaces, die die Funktionen der Treiber aufbereiten und in Funktionen gliedern, die es den oberen Schichten erlauben, ohne großen Aufwand und Kenntnis der Implementierungsdetails (konkreter Ein- beziehungsweise Ausgangs-Pin, Timing, usw.) auf die Hardware zuzugreifen. Neben den Treibern und Interfaces befindet sich im Background Level auch das Echtzeitbetriebssystem Contiki OS. Dieses beinhaltet unter anderem einen Scheduler, eine Prozessverwaltung und den Kommunikationsstack *Rime* (siehe Abbildung 6.4.1.5).

Schließlich folgt auf der höchsten Hierarchieebene das Agent Level. Hier befindet sich zunächst das AgentRTE, eine Laufzeitumgebung für Agenten. Dieses ist weitgehend hardwareunabhängig. Lediglich bei der Prozessverwaltung gibt es noch Unterschiede die in Unterabschnitt 6.4.3 noch näher betrachtet werden. Aufgaben des AgentRTE sind vor allem die Verwaltung aller Agenten auf der Plattform und deren Scheduling, sowie der Austausch von Nachrichten untereinander.

Das letzte Glied in der Kette bilden letztendlich die Agenten. Sie werden vom AgentenRTE verwaltet und sind grundsätzlich hardwareunabhängig. Die Agenten bilden die echte Be-

triebslogik des Systems ab und kommunizieren dafür untereinander mit Nachrichten. Auf jedem Modul gibt es einen Platform-, einen Order- und einen Routing-Agenten. Dazu können auf den Volksbots ein und auf den Rampen bis zu vier Paket-Agenten registriert sein.

In den folgenden Abschnitten wird nun die Implementierung des Materialflusssystems anhand dieser Architektur erläutert, beginnend beim Echtzeitbetriebssystem Contiki, über die Treiber und Interfaces hin zum AgentenRTE und schließlich den Agenten.

#### 6.4.1 Contiki

Contiki ist ein quelloffenes Echtzeitbetriebssystem (RTOS: Real Time Operating System), das in dieser Projektgruppe auf den MICAZ-Modulen eingesetzt wird [Contiki:2014:Online]. Es ist speziell für die Anforderungen des Internet of Things und von Wireless Sensor Networks zugeschnitten und bietet einen einfachen ereignisgesteuerten Betriebssystemkern mit sogenannten Protothreads (Threads, die sich einen gemeinsamen Stack teilen und daher schnell gewechselt werden können), optionalem präemptives Multithreading, Interprozess-Kommunikation via Message-Passing mit Events, eine dynamische Prozessstruktur mit Unterstützung für das Laden und Beenden von Prozessen und einen nativen Kommunikationsstack für die drahtlose Kommunikation gemäß dem IEEE-Standard 802.15.4 [IEEE802154:2014:Online].

##### 6.4.1.1 Build-Vorgang

Es existieren Implementierungen von und Treiber für Contiki für eine Vielzahl von Plattformen. Dazu gehören neben MICAZ-Modulen auch etwa der *MSP430x* von Texas Instruments oder der *Atmega128 RFA1* von Atmel. Für welche Plattform ein Contiki-System und die darauf geplanten Anwendungen gebaut wird, wird zur Compile-Zeit entschieden. Das heißt, um die selbe Anwendung mit Contiki auf mehrere Plattformen zu bringen, muss die Anwendung für jede Zielplattform neu gebaut werden.

Für jede Plattform existiert dafür ein eigener Ordner im *platforms*-Verzeichnis der Contiki-Quelldateien. Um nun das Zielsystem zu wählen, muss lediglich das TARGET beim Aufruf des entsprechenden Makefiles angegeben werden und das Build-System inkludiert automatisch die passenden Treiber und Definitionen.

Projektdateien können über die Makefile-Variable *PROJECT\_SOURCEFILES* hinzugefügt werden. Listing 1 zeigt exemplarisch das Makefile der Rampen.

```
1 # -*- Den Contiki-Pfad hier eintragen -*-
2
3 CONTIKI = ../../contiki-2.7/
4
```

```

5 all: MicazFlow
6
7 PROJECT_SOURCEFILES += drivers/bolt_drv.c interface/bolt_int.c
8 PROJECT_SOURCEFILES += drivers/photosensor_drv.c interface/photosensor_int.c
9 PROJECT_SOURCEFILES += drivers/proficonn_driver.c
10 PROJECT_SOURCEFILES += drivers/radio_drv.c ../shared/interface/
    CommunicationInterface/CommunicationInterface.c
11 PROJECT_SOURCEFILES += ../shared/agents/RoutingAgent/RoutingAgent.c
12 PROJECT_SOURCEFILES += ../shared/agents/PlatformAgent/PlatformAgent_Ramp.c
13 PROJECT_SOURCEFILES += ../shared/agents/OrderAgent/OrderAgent.c
14 PROJECT_SOURCEFILES += ../shared/agents/PackageAgent/PackageAgent.c
15 PROJECT_SOURCEFILES += ../shared/AgentRTE/AgentRTE.c
16
17 include $(CONTIKI)/Makefile.include

```

Listing 1: Makefile des Contiki-Systems der Rampen

#### 6.4.1.2 Prozesse

Prozesse in Contiki implementieren folgen einem Konzept namens Protothreads. Dies erlaubt es Prozessen, ohne den Speicher-Overhead und die langen Prozesswechselzeiten von normalen Threads auszukommen, indem sie sich einen gemeinsamen Stack auf dem Hauptspeicher teilen. Einzige Einschränkungen dieser Entwicklung sind, dass in Prozessen keine Switch-Case-Anweisungen auftreten dürfen und dass nur statische und globale Variablen zwischen zwei Aufrufen erhalten bleiben. Dynamisch erzeugte Variablen werden dagegen überschrieben. Entsprechend sollte der Zustand eines Prozesses mithilfe von statischen Variablen gespeichert werden. Listing 2 zeigt eine solche statische Variable (i) und den vollständigen Aufbau eines Prozesses.

```

1 PROCESS(example_process, "Example process");
2 AUTOSTART_PROCESSES(&example_process);
3
4 PROCESS_THREAD(example_process, ev, data)
5 {
6     PROCESS_BEGIN();
7     static uint8_t i = 1;
8     while(1) {
9         PROCESS_WAIT_EVENT();
10        printf("Received event number: %d\n", i);
11        i++;
12    }
13    PROCESS_END();
14 }

```

Listing 2: Einfacher Beispiel-Prozess in Contiki

In Zeile 1 wird der Prozess initialisiert und in Zeile 2 automatisch beim Boot von Contiki gestartet. Zeile 4 beinhaltet die Deklaration. So können andere Prozesse diesem Prozess Events (mit oder ohne Daten) schicken, auf die unser Beispielprozess mit ev und data zugreifen kann. Zeile 6 kennzeichnet den Beginn der tatsächlichen Ablauflogik. Code über dieser Zeile wird bei jedem Prozessaufruf ausgeführt, dies wird jedoch in den meisten Fällen nicht benötigt. Zeile 13 schließlich beendet den Prozess und entfernt ihn aus der Prozess-Liste des Kernels. In diesem Beispiel wird die Zeile jedoch nie erreicht, sodass der Prozess immer wieder aufgerufen wird, bis er von einem anderen Prozess beendet wird.

#### 6.4.1.3 Prozesskommunikation

In Contiki kommunizieren Prozesse über Events. Auch der Kernel versendet Events, um Prozesse über ihren Zustand (Init, Continue, Exit) oder über abgelaufene Timer zu informieren. Zur Identifikation werden dabei Event IDs genutzt. Die Event IDs 0-127 können vom Benutzer frei vergeben werden, während die Prozess IDs ab 128 vom System genutzt werden. Grundsätzlich unterscheidet Contiki zwischen synchronen und asynchronen Events.

- **Asynchrone Events** werden vom Kernel in einer Warteschlange gespeichert. Die Scheduling-Funktion des Kernels läuft nach Systemstart in einer Endlosschleife. In jedem Durchlauf wird ein Event aus der Schlange entnommen und an den Zielprozess weitergeleitet.
- **Synchrone Events** gleichen einem Funktionsaufruf. Sie werden ohne Umweg über die Warteschlange direkt an den Empfänger-Prozess zugestellt [Contiki:2014:Online]. Mit der Funktion `process_post_sync(&example_process, EVENT_ID, msg)` wird gezielt ein Prozess aufgerufen (ein Broadcast ist nicht möglich). Während der aufgerufene Prozess aktiv ist, blockiert der Aufrufer und setzt seine Ausführung erst fort, wenn der aufgerufene Prozess die Kontrolle wieder abgibt.

Um auf Events zu reagieren, können in Prozessen die folgenden Funktionen genutzt werden:

- `PROCESS_WAIT_EVENT()` - Wartet auf ein beliebiges Event, bevor die Ausführung fortgesetzt wird.
- `PROCESS_WAIT_EVENT_UNTIL(condition)` - Wartet auf ein beliebiges Event, setzt die Ausführung aber nur fort, wenn die Bedingung erfüllt ist.
- `PROCESS_WAIT_UNTIL()` - Wartet, bis die Bedingung erfüllt ist. Muss den Prozess nicht zwangsläufig anhalten.

Prozesse können neben Events auch über Polling-Anfragen kommunizieren. Polls werden bei der Bearbeitung von Hardware-Interrupts genutzt, da Interrupt-Handler keine Events absetzen dürfen. Sie können als Events mit erhöhter Priorität betrachtet werden. Ein Prozess, der einen Poll erhalten hat, wird in der Warteschlange für Prozesse priorisiert. [Contiki:2014:Online, Walter:2010].

#### 6.4.1.4 Scheduling und Timer

Grundsätzlich nutzt Contiki ein Event-getriebenes Modell von Nebenläufigkeit, wobei einzelne Events nach dem Run-To-Completion (RTC) Prinzip abgearbeitet werden. Das heißt, einmal angelaufen können Prozesse nur noch von Hardware-Interrupts unterbrochen werden oder selbst die Kontrolle abgeben. Dies ermöglicht es, alle Prozesse auf dem selben Stack arbeiten zu lassen und so Hauptspeicher zu sparen. Auf diese Weise muss kaum Speicher dynamisch alloziert werden. Außerdem werden so Race-Conditions auf geteilten Speicher nahezu ausgeschlossen. Dabei haben alle Prozesse und Events vorerst die gleiche Priorität und werden streng nacheinander in Reihenfolge abgearbeitet.

Es existiert eine Bibliothek die auch echte Threads mit jeweils einzelnen Stacks ermöglicht. Aufgrund des ohnehin knappen Hauptspeichers auf den MICAZ-Modulen wurde diese Möglichkeit jedoch in der Projektgruppe nicht weiter betrachtet.

Ein Problem der Event-getriebenen Nebenläufigkeit ist jedoch das Reaktionsvermögen auf Echtzeitanforderungen und externe Events: Sollte ein Prozess eine aufwändige Berechnung durchführen, kann es zu spät sein, bis er die Kontrolle abgibt. Aus diesem Grund führt Contiki eine zweite Prioritätsebene ein, sogenannte Polls. Diese werden zwischen asynchron auftretende Events geplant und rufen in Reihenfolge einer Priorität alle Prozesse auf, die ein Polling-Flag gesetzt haben. Üblicherweise sind dies insbesondere hardwarenahe Prozesse, die auf Änderungen an den Ein- und Ausgangspins beziehungsweise auf Timer reagieren müssen.

#### 6.4.1.5 Der Rime Kommunikationsstack

Die drahtlose Kommunikation in Contiki erfolgt über einen leichtgewichtigen Netzwerystack namens *Rime* [Dunkels:2007:Proc]. . Dieser übergibt seine Daten an und erhält seine Daten von der sogenannten *Charmeleon*-Architektur

Der *Rime*-Stack implementiert das Network- und MAC- (beziehungsweise Data Link-)Layer des ISO OSI Referenzmodells. Darunter folgt eine sogenannte *Radio Duty Cycling*-Schicht (RDC), die der Stromersparnis in drahtlosen Sensornetzwerken dient: Es ermöglicht, die Übertragungs- und Empfangseinheit des Moduls auszuschalten, während es nicht benötigt wird.

Auf der physikalischen Schicht schließlich wird die Übertragungseinheit über die Ein- und Ausgangspins angesteuert. Im Falle des MICAZ-Moduls ist dies in CC2420-Chip für paketbasierte Funkübertragung auf einer Frequenz von 2.4 GHz [CC2420:2014:Online].

Abbildung 41 zeigt den kompletten Stack, ausgehend vom Radio Driver, der in der Projektgruppe implementiert wurde bis hin zum Treiber des Funkmoduls.

