Carl von Ossietzky Universität Oldenburg
Fakultät II - Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Untersuchung zur Steigerbarkeit von Flexibilität, Performanz und Erweiterbarkeit von Fahrerlosen Transportsystemen durch den Einsatz dezentraler Steuerungstechniken

Dissertation zur Erlangung des Grades eins Doktors der Ingenieurwissenschaften
vorgelegt von

Christoph Schwarz

07. April 2014

Gutachter: Prof. Dr. Jürgen Sauer, Carl von Ossietzky Universität Oldenburg
Prof. Dr. Sascha Ossowski, Universidad Rey Juan Carlos Madrid
Datum der Disputation: 05.06.2014

Christoph Schwarz
Lerigauweg 84
26131 Oldenburg

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Oldenburg, den 7. April 2014

_____

Christoph Schwarz

Zusammenfassung

In der vorliegenden Arbeit wird untersucht, in wieweit Flexibilität, Performanz und Erweiterbarkeit von Fahrerlosen Transportsystemen durch den Einsatz dezentraler Steuerungstechniken gesteigert werden können. Nach einer Einführung in die Problemstellung und den aktuellen Stand der Technik im Bereich der Fahrerlosen Transportsysteme werden zunächst die wichtigsten Grundlagen für die Arbeit beschrieben. Das sind neben den Fahrerlosen Transportsystemen der Einsatz von dezentralen Steuerungstechniken in der Logistik, Multi-Agenten-Systeme sowie deren Simulation, Verhandlungen und abschließend Graphen sowie der klassische Dijkstra-Algorithmus zum Finden von kürzesten Pfaden in einem Graphen.

Im zweiten Teil werden die Anforderungen an die Arbeit vorgestellt. Diese speisen sich unter anderem aus dem Forschungsprojekt „Dezentrale, agentenbasierte Selbststeuerung von Fahrerlosen Transportsystemen (FTS)“ in dem ein Großteil der Arbeit entstanden ist. Die wichtigste Anforderung ist hier die vollständige Dezentralität. Alle entwickelten Verfahren sollen dezentral umsetzbar sein. Neben dieser Kernanforderung soll sowohl die allgemeine Systemperformanz auf einem möglichst hohen Niveau liegen als auch eine möglichst hohe Flexibilität des Systems beziehungsweise eine möglichst einfache Erweiterbarkeit gewährleistet werden. Diese allgemeinen Anforderungen werden noch für die beiden wichtigsten Teilbereiche der Arbeit, die Auftragsvergabe und das Routing präzisiert. Anschließend werden einige verwandte Arbeiten in den Themenfeldern der Arbeit vorgestellt. Hierzu gehören Arbeiten aus dem Bereich Selbststeuerung in der Intralogistik, beispielsweise der dezentralen Auftragsvergabe in Multi-Agenten-Systemen, sowie dem Routing für Fahrerlose Transportsysteme mit dem Schwerpunkt auf Konflikt vermeidendem Routing.

Anschließend wird der Entwurf für die in der Arbeit vorgestellte Lösung der Problemstellung beschrieben. Der allgemeine Entwurf umfasst die zu erstellenden Komponenten, die prinzipiellen Abläufe innerhalb des Systems, die zu entwickelnden Agenten, die dezentrale Auftragsvergabe sowie die dezentrale Routenplanung. Der prinzipielle Ablauf wird anhand eines neu ins Systems eingehenden Auftrags beschrieben. Dieser wird als Auktion ausgeschrieben und an einen Agenten vergeben, welcher diesen Auftrag dann einplant und schließlich fährt. Zur Realisierung dieses Ablaufs werden im Weiteren verschiedenen Agenten konzipiert. Die wichtigsten sind hier der FTF-Agent, der die Steuerung der Fahrzeuge übernimmt, sowie der OrderDisposer der für die Vergabe der Aufträge zuständig ist. Neben den Agenten werden auch das Kommunikationskonzept für die Agenten und die wichtigsten Nachrichten beschrieben.

Der Entwurf für die Auftragsvergabe beschreibt neben der Auftragseinlastung sowie der Laststruktur verschiedene Auktionsverfahren zur dezentralen Auftragsvergabe. Der allgemeine Ablauf ist hier jeweils gleich: Der zuständige OrderDisposer informiert alle FTF-Agenten über den neuen Auftrag woraufhin diese ein Angebot berechnen und als Antwort zurückschicken. Der Auftrag wird dann an den Agenten, der das beste Gebot abgegeben hat vergeben. Im ersten beschriebenen Verfahren besteht das Angebot hier aus dem frühest möglichen Zeitpunkt zu dem der Agent den Auftrag starten kann. Im zweiten Verfahren besteht das Angebot aus einer gewichteten Summe verschiedener Größen wie dem frühst möglichen Startzeitpunkt, den verwendeten Ressourcen (zum Beispiel Fahrstühle) oder der benötigten Wegstrecke. Im dritten Verfahren wird den Agenten die Möglichkeit eingeräumt, bereits zugewiesene Aufträge zurückzugeben um bessere Angebote für den gerade ausgeschriebenen Auftrag abgeben

zu können. Zurückgegebenen Aufträge werden dann erneut vom System ausgeschrieben. Damit es bei dieser Art der Auftragsvergabe nicht zu einem Liegenbleiben (Verhungern) von Aufträgen kommt, wurde die dynamische Priorität als weitere Einflussgröße in die Angebotsberechnung integriert. Die dynamische Priorität wird durch die Größen Platzmangel an der Station, Dringlichkeit des Auftrags sowie Anzahl der bisherigen Rückgaben bestimmt.

Der Entwurf für das Routing besteht als Basis aus einem Algorithmus für das dezentrale, Konflikt vermeidende Routing. Hierfür sperren Agenten geplante Strecken mit Reservierungen, die sie den anderen Agenten mit der Hilfe von Nachrichten mitteilen. Wenn ein Agent nun seine Route plant, benutzt er den Free-Time-Windows Algorithmus welcher eine konfliktfreie Route, also eine Route, die alle gemachten Reservierungen respektiert, berechnet. Um die Gesamtsystemperformanz zu verbessern wird dann noch das Konzept der Verhandlungen eingeführt. Hier können Agenten über bereits gemachte Reservierungen verhandeln, um ihre eigene Route zu verbessern. Es werden ein allgemeines Konzept sowie zwei Beispielverhandlungen vorgestellt.

Für die Evaluierung der vorgestellten Konzepte wurden mehrere Simulationsexperimente durchgeführt. Die hierfür notwendige Simulation wurde im Rahmen des oben angesprochenen Forschungsprojekts entwickelt. Kern der Simulation ist der Scheduler welcher die Simulation taktet in dem er einmal pro Zeitschritt alle Agenten aktiviert, so dass diese für die Dauer des Zeitschritts handeln können. Das zweite Kernelement der Simulation sind die Agenten, welche, von einigen Hilfsagenten abgesehen, den bereits beschriebenen Agenten aus der Konzeption entsprechen. Für die Modellierung des Szenarios wurden annotierte Graphen verwendet. Wegabschnitte des entsprechenden Layouts werden hier als Kanten und Stationen sowie Kreuzungen als Knoten modelliert. Verfahren zur Konflikterkennung sowie die Implementierung des konzipierten Nachrichtensystems vervollständigen die Beschreibung der Implementation der Simulation.

Zur Evaluation der konzipierten und implementierten dezentralen Steuerung wurden drei Klassen von Experimenten durchgeführt. Mit der ersten Klasse wurde die Systemperformanz mit der Performanz einer in der Industrie eingesetzten zentralen Steuerung verglichen. Hierfür wurde ein Fahrerloses Transportsystem in einer Getränkeabfüllanlage verwendet. Die Originalsimulation sowie die zentrale Steuerung wurde hier von einem Industriepartner aus dem Forschungsprojekt „Dezentrale, agentenbasierte Selbststeuerung von Fahrerlosen Transportsystemen (FTS)“ zur Verfügung gestellt. Es kann gezeigt werden, dass die entwickelte dezentrale Steuerung in diesem Szenario eine bessere Systemperformanz erreicht, als die genutzte zentrale Steuerung. In der zweiten Klasse von Experimenten wurden im gleichen Szenario verschiedene Lastsituationen jeweils mit verschiedenen Parametersets für die Auftragsvergabe simuliert. Es kann gezeigt werden, dass eine geeignete Parameterwahl bei der dezentralen Auftragsvergabe zu einer Anpassung des Systemverhaltens verschiedene Richtungen führen kann. Mit der dritten Klasse von Experimenten sollte die Erweiterbarkeit des Systems gezeigt werden. Hierfür diente eine Fragestellung aus dem Forschungsprojekt CogniLog als Grundlage. Es sollte untersucht werden, inwieweit ein dynamisch rekonfigurierbares Förderband eine lohnende Erweiterung für den Betrieb einer CrossDocking-Halle ist, in der der Warentransport bisher ausschließlich mit Unstetigförderern realisiert wurde. Um die Fragestellung beantworten zu können war die Konzipierung und Implementierung eines Modells für Stetigförderer sowie die Erweiterung der konzipierten Auftragsvergabe notwendig. Es kann gezeigt werden, dass die Fragestellung mit nur geringem Erweiterungsaufwand zu beantworten war.

Abgeschlossen wird die Arbeit durch ein kurzes Fazit sowie ein Ausblick auf mögliche Erwei-

terungen der entwickelten Konzepte der Arbeit.

Abstract

In the present work it is investigated to what extent the flexibility, performance, and extensibility of automated guided vehicle systems can be increased by the use of decentralized control techniques. After an introduction to the problem and the current state of the art in the field of automated guided vehicle systems the most important foundations for the work are described. These are in addition to the automated guided vehicle systems, the use of decentralized control techniques in logistics, multi-agent systems and their simulation, negotiations and finally graphs and the classical Dijkstra algorithm for finding shortest paths in a graph.

In the second part the requirements for the work are presented. A large part of the requirements came from the research project „Dezentrale, agentenbasierte Selbststeuerung von Fahrerlosen Transportsystemen (FTS)“ (distributed, agent-based self-control of automated guided vehicle systems) in which a majority of the work was created. The most important requirement is the complete decentralization. All the developed methods should be implemented locally. In addition to this core requirement the general system performance should be on the highest possible level. Furthermore a high flexibility of the system and a simple as possible extensibility of the system should be archived. These general requirements are then specified for the two most important parts of the work, order disposition and routing. After specifying the requirements some related work in the fields of this work is presented. This includes work in the field of self-control in intra-logistics, for example, the decentralized order disposition in multi-agent systems, as well as the routing for automated guided vehicle systems with a focus on conflict free routing.

Afterwards the approach for solving the described problem is outlined. The overall design for solving this problem includes the basic processes within the system, the developed agents, the decentralized order disposition, and the distributed route planning. The basic procedure is described in terms of a new order which enters the system. This order is disposed to an vehicle with the help of an auction. This vehicle plans steps to fulfil this order and finally fulfils the order. To implement this behavior different agents were designed. The most important agents here are the FTF agents which take control of the vehicles and the OrderDisposer agent who is responsible for the auctions. In addition to the agents the communication concept for the agents and the most important messages are described.

The design for the order disposition consists of the way in which orders are coming into the system and different auction based methods for the actual disposition of the orders. The general process is the same in each case here: The responsible OrderDisposer informs all FTF agents about the new order whereupon these agents calculate an offer and send it back as a response. The order is then awarded to the agent who has made the best bid. In the first method described here the offer is the earliest possible time at which the agent can start the order. In the second method the offer is a weighted sum of different variables like the earliest possible start time, the resources needed (for example, elevators) or the distance which had to be driven in order to fulfill the order. In the third method, the agents are given the opportunity to return orders which were already assigned to them if they can make better offers for the currently proposed order when doing so. Returned orders are then disposed again by the system. In order to prevent orders from getting returned over and over again, the dynamic priority has been integrated as a further influencing factor in the calculation of

offers. The dynamic priority is determined by the buffer size at the station, the urgency of the order and the number of previous returns of the order.

The basis for the design for the routing is an algorithm for decentralized, conflict avoiding routing called free-time-windows algorithm. For this algorithm the agents block all planned routes with reservations which they communicate to the other agents. In order to improve the overall system performance the concept of negotiations is introduced. Here agents can negotiate about already made reservations in order to improve their own route. A general concept and two example negotiations are presented.

In order to evaluate the presented concepts several simulation experiments were performed. The necessary simulation was developed within the above mentioned research project. Core of the simulation is the scheduler which clocks the simulation by activating all agents once per time step. The activated agents can then act for the duration of the time step. The second core element of the simulation are the agents which, apart from some auxiliary agents, meet the agents from the design described above. For the modeling of the scenario annotated graphs were used. Ways in the corresponding layouts are modeled as edges and crossings and stations are modeled as nodes. Methods for conflict detection and the description of the designed message system complete the description of the implementation of the simulation.

To evaluate the designed and implemented decentralized control, three sets of experiments were performed. In the first set of experiments, the system performance was compared to the performance of a central control system used in the industry. For this purpose an automated guided vehicle system installed in in a bottling plant was used. The original simulation as well as the central control were provided by an industrial partner from the above mentioned research project. It could be shown that, in this scenario, the developed decentralized control system achieves better system performance than the central control system. In the second set of experiments, various load situations were simulated with different parameter sets for the order disposition. It could be shown that the choice of parameters for the decentralized order disposition can change the system behavior in different directions. In the third set of experiments, the extensibility of the system was studied. For this study a question from the research project CogniLog was the basis. It should be investigated to what extent a dynamically reconfigurable conveyor belt is a worthwhile extension for the operation of a cross-docking hall in which before the transport of goods has been realized exclusively with fork lifts. In order to answer the question, the design and implementation of a model for continuous conveyors as well as the extension of the designed order disposition was necessary. It could be shown that only a small amount of effort was necessary for these extensions.

The work is completed by a short conclusion and an outlook on possible extensions of the developed concepts.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einleitung

Die Anforderungen an die Intralogistik steigen mit zunehmender Globalisierung und mit der Zunahme des Trends zu kleineren, individuellen Bestellungen. Zunehmend wichtiger werden hierbei die Kriterien Flexibilität, Robustheit sowie Erweiterbarkeit bzw. Veränderbarkeit im Vergleich zu den klassischen Kennzahlen wie Durchsatz oder Durchlaufzeit (vgl. zum Beispiel [GH10] und [Wil13]). Um diesen steigenden Anforderungen gerecht zu werden, wird auch in der Intralogistik verstärkt Vollautomatisierung eingesetzt (vgl. zum Beispiel [Ech11]). Ein Beispiel für voll automatisierte Komponenten in der Intralogistik sind Fahrerlose Transportsysteme (FTS). Die Zahl der jährlich installierten Systeme wächst in den letzten Jahren stetig (vgl. [Ull10]). Insgesamt wurden alleine durch europäische Hersteller 3650 fahrerlose Transportsysteme mit insgesamt 29.500 Fahrzeugen weltweit in Betrieb genommen (siehe [Jun09]).

Fahrerlose Transportsysteme befinden sich seit der Inbetriebnahme des ersten Systems im Jahr 1953 in einem Prozess der stetigen Weiterentwicklung (vgl. zum Beispiel [Ull11]). Gerade in den Bereichen der (Selbst-)Lokalisierung wurden große Fortschritte gemacht (siehe Abschnitt 2.1.1). Im Bereich der Steuerung von Fahrerlosen Transportsystemen wird allerdings weiterhin auf erprobte, zentrale Steuerungen gesetzt. Es wurden bereits einige Forschungsergebnisse veröffentlicht, die sich mit der Dezentralisierung von Teilen der Steuerung beschäftigen. Beispielsweise wurde untersucht, in wie fern sich die Auftragsvergabe dezentralisieren lässt (siehe zum Beispiel [BWHM04]) oder welche Entitäten sich als Agenten modellieren lassen (siehe zum Beispiel [MHH08]). Weiterhin wurden dezentrale Steuerungen für Stetigfördersysteme vorgestellt.

Im Rahmen des Projekts „Dezentrale, agentenbasierte Selbststeuerung von Fahrerlosen Transportsystemen (FTS)“, an dem der Autor der vorliegenden Arbeit mitgewirkt hat, sollten die Möglichkeiten einer komplett dezentralen Steuerung untersucht werden. Das Projekt lief vom 1. Juli 2011 – 30. Juni 2013. Das Projekt wurde mit Mitteln aus dem Haushalt des Bundesministeriums für Wirtschaft und Technologie (BMWi) über die Arbeitsgemeinschaft industrieller Forschungsvereinigungen „Otto von Guericke“ (AiF) e. V. im Auftrag der Bundesvereinigung Logistik (BVL) e.V. gefördert. Neben dem OFFIS - Institut für Informatik aus Oldenburg wurde das Projekt vom IPH - Institut für integrierte Produktion Hannover bearbeitet. Begleitet wurde das Projekt durch einen Ausschuss bestehend aus Vertretern von Firmen aus der Branche der Fahrerlosen Transportsysteme sowie aus Zuliefererbranchen namentlich Götting KG, LinogistiX GmbH, Nuyts GmbH, Simplan Integrations GmbH,

software4production GmbH und STILL GmbH. Die zentrale Frage des Projekts lautetet ob eine komplett dezentrale Steuerung eines Fahrerlosen Transportsystems möglich ist und wenn ja, wie sie im Vergleich zu einer zentralen Steuerung abschneidet. Die wichtigsten Kriterien dieses Vergleiches sind die Anzahl an Fahrerlosen Transportfahrzeugen, die benötigt werden, um die anfallenden Auftragslast zu bewältigen, die durchschnittliche Auftragsdauer, die zurückgelegte Strecke der Fahrzeuge und der Leerfahrtanteil der Fahrzeuge.

Um einen realistischen Vergleich zu ermöglichen wurde vom Industrieausschuss ein Referenzszenario ausgewählt. Es handelt sich dabei um eine Getränkeabfüllanlage in der die anfallenden Transporte vollständig durch ein Fahrerloses Transportsystem realisiert werden sollen. Das Layout der Anlage ist in Abbildung 1.1 dargestellt. Neben dem Szenario wurde dem Projekt auch eine Simulation mit einer erprobten, in der Industrie eingesetzten, zentralen Steuerung überlassen. Hierdurch konnten alle relevanten Kennzahlen des Transportsystems ermittelt und verglichen werden. Ein Großteil der Transportaufträge startet im Szenario in den Bereichen oben links und oben rechts, so dass die Fahrspuren, die zu diesen Bereichen führen stark frequentiert sind. Für das konfliktvermeidende Routing stellt die starke Frequentierung dieser Fahrspuren ein Problem da. In der zentralen Steuerung, die dem Projekt von den Industriepartnern überlassen wurde, wird dieses Problem durch eine sorgfältige Planung und Abstimmung auf die Anzahl der eingesetzten Fahrzeuge (in diesem Fall operieren acht Fahrerlose Transportfahrzeuge im Szenario) sowie der auftretenden Auftragslast gelöst. Diese Lösung funktioniert zwar, wie durch die Installation zahlreicher erfolgreich arbeitender Anlagen durch das Unternehmen gezeigt wurde, sie ist aber sehr unflexibel, wenn sich Systemeigenschaften ändern. So würde bei einer deutlichen Änderung des Auftragsaufkommens oder bei einer Aufstockung der Fahrzeuganzahl eine aufwändige Neuprogrammierung notwendig werden. Ebenfalls nur mit einer Neuprogrammierung möglich wäre das Verändern des gewünschten Systemverhaltens. Soll beispielsweise in einer Phase geringerer Auslastung der Anlage die Priorität anstatt auf möglichst kurzer Durchlaufzeit / möglichst hohem Durchsatz auf einen möglichst Energie effizienten Betrieb gelegt werden, so ist das mit der zentralen Steuerung nicht ohne größere Änderungen möglich.

Aus diesen Einschränkungen der zentralen Steuerung ergaben sich die folgenden Forschungsfragen:

- Wie schneidet eine dezentrale Steuerung in Bezug auf die benötigte Anzahl an Fahrzeugen, die durchschnittliche Auftragsdauer, die zurückgelegte Strecke und den Leerfahrtanteil im Vergleich zu einer zentralen Steuerung ab?
- Kann eine dezentrale Steuerung einfacher auf eine Erweiterung bzw. Veränderung des Szenarios angepasst werden als eine zentrale Steuerung?
- Kann eine dezentrale Steuerung flexibler auf sich ändernde Anforderungen an das Systemverhalten reagieren als eine zentrale Steuerung?

In der vorliegenden Arbeit wird versucht, diese drei Fragen zu beantworten.