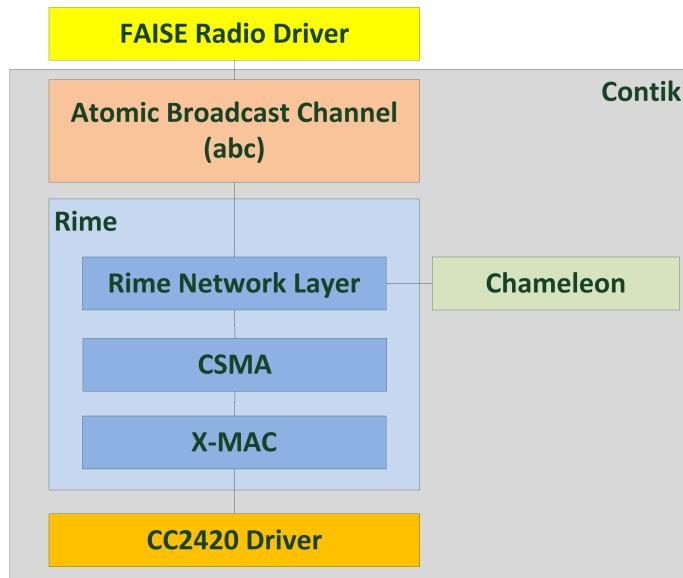


Abbildung 41: Der Rime Netzwerkstack [Dunkels:2007:Proc]

Der Radio Driver der Projektgruppe implementiert ein einfache Network-Flooding, auf das im nächsten Abschnitt näher eingegangen wird. Dieses greift schließlich auf einen *Atomic Broadcast Channel* (abc) zu. Dieser einfachste Channel in Contiki fügt der Nachricht lediglich die ID des Senders und eine Time-To-Live Angabe für das Network-Flooding als Header-Informationen hinzu. Der Channel sendet das Paket an den *Rime Network Layer*. Dieser ruft den Chameleon-Service auf. Dieser sorgt für eine Trennung von Header-Informationen und Ebenen und reduziert den Header, indem er redundante Informationen entfernt.

Anschließend wird das Paket der *Carrier Sense Multiple Access*-Schicht (CSMA) übergeben. Da drahtlose Kommunikation immer über ein geteiltes Medium erfolgt, muss vor dem Senden geprüft werden, ob nicht gerade ein anderer Knoten sendet. Ist dies der Fall, wird mit der Übertragung gewartet, bis das Medium wieder frei ist.

Wurde ein freies Medium erkannt, wird das Paket an den RDC-Treiber übergeben. Im Falle unser Projektgruppe implementiert dieser das X-Mac-Protokoll für energiesparende Kommunikation in drahtlosen Sensornetzwerken [Buettner:2006:Proc]. Dieses nutzt Early Acking und möglichst frühe Übertragung der Adresse, um die nötigen Wachzeiten der Übertragungseinheiten der teilnehmenden Module möglichst gering zu halten und so Strom

zu sparen. X-Mac sorgt weiterhin dafür, dass auch der Empfänger aufgeweckt und damit empfangsbereit ist. Ist dies der Fall, wird das Paket schließlich an den CC2420-Driver übergeben, der die passenden Ausgangspins bedient, um den externen Funkchip anzusprechen.

Ein eingehendes Paket nimmt exakt den umgekehrten Weg: Der CC2420-Chip löst einen Interrupt aus, sobald er ein Paket empfangen hat. Dieses wird von jeder der genannten Schichten verarbeitet und bis zum Radio Driver aus der Projektgruppe weitergeleitet.

#### 6.4.2 Background Level: Treiber, Services und Interfaces

Im Background Level werden Treiber, die sich um die Schnittstellen/Pins des Controllers wie z. B. Protokolle angeschlossener Devices [Stasch:Hahn] kümmern, Services, deren Aufgabe es ist, die Funktionen der Treiber zu sinnvollen Einheiten zusammenzufassen, und schließlich Interfaces, die genutzt werden, um die Funktionen der Services dem AgentenRTE zur Verfügung zu stellen und gleichzeitig plattformabhängige Implementierungsdetails zu maskieren, implementiert [Stasch:Hahn]. Im Laufe der Projektarbeit wurden unterschiedliche Treiber, Services und Interfaces entwickelt. Der folgende Abschnitt soll diese einzeln näher beleuchten.

##### 6.4.2.1 Treiber Bolzen

Folgende Funktionen werden im Treiber für die Bolzen bereitgestellt:

- void BoltDriver\_init(void)
- void BoltDriver\_up(unsigned char boltv)
- void BoltDriver\_down(unsigned char boltv)
- uint8\_t BoltDriver\_get(void)

Die Funktion `void BoltDriver_init(void)` dient zur Initialisierung. Dabei werden die Ports der MICAZ-Module, die für die Bolzensteuerung nötig sind, als Ausgänge festgelegt und gleichzeitig auf HIGH-Aktiv gesetzt.

Zum Öffnen der Bolzen wird die Funktion `void BoltDriver_down(unsigned char boltv)` genutzt. Ihr wird ein Vektor übergeben, der die Pinbelegung von den zu öffnenden Bolzen enthält. Da die Bolzen beim öffnen jeweils eine Anfangsspannung von 24 Volt benötigen, muss darauf geachtet werden, dass die Bolzen nacheinander geöffnet werden. Im ersten Schritt fragt die Funktion ab, ob Bolzen 1 geöffnet werden soll. Bei erfolgreicher Prüfung, werden Pin 1 und 2 der ersten Bolzen angeschaltet, wodurch eine Anfangsspannung von 24 Volt anliegt und der Bolzen sich öffnen kann. Damit das Öffnen erfolgreich sichergestellt werden kann, wird

dieser Zustand über 60 Taktzyklen gehalten, dann wird der zweite Pin wieder ausgeschaltet. Jetzt liegen über den angeschlossenen Verstärker noch 12 Volt Spannung an, was ausreicht um den Bolzen offen zu halten, ohne die Schaltung zu überhitzen. Im nächsten Schritt wird überprüft ob beide Bolzen gleichzeitig geöffnet werden sollen. Ist dies der Fall, dann wird wieder für 60 Taktzyklen im System gewartet. Im Anschluss daran wird kontrolliert, ob Bolzen 2 geöffnet werden soll. Liegt hier eine positive Bewertung vor, wird mit den Pins von Bolzen 2 analog zu Bolzen 1 vorgegangen.

Die Funktion `void BoltDriver_up(unsigned char boltv)` schließt die Bolzen. Dieser Funktion wird ebenfalls ein Vektor übergeben, der die Belegung der Bolzen enthält, die geschlossen werden sollen. Dabei wird geprüft, welche Bolzen geschlossen werden sollen. Anschließend werden dann jeweils beide Pins der Bolzen auf LOW gezogen, ein sukzessives *Herunterfahren* der Spannung wie beim Öffnen ist nicht nötig.

Mit der Funktion `uint8_t BoltDriver_get(void)` kann die Bolzenstellung abgefragt werden. Dabei wird einfach überprüft, ob die Ausgänge des MICAZ-Modul an denen die Bolzen angeschlossen sind HIGH sind. Beim Rückgabevektor steht eine Eins im Vektor für einen offenen und eine Null für einen geschlossenen Bolzen.

**6.4.2.2 Interface Bolzen** Das Bolzen-Interface greift direkt auf die Treiber der Bolzen zu und stellt mehrere Funktionen zur Verfügung, durch die Prozesse gestartet werden. Die unterschiedlich bereitgestellten Funktionen im Bolzen-Interface sind:

- `void BoltInterface_init(void);`
- `void BoltInterface_release(void);`
- `void BoltInterface_release_and_separate(void);`
- `void BoltInterface_separate(void);`

Durch die unteren drei Funktionen werden folgende Prozesse gestartet:

- `PROCESS_NAME(bolt_int_release)`
- `PROCESS_NAME(bolt_int_release_and_separate)`
- `PROCESS_NAME(bolt_int_separate)`

Der Prozess `bolt_int_release` ist dafür zuständig, die untersten zwei Bolzen für einen bestimmten Zeitraum zu öffnen und dann wieder zu schließen. Dafür existieren zwei Zustände. Im ersten Zustand wird die Funktion `BoltDriver_up(unsigned char boltv)` aufgerufen. So wird sichergestellt, dass die oberen beiden Bolzen geschlossen sind, um nur exakt ein Paket freizugeben. Nach dem Aufruf der Funktion wird ein Timer gestartet. Zustand zwei ruft,

sobald der Timer abläuft, die Funktion *BoltDriver\_down(unsigned char boltv)* aus dem Treiber mit einem Vektor für die unteren beiden Bolzen auf und das Paket wird freigegeben. Im Anschluss daran, wird wieder ein Timeout gesetzt und gewartet. Nach dem Durchlauf der zwei Zustände wird vorm Prozessende die Funktion *BoltDriver\_up(unsigned char boltv)* aus den Treibern aufgerufen, um die unteren Bolzen wieder zu schließen.

Um Pakete nicht nur von der Rampe auszugeben, sondern auch gleich das nächste Paket zu separieren, ist der Prozess *bolt\_int\_release\_and\_separate* zuständig. Dabei werden zu Anfang des Prozesses vier unterschiedliche Zustände durchlaufen. Die ersten drei Zustände sind dafür da, das Paket von der Rampe zu entfernen. Hierbei wird genau wie oben beim der Funktion *bolt\_int\_release* vorgegangen. Zustand vier wird im Anschluss daran dann benutzt, die oberen Bolzen wieder zu öffnen, damit das nächste Paket nachrutschen kann. Vorm Abschluss des Prozesses werden diese dann wieder durch die Treiberfunktion *BoltDriver\_up(unsigned char boltv)* geschlossen.

Mit dem dritten Prozess wird es ermöglicht, die oberen zwei Bolzen zu öffnen. Beim Prozessdurchlauf wird im Zustand eins sichergestellt, dass die unteren beiden Bolzen geschlossen sind. Im zweiten Zustand werden dann die oberen zwei Bolzen geöffnet, was ein Nachrutschen von Paketen ermöglicht. Bevor der Prozess dann beendet wird, werden die beiden Bolzen wieder geschlossen. Der Prozess ermöglicht es somit, ein Paket zu separieren.

#### 6.4.2.3 Treiber Externer Speicher

#### 6.4.2.4 Interface Externer Speicher

#### 6.4.2.5 Service Externer Speicher

**6.4.2.6 Treiber Lichtschränken** Durch den Lichtsrankentreiber werden zwei Funktionen zur Verfügung gestellt. Zum einen ist es die Funktion *void photosensor\_drv\_init(void)*. In dieser werden alle nötigen Eingänge der MICAZ-Module initialisiert. Zum anderen wird durch den Treiber die Funktion *uint8\_t get\_photosensors()* implementiert. Durch die Funktion kann der Zustand der Lichtschränken abgefragt werden. Dabei steht eine 1 für eine unterbrochene und eine 0 für eine offene Lichtschanke.

**6.4.2.7 Interface Lichtschränken** Über das Interface zur Lichtschanke werden folgende drei Funktionen bereitgestellt:

- void PhotosensorInterface\_init(void)

- `uint8_t PhotosensorInterface_is_bay_occupied(uint8_t i)`
- `uint8_t PhotosensorInterface_num_packages(void)`

Durch die Funktion `void PhotosensorInterface_init(void)` werden die Lichtschranken über den Treiber initialisiert. Dafür wird lediglich die Funktion `void photosensor_drv_init(void)` aus dem Treiber aufgerufen.

Die zweite oben aufgeführt Funktion wird genutzt, um zu überprüfen, ob eine bestimmte Lichtschranke unterbrochen wurde oder nicht. Dafür wird der Funktion ein Wert übergeben, der die zu überprüfende Lichtschranke enthält. Ist der übergebende Wert größer als vier, mehr Lichtschranken wurden nicht verbaut, oder kleiner als eins wird eine 0 zurückgegeben, was als nicht ausgelöste Lichtschranke zu interpretieren ist.

**6.4.2.8 Treiber Funkmodul** Der Treiber für das Funkmodul ist zuständig für die korrekte Funktionalität der drahtlosen Kommunikation zuständig und greift dafür auf den Contiki-Kommunikationsstack *RIME* zu. Auf das Coktiki-interne Atomic Broadcast Protokoll wird durch den Treiber ein Networkflooding eingesetzt, das eine zuverlässigere Nachrichtenübermittlung ermöglicht. Dabei werden folgende Funktionen zur Verfügung gestellt:

- `void radio_open(struct radio_conn *c, uint16_t channel, const struct radio_callbacks *u)`
- `void RadioDriver_sendmessage(uint8_t* msgtext, uint8_t length)`
- `void RadioDriver_receivemessage(struct radio_conn *c)`
- `void RadioDriver_init(void)`

**6.4.2.9 Treiber UART-Schnittstelle** Der UART-Treiber ermöglicht die Kommunikation der *Micaz*-Module über den USB-Port. Dafür wurden vom Treiber die folgenden Funktionen entwickelt.

- `void UartDriver_send(uint8_t *buf, uint8_t size)`
- `int UartDriver_line_input_byte(unsigned char c)`
- `void UartDriver_line_init(void)`

und zusätzlich der Prozess

- `UartDriver_recv_process`

Die Initialisierung der UART-Schnittstelle findet in der Methode `void UartDriver_line_init(void)` statt. Dabei wird ein Ringbuffer, der für den Empfang der Zeichen zuständig ist, initialisiert. Außerdem wird der für die Verarbeitung verantwortliche Prozess gestartet.

Die Funktion `void UartDriver_send(uint8_t *buf, uint8_t size)` wird für das Senden über die UART-Schnittstelle benötigt. Dabei wird ein Zeiger auf die Adresse der zusendende Nachricht und die Länge der Nachricht an die Funktion übergeben. Zu Anfang der Funktion wird geprüft, ob die zu sendende Nachricht kleiner als fünf Byte ist. Wenn die Nachricht kürzer als fünf Byte ist, dann wird die Nachricht mit Bytes mit dem Wert 0x00 aufgefüllt. Die Nachricht wird dann byteweise über die UART-Schnittstelle ausgegeben.

Die Methode `int UartDriver_line_input_byte(unsigned char c)` wird zum einlesen über die UART-Schnittstelle verwendet. Es handelt sich hierbei um einen Callback, der von einem Interrupt immer dann aufgerufen wird, wenn ein Byte empfangen wird. Dafür wird überprüft, ob ein Überlauf vorliegt. Wenn dies der Fall ist, wird das Zeichen komplett bis zum Zeilenende ignoriert und der Parameter für den Überlauf auf 0 gesetzt. Wenn genug Platz im Buffer vorhanden ist, wird das empfangene Zeichen eingelesen und im Buffer gespeichert. Kann das Zeichen nicht mehr gespeichert werden, wird der Parameter für den Überlauf auf 1 gesetzt. Zum Schluss wird der Prozess `UartDriver_recv_process` gepollt. Dieser ist für die Verarbeitung der empfangenen Nachricht verantwortlich. Dabei wird das eingehende Byte in einen Buffer geschrieben. Wenn der Buffer eine Länge von fünf Byte hat, wird dieser an das Communication Interface weitergereicht.

**6.4.2.10 Interface UART-Schnittstelle** Durch das UART-Interface werden folgende Funktionen zur Verfügung gestellt:

- `void UARTInterface_drive_to_entry(uint16_t rampid)`
  - Gibt den Befehl aus, zum Eingang der Rampe mit der Rampen-ID rampid zu fahren
- `void UARTInterface_drive_to_exit(uint16_t rampid)`
  - Gibt den Befehl aus, zum Ausgang der Rampe mit der Rampen-ID rampid zu fahren
- `void UARTInterface_how_cost_job(uint16_t startid, uint16_t targetid)`
  - Stellt die Frage, wie hoch die Kosten sind ein Paket von Startrampe (startid) zur Zielrampe (targetid)
- `void UARTInterface_give_package(void)`
  - Befehl Paket an die Rampe abzugeben

- void UARTInterface\_take\_package(void)
  - Befehl Paket von Rampe entgegenzunehmen

Alle Funktionen sind dazu nötig um Befehle an die Teilgruppe Drive zusenden. Mit diesen können Nachrichten an den Volksbot über die USB-Schnittstelle versandt werden.

### 6.4.3 Agenten RTE

Auf der nächsten Hierarchieebene über den Interfaces liegt die Laufzeitumgebung der Agenten (AgentRTE). Es ist die erste plattformunabhängige Ebene über den Interfaces und dem Betriebssystem. Vergleicht man die Implementation für STASH-Controller und MICAZ-Module, unterscheidet sich das AgentenRTE lediglich im Prozessmodell der Agenten, das abhängig vom Betriebssystem ist: Während die STASH-Controller mit dem MSP430-Mikrocontroller auf SYS/BIOS mit präemptivem Scheduling stetzen, läuft auf den MICAZ-Modulen ein nicht-präemptives Contiki-System. Die Agenten auf den Stash-Controllern werden daher jeweils durch einen eigenen Prozess repräsentiert, während es auf den MICAZ-Modulen einen gemeinsamen Prozess gibt, der Agenten über Funktionszeiger aufruft. Wir gehen hier nur auf die Implementierung auf den MICAZ-Modulen ein.

Neben dem Aufruf der einzelnen Agenten ist das AgentenRTE auch für das Registrieren und Terminieren der Agenten und den Austausch beziehungsweise die Verteilung von Nachrichten verantwortlich.

#### 6.4.3.1 Verwaltung der Agenten

Agenten werden mit ID und Typ registriert. Nach Konvention bekommt dabei der Plattform-Agent stets die ID des Moduls, der Order-Agent eine um eins erhöhte ID und der Routing-Agent eine um zwei erhöhte ID. Hat Beispielsweise das Modul die ID 0x0110, so hat der Plattform-Agent ebenfalls die ID 0x0110, der Order-Agent die ID 0x0111 und der Routing-Agent schließlich die ID 0x0112. Die Registrierung dieser Agenten erfolgt in der Main-Methode des Systems. Die Nummerierung der Paket-Agenten ist unabhängig von dieser Konvention, allerdings muss darauf geachtet werden, dass ihre IDs nicht mit den übrigen Agenten übereinstimmen.

Registrierte Agenten werden in ein Array aus Agenten-Strukturen eingetragen und dort verwaltet. Listing 3 zeigt die Agenten- und Agenten-Callback-Strukturen. Außerdem wird einmalig ihre Init-Methode aufgerufen.

Bei der Ausführung wird nun bei jedem Prozessaufruf des Agenten-Prozesses eine Zählervariable erhöht, die jeweils die Main-Methode des nächsten Agenten aufruft. Da Contiki ein

nicht-präemptives Scheduling nutzt, dürfen die Agenten nicht blockieren, sondern müssen die Kontrolle an den Haupt-Prozess zurückgeben, sprich die Main-Methode muss terminieren.

```

1 struct Agent {
2     uint8_t local_id; // lokale Position für Nachrichten-Buffer
3     uint8_t agent_type; // Typ des Agenten: Order-Agent, Routing-Agent, ...
4     uint16_t agent_id; // global eindeutige ID
5     const struct AgentCallbacks *cb; // Funktionszeiger
6     uint8_t data[4]; // Daten, die zwischen Aufrufen gesichert werden müssen
7 };
8
9 struct AgentCallbacks {
10     void (*agent_init)(struct Agent* a); // Ausführung bei Initialisierung
11     void (*agent_main)(struct Agent* a); // Ausführung bei jedem Aufruf
12     void (*agent_kill)(struct Agent* a); // Ausführung bei Terminierung
13 };

```

Listing 3: Agenten Strukturen in C

#### 6.4.3.2 Austausch von Nachrichten

Der Austausch von Nachrichten und damit die Kommunikation zwischen Agenten ist unerlässlich in einem Multiagentensystem. Auch diese Aufgabe kommt dem AgentenRTE zu. Die Laufzeitumgebung prüft den Empfänger einer Nachricht und übergibt sie schließlich an den richtigen Agenten. Der Aufbau einer solchen Agenten-Nachricht wird durch eine Struktur beschrieben, die in Listing 4 abgebildet ist. Der Kopf einer Nachricht ist demnach 15 Byte groß, die Nutzdaten können bis zu 26 Byte betragen.

```

1 struct Communication_Message {
2     uint16_t destination_id; // Empfänger-ID oder Gruppen-ID
3     uint16_t source_id; // Sender-ID
4     int8_t priority; // Agent, Device, NoMessage
5     uint16_t type; // Nachrichten-Typ
6     uint16_t agentType; // nur für die Registrierung von Agenten: Agenten-Typ
7     uint16_t messageID; // Sender und Message-ID kennzeichnen eine Nachricht
9         eindeutig
8     struct ThreeByte_ID conversationID; // Genutzt für Auktionen zwischen
10        Volksbots
9     int8_t dataLength; // Länge der Nutzdaten in Bytes
10     uint8_t data[MAX_MESSAGE_DATA_SIZE];
11 };

```

Listing 4: Struktur einer Agenten-Nachricht in C

Sendet ein Agent eine solche Nachricht, übergibt er sie dem AgentRTE. Dieses prüft, ob sich der Empfänger-Agent auf dem eigenen Modul befindet. Ist das der Fall, wird die Nachricht in der Warteschlange für eingehende Nachrichten gespeichert und kann dort vom Empfänger abgerufen werden. Wenn der Agent nicht auf der Plattform registriert ist, wird die Nachricht an das CommunicationInterface weitergeleitet, das sich zum die Verteilung im Netzwerk kümmert. So gelangt die Nachricht auch über die Modulgrenzen hinweg zum richtigen Agenten.

Eingehende Nachrichten werden vom CommunicationInterface an das AgentRTE weitergereicht. Dafür wird zunächst angefragt, ob sich der Ziel-Agent beziehungsweise einer der Ziel-Agenten im Falle von Gruppennachrichten auf der Plattform befindet. Ist dies der Fall wird die Nachricht in der Warteschlange für eingehende Nachrichten gespeichert.

Ein großes Problem bei Nachrichtenaustausch stellt der mit 4 KB sehr begrenzte Arbeitsspeicher der MICAZ-Module dar. Entsprechend ist der Platz in der Warteschlange für eingehende Nachrichten begrenzt, pro Agent können nur drei Nachrichten gespeichert werden. Um den Speicher nicht überlaufen zu lassen und dadurch Nachrichten zu verlieren, wird daher das Senden von Nachrichten durch ein Token-System begrenzt. Pro Agent kann so in jedem Durchlauf nur eine Nachricht versendet werden. Nutzt ein Agent dies nicht aus, kann sein Token auf einen anderen Agenten übertragen werden. In der Praxis schränkt diese Restriktion die Agenten in ihrer Funktionalität jedoch nicht ein. Meist erfordert eine eingehende Nachricht nur eine direkt Antwort oder die Benachrichtigung eines anderen Agenten. Müssen einmal doch zwei oder mehr Nachrichten als Reaktion versendet werden, geschieht dies mithilfe von Zwischenzuständen und Flags, die beim nächsten Durchlauf aktiv werden.

## 6.5 Agenten

Oberhalb des AgentRTE sind die Agenten implementiert. Sie bilden die oberste Ebene des Systems und implementieren die eigentliche Funktionalität. Dabei greifen sie auf das AgentenRTE und die verschiedenen Interfaces zu und kommunizieren untereinander über Agenten-Nachrichten. Auf jedem Modul agieren ein Plattform-Agent, der die Sensoren und Aktoren der Plattform steuert, ein Order-Agent, der zusammen mit anderen Order-Agenten die Betriebslogik des Materialflusssystems darstellt, ein Routing-Agent, der gemeinsam mit anderen Routing-Agenten die Pakete durch das System leitet und eventuell bis zu vier Paket-Agenten, die die physischen Pakete repräsentieren und durch das System wandern. Die einzelnen Agenten sind im Folgenden beschrieben.

### 6.5.1 Paket-Agent

Ein Paket-Agent repräsentiert ein physisches Paket. Seine ID ist gleichzeitig auch Paketnummer. Beim Eintritt in das System wird vom Plattform-Agenten ein neuer Paket-Agent initialisiert. Wechselt ein Paket von einem Modul auf das nächste, so wandert auch der Paket-Agent auf das neue Modul. Dies wird erreicht, indem der Agent auf dem einen Modul beendet und auf dem nächsten mit seinem Ziel und seiner ID neu initialisiert wird. Dies ist möglich, da sich verschiedene Pakete nur durch ihr Ziel und ihre ID voneinander unterscheiden. Die Übertragung der Paket-Agenten wird von den Plattform-Agenten der jeweiligen Module übernommen.

Das Ziel von Paket-Agenten ist dynamisch. Es wird von den Order-Agenten verwaltet. Kommt ein neuer Auftrag ins System, wird vom Order-Agenten, der den Auftrag verwaltet, eine Nachricht an das entsprechende Paket gesendet. Erhält der Paket-Agent eine solche Nachricht, wird dem Order-Agenten der Empfang bestätigt und das eigene Ziel wird angepasst.

Hat der Paket-Agent ein gültiges Ziel und wird ihm vom Plattform-Agenten per Flag die Erlaubnis erteilt, sendet er dem Routing-Agenten seiner Plattform eine Routing-Anfrage, bestehend aus dem Ziel des Pakets. Wenn diese nicht abgelehnt wird, wartet der Paket-Agent, bis sein Ziel geändert wird, oder er sich auf einer Plattform befindet, die nicht seinem aktuellen Ziel entspricht und er die erneute Erlaubnis bekommt, eine Routing-Anfrage zu stellen. Wird die Anfrage dagegen abgelehnt, stellt er beim nächsten Aufruf eine neue, bis die Anfrage vom Routing-Agenten angenommen wird.

Schließlich kann der momentane Zustand des Pakets über eine sogenannte *UPDATE\_PHYSICAL*-Nachricht von einem Gateway abgefragt werden. Der Agent antwortet darauf mit der ID der Plattform, auf der er sich zur Zeit befindet und dem eigenen Ziel.

### 6.5.2 Plattform-Agent

Der Plattform-Agent ist der einzige plattformabhängige Agent des Systems. Während alle anderen Agenten auf allen Modultypen identisch sind, ist der Plattform-Agent modulspezifisch. Seine Aufgabe ist die Steuerung und Überwachung seines Moduls.

Beiden gemeinsam ist jedoch die Übertragung, also das Beenden und die erneute Initialisierung, von Paket-Agenten. Diese wird immer vom Volksbot initialisiert. Es wird eine Anfrage an den Plattform-Agenten der jeweiligen Rampe gesendet, entweder nach einem Lagerplatz oder nach einem speziellen Paket, abhängig davon, ob ein Paket abgelegt oder aufgenommen werden soll. Soll ein Paket abgelegt werden, muss die Anfrage bestätigt werden, bevor

ein Registrierungs-Auftrag mit den Details des Paket-Agenten gesendet wird und der Agent auf seiner momentanen Plattform terminiert wird. Soll ein Paket aufgenommen werden, prüft die Rampe, ob das Paket mit der geforderten ID vorhanden ist und ausgegeben werden kann und sendet im Erfolgsfall ebenfalls einen Registrierungs-Auftrag und terminiert seinerseits den entsprechenden Paket-Agenten, der dann auf dem Volksbot neu initialisiert wird.