1.1 Aufbau der Arbeit

Im nächsten Kapitel werden die Grundlagen für die vorliegende Arbeit beschrieben. Die wichtigste Grundlage ist hier sicher das Fahrerlose Transportsystem als Spezialfall eines Materialflusssystems. Es wird neben einem kurzen Überblick über die Geschichte dieser Systeme

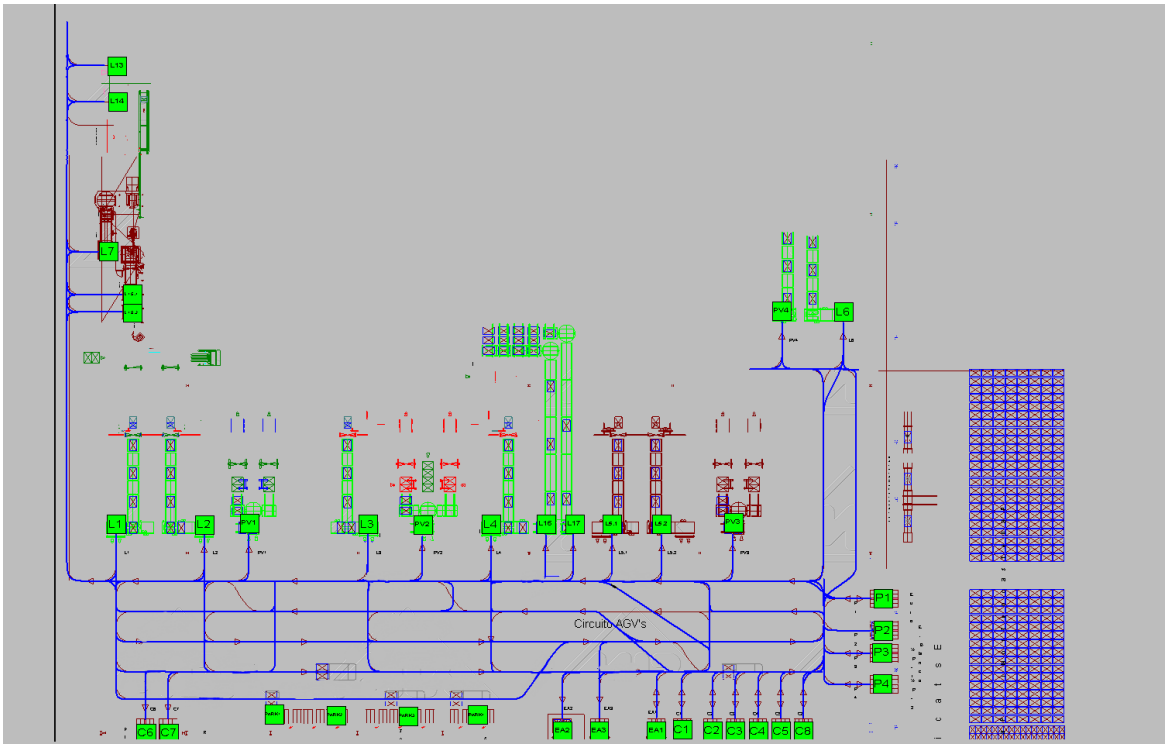


Abbildung 1.1: Layout der Getränkeabfüllanlage

insbesondere der Aufbau und die Funktionsweise sowie die aktuelle Steuerung beschrieben. Weiterhin wird eine kurze Einführung in dezentrale Steuerungskonzepte im Allgemeinen und Agenten im Speziellen gegeben. Nach einem Überblick über klassische Auktionsverfahren sowie über Verhandlungen in Multi Agenten Systemen wird das Kapitel mit einer Einführung in Graphen sowie dem Dijkstra Algorithmus zum Finden von kürzesten Wegen in einem Graphen abgeschlossen.

Im dritten Kapitel der Arbeit werden die genauen Anforderungen an die zu entwickelnde, dezentrale Steuerung und den dezentralen Aufbau herausgearbeitet. Weiterhin wird deutlich gemacht, wie die sich die drei oben formulierten Forschungsfragen beantworten lassen, bzw. was für eine Beantwortung gezeigt werden muss. Weiterhin werden in diesem Kapitel verwandte Arbeiten aus den Themenfeldern, die in dieser Arbeit behandelt werden, vorgestellt.

Im vierten Kapitel wird der Entwurf präsentiert. Es wird hier sowohl auf den allgemeinen Entwurf zum Aufbau eines dezentral gesteuerten Fahrerlosen Transportsystems als auch auf die entwickelten Konzepte für die Umsetzung mit Hilfe von Agenten eingegangen. Weiterhin werden die entwickelten Verfahren für die Routenplanung und die Auftragsvergabe beschrieben. Am Ende des Kapitels wird die Konzeption der Simulation, mit der die entwickelten Verfahren evaluiert werden sollen beschrieben.

Im fünften Kapitel wird kurz auf die Implementierung der im vierten Kapitel vorgestellten Konzepte eingegangen. Da die Implementierung sehr nah am Entwurf liegt, wird hier vor allem auf einige relevante Details eingegangen.

Im sechsten Kapitel werden die drei Forschungsfragen beantwortet. Zu Beantwortung der Frage, welche logistischen Kennzahlen (Anzahl benötigter Fahrzeuge, Auftragsdauer,

zurückgelegte Strecke, Leerfahrtanteil) sich für eine dezentrale Steuerung im Vergleich zu einer zentralen Steuerung ergeben, wurden Experimente durchgeführt, in welchen die entwickelte dezentrale Steuerung mit der vom Industrieausschuss überlassenen, zentralen Steuerung verglichen werden. Für die Untersuchung der Flexibilität der dezentralen Steuerung wurde eine Reihe von Experimenten mit verschiedener Auftragseinlastung und unterschiedlichen Parameterkombinationen durchgeführt und es wurde untersucht, ob sich das Systemverhalten durch die Anpassung der Parameter in die gewünschte Richtung verändern lässt. Um herauszufinden, wie leicht das System erweiterbar bzw. veränderbar ist, wurden zum einen die oben beschriebenen Experimente mit unterschiedlichen Anzahlen an Transportfahrzeugen durchgeführt, zum anderen wurde das System um eine neue Komponente (einen rekonfigurierbaren Stetigförderer) erweitert.

Das abschließende siebte Kapitel gibt eine kurze Zusammenfassung über die erzielten Ergebnisse und die Erkenntnisse, die sich daraus ergeben haben. Außerdem stellt es offene Anknüpfungspunkte sowie Erweiterungsmöglichkeiten vor, die sich aus den Ergebnissen und Erkenntnissen dieser Arbeit ergeben.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen vorgestellt, die für das Verständnis der vorliegenden Arbeit notwendig sind. Zunächst wird eine Einführung in Materialflusssysteme im Allgemeinen und in Fahrerlose Transportsysteme im Speziellen gegeben. Anschließend wird der Einsatz dezentraler Techniken in der Logistik beschrieben. Im weiteren werden die wichtigsten Grundlagen aus dem Bereich der Agenten und der Multi-Agenten-Simulation vorgestellt. Nach einem Überblick über klassische Auktionsverfahren sowie über Verhandlungen in Multi Agenten Systemen wird das Kapitel mit einer Einführung in Graphen sowie dem Dijkstra Algorithmus zum Finden von kürzesten Wegen in einem Graphen abgeschlossen.

2.1 Materialflusssysteme

Materialflusssysteme werden zum Transport von Gütern sowohl in Fertigungsstätten als auch in Lager- und Distributionszentren eingesetzt. Häufig werden die Waren dabei in Form von Stückgütern, genannt Kollo (vom italienischen collo, im Englischen trading unit) transportiert (siehe [Hei11]). Der Transport der Güter erfolgt dabei entweder durch Stetigförderer (z.B. Bandförderer, Schleppkette oder Kreisförderer) oder durch Unstetigförderer (z.B. Kran, Gabelstapler, FTF). Stetigförderer sind dabei ortsgebunden und wenig flexibel. Sie eigenen sich gut für großes und konstantes Transportaufkommen. Unstetigförderer sind flexibler Einsetzbar und eigenen sich für kleines und mittleres, sowie sich oft änderndes Transportaufkommen. Ausführliche Beschreibungen der Unterschiede sowie Praxisbeispiele und typische Kennzahlen finden sich z.B. in [Mar06].

Klassische Materialflusssysteme sind dabei in der Regel wenig flexibel:

> Heute eingesetzte Systeme sind sowohl mechanisch als auch steuerungstechnisch und organisatorisch meist auf spezifische Anforderungen ausgelegt und für diese optimiert. (siehe [Hei11, Seite 7])

In den letzten Jahren lässt sich aber ein Trend zu flexibleren Systemen feststellen. Systeme sollen wandelbarer werden und sich veränderten Umständen schnell anpassen können. In [WHG06] und [Hei06] werden als Kriterien für solche Eigenschaften Fördergutflexibilität, Layoutflexibilität und Durchsatzflexiblität genannt.

2.1.1 Fahrerlose Transportsysteme (FTS)

Die wichtigsten Definitionen für diesen Abschnitt, die des Fahrerlosen Transportfahrzeugs sowie die des Fahrerlosen Transportsystems stammen aus der VDI[1]-Richtlinie 2510 - "Fahrerlose Transportsysteme":

Definition 2.1. Fahrerloses Transportfahrzeug
Ein Fahrerloses Transportfahrzeuge ist ein flurgebundendes, automatische Fahrzeuge mit eigenem Fahrantrieb welches zum Transport von Gütern aber nicht zum Transport von Personen eingesetzt werden kann.

Definition 2.2. Fahrerloses Transportsystem
Ein Fahrerloses Transportsystem ist ein System aus mehreren Fahrerlosen Transportfahrzeugen, welches dem innerbetrieblichen, flurgebundenden Materialtransport dient.

Ein Fahrerloses Transportfahrzeug zum Transport von Paletten ist in Abbildung 2.1 abgebildet.

Abbildung 2.1: Ein Fahrerloses Transportfahrzeug (Quelle: Carmenter[2])

[1] Verein deutscher Ingenieure www.vdi.de
[2] Veröffentlicht unter CC-Lizenz: `http://commons.wikimedia.org/wiki/File:AGVs_amarillos.jpg`

Fahrerlose Transportsysteme werden für verschiedene Transportaufgaben eingesetzt. Die Abbildungen 2.2 und 2.3 zeigen Fahrerlose Transportfahrzeuge für verschiedene Einsatzzwecke. Weitere Einsätze finden sich zum Beispiel in Krankenhäusern oder auf Flughäfen.

(a) FTF als Unterfahrschlepper (Quelle: Joerg Hammes)

(b) FTF mit Gurtförderern (Quelle: Thomas Albrecht, Fa. FROG)

Abbildung 2.2: Verschiedene FTF I[3]

(a) FTF als mobile Montageplattform (Quelle: Fa. Bleichert)

(b) Outdoor FTF zum Transport von Betonsteinen (Quelle: Betonsteinwerk Lintel)

Abbildung 2.3: Verschiedene FTF II[4]

Die allgemeinen Eigenschaften hängen stark vom Einsatzzweck ab. Die Tabelle 2.1 zeigt die Bandbreite wichtiger Eigenschaften.