Außerdem geben beide auf eine *UPDATE\_PHYSICAL*-Nachricht die IDs aller Paket-Agenten zurück, die sich derzeit auf dem Modul befinden.

Im Folgenden werden nun die plattformspezifischen Eigenschaften der Plattform-Agenten beschrieben.

#### **6.5.2.1 Plattform-Agent der Rampen**

Der Plattform-Agent auf einer Rampe ist neben der Verwaltung der Paket-Agenten auch für die Steuerung und das Auslesen der Bolzen über das Bolt\_Interface beziehungsweise Lichtschranken über das Photosensor\_Interface verantwortlich. Er sorgt dafür, dass die Pakete korrekt vereinzelt werden und verwaltet ihre Reihenfolge. Außerdem prüft er auf die Anfrage eines Volksbots, ein Paket abzulegen, anhand der Lichtschranken, ob noch ein Platz auf der Rampe zur Verfügung steht.

#### **6.5.2.2 Plattform-Agent der Volksbots**

Der Plattform-Agent auf einem Volksbot kommuniziert mit dem Laptop, der den Volksbot steuert. Er erhält eine Nachricht, wenn der Volksbot seine Ziel-Position erreicht hat. Daraufhin initiiert er die Übergabe des Paketes. War die Übergabe des Paket-Agenten erfolgreich, sendet der dem Laptop eine Nachricht über die serielle UART-Nachricht, um das Fließband auf dem Volksbot zu starten und die physische Übernahme des Pakets zu starten.

#### **6.5.3 Order-Agent**

Die Order-Agenten übernehmen die Rolle eines zentralen Materialflussrechners im Materialflusssystem. Ihre Aufgabe ist die Verarbeitung von Aufträgen, sprich die Zuweisung von Zielen an Pakete. Aufträge bestehen aus Paket- und Ziel-ID und werden über ein Gateway in das System eingegeben. Dafür wird eine entsprechende Agenten-Nachricht an einen einzelnen Order-Agenten gesendet, der den Empfang bestätigt oder ablehnt, falls kein Platz in seinem Speicher zur Verfügung stand. In diesem Fall muss ein anderer Order-Agent mit dem Auftrag betraut werden. Ein Auftrag wird mit seiner Paket- und Ziel-ID sowie einem Status („zu bearbeiten“ beziehungsweise „in Verteilung“) in einer Warteschlange ablegt.

Hat der Order-Agent in einem Aufruf keine Nachricht erhalten, durchsucht er diese Warteschlange nach zu bearbeitenden Aufträgen und sendet eine Nachricht an das entsprechende Paket, sein Ziel zu ändern. Wird der Empfang dieser Nachricht vom Paket bestätigt, wird der Auftrag gelöscht, von nun an ist das Paket für die Erreichung seines Ziels verantwortlich. Sind alle Aufträge in Verteilung muss davon ausgegangen werden, dass die entsprechenden Nachrichten nicht angekommen sind oder die Pakete noch nicht im System sind. Daher wird in diesem Fall der Status aller Aufträge zurückgesetzt und es wird erneut versucht, Nachrichten an die einzelnen Pakete zu senden.

#### 6.5.4 Routing-Agenten

Die Routing-Agenten kümmern sich um die Wegplanung der Pakete im Materialflusssystem. Sie suchen nach einem Volksbot, der ein Paket möglichst günstig zu seinem Ziel bringen kann. Dafür führen sie untereinander eine Auktion durch, bei der ein Transportauftrag zu möglichst geringen Kosten an einen Volksbot vergeben wird. Ein Routing-Agent reagiert dabei auf die Routing-Anfrage eines Paket-Agenten. Ein Routing-Agent kann gleichzeitig nur an einer Auktion teilnehmen beziehungsweise diese initiieren. Hiermit wird verhindert, dass ein Routing-Agent gleichzeitig zwei Auktionen gewinnt und deshalb eine der Auktionen zurückgerollt und wiederholt werden muss.

Die Routing-Agenten werden als kommunizierende Zustandsautomaten implementiert. Zustandsübergänge können durch eingehende Nachrichten oder Timer ausgelöst werden. Abbildung 42 zeigt den zugrunde liegenden Automaten.

Ein Routing-Agent startet stets in Zustand 0. In diesem Zustand wartet er auf eingehende Routing-Anfragen. Diese können entweder von Paketen auf der eigenen Plattform oder von anderen Routing-Agenten kommen. Sie unterscheiden sich im ersten Byte der Conversation-ID einer Agenten-Nachricht. Hier ist die Auktions-ID gespeichert, die für neue Routing-Anfragen von Paketen erst durch den Routing-Agenten bestimmt werden muss und daher auf 0 gesetzt ist.

Bei Eingang einer Routing-Anfrage durch ein Paket wird eine neue Routing-Anfrage an alle Routing-Agenten versendet, ein Timer gestartet, der abläuft, wenn die Bearbeitungszeit erreicht ist, und in Zustand 3 übergegangen. Kommt die Anfrage von einem anderen Routing-Agenten prüft der Agent, ob das Ziel erreichbar ist. Für Module auf einem Volksbot ist dies immer wahr, für Module an Rampen immer falsch. Ist das Ziel erreichbar, wird eine UART-Nachricht an den Volksbot gesendet, der die Kosten der Fahrt vom anfragenden Routing-Agenten zum Ziel des Pakets berechnen soll. Anschließend geht der Automat in Zustand 1 über.

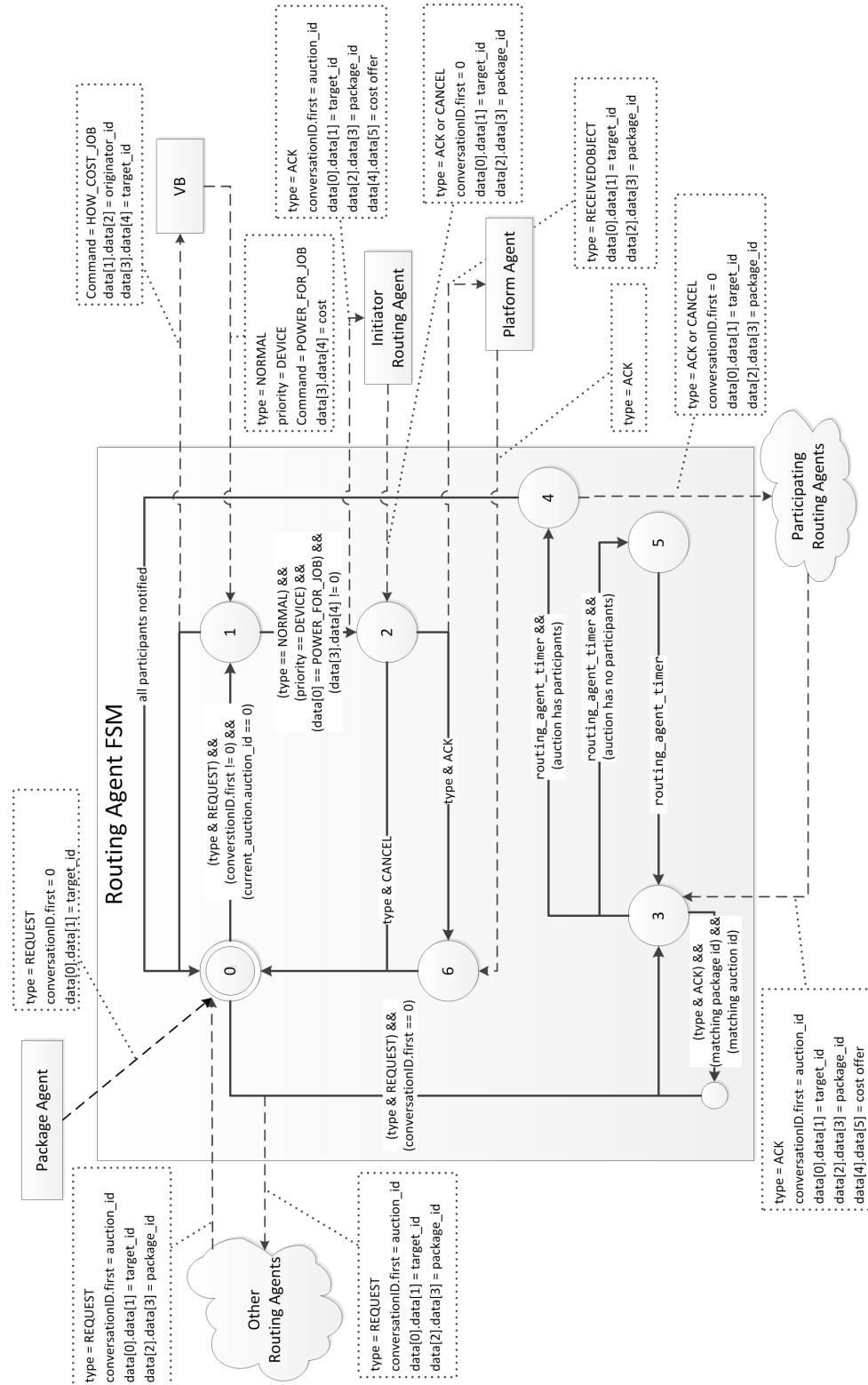


Abbildung 42: Zustandsautomat des Routing Agenten

In Zustand 1 wartet der Agent auf die Antwort des Volksbots auf seine Kostenanfrage. Geht diese ein und sind die Kosten größer als null, bedeutet dies, dass das Ziel erreichbar ist. Der Agent sendet anschließend diese Kosten an den Initiator der Auktion und geht in Zustand 2 über, wo er auf die Antwort des Initiators wartet. Wird das Angebot bestätigt, wird der Volksbot beauftragt, sich zum Ausgang des Initiators zu bewegen und der Automat geht in Zustand 6 über. Wird die Anfrage abgelehnt, geht der Automat zurück in Zustand 0 und wartet auf neue Anfragen. In Zustand 6 wartet der Routing Agent auf eine Nachricht des Plattform-Agenten, der bestätigt, dass das Paket abgegeben wurde und neue Anfragen angenommen werden können.

In Zustand 3 wartet der Routing-Agent auf Angebote von anderen Routing-Agenten, nachdem er eine Routing-Anfrage für ein Paket auf der eigenen Plattform verschickt hat. Er speichert diese Angebote mit Absender-ID und Kosten. Läuft schließlich der Bearbeitungs-Timer ab, wird geprüft ob mindestens ein Angebot eingangen ist. Ist das der Fall, wird das beste Angebot bestimmt und der Agent geht in Zustand 4 über. Andernfalls wird ein neuer Timer gesetzt, bis die Anfrage wiederholt wird und der Agent geht in Zustand 5. In Zustand 5 wartet der Agent auf den Ablauf des Timers, versendet die Routing-Anfrage erneut an alle Routing-Agenten und geht zurück in Zustand 3. Zustand 4 dagegen bleibt aktiv, bis alle Teilnehmer der Auktion benachrichtigt wurden. In jedem Aufruf des Agenten wird eine Nachricht mit Cancel oder Acknowledge an einen weiteren Teilnehmer der Auktion gesendet. Sind alle Teilnehmer benachrichtigt, geht der Agent zurück in Zustand 0 und wartet auf neue Anfragen.

## 6.6 Validierung

Das folgende Kapitel gibt einen kurzen Überblick über die Ergebnisse der Teilgruppe Materialfluss. Dazu werden zu Anfang die erreichten Funktionalitäten aufgezeigt. Im Anschluss daran werden die wesentlichen Probleme und Herausforderungen, die während der Projektarbeit aufgetreten sind, aufgezeigt. Zum Schluss des Kapitels wird ein Ausblick gegeben, in dem beschrieben wird, welche weiterführenden Funktionalitäten aus unserer Sicht durch spätere Projektgruppen noch entwickelt werden können.

### 6.6.1 Erreichte Funktionalität

Während der Projektarbeit wurde ein lauffähiges System für den Materialfluss auf Agentenbasis fertiggestellt. Zunächst wurde dafür ein vollständiges Agenten-System auf den unterschiedlichen Modulen umgesetzt. Es erlaubt die parallele Arbeit von bis zu sieben Agenten pro MICAz-Modul und den Austausch von Agenten-Nachrichten sowohl innerhalb eines

Moduls als auch modulübergreifend. So ermöglicht es die Überwachung und Lokalisierung aller Rampen und Pakete, die sich im System befinden.

Ein am Eingang liegendes Paket kann durch das System bis zu einem gewünschten Ausgang transportiert werden. Dazu gehört zum einen, dass die Module untereinander über ihre Funkschnittstelle kommunizieren können. Es wurde außerdem die Kommunikation zwischen den MICAZ-Modulen und Volksbots über ein definiertes Kommunikationsprotokoll erfolgreich umgesetzt. Ein weiteres Merkmal in der Funktionalität ist das gelungene Zusammenspiel zwischen Rampen und Modulen, also die Abfrage und die Ansteuerung von Sensorik und Aktorik. Dies erlaubt die Vereinzelung von Paketen durch gezieltes öffnen und schließen der Bolzen. Außerdem wird die Überwachung der einzelnen Rampenplätze durch die angebrachten Lichtschranken ermöglicht. Eine weitere erfolgreich umgesetzte Funktionalität, ist die Möglichkeit Pakete durch das System routen zu lassen. Dabei wird ein verteiltes Protokoll auf Basis von kommunizierenden Zustandsautomaten mit plattformübergreifender Kommunikation der Agenten eingesetzt.

### 6.6.2 Probleme und Herausforderungen

Während der Projektdurchführung sind wir auf einige Probleme und Herausforderungen gestoßen. Auf diese soll in diesem Abschnitt noch einmal ein besonderes Augenmerk gelegt werden.

In der Produktbeschreibung der MICAZ-Module wurde eine Funkreichweite von bis zu 100 Metern beworben. Beim Testen der Funkfunktionalität ist aber sehr schnell deutlich geworden, dass dies mit den Modulen nicht oder nur unter optimalen Laborbedingungen möglich sein würde. Es wurden lediglich Entfernung von zwei bis drei Metern erreicht, wenn die Module während des Funkvorganges im Raum bewegt wurden. Bei starren Positionen wurden Entfernungen von bis zu 5 Metern auf freier Fläche erreicht. Durch Recherche in Datenblättern stellte sich heraus, dass der Flaschenhals der Funkübertragung sehr wahrscheinlich in den mitgelieferten Antennen zu finden sei. Es wurde deshalb neue Antennen mit einem Gewinn von bis zu acht dBi an die MICAZ-Module angeschlossen. Außerdem wurde ein selbstentwickeltes Networkflooding-Protokoll implementiert. Bei diesem Verfahren wird jede Nachricht durch ihre Sender-ID und eine Message-ID identifiziert und außerdem wird eine Time-To-Live (kurz TTL) Information angefügt. Wenn ein Empfänger eine ihm noch unbekannte Nachricht empfängt, überprüft er, ob er selbst das beziehungsweise ein Ziel der Nachricht ist. Wenn ja, verarbeitet er die Nachricht. Ansonsten wird die TTL dekrementiert und die Nachricht nochmals an alle erreichbaren Empfänger gesendet. Diese beiden Maßnahmen erlaubten auch in der Praxis eine sehr stabile Funkverbindung zwischen allen Modulen, die gleichzeitig nicht durch zu viel Overhead beeinträchtigt wurde.

Eine weitere Herausforderung lag im geringen Arbeitsspeicher der Module. Dieser liegt bei lediglich zwei Kilobyte, die allein vom Betriebssystem und dem Kommunikationsstack schon zu etwa 50% genutzt werden. Dies führte dazu, dass komplexe Routing-Protokolle nicht durchführbar sind. Die Herausforderung wurde dadurch gemeistert, dass ein nachrichtenbasiertes Routing durchgeführt wird, bei dem nahezu alle relevanten Informationen in den einzelnen Agenten-Nachrichten und im Zustand des zugrunde liegenden Automaten gespeichert werden können.

Die dritte Herausforderung, die hier kurz beschrieben werden soll ist das Debugging in verteilten Hardware-Systemen. Eine genaue Programmabfolge kann in solchen Systemen nur schwer beobachtet werden und Fehler im Ablauf treten durch die nicht-synchronisierten Systeme häufig nicht-deterministisch auf. Eine große Hilfe war hier der JTAGICE3 von Atmel, der es durch das Setzen von Haltepunkten im Programmcode ermöglicht, Zustände von Variablen und Ausgangspins direkt im laufenden System zu überprüfen, zu analysieren und gar zu verändern.

### 6.6.3 Ausblick

Im folgendem Abschnitt soll ein kurzer Überblick darüber gegeben werden, durch welche Entwicklungen das Projekt in späteren Arbeiten und Projektgruppen erweitert werden kann.

Hierzu zählt unter anderem das Einbeziehen von Zwischenlager-Rampen in das Routing der Pakete. So kann etwa berechnet werden, ob es kostengünstiger ist, ein Paket vom Eingang zum Ausgang über eine Zwischenrampe zu liefern oder nicht. Außerdem können Pakete, denen noch kein Ziel im System zugewiesen wurde, hier zwischengelagert werden, um Ein- und Ausgangs-Rampen nicht unnötigerweise zu blockieren. Das Routing kann noch weiter ausgebaut werden, indem Zeitslots und Reservierungen auf Plattformen eingeführt werden werden. So können die Durchlaufzeiten der Pakete und die Gefahr von Deadlocks verringert werden.

Weiterhin kann ein hybrider Modus mit der computergestützten Simulation entwickelt werden. So kann der Zustand der physischen Zelle in der Simulation visualisiert werden und es können virtuelle Pakete aus der Simulation in der physischen Zelle berücksichtigt werden, um das tatsächliche Verhalten beobachten zu können. Für diesen Punkt müsste ein neues Protokoll definiert, abgestimmt und eingebunden werden. Die technischen Grundlagen für ein Gateway, das es ermöglicht, die gesamte drahtlose Kommunikation zu verfolgen, sind bereits vorhanden.

Schließlich ist es möglich, mit den Volksbots nicht nur Rampen, sondern auch existierende Stetigförderer zu bedienen. Für diese muss das AgentenRTE teilweise angepasst und es müssen dedizierte Plattformagenten für jeden Stetigförderer entwickelt werden. So kann das Gesamtsystem deutlich erweitert werden: Durch die Möglichkeit der Stetigförderer, mehrere Pakete in unterschiedlichen Richtungen auszugeben, werden Durchsatz und Flexibilität deutlich erhöht.

## 7 Teilbericht Fahrzeuge

Ziel der Teilgruppe Fahrzeuge ist es, einen autonomen Transport zwischen den Rampen zu realisieren. Dafür sollen die Volksbots in der Lage sein, auf die eingehenden Aufträge zu reagieren. Dies beinhaltet die Teilnahme an Auktionen (Jobverteilung), welche durch eine Aufwandsabschätzung des Auftrages eines jeden Volksbots entschieden werden. Nach der Zuweisung an den besten geeigneten Volksbot soll dieser das Paket von der entsprechenden Startrampe holen und zur Zielrampe transportieren.

Folgendes wird in den nächsten Unterkapiteln behandelt:

- Anforderungskatalog an das Endsystem
- Beschreibung der Komponenten
- Werkzeuge
- Architektur
- Implementierung
- Erreichte Funktionalitäten
- Herausforderungen
- Ausblick

### 7.1 Anforderungen

- **Aufträge:** Aufträge enthalten den Start- und Zielpunkt, welche in der Umgebungskarte gesetzt werden. Diese werden für weitere Berechnungen verwendet.
- **Bieten:** Ein Volksbot ist in der Lage an einer Auftragsverteilung teilzunehmen, sofern er noch keinen Auftrag ausführt und sein Energievorrat nicht das kritische Minimum erreicht hat. Die Teilnahme beinhaltet eine Aufwandsschätzung anhand einer Distanzfunktion, sowie dem aktuellen Energiezustand.
- **Planung:** Bei der Erteilung eines Auftrages und dem Bieten für einen Auftrag, muss der Volksbot die Route vom aktuellen Standort zum Startpunkt berechnen. Dies gilt ebenso vom Startpunkt zum Endpunkt.
- **Navigation:** Für die Routenplanung soll der kürzeste Weg verwendet werden, sofern möglich die direkte Verbindung zum gewünschten Ziel. Die Planung, Navigation und Positionierung erfolgt dabei auf dem entsprechenden Notebook.

- **Positionierung:** Die Umgebungskarte ist jedem Volksbot bekannt. Bei Bewegungen aktualisiert dieser seine Position mittels Odometrie, um die geplante Route korrekt zu befahren. Zur lokalen Unterstützung werden die Laserscanner verwendet werden. Startpunkt eines Volkssbots wird statisch definiert.
- **Synchronisation:** Jegliche Information eines Volkssbots wird an die Simulation übermittelt. Dies beinhaltet neben der aktuellen Position auch den Ladezustand, sowie den Energievorrat eines Volkssbots.
- **Feinsteuerung:** Das genaue Heranfahren an ein Objekt, sei es die Ladestation oder eine Rampe, erfolgt mit Hilfe der Laserscanner.
- **Überabe:** Bei Einnahme der korrekten Position zum Be- oder Entladen des Volkssbots findet ein Datenaustausch mit der entsprechenden Rampe statt. Es folgt eine kooperative Interaktion beider, bis der Volksbot das Paket erhalten oder abgeladen hat.
- **Energiemanagement:** Sobald ein Volksbot seinen kritischen Energiezustand erreicht, werden alle Auftragsverteilungen ignoriert und der Bot setzt die Dockingstation als primäres Ziel. Sollten mehrere Bots die Ladestation ansteuern, oder diese bereits belegt sein, so wird der Folgeablauf durch eine Queue oder ein anderes Verfahren geregelt.
- **Kollisionsvermeidung:** Sobald eine Kollision mit einem festen oder mobilen Objekt erkannt wird, wird eine Neuberechnung, Umplanung oder ein Ausweichverfahren eingeleitet um entsprechend zu reagieren.

## 7.2 Beschreibung der Komponenten

Bei dem Volkssbot handelt es sich um einen modular aufgebauten und mobilen Roboter, der vom Fraunhofer-Institut IAIS entwickelt wurde. Die in diesem Projekt verwendeten Prototypen bestehen aus drei zentralen Elementen.

- **Fahleinheit** Die Fahleinheit wird aus zwei Maxonantrieben links und rechts, sowie dem Gerüst des Bots gebildet. Vorne befindet sich ein SICK LMS100 Laserscanner, hinten ein SICK TiM310 Laserscanner. Angesteuert werden die Hardwarekomponenten mit Hilfe von vier Epos2 Controllern, die mit Hilfe einer CAN-Verbindung ansteuerbar sind.
- **Hub-Förderband** Zum Transportieren der Pakete ist der Volkssbot mit einer Hubeinheit ausgestattet, die zum Schutz vor Überdrehungen zwei Hallsensor auf beiden Seiten beinhaltet. Des weiteren befindet sich auf der Hubeinheit ein Förderband, an dem sich zwei Lichtschranken befinden. Diese sollen zur Ermittlung des Beladungszustandes dienen.

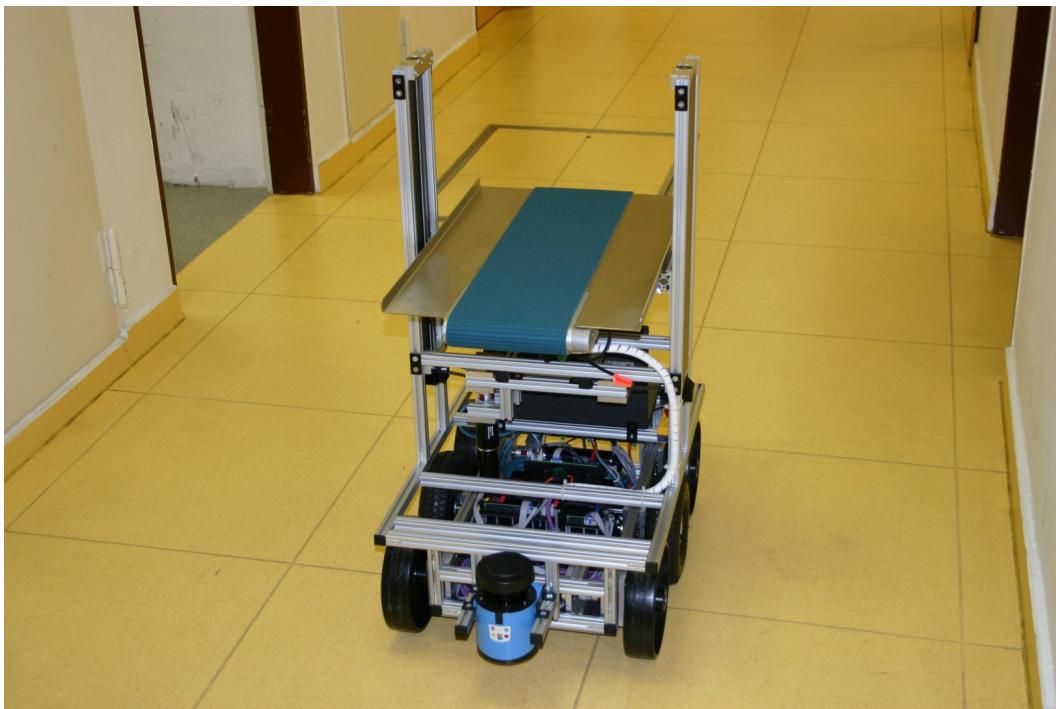


Abbildung 43: Darstellung eines Volksbots

- **Steuereinheit** Gesteuert wird das System mit Hilfe eines Notebooks, welches unter Ubuntu mit Hilfe des Robot Operating System (ROS) die Hardware anspricht. Auf dem Notebook werden ausserdem sämtliche Berechnungen durchgeführt. Neben der Ansteuerung der EPOS2 Controller wird der SICK LMS100 über eine Ethernet Schnittstelle angeschlossen. Zur Kommunikation mit den Rampen wird zusätzlich ein MI-CAZ Modul verwendet, welches mittels einer USB Schnittstelle mit dem Notebook verbunden ist.

### 7.3 Werkzeuge

Beschreibung ROS. Wichtige Funktionen in ROS:

- Publish
- Subscribe
- ...