Um 1953/54 wurde von der amerikanischen Firma Barrett Vehicle Systems erstmals ein automatischer Schlepper präsentiert. Das Fahrzeug war in der Lage, selbsttätig einem weißen, auf den Boden aufgebrachten Farbstreifen zu folgen. Seit Ende der 1960er Jahre wurden Fahrzeuge speziell für den Einsatz als Fahrerloses Transportsystem entwickelt. Getrieben durch die Anforderungen der unterschiedlichen Domänen wurden Fortschritte sowohl in der Technik als

[3] a): `http://commons.wikimedia.org/wiki/File:Carobot_FTS.jpg` b): Veröffentlicht unter CC-Lizenz: `http://commons.wikimedia.org/wiki/File:FTF_fuer_2_KLT.jpg`

[4] a): Veröffentlicht unter CC-Lizenz: `http://commons.wikimedia.org/wiki/File:FTF_Montarail1.jpg` b): Veröffentlicht unter CC-Lizenz: `http://commons.wikimedia.org/wiki/File:FTF_Steintransport.jpg`

Eigenschaft	typische Werte
Anzahl FTF je System	ein bis mehrere hundert
Tragfähigkeit eines FTF	wenige Kilogramm bis über 50 t
Fahrgeschwindigkeit	typischerweise ca. 1 m/s, aber auch abweichende Werte möglich; die Maximalgeschwindigkeit ist durch das Bremsvermögen begrenzt
Fahrkurslänge	wenige Meter bis über 10 km
Anzahl der Lastwechselstationen	unbegrenzt
Anlagensteuerung	manuell bis vollautomatisch, stand-alone oder in komplexen Materialflusssysteme integriert
Einsatzdauer	sporadisch bis "rund um die Uhr"
Antriebskonzept	elektromotorisch, mit oder ohne Batterie; verbrennungsmotorisch

Tabelle 2.1: Eigenschaften von FTS - nach [Ull10]

auch der Steuerung erzielt. (vgl. [Ull10]) Abbildung 2.4 zeigt einige Entwicklungsmeilensteine von 1950 bis 2000 in den Bereichen Fahrzeugtechnik, Anwendungsgebiete und Steuerungstechnik.

Der allgemeine Aufbau für die Steuerung von Fahrerlosen Transportsystemen ist in Funktionsblöcke unterteilt (vgl. zum Beispiel [Ull10]). Abbildung 2.5 stellt die vorhandenen Blöcke dar. Die Blöcke Bahn- und Achsregelung und Lastaufnahmemittelsteuerung sowie die entsprechenden Unterblöcke stehen hier für die Hardwarefunktionen der Fahrzeuge, genauer für die Aktorik und Sensorik die für die Bewegung und die Lastauf- und abnahme zuständig sind. Die Bedienung und Fehleranzeige sowie Diagnose und Service Blöcke bilden die Schnittstellen zu menschlichen Bedienern im System. Die Ablaufsteuerung ist der zentrale Funktionsblock eines Fahrerlosen Transportfahrzeugs welcher die Funktionen des Fahrzeugs kapselt. Die Datenübertragung ist für die Übertragung von Daten sowohl zu menschlichen Bedienern als auch zur Leitsteuerung beziehungsweise gegebenenfalls zu anderen Fahrzeugen oder Systemteilnehmern zuständig. Die gestrichelten Blöcke Auftragsverwaltung und Wegsteuerung sind für die logische Planung der Fahrzeugaktionen zuständig. Diese Funktionen können Teil des Fahrzeugs sein (dann sind die Blöcke vorhanden), sie können aber auch vollständig in die Leitsteuerung ausgelagert sein (dann sind die Blöcke nicht vorhanden).

Für die Umsetzung der einzelnen Blöcke gibt es in der Regel verschiedene technische Möglichkeiten. Für die Datenübertragung gibt es beispielsweise die induktive Übertragung, Infrarot, Schmalbandfunk oder Breitbandfunk (WLAN).

Die Orientierung kann ebenfalls durch verschiedene Verfahren erreicht werden. Mögliche Navigationsverfahren wären zum Beispiel:

- Odometrie
- Spurführung mit kontinuierlicher Leitlinie
- Rasternavigation

[5] Gemeinfrei: `http://de.wikipedia.org/w/index.php?title=Datei:Geschichte_der_FTS.png`

[6] Gemeinfrei: `http://de.wikipedia.org/w/index.php?title=Datei:Funktionsbl\protect\unhbox\voidb@x\bgroup\U@D1ex{\setbox\z@\hbox{\char127}\dimen@-.45ex\advance\dimen@\ht\z@}\accent127\fontdimen5\font\U@Do\egroupcke_FTF-Steuerung.png`

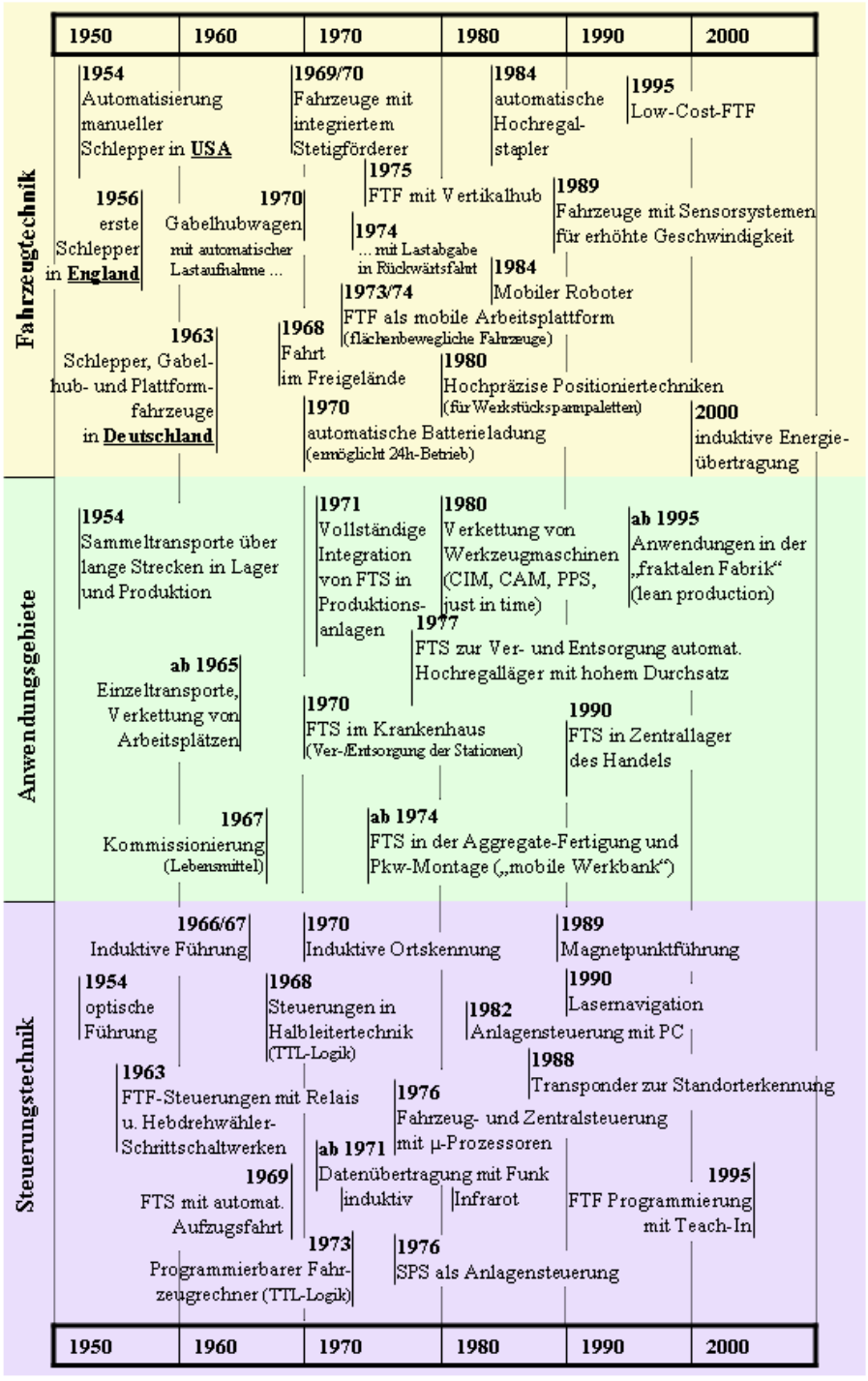


Abbildung 2.4: Geschichte der FTS (Quelle: Peter Gunsser[5])

- Lasernavigation

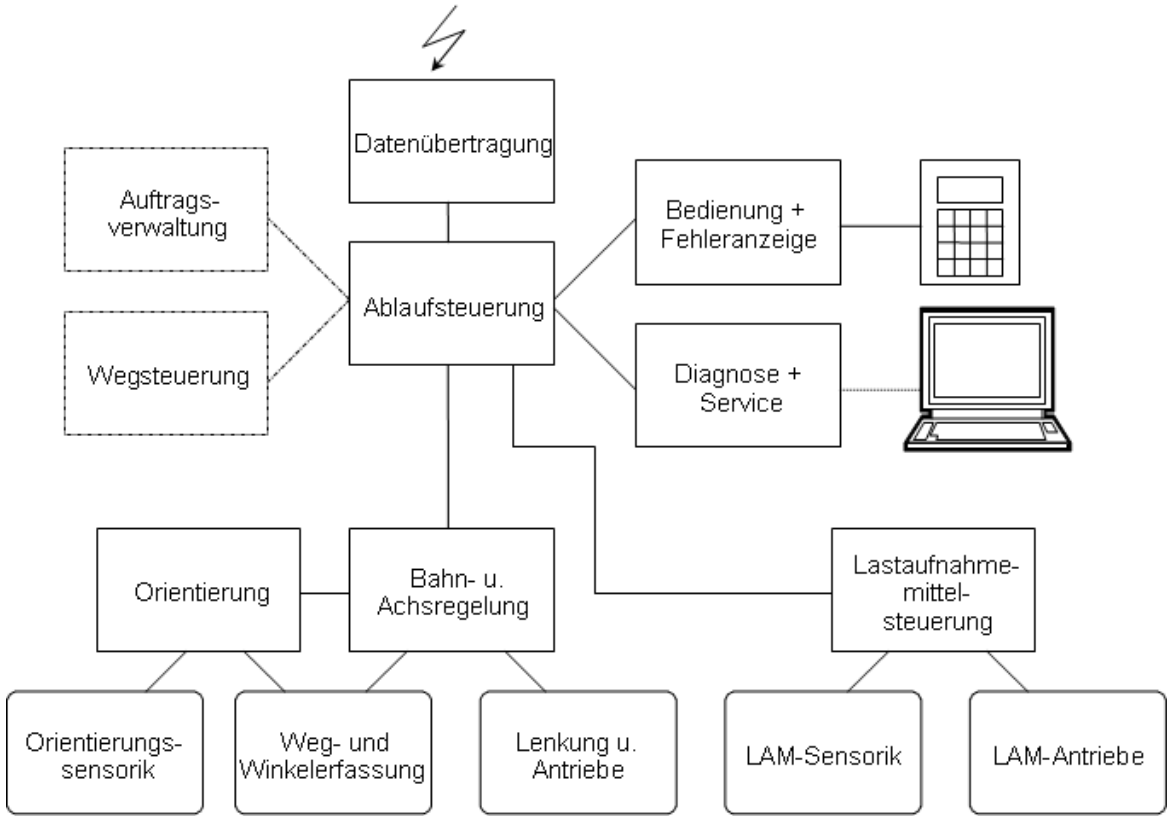


Abbildung 2.5: Funktionsblöcke der FTF-Steuerung (Quelle: Thomas Albrecht[6])

- GPS-Varianten

Bei der Odometrie wird versucht die aktuelle Position ausschließlich aus dem Wissen der Startposition sowie aus der gemessenen Bewegung zu Berechnen. Die Richtung wird hierbei in der Regel durch Messung des Lenkwinkels, die zurückgelegte Strecke durch die Messung der Radumdrehungen bestimmt. Das Verfahren gilt allgemein als relativ ungenau (vgl. [Ull10]).

Bei der Spurführung mit kontinuierlicher Leitlinie wird eine Leitlinie fest in den Boden integriert. Die Fahrzeuge folgen dieser Leitlinie dann mit Hilfe von Sensoren. In der Leitlinie integriert sind spezielle Referenzpunkte, an Hand derer das Fahrzeug seine Position bestimmen kann. Für die konkrete Ausgestaltung dieser Technik gibt es mehrere Möglichkeiten. Die Leitlinien können beispielsweise in Form von Magnetstreifen, weißen Farbstreifen oder elektrischen Feldern ausgebracht werden. Die Sensorik besteht dementsprechend entweder aus Magnetsensoren, Kameras oder Antennen. Diese Technik ist erprobt und in der Regel zuverlässig. Allerdings ist es sehr aufwändig, Änderungen am Fahrkurs vorzunehmen, da das gleichbedeutend damit ist, neue Leitlinien anzubringen (vgl. [Ull10]).

Bei der Rasternavigation wird auf eine Kombination aus Odometrie und Leitlinie gesetzt. Anstatt eine kontinuierliche Leitlinie aufzubringen, wird das Layout "gerastert". Hierfür wird in regelmäßigen Abständen ein Stützpunkt angebracht. Für die Anbringung gibt es erneut die Möglichkeiten Magneten, elektrische Felder oder markante, optische Merkmale (Schachbrettmuster) zu nutzen. Das Fahrzeug orientiert sich nun maßgeblich mit Hilfe der Odometrie und überprüft seine Position jedes mal, wenn es einen Stützpunkt erreicht. Je weiter die einzelnen Stützpunkte auseinander liegen, desto genauer muss folglich die Odometrie kalibriert sein.

Die Rasternavigation ist ebenfalls seit Jahren erprobt und bewährt. (vgl. [Ull10]).

Bei der Lasernavigation ist auf jedem Fahrzeug ein Laserscanner angebracht. Weiterhin sind in dem Layout (der Halle) eine beliebige Anzahl an Reflektoren angebracht deren genaue Position dem Fahrzeug bekannt sind. Das Fahrzeug misst nun die Entfernung zu drei dieser Reflektoren und kann daraus durch das Lösen eines Gleichungssystems mit drei Unbekannten seine eigene Postion bestimmen. Das Verfahren ist in Abbildung 2.6 angedeutet.

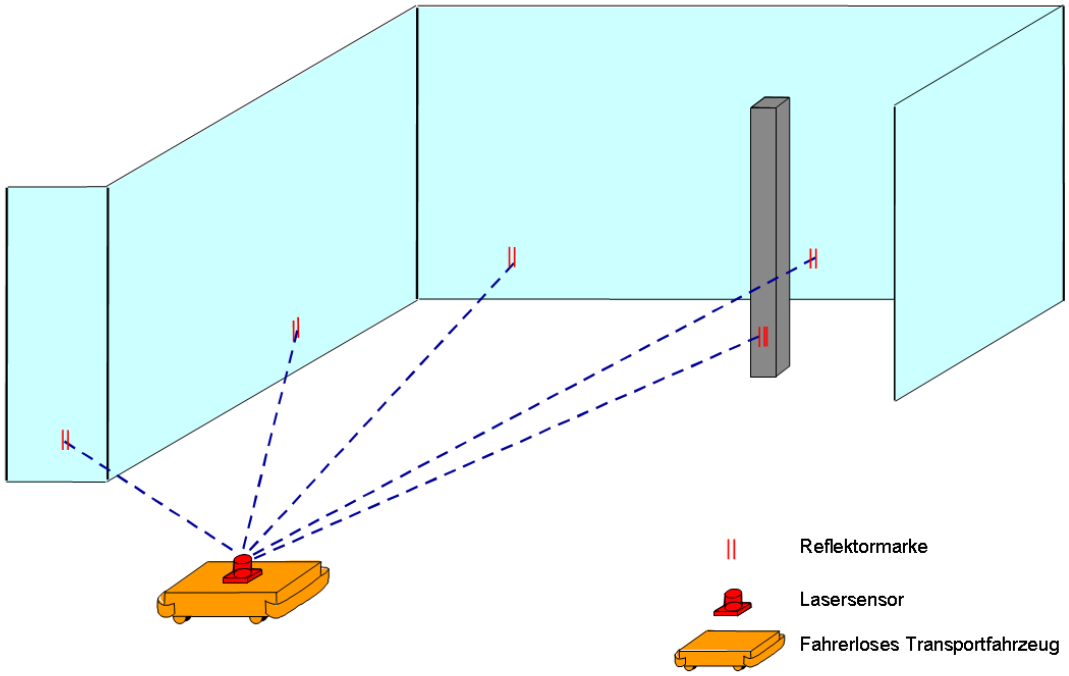


Abbildung 2.6: Funktionsprinzip der Lasernavigation (Quelle: Thomas Albrecht[7])

Die Navigation mit GPS wird oft im Outdoor Bereich angewandt. Hier kommt insbesondere dGPS (Differential Global Positioning System) zum Einsatz. Bei diesem Verfahren werden Referenzstationen aufgebaut deren genaue Position bekannt ist. Die notwendigen Korrekturen die sich aus der per GPS bestimmten Position und der tatsächlichen Position ergeben werden per Funk an die Fahrzeuge gesendet, die diese Korrekturen auf die von ihnen bestimmten Positionen anwenden. Durch dieses Verfahren kann die Bestimmungsgenauigkeit von ca. 10 Metern auf ca. 10 Zentimeter verbessert werden (siehe zum Beispiel [O'C98]). Allerdings ist der Aufbau einer Referenzstation kostenintensiv.

Fahrerlose Transportfahrzeuge unterscheiden sich auf Grund verschiedener Fahrwerkskonzepte hinsichtlich ihrer möglichen Bewegungen. Entscheidend hierfür ist die Anzahl der Freiheitsgrade des jeweiligen Fahrwerks. In der Folge können die Fahrzeuge entweder linienbeweglich oder flächenbeweglich sein. Linienbewegliche Fahrzeuge haben einen deutlich höheren Kurvenradius und sind nicht in der Lage, im Stand zu drehen. Flächenbewegliche Fahrzeuge können sich auch während sie stehen drehen, erfordern aber einen erhöhten Konstruktionsaufwand und sind daher teurer. Abbildung 2.7 zeigt Beispiele von Fahrwerken, wie sie in Fahrerlosen Transportsystemen zum Einsatz kommen.

[7] Gemeinfrei: `http://de.wikipedia.org/w/index.php?title=Datei:Lasernavigation.png`

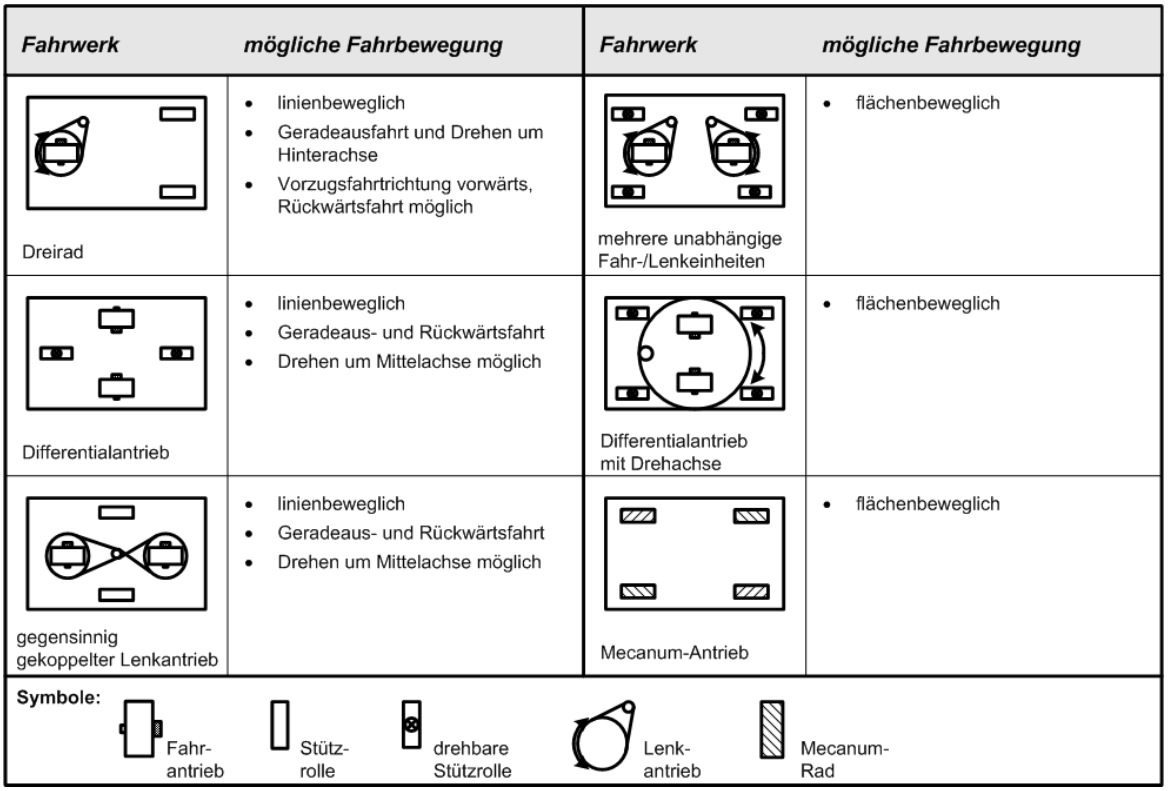


Abbildung 2.7: Verschiedene Fahrwerkstypen von FTF (Quelle: Thomas Albrecht[8])

Auch für die Lenkung gibt es mehrere Möglichkeiten. Eine Möglichkeit ist die klassische Lenkung wie von Automobilen her bekannt, bei der, entweder elektromechanisch oder hydraulisch, ein bestimmter Lenkwinkel eingestellt wird. Weiterhin kann die Lenkung mittels Differentiallenkung erfolgen. Hier drehen die Antriebsräder auf der einen Seite des Fahrzeugs schneller als auf der anderen was zu einer entsprechenden Einlenkung in Richtung der langsamer drehenden Räder führt.