### 7.3.1 Robot Operating System (ROS)

### 7.3.2 Was ist ROS

Es gibt viele Robotik-Frameworks, die spezifisch für präzise Anwendungen, für Prototypen, erstellt wurden. ROS strebt da eher das Allgemeine an. Das »Robot Operating System« (ROS) ist ein Open-Source Framework für individuelle Roboter, das sich in der Robotikforschung in den letzten Jahren etabliert hat und ein großes Repertoire an Software-Komponenten und -Werkzeugen für Robotikapplikationen bietet.

Die Entwicklung begann 2007 am Stanford Artificial Intelligence Laboratory im Rahmen des Stanford-AI-Robot-Projektes (STAIR). Heute wird es hauptsächlich am Robotik Institut Willow Garage weiterentwickelt. Seit April 2012 wird ROS von der neu gegründeten, gemeinnützigen Organisation Open Source Robotics Foundation (OSRF) unterstützt. Die Bibliotheken von ROS setzen auf Betriebssysteme wie Linux, Mac OS X oder Windows auf. ROS ist nicht von einer spezifischen Sprache abhängig. Heutzutage gibt es 3 Grundbibliararies für ROS, die jeweils auf Python, Lisp und C++ ausgerichtet sind. Zwei Experimentier-Libraries sind für Java und Lua erhältlich.

### 7.3.3 Was will ROS?

- ROS will unterstützen, Code für Forschung und Entwicklung wiederzuverwenden
- loser Verbund von individuellen Programmteilen (Nodes)
- einzelne Programmteile können einfach geteilt und verbreitet werden (Packages und Stacks)
- ROS stellt Repositories zu Verfügung, um dort Code zu teilen [ROS:2014:Online] (<http://www.ros.org/browse>)

### 7.3.4 Was kann ROS?

Die Hauptbestandteile und Hauptaufgaben von ROS sind Hardwareabstraktion; Gerätetreiber; Implementierung von viel genutzten Funktionalitäten; Inter-Prozess-Kommunikation; Paket-Management

### 7.3.5 Aufgaben des ROS

- Interprozesskommunikation (IPC)
  - Problematik der Kommunikation zwischen verschiedenen Systemen des Roboters
  - Sicherheitseinstellung bei der Übertragung
  - Anforderung an die Geschwindigkeit / Schnelligkeit der Kommunikation
  - Koordination von Nachrichten durch zentralen Master
- Paketverwaltung – Packages
  - ROS ist durch Softwarepakete (sogn. Packages) aufgebaut
  - Ein Package beinhaltet Laufzeitprozesse (Nodes); ROS abhängige Bibliotheken; Datensätze; Konfigurationsdateien; 3rd Party Software
  - Packages sind dazu, da um Code wiederverwendbar zu machen
- Paketverwaltung – Stacks
  - Sammlung von Paketen (Packages)
  - Der Sinn ist, dass Stacks die Verteilung und Verwendbarkeit von Code vereinfachen
  - Meist viele Packages ähnlicher Aufgaben in einem Stack verpackt
- Message (msg)
  - Messages werden verwendet um unter ROS Nachrichten zwischen Knoten und Topics auszutauschen
  - Dafür verwendet ROS eine einfache Beschreibung der Datentypen in Textdateien
  - Durch diese Beschreibung kann für unterschiedliche Sprachen Code autogeneriert werden
  - Diese sind in .msg-Dateien im msg- Unterverzeichnis eines ROS-Pakets abgelegt
  - Eigene Message-Typen sind mit Paket Ressource-Namen bezeichnet
  - Standard Messages sind mit std\_msg/msg/String.msg bezeichnet
- Service
  - ROS verwendet eine eigene vereinfachte Service Description Language ("srv") für die Beschreibung von ROS Service-Typen

- Setzt direkt auf die ROS msg-Format auf
  - Ermöglicht die Anfrage / Antwort-Kommunikation zwischen den Knoten
  - Service-Beschreibungen sind in .srv-Dateien im srv- Unterverzeichnis eines Pakets gespeichert
  - Service-Beschreibungen werden für die Verwendung mit dem Paket Ressourcen-Namen bezeichnet
  - Z. B.: wird die Datei robot\_srvs/srv/SetJointCmd.srv als Service robot\_srvs/SetJointCmd bezeichnet
- Notes
    - Der Nachrichtenaustausch findet bei Nodes durch 3 Möglichkeiten statt: Parameter;Topics; Services
    - Nodes werden wie in einem Graph angeordnet
    - In einem System laufen viele Nodes Parallel
    - Diese werden zu Beginn gestartet
    - Beispiele sind Nodes für: Laserscanner; Kinect; Pfadplanung
  - Topica
    - Topics verhalten sich wie ein virtuelles BUS-System Nodes können von Topics lesen (subscribe)
    - Nodes können an Topics senden (publish)
    - Es gibt keine Begrenzung wie viele Nodes publish oder subscribe auf ein Topic machen

### 7.3.6 ROS-Datensystem

ROS-Ressourcen sind in rangmäßiger Gliederung eingeordnet. Zwei Konzepte sind zu verstehen:

- **Le package:** Es handelt sich hier um die Zentraleinheit der Softwareorganisation von ROS. Ein Package ist ein Verzeichnis der die Knoten beinhaltet (wir werden hier unten erklären, was ein Knoten ist) sowie die externen Librairies, Daten und XML Konfigurationsdateien die manifest.xml genannt wird.

- **Stack:** Stack bezeichnet eine Sammlung von Packagen. Sie ermöglicht mehrere Funktionen wie Navigation, Lokalisierung und viele mehr. Ein Stack beinhaltet mehrere Verzeichnisse sowie eine Konfigurationsdatei die stack.xml genannt wird.
  - Vorhandene wichtige Stacks
    - \* TF – Koordinatentransformation
    - \* Navigationstack
    - \* URDF - Modelle
    - \* Beispiel Navigationstack
      - Wertet Sensordaten aus z.B.: Laserdaten
      - Baut daraus mit gmapping (ebenfalls ein ROS-Stack) eine Begehbarkeitskarte
      - Warum? Zur Kollisionsvermeidung
      - Bei erfolgreicher Erstellung einer Map kann dann ein Ziel übergeben werden (Pfadplanung durch Navigationstack, Kollisionsvermeidung, Reaktion auf sich ändernde Umgebung, Aufbau einer globalen Karte)

### 7.3.7 Vorteile und Nachteile des ROS

#### 7.3.7.1 Vorteile

- Nachrichten-basierte Software Architektur
  - Verschiedene Komponenten sind unabhängig voneinander mit dem System verbunden
  - Unterschiedliche Komponenten können miteinander verbunden werden, ohne jedes Mal das Programm neu zu Kompilieren
  - Netzwerkfähigkeit
  - Einfaches Debugging und Simulieren
- Absturz eines Nodes führt nicht zum Absturz des ganzen Systems
- Für ROS lässt sich in mehreren Sprachen programmieren
- ROS hat eine große Community, die viele Daten und Programme zu Verfügung stellen

### 7.3.7.2 Nachteile

- Durch Nachrichten-basierte Systemarchitektur Bottleneck bei großer Datenmenge
- Steuerung des Systems über Kommandozeile

## 7.4 Architektur

Der systematische Aufbau des Volksbots ist in Abbildung 44 dargestellt. Die Funktionen des Roboters basieren auf der Auswertung und Ansteuerung der Sensoren bzw. Aktoren. Alle Operationen, wie z.B. die Lokalisation, die Routenplanung, der Vorgang der Paketübergabe, laufen auf dem Robot Operating System und nutzen die Daten der Sensorik zur geeigneten Ansteuerung der Aktoren. Gesteuert werden die Operationen über die externe Kommunikationsebene, welche aus einem MICAz-Modul besteht. Hier werden Aufträge empfangen und auf der Operationsebene in zugehörige Ziele übersetzt.

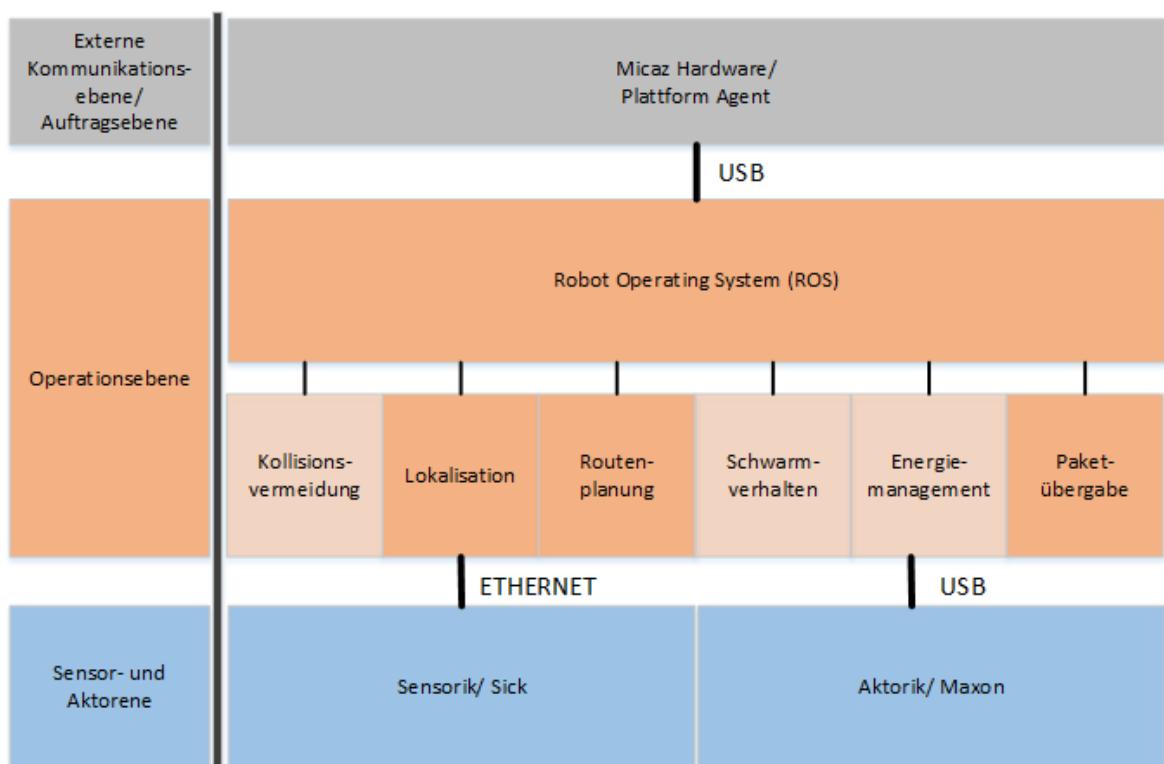


Abbildung 44: Architektur des Volksbot

Die in dunkleren rot hervorgehobenen Funktionen der Operationsebene deuten deren höhere Wichtigkeit an und geben ein erstes Anzeichen darauf welche Funktionen erfolgreich umgesetzt werden konnten.

## 7.5 Implementierung

Im folgenden Abschnitt wird auf die Implementierung der Funktionen des Transportroboters eingegangen. Dabei werden die drei Teilabschnitte, Inbetriebnahme der Hardware, Implementierung der Navigation und Implementierung der Paketübergabe, betrachtet. Der Großteil der Implementierung wurde in den ROS-Packages „epos2-control“ und „simple-navigation-goals“ realisiert.

### 7.5.1 Inbetriebnahme der Hardware

Der erste Teil der Implementierung sah die Bereitstellung von Treibern für die komplette Hardware des Volksbots vor. Die Initialisierung und Ansteuerung der an den EPOS2-Controllern angeschlossenen Peripherie erfolgte unter der Verwendung der EPOS2 Bibliothek. Diese Bibliothek verfügt über alle benötigten Funktionen zum Ansteuern und Auslesen der Motoren und Lichtschranken, welche mit den EPOS-Controllern verbunden sind. Die Funktionen wurden dem ROS-Package „epos2-control“ zusammengeführt und sind in der Epos2MotorController-Node des Packages enthalten. Die Verwendung der Laserscanner, welche nicht mit den EPOS2-Controllern verbunden sind, wurde durch die Implementierung und Parametrisierung von im ROS vorhandenen Treibern gewährleistet. Dem an der Front des Roboters angebrachten SICK LMS100 Laserscanner, wurde dafür unter Windows eine feste IP-Adresse zugeordnet. Mit Hilfe dieser bekannten Adresse, konnte auch unter Ubuntu die Verbindung über Ethernet mit dem Laserscanner durch das Package „LMS1xx“ hergestellt werden.