Bei der Lastmittelaufnahme (LAM) wird zwischen passiven und aktiven LAM unterschieden. Passive LAM bieten nur eine Abstellfläche und möglicherweise noch Maßnahmen zur Lastsicherung wie Seile, Ketten, Netze oder ähnliches. Aktive LAM verfügen über eigenen Sensorik und Aktorik mit der die Last erkannt und aufgenommen wird. Beispiele hierfür wären Rollenbahnen, Gurtförderer oder eine Gabel.

Die Bedienung und Fehleranzeige einzelner Fahrzeuge ist stark abhängig vom Einsatzszenario. Abbildung 2.8 zeigt einige Bedienelemente für die Steuerung von Fahrerlosen Transportfahrzeugen am Fraunhofer-Institut für Materialfluss und Logistik (IML)[9] in Dortmund.

Für die Steuerung des Gesamtsystems ist die sogenannte FTS-Leitsteuerung zuständig. Laut der Definition der VDI-Richtlinie 4451, Blatt 7 "Kompatibilität von Fahrerlosen Transportsystemen, Leitsteuerung für FTS" besteht eine FTS-Leitsteuerung 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 Integra-

[8] Gemeinfrei: `http://de.wikipedia.org/w/index.php?title=Datei:Fahrwerke.png`
[9] http://www.iml.fraunhofer.de/
[10] Veröffentlicht unter CC-Lizenz: `http://commons.wikimedia.org/wiki/File:Bedienelemente.png`

Abbildung 2.8: FTF Bedienelemente (Quelle: Thomas Albrecht, Fraunhofer IML, Dortmund[10])

tion des FTS in die innerbetrieblichen Abläufe. Eine solche Leitsteuerung ist üblicherweise in Funktionsblöcke, ähnlich wie die der Fahrerlosen Transportfahrzeuge, unterteilt. Abbildung 2.9 stellt den Aufbau einer Leitsteuerung dar. Je nach Größe und Komplexität der Anlage können einzelne Blöcke wegfallen oder mit anderen zusammengelegt werden. Die Benutzerschnittstelle ist zuständig für die Kommunikation der Leitsteuerung mit übergeordneten Systemen sowie menschlichen Benutzern. In der internen Materialflusssteuerung wird die Planung und Logik des Systems umgesetzt. Hier wird der gesamte Ablauf der Materialbewegungen geplant. In der Transportauftragsabwicklung werden zum einen die einzelnen Transportaufträge gespeichert (auch nach dem sie abgearbeitet wurden), zum anderen erfolgt die Zuordnung der Aufträge an die Fahrzeuge (Fahrzeugdisposition). Eine der wichtigste Aufgabe der Leitsteuerung ist die Koordination der Fahrerlosen Transportfahrzeuge. Sie findet im Block Fahrauftragsabwicklung statt. In der Regel basiert die Fahrauftragsabwicklung auf auf einer Verkehrsleitsteuerung. Die Verkehrsleitsteuerung sorgt in Mehrfahrzeuganlagen für eine sichere Verkehrsregelung, insbesondere im Bereich von Kreuzungen und Einmündungen. In der Regel wird hier mit Blockbereichen gearbeitet. Wenn ein Fahrzeug in einen solchen Blockbereich (zum Beispiel eine Kreuzung oder einen engen Weg) einfährt, so wird dieser für andere Fahrzeuge gesperrt (vgl. zum Beispiel [Ull10]). Wie schon bei den Funktionsblöcken der Fahrerlosen Transportfahrzeuge angesprochen, ist die genaue Zuständigkeitsaufteilung für die Wegsteuerung und die Auftragsvergabe zwischen Fahrzeugen und Leitsteuerung nicht in jedem System gleich. Die Wegsteuerung kann beispielsweise vollständig bei der Leitsteuerung liegen oder die Leitsteuerung kann nur bestimmte Zielpunkte vorgeben und die Feinplanung

der Wegsteuerung den Fahrzeuge überlassen. Die Service-Funktionen stellen schließlich Statistiken sowie Visualisierungen und Diagnosefunktionen für die Anlage zur Verfügung. Einige Leitsteuerungen erlauben auch die Simulation des Systems oder die Modellierung eines neuen Systembestandteils.

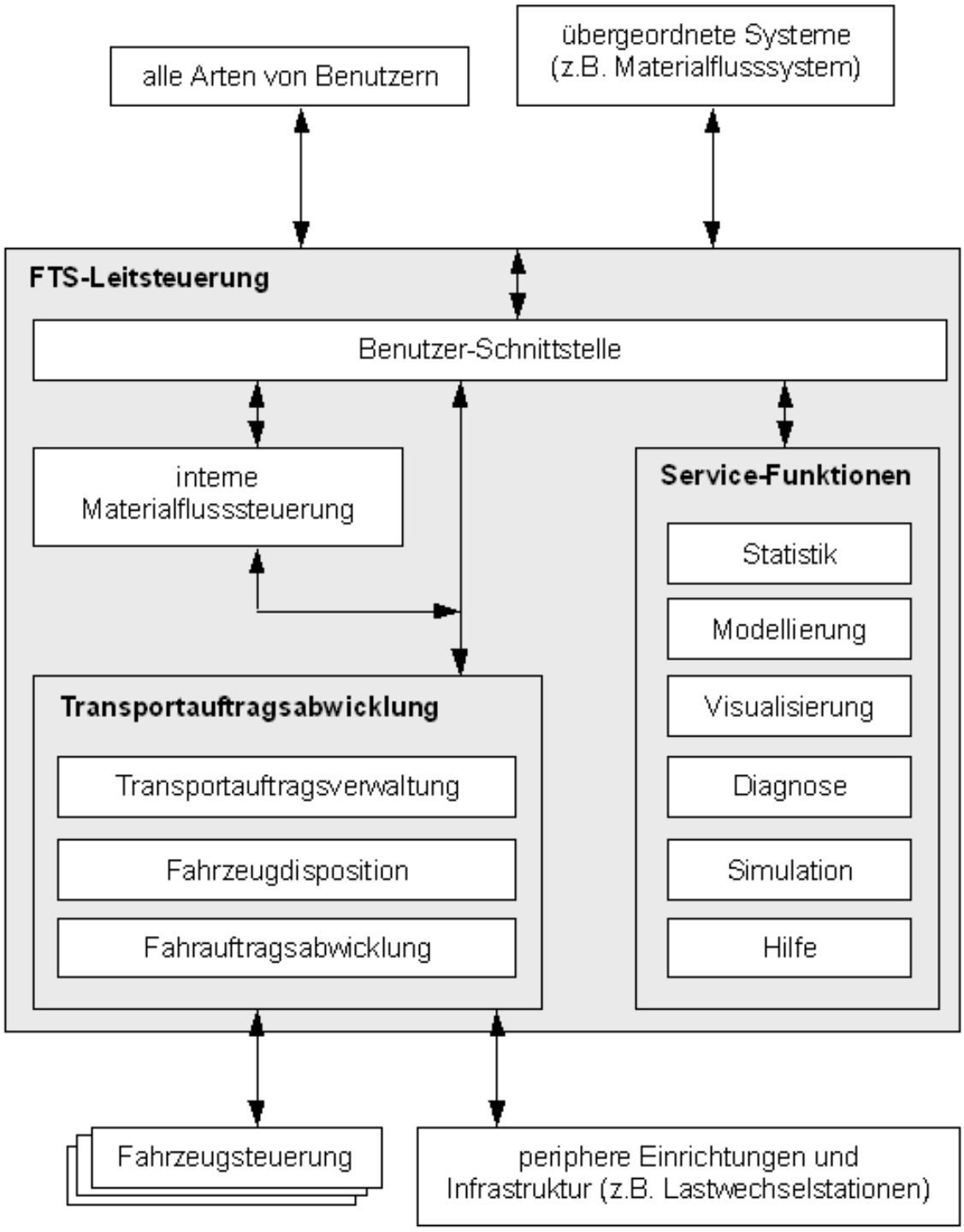


Abbildung 2.9: FTS-Leitsteuerung (Quelle: Thomas Albrecht [11])

2.2 Dezentrale Steuerung in der Logistik

Dezentrale Steuerung ist ein Thema, das in vielen Forschungsgebieten untersucht wird. Die Bedeutung unterscheidet sich hier naturgemäß je nach Domäne. So wird in den Sozial- oder Systemwissenschaften von einer Dezentralisierung der Steuerung gesprochen, wenn politische oder wirtschaftliche Macht auf mehrere Entitäten aufgeteilt wird. Aufgekommen ist der Begriff erstmals in den 1820er Jahren um die politischen Strukturveränderungen im Frankreich nach der Revolution zu beschreiben (siehe [Sch07]). Im frühen 20. Jahrhundert wurde das Konzept der dezentralen Steuerung von Leuten gefordert, die die größtenteils zentral gesteuerte Großindustrie für das Zerstören mittelständischer Kleinunternehmer verantwortlich machten (siehe [Loo04]). Aus diesen Theorien heraus hielt das Konzept der dezentralen Steuerung Einzug in die Ökonomie. Hier wurde unter anderem diskutiert, für welche Wirtschaftssysteme eine dezentrale Steuerung Vorteile bringen kann (siehe zum Beispiel [Dev10]).