### 7.5.2 Implementierung der Navigation

Nach erfolgreicher Inbetriebnahme der Hardware des Volksbots wurde die Odometrieberechnung anhand der zurückgelegten Strecke der Räder implementiert. Die Berechnung der zurückgelegten Wegstrecke und der Drehung des Roboters erfolgt über folgende Formeln[Der:2000]:

$$\Delta s = \frac{\Delta R + \Delta L}{2} \quad (1)$$

$$\Delta \alpha = \frac{\Delta R - \Delta L}{D} \quad (2)$$

Die Daten der Wegstrecke und der Drehung des Roboters werden genutzt, um mit folgenden Formeln die x- und y-Position des Roboters zu berechnen:

$$x = x(t-1) + (\Delta s * \cos(\alpha(t-1) + \Delta\alpha)) \quad (3)$$

$$y = y(t-1) + (\Delta s * \sin(\alpha(t-1) + \Delta\alpha)) \quad (4)$$

Der Drehwinkel des Roboters ergibt sich aus der Addition des vorherigen Winkels mit der zuletzt durchgeführten Änderung des Drehwinkels.

```

1 void TankSteering::odomCallback(const ros::TimerEvent& event) //Funktionsaufruf
    durch Timer
2 {
3
4     double pos_delta[2], temp_pos[2];
5     for (int i = left; i <= right; i++) { //left entspricht dem Wert 0, right 1
6         temp_pos[i] = tankSettings.epos[i]->getAbsolutePosition() * tankSettings.
            wheelPerimeter; //Bestimmung der zurueckgelegten Strecke beider Raeder
7     }
8
9     for (int i = left; i <= right; i++) {
10        pos_delta[i] = temp_pos[i] - pos.lastPosition[i]; //Unterschied von
            jetziger zur vorheriger Position
11        pos.lastPosition[i] = temp_pos[i];
12    }
13
14    double polar_s = (pos_delta[right] + pos_delta[left]) * 0.5; //Berechnung der
        Wege laenge
15    double polar_theta = (pos_delta[right] - pos_delta[left]) / tankSettings.
        axisLength; // Berechnung der Drehung
16
17    pos.now.x = pos.last.x + polar_s * cos(pos.last.theta + polar_theta); //
        Berechnung und Aktualisierung der x-Position
18    pos.now.y = pos.last.y + polar_s * sin(pos.last.theta + polar_theta); //
        Berechnung und Aktualisierung der y-Position
19
20    pos.now.theta = pos.last.theta + polar_theta; //Aktualisierung der
        eingenommenen Ausrichtung
21
22 }
```

Listing 5: Implementation der Odometrieberechnung

Die Daten der Odometrie werden zur Lokalisierung des Roboters innerhalb einer mit dem SICK LMS100 Laserscanners erstellten Umgebungskarte verwendet. Die Implementierung erfolgte ebenso wie bei der Entwicklung der Treiber im „epos2-control“-Package. Bei der Erstellung der Umgebungskarte wurde auf das „Gmapping“-Package von ROS zurückgegriffen. Der darin enthaltene Algorithmus nutzt die Daten des Laserscanners, um während der Fahrt des Roboters aus seiner erfassten Umgebung eine Karte zu erstellen. Zur Visualisierung der Karte und der Daten des Laserscans wurde „Rviz“, ein Visualisierungstool innerhalb der ROS-Umgebung verwendet. Mit Hilfe von Rviz ist es neben der Visualisierung unter anderem möglich die initiale Position des Roboters, sowie Navigationsziele innerhalb der Karte festzulegen. (Screenshot Rviz) Mit dem Ziel den auftretenden Abweichungen der Odometrieberechnung entgegenzuwirken, wurde das „adaptive Monte Carlo Localisation“-Package (AMCL) implementiert. Vereinfacht formuliert, nutzt dieses Verfahren die Daten des Laserscans und der Karte, um mit Hilfe der Merkmale des aktuellen Scans und der zugrundeliegenden Daten der Kartenrepräsentation eine Schätzung der Position des Roboters auszuführen. [Bischoff:2004] Eine funktionsfähige Selbstlokalisierung ist die Grundlage für eine erfolgreiche autonome Navigation in der Umgebung des Roboters. Das ROS-Framework stellt mit dem „move-base“-Package die nötigen Funktionen für die Navigation bereit. Das Package hat den Dijkstra-Algorithmus zur Wegplanung implementiert und nutzt zwei parametrisierbare Costmaps, um Eigenschaften wie z.B. den Mindestabstand zu Hindernissen oder das Verhalten bei Planungsfehlern festzulegen. Nach der Planung des Weges werden automatisch die passenden Steuerbefehle generiert. Falls der Volksbot in eine unvorhergesehene Situation gerät und seine geplante Route nicht mehr gültig ist, kommen Rettungs-Funktionen des Packages zum Einsatz. Dabei wird eine Rotation um die eigene Achse des Roboters durchgeführt, um einen geeigneten neuen Weg zu finden.

### 7.5.3 Implementierung der Paketübergabe

Damit der Austausch der Pakete mit den Komponenten des Materialflusses erfolgen kann, musste die Kommunikation über die MICAz-Module und Automatismen zur Anpassung der Hub-Position an die Höhe der jeweiligen Rampe implementiert werden. Nachdem ein Auftrag vom Materialfluss empfangen wurde, wird die Annahme des Auftrags dem Materialfluss bestätigt und der Auftrag in ein passendes Navigationsziel auf der Umgebungskarte umgesetzt. Für den Prozess des Entschlüsselns von den Nachrichten des Materialflusses wurde ein ROS-Package namens „Simple-Navigation-Goals“ implementiert. Neben der Umsetzung von Navigationszielen sendet dieses Package ROS-interne Nachrichten, welche Informationen über die gewünschte Position des Hubs und das Erreichen der Zielposition enthalten. Sobald die Zielposition erreicht wurde, beginnt die Hubsteuerung mit der Anpassung der Hubposition an die geforderte Höhe. Anschließend folgt eine Benachrich-

tigung an den Materialfluss und das Förderband wird in Gang gebracht. Die Auswertung der Lichtschranken bewirkt den Haltevorgang des Förderbands und das nächste Navigationsziel wird festgelegt.

#### 7.5.4 Struktur der einzelnen Funktionalitäten

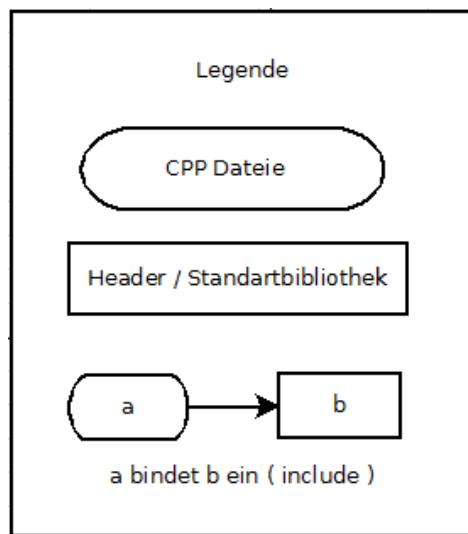


Abbildung 45: Legende zu Abb. 46 und 47

#### 7.6 Erreichte Funktionalitäten

##### Beschreibung eines Szenarios mit dem bisher implementierten Volksbot:

Mit Hilfe des Sick LMS100 Laserscanners wurde eine statische Umgebungskarte generiert, welche den Volksbots bekannt ist. Innerhalb dieser Karte hat ein Volksbot einen zuvor definierten Startpunkt den er für seine eigene Lokalisierung benötigt. Mit Hilfe des MICAZ Moduls kann der Volksbot Aufträge von den Rampen annehmen und beginnt mit der Abarbeitung, indem er mit dem DijkStra Suchalgorithmus die Route bestimmt. Während der Fahrt arbeitet der Volksbot mit Odometrie und dem Laserscanner um seine aktuelle Position zu bestimmen und seine Route aktuell zu halten, ausserdem passt er die Höhe seines Hubs dem Auftrag an. Das bedeutet das er entsprechend seiner Auftragsart den Hub hoch oder runter fährt, damit er das Paket annehmen oder abgeben kann. Sobald der Volksbot die Zielrampe erreicht hat, richtet er seine Position aus und nähert sich mit einem gewissen Sicherheitsabstand der Rampe. Zur Annahme oder Abgabe des Paketes wird das Förderband in Bewegung gesetzt und läuft solange, bis die Sensoren, welche sich an dem Förderband

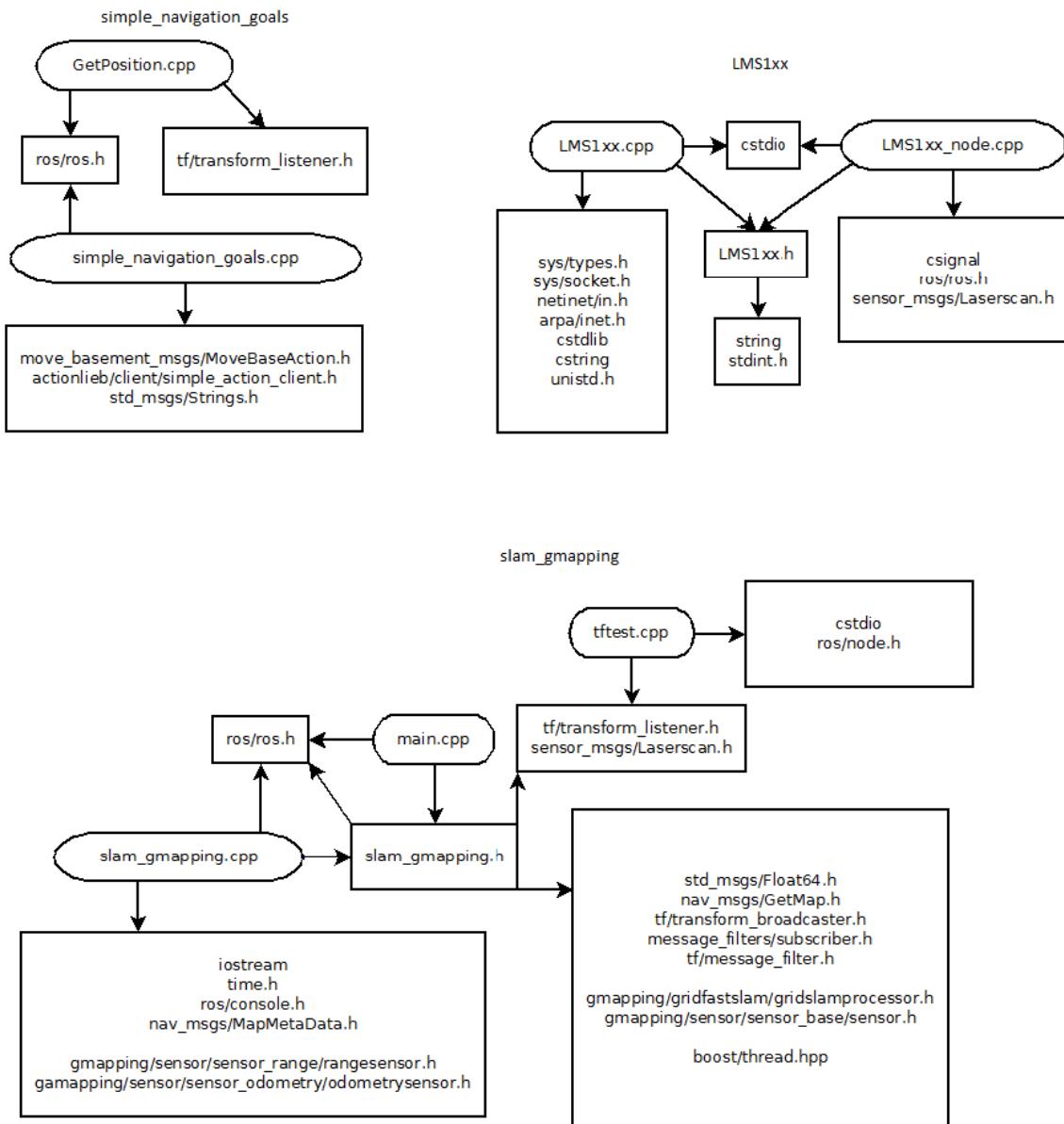


Abbildung 46: eingebundene Funktionen in ROS

befinden, das Paket vollständig auf dem Förderband lokalisiert haben oder sich bei der Abgabe kein Paket mehr auf dem Förderband befindet. Sollte der Volksbot ein Paket erhalten haben, so berechnet er die neue Route zum Ziel und fährt diese ab, ansonsten wartet er auf den nächsten Auftrag.

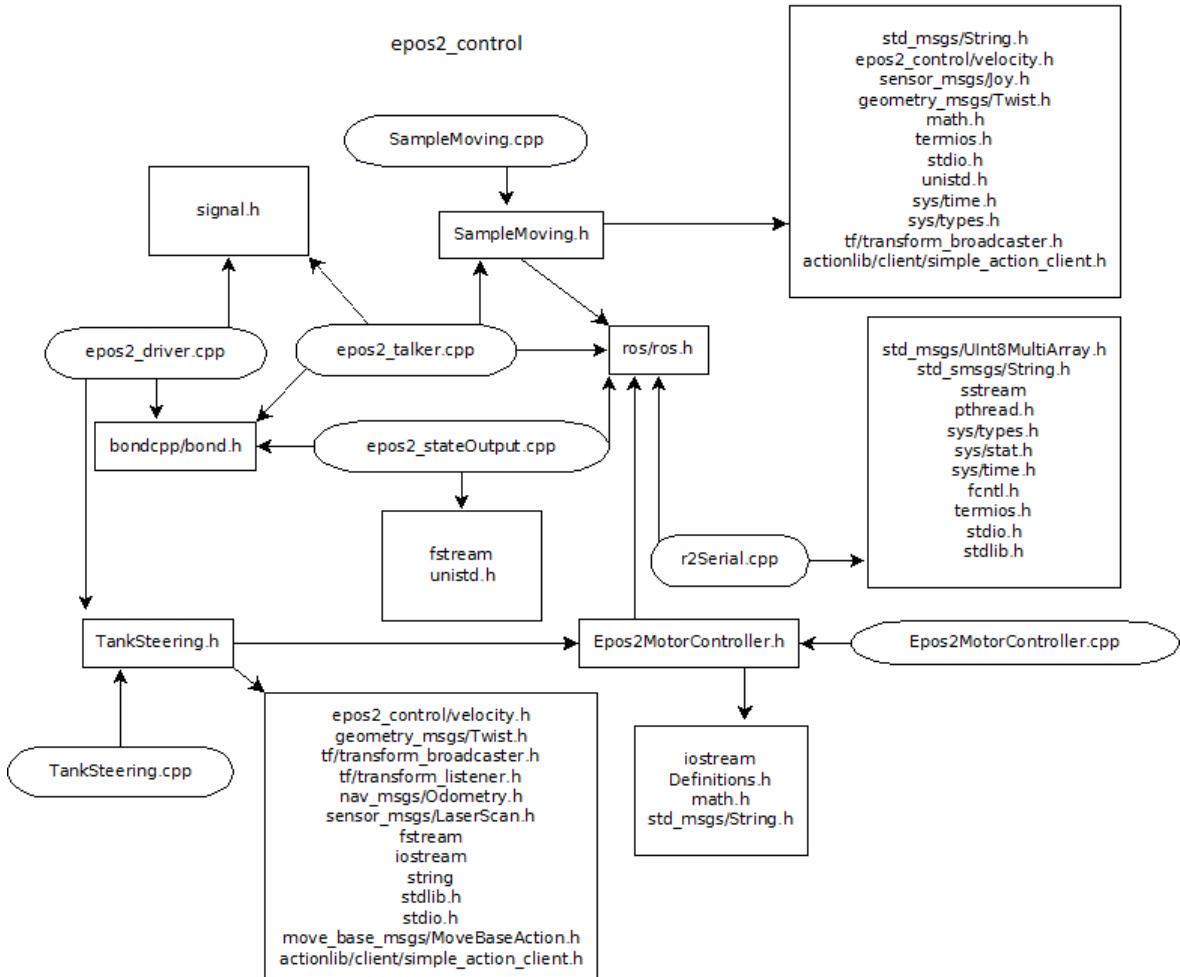


Abbildung 47: Hauptsteuerung des Systems ( Epos2 Control )

## 7.7 Herausforderungen

Bei der Implementierung der Funktionen des Volksbot traten einige Herausforderungen auf, deren Lösungen besonderen Aufwand hervorriefen oder nicht umgesetzt wurden. Ein Beispiel dafür ist die Selbstlokalisierung des Roboters durch die Odometrieberechnung und dessen Verbesserung durch die adaptive Monte Carlo Lokalisation (AMCL). Aufgrund der Fehleranfälligkeit der einfachen Odometrie durch Einflüsse wie die Beschaffenheit des Bodens oder Unterschiede im Reifendruck, ist der Einsatz von Algorithmen wie die adaptive Monte Carlo Lokalisation, welcher die Daten des Laserscanners nutzt, notwendig. Die Qualität der Odometrie musste durch das Kalibrieren des Parameters der Achsenlänge des Roboters erhöht werden, um die nötige Voraussetzung für einen funktionierenden AMCL-Algorithmus zu schaffen. Eine gute Odometrie lässt sich dadurch erkennen, dass die Punkte des Laserscans auch bei Bewegung weiterhin mit den Wänden der Umgebungskarte übereinstimmen.

Durch die Verwendung der Selbstlokalisierung über AMCL führt der Roboter während seiner Navigation Drehungen um die eigene Achse aus, um seine Position zu berechnen. Dieses Verhalten führt dazu, dass der Roboter deutlich mehr Zeit als vermutet benötigt, um seinen Weg zurückzulegen. Bei der Parametrisierung für die Wegplanung und des AMCL-Algorithmus musste ein Gleichgewicht zwischen Schnelligkeit und Genauigkeit gefunden werden, um ein stabiles System zu schaffen. Durch die Vielzahl der Möglichkeiten war die Suche nach geeigneten Einstellungen problematisch.

Eine weitere Herausforderung stellt die Erkennung von Hindernissen dar, die ober- oder unterhalb des Laserscanners in den Raum ragen. Zur Lösung dieses Problems könnte ein weiterer bildgebender Sensor genutzt werden, um den gesamten Raum nach potenziellen Hindernissen zu untersuchen.

Mit großem Aufwand war die korrekte Ansteuerung der Maxon Controller verbunden. Besonders die Parametrisierung des kleineren Maxon Controllers unter ROS, welcher für den Betrieb des Förderbandes verwendet wird, stellte sich als Problem heraus. Schon kleinere Abweichungen der Parameter für Spannungs- und Beschleunigungswerte in der Ansteuerung des Controllers führten zu Systemabstürzen.

Die Programmierung und Parametrisierung der Teifunktionen musste mit Blick auf die CPU-Auslastung der Steuereinheit geschehen, da einige Berechnungen bei falscher Verwendung zu sehr hohen Auslastungen und einer unzureichenden Performance des Roboters führten.

## 7.8 Ausblick

Der Volksbot sollte um folgende Funktionalitäten erweitert werden:

- **Genauigkeit bei der Übergabe:** Die Genauigkeit des Heranfahrens an eine Zielposition, wie z.B. den Ausgang einer Rampe, muss verbessert werden, um eine erfolgreiche Übergabe zu gewährleisten. Der Volksbot stürzt bei direktem anfahren an die Rampe ab, da der Bewegungsraum eingeschränkt wird.
- **Kollisionserkennung mobile Hindernisse - Schwarmverhalten:** Der Volksbot sollte Hindernisse, die sich selbstständig bewegen, erkennen und entsprechend reagieren. Dies könnte beim erkennen eines Objektes mittels einer Neuplanung der Route, einer kurzen Unterbrechung der Fahrt oder eines Ausweichmanövers umgesetzt werden.
- **Rückkopplung an Simulation:** Die Volksbots sollten ihre aktuelle Position, sowie den Beladungs- und Energiezustand an die Simulation zurückgeben.

- **Fahralgorithmus:** Es können alternative Alorithmen implementiert werden, um die Berechnungszeiten zu verkürzen oder optimalere Routen zu finden.
- **Kostenabschätzung:** Eine optimale Berechnung zur Bestiimmung der benötigten Zeit, Strecke und Energieverbrauchs würde die Jobverteilung effizienter gestalten.
- **Ladestation:** Die Ladestation muss angebracht und getestet werde. Danach fehlt die automatische Erkennung des kritischen Zustandes, sowie das selbstständige anfahren der Ladestation, damit der Akku geladen werden kann. Hierbei handelt es sich um einen Prototypen der noch getestet werden muss.

## 8 Integration

In den vorherigen Abschnitten wurden die Komponenten der physischen Zelle beschrieben. In diesem Kapitel soll die Integration der Fahrzeuge in den Materialfluss erläutert und anhand eines zusammenhängenden Szenarios beschrieben werden. Hierbei sollen die technische Abläufe (Verweise auf Implementierungen und Zwischensequenzen) verdeutlicht werden.

### 8.1 Ablauf in physischer Zelle

Draft:

- Paket kommt ins System
  - Zielzuweisung
- Micaz/ Rampe gibt Paket an Volksbot
- Volksbot fährt Ziel an
- Micaz/ Rampe empfängt Paket

Sequenzdiagramm....

## 9 Ergebnisse

Dieses Kapitel gibt einen abschließenden Überblick über die erreichten Ergebnisse der Projektgruppe. Zunächst sollen zusammenfassend beschrieben werden, welche Komponenten erfolgreich entwickelt wurden. Im Fazit werden die aus der Projektarbeit gewonnenen Erkenntnisse und Erfahrungen beschrieben. Der letzte Teil dieses Kapitels beinhaltet eine Vision, in der potenzielle Weiterentwicklungen und Ideen aufgegriffen werden.

### 9.1 Realisierte Ziele

Die Teilgruppe Simulation hat ein Simulationstool erstellt, das als Webanwendung realisiert wurde. Das Tool erlaubt das Erstellen von Szenarien mit einer vom Nutzer festgelegten Anzahl von Fahrzeugen und Rampen. Auch kann ein Nutzer eine beliebige Anzahl an Aufträgen generieren, die als Input für das Simulieren eines Szenarios dienen. Die Aktionen der Akteure werden serverseitig durch das Multiagentenframework JADE realisiert und clientseitig dargestellt. Jeder Akteur wird durch mehrere Agenten realisiert, die Zielsuche und Suche eines Transportmittels autonom und ohne zentrale Steuerung durchführen. Die Agenten der Fahrzeuge sind in der Lage mit einem selbst erstellten Pathfinding-Algorithmus Routen zu berechnen und diese anschließend abzufahren. Für einen Simulationsdurchlauf werden Daten über Paketdurchlaufzeit und Auslastung der Roboter mitgeloggt, die in der Statistik eingesehen werden können.

### 9.2 Fazit

Im Rahmen der Projektgruppe wurde von zwölf Teilnehmern ein System entwickelt, dass auf die drei Teilgruppen Materialfluss, Fahrzeuge und Simulation verteilt wurde. Durch die gemeinsame Arbeit an einem Projekt konnten Erfahrungen und Erkenntnisse auf dem Gebiet der Projektarbeit und Softwareentwicklung gewonnen werden. Ein wesentlicher Erfolgsfaktor eines Projekts ist das effektive Nutzen der vorhanden Zeit bei einem Projekt mit begrenzter Dauer. Grundsätzlich wurden Aufgaben sorgfältig und möglichst schnell durchgeführt. Für zukünftige Projekte empfiehlt es sich jedoch das Zeitmanagement zu optimieren, indem versucht wird Aufgaben noch weiter zu parallelisieren. Im Rahmen der Anforderungserhebung hatte sich bereits herausgestellt, dass bestimmte Technologien, wie beispielsweise ein Multiagentensystem zu einem späteren Zeitpunkt benötigt werden. Die Einarbeitung in benötigte Technologien erfolgte jeweils kurz vor der Implementierung einer Technologie, wodurch sich die Entwicklungszeit verlängerte. Um die Entwicklungszeit

zu verkürzen, sollten Teilnehmer eines Projekts sich vor der Implementierung einer Komponente in die benötigte Technologie einarbeiten und diese für die anderen Teilnehmer aufbereiten.

Die Durchführung des Projekts mithilfe des Scrum Vorgehensmodell beinhaltete Sitzungen mit den Auftraggebern in denen die festgelegten Anforderungen mit den tatsächlich realisierten abgeglichen wurden. So konnten Eigenschaften des Produkts, die nicht den Vorstellungen der Auftraggeber entsprachen, direkt und mit geringerem Aufwand korrigiert werden. Ein solcher Abgleich sollte unabhängig vom dem gewählten Vorgehensmodell regelmäßiger Bestandteil eines Projekts sein.

Für die Implementierung der Agenten sowohl im physischen als auch im softwarebasierten System, wurden die Funktionalitäten, die ein Fahrzeug oder eine Rampe besitzen soll, auf mehrere Agenten verteilt. Ziel war es die Agentensysteme zu modularisieren. Modularisierung soll grundsätzlich die Wartbarkeit und Übersichtlichkeit eines Systems erhöhen. Jedoch hat sich gezeigt, dass eine Modularisierung auch Nachteile haben kann. Durch die Vielzahl der Agenten und die Vielzahl an Nachrichten, die ausgetauscht werden, wurde die Testbarkeit des Systems verringert, da auftretende Fehler schwer einzelnen Methoden der Agenten zugeordnet werden konnten. Für zukünftige Projekte sollte differenzierter betrachtet werden, ob eine Modularisierung unter Berücksichtigung der systemspezifischen Eigenschaften sinnvoll ist.

### 9.3 Vision

Für die Simulationssoftware bieten sich folgende Optimierungsmöglichkeiten: Die Fahrzeuge fahren selbstständig zu einem Ziel ohne dabei Kollisionen mit anderen Fahrzeugen zu vermeiden. Die Fahrzeuge fahren durcheinander durch. Um Kollisionen vorzubeugen, könnten Pfade entweder reserviert werden oder Roboter könnten untereinander Nachrichten austauschen, um festzustellen, ob ein Feld frei ist oder nicht. Um den Kommunikationsaufwand zu verringern, empfiehlt es sich Pfade zu reservieren. Weiterhin sollte es die Möglichkeit geben Roboter zur Laufzeit hinzuzufügen, da ein physisches System auch dynamisch skaliert werden kann, indem ein weiterer Roboter hinzugefügt wird. Durch die Implementierung der genannten Funktionalitäten, wird das System noch weiter an das physische System angeglichen und eine Simulation erzeugt realistischere Ergebnisse.

Bezogen auf das Gesamtsystem bietet es sich an das physische und das virtuelle Teilsystem mithilfe eines hybriden Modus miteinander zu verknüpfen. Ein solcher Hybridmodus bietet viele weitere Möglichkeiten und könnte auf unterschiedliche Weisen realisiert wer-

den. Beispielsweise könnte die Software das physische System steuern, indem generierte Aufträge an das reale System geschickt werden. Auch könnten die Aktionen der Fahrzeuge und Rampen in der Software visualisiert werden. Besteht ein Teil der Visualisierung aus der Darstellung der realen Akteure und der andere aus rein virtuellen Akteuren, so könnten physische und virtuelle Akteure ein Gesamtsystem bilden, dass die Skalierbarkeit des physischen Systems erhöht. Dies setzt jedoch voraus, dass die Akteure aus beiden Systemen in ihren Eigenschaften (Geschwindigkeit etc.) weitestgehend vollständig aneinander angeglichen sind.