Dezentrale Steuerungen haben in den letzten Jahren auch in der Anlagenautomatisierung an Bedeutung gewonnen (siehe zum Beispiel [GH13]). Hier ist das Ziel oft die Komplexität der Programmierung zu reduzieren. Anstatt ein einziges Programm welches auf einem zentralen Steuerrechner läuft schreiben zu müssen, werden kleinere Programme geschrieben, die für die Steuerung einzelner Elemente (zum Beispiel einer Maschine oder eines Förderbandes) zuständig sind. Die Kommunikation zwischen diesen Teilprogrammen läuft üblicherweise über ein BUS (Binary Unit System).

Auch in der Logistik wurde in den vergangenen Jahren verstärkt im Bereich der dezentralen Steuerung geforscht. Der Grund hierfür ist laut Bernd Scholz-Reiter et.all. "Künftig werden Aspekte wie Flexibilität, Adaptivität und Proaktivität im Vordergrund stehen. Dies ist nur durch Dezentralisierung und Autonomie der logistischen Entscheidungsprozesse zu erreichen."(siehe [SRFR$^+$05]).

Bei einer dezentralen Steuerung ist die dezentrale Auftragsvergabe eine wichtige Aufgabe in nahezu jedem System (siehe z.B. [Fer98]). Durch die Auftragsvergabe werden die im System auftretenden Aufgaben auf die verschiedenen Entitäten, die diese lösen können, verteilt. Bei der dezentralen Auftragsvergabe gibt es hierbei keine zentrale Stelle, die diese Verteilung übernehmen würde, sondern die autonomen Entitäten müssen dieses Problem selbstständig lösen. Eine dezentrale Auftragsvergabe ist dynamisch, wenn die Zuordnungen von Aufgaben zu handelnden Entitäten zurückgenommen werden können, die Aufgaben also neu zugeordnet werden können.

Die dezentrale Auftragsvergabe wird oft durch Heuristiken oder durch Ansätze aus der Spieltheorie realisiert (siehe z.B. [JFL$^+$01]). Diese Techniken setzen normalerweise voraus, dass die Menge möglicher Aufgaben, die Menge der handelnden Entitäten und die Resultate von jeder möglichen Zuordnung im voraus bekannt sind. Da dies im Fall der dezentralen FTS Steuerung nicht der Fall ist, sind diese Techniken für die vorliegende Arbeit nicht von Interesse.

Eine weitere Klasse von Verfahren sind die auktionsbasierte Mechanismen. Unter den meisten bekannten und verbreiteten ist das Contract Net Protokoll (CNET) welches von Smith vorgestellt wurde (siehe [Smi80]). Beim CNET Protokoll nehmen einige Agenten (siehe Abschnitt 2.3) die Rolle von Managern (oder Initiatoren) und andere die Rolle von Bietern (oder Teilnehmern) ein. Abbildung 2.10 zeigt den Vorgang des FIPA[12] -CNET Protokolls. Im Fall

[11] Gemeinfrei: `http://de.wikipedia.org/w/index.php?title=Datei:FTS-Leitsteuerung.png`

[12] Foundation for Intelligent Physical Agents: http://www.fipa.org/

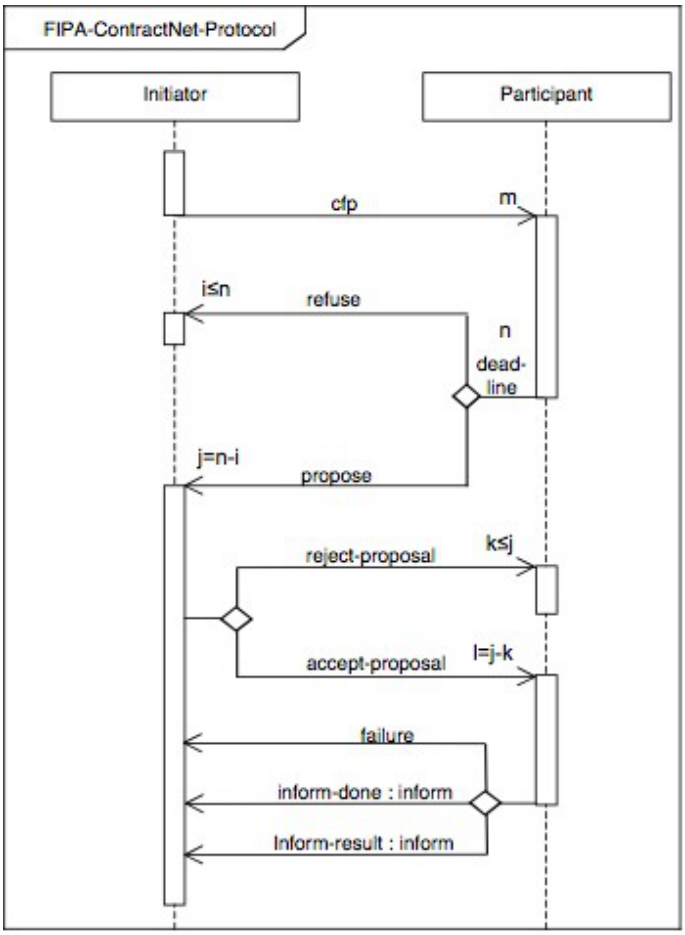


Abbildung 2.10: FIPA-ContractNet-Protocol (Quelle: [FIP01])

eines FTS-Systems könnten die Manager zum Beispiel Ladestationen, Batteriestationen, Fertigungsanalgen oder ähnliches sein. Die Fahrzeuge würden die Rolle der Bieter übernehmen. Wenn eine neue Aufgabe (zum Beispiel der Transport eines Gutes) erzeugt wird, informiert der zuständige Manager die Bieter über diese Aufgabe woraufhin diese ein Gebot abgeben. Dieses Gebot kann aus der Zeit, die sie für die Erfüllung der Aufgabe benötigen würden, der Menge an Energie, die diese Aufgabe verbrauchen würde oder aus der Anzahl der zu fahrenden Kilometer bestehen. Nach einer definierten Wartezeit weist der Manager dann die Aufgabe dem Fahrzeug zu, das das beste Gebot abgeben hat. Das Contract Net Protokoll setzt das Prinzip der Verhandlung (siehe Abschnitt 2.4) für Agenten um.

Weitere Verfahren, die speziell in letzter Zeit im Bereich fahrerloser Transportsysteme entwickelt wurden, werden in Abschnitt 3.3.2 vorgestellt.

2.3 Agenten

Die am weitesten verbreitete Technik zur Dezentralisierung der Steuerung in der Logistik und Intralogistik ist die der Agenten (siehe zum Beispiel [GG05], [NK12], [KG13] oder [Tra07]).

Es gibt verschiedene, ähnliche Definitionen des Begriffs „Agent". Eine der generellsten und am meisten verbreiteten Definitionen ist die von Wooldridge:

> Ein Agent ist ein Computersystem das sich in einer bestimmten Umgebung befindet und welches fähig ist, eigenständige Aktionen in dieser Umgebung durchzuführen, um seine (vorgegebenen) Ziele zu erreichen. (vgl. [Woo02])

Die Entwicklung zu Agenten wurde aus vielen verschiedenen Bereichen heraus getrieben. Abbildung 2.11 zeigt einige der wichtigsten Wissenschaften und deren jeweiligen Einfluss auf die Agenten auf. Nicht jeder Sofwareagent muss hier jede Eigenschaft aufweisen. So gibt es Agenten die nicht lernfähig sind oder solche die keine Kommunikation beherrschen. Wenn ein Agent jedoch eine der abgebildeten Eigenschaften besitzt, so stammen die Grundlagen dafür oft aus der angegebenen Wissenschaft (vgl. [BZW97]).

Der prinzipielle Aufbau eines Agenten besteht in der Regel aus den vier Komponenten interner Zustand, Weltbild, Ziele und Plan wie in Abbildung 2.12 dargestellt wird. Die Ziele geben an, was der Agent erreichen möchte. Im Fall eines Agenten, der ein Fahrerloses Transportfahrzeug steuert, könnte das zum Beispiel ein möglichst effizienter Transport von Transportgütern sein. Der interne Zustand gibt an, was der Agent über sich selbst weiß. Das könnte zum Beispiel die aktuelle Position des Fahrzeugs, der Batterieladezustand oder die ihm aktuell zugewiesenen Transportaufträge sein. Das Weltbild gibt an, was der Agent über seine Umgebung zu wissen glaubt. Das könnte zum Beispiel die Position anderer Fahrzeuge im System, das Hallenlayout oder die Belegungsinformationen der Ressourcen im System sein. Das Weltbild kann durchaus unvollständig oder unkorrekt sein. So kann das Weltbild eines Agenten beispielsweise von Sensorauswertungen oder empfangenen Nachrichten abhängen. Ist diese Sensorauswertung fehlerhaft, beispielsweise auf Grund einer verdreckten Kamera oder eines verdreckten Laserscanners, oder wenn er für ihn bestimmte Nachrichten nicht empfangen hat, beispielsweise auf Grund von einer Störung im WLAN, können dem Agenten relevante Informationen für das Weltbild fehlen.

Aus den drei Komponenten interner Zustand, Weltbild und Ziele erzeugt der Agent einen Plan. Ein Plan ist in der Regel eine Kette von Aktionen mit denen der Agent sein Ziel erreichen

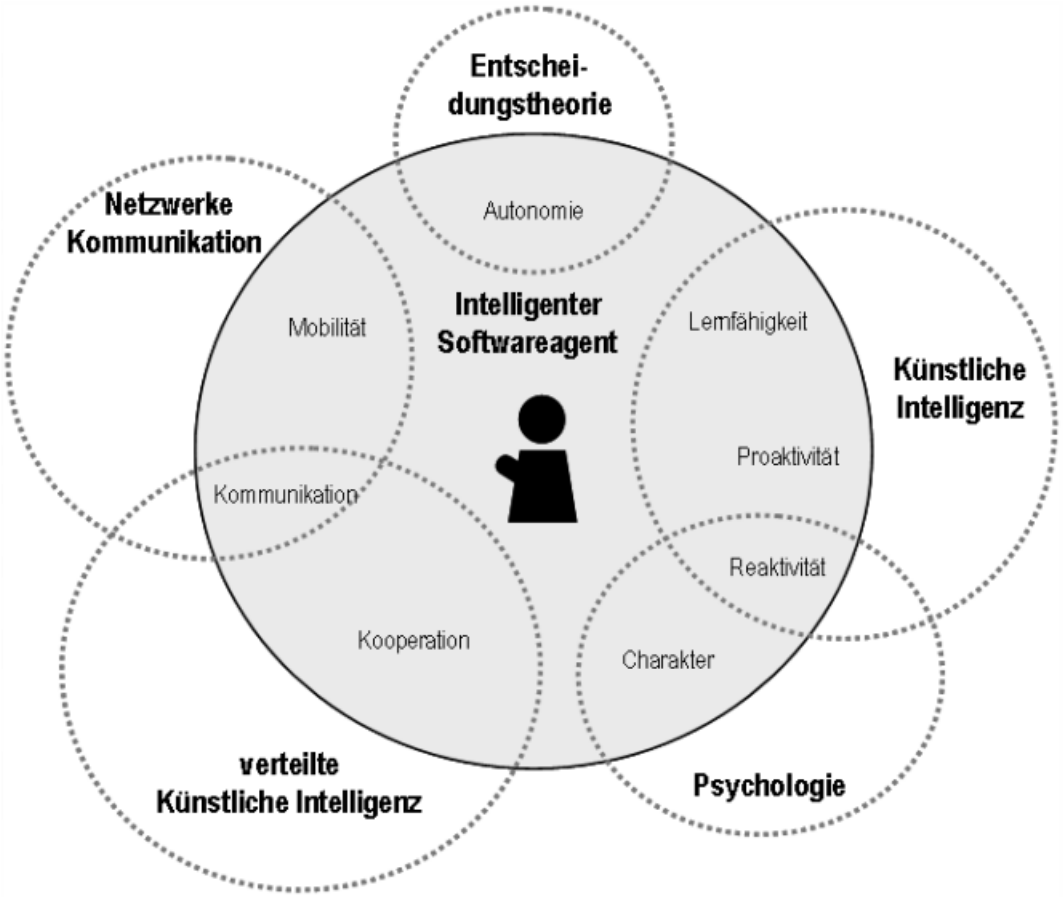


Abbildung 2.11: Einflussgebiete auf die Softwareagenten (Quelle: [BZW97]

will. Im Falle eines Agenten, der ein Fahrerloses Transportfahrzeug steuert könnte ein Plan beispielsweise aus Lade- und Entladeaktionen sowie aus dem Abfahren einer bestimmten Route bestehen. Immer wenn sich an einer der drei Komponenten interner Zustand, Weltbild oder Ziele in der Art etwas ändert, dass der aktuelle Plan nicht mehr zulässig ist oder nicht mehr in der Lage dazu ist, die Ziele des Agenten zu erreichen, muss der Agent seinen Plan anpassen oder ihn verwerfen und einen neuen erzeugen.

Die Verwendung von Agenten ermöglicht es in der Logistik Entscheidungen an dem Ort zu treffen, an dem die Entscheidung eine Auswirkung hat (siehe zum Beispiel [LTSK06] oder [JL08]).

2.3.1 Multi-Agenten basierte Simulationen

Da für die Untersuchung der drei Fragen

- Wie schneidet eine dezentrale Steuerung in Bezug auf die benötigte Anzahl an Fahrzeugen, die durchschnittliche Auftragsdauer, die zurückgelegte Strecke und den Leerfahrtanteil im Vergleich zu einer zentralen Steuerung ab?
- Kann eine dezentrale Steuerung einfacher auf eine Erweiterung bzw. Veränderung des Szenarios angepasst werden, als eine zentrale Steuerung?

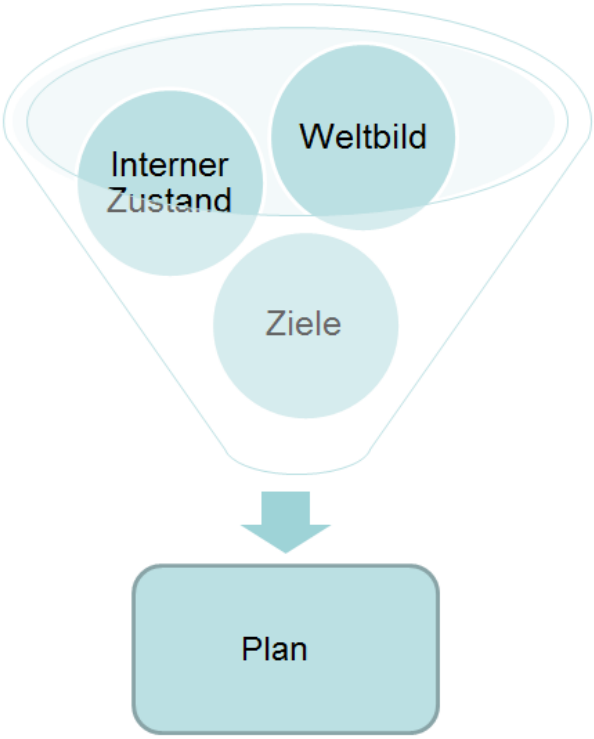


Abbildung 2.12: Aufbau eines Agenten

- Kann eine dezentrale Steuerung flexibler auf sich ändernde Anforderungen an das Systemverhalten reagieren, als eine zentrale Steuerung?

nicht die Möglichkeit gegeben ist ein reales System zu nutzen, muss für die Evaluierung auf eine Simulation zurückgegriffen werden.

Zur Simulation dezentraler Steuerungen in der Logistik bieten sich Mutli-Agenten-Simulationen an. Multi-Agenten basierte Simulationen simulieren Vorgänge in der realen Welt mit Hilfe von (Software-)Agenten.

Es gibt verschiedene Methoden das zeitliche Fortschreiten einer Simulation zu realisieren. Im wesentlichen wird dabei zwischen Discrete Event Simulationen und Continuous Simulationen unterschieden. Bei der Discrete Event Simulation (DES) springt die Simulation von einem festgelegten Zeitschritt zum nächsten. Änderungen im Systemzustand geschehen immer nur zu diesen diskreten Zeitschritten. Änderungen zwischen zwei Zeitschritten geschehen nicht (vgl. [Rob04]). Im Gegensatz dazu basieren Continous Simulationen auf Aktivitäten. Die Zeit wird in kleine Scheiben eingeteilt und der Zustand des Systems wird abhängig von den Aktivitäten, die innerhalb dieser Zeitscheiben durchgeführt werden, verändert (vgl. [Mat08]).

Die Simulation von Materialflusssystemen wird in der Regel als Discrete Event Simulation durchgeführt (siehe zum Beispiel [SSTW09]).

2.3.1.1 Multi-Agenten basierte Discrete Event Simulation

Multi-Agenten basierte Discrete Event Simulation kombiniert Agenten mit einer Discrete Event Simulation. Die Simulation springt jeweils von einem diskreten Zeitpunkt zum nächsten und informiert die Agenten über das Voranschreiten der Zeit. Die Agenten nehmen dann die Veränderungen ihrer Umgebung wahr, passen ihre Pläne an die veränderte Umgebung an und handeln entsprechend. Dieser Ansatz der Simulation wurde bereits in einigen Logistik-Kontexten eingesetzt (siehe zum Beispiel [BWG+06] oder [GLWB06]).

MASON Ein Beispiel für ein System zur Multi-Agenten basierte Discrete Event Simulation ist MASON (vgl. [LCrPS04]). Erklärtes Ziel der Entwicklung von MASON ist die Vereinfachung von Simulationen komplexer adaptiver Systeme, evolutionärer Algorithmen, Methoden der Selbstorganisation, des Physical Modelling sowie des Maschinellen Lernens. Auf der MASON-Homepage[13] findet sich eine breite Auswahl von Projekten aus den genannten Bereichen, die die Bibliothek verwenden. Die MASON-Bibliothek ist modular aufgebaut. Den Kern bildet die Model Library die die grundlegenden Hilfsmittel von MASON bereitstellt. Das sind im Wesentlichen ein Scheduler zur zeitlichen Steuerung von Agenten sowie sogenannte Grids, die die Modellierung der Umgebung ermöglichen, in denen sich die Agenten bewegen. Ein Vorteil von MASON gegenüber anderen Simulations-Frameworks ist die strikte Trennung zwischen Modell und Visualisierung. Die Visualisierung wird von den MASON GUI Tools bereitgestellt. Diese ermöglichen ein dynamisches Hinzufügen, Verändern und Entfernen von Visualisierungs-Objekten zu Agenten und Elementen der Umgebung. Weiterhin kann die Simulation über eine grafische Oberfläche pausiert, gespeichert und geladen werden (inklusive des kompletten Simulationszustands). Durch die Verwendung von Java Beans (siehe zum Beispiel [Zwi04]) zur Beschreibung der Felder der Agenten ist es außerdem möglich, Felder der Agenten über die grafische Oberfläche während der laufenden Simulation zu inspizieren und sogar zu manipulieren.

Der Scheduler realisiert in MASON eine schritt-basierte Simulation. Methoden eines Agenten können entweder einmalig in einem bestimmten Simulationsschritt in der Zukunft ausgeführt werden oder die Ausführung erfolgt als Sequenz in bestimmten Zeitintervallen. Die eigentliche Logik wird hier in den Agentenklassen implementiert, welche auf die Aufrufe durch den Scheduler in geeigneter Weise reagieren müssen.

Grids dienen in MASON der Umgebungsmodellierung. Sie organisieren Felder (Fields) in 2- oder 3-dimensionale Matrizen. Ein Feld beinhaltet wiederum ein Attribut (z.B. ein Integer- oder Double-Wert) oder es können Objekte darauf platziert werden. Beispiele hierfür sind ObjectGrid, SparseGrid, DoubleGrid und IntGrid. Grids beinhalten Methoden zur Bestimmung von Nachbarschaftsbeziehungen. So kann beispielsweise ermittelt werden, welche Objekte sich in einem angegebenen Umkreis von einem bestimmten Feld befinden. Diese Funktionalitäten sind sehr nützlich, um weitere Agenten zu finden, mit denen kooperiert oder kommuniziert werden soll. Ein Simulationsmodell kann aus mehreren übereinander angeordneten Grids bestehen. Nachbarschaftsbeziehungen können jedoch nicht Grid-übergreifend ermittelt werden.

Im Umfeld dieser Arbeit wurden mehrere Intralogistik-Simulationen mit MASON realisiert. In [SH12] wurden die Entscheidungen von Gabelstaplerfahrern in einer Warenumschlaghalle mit einer auf MASON basierenden Simulation untersucht. In [SHS12] wurden ähnliche

[13] http://www.cs.gmu.edu/ eclab/projects/mason/

Untersuchungen in einem Container Port durchgeführt.

2.3.1.2 Multi-Agenten basierte Continous Simulation

Multi-Agenten basierte Continous Simulationen basieren in der Regel auf den Auswirkungen von Aktivitäten. Anstatt dass der Scheduler der Simulation die diskreten Zeitschritte vorgibt, tragen alle Agenten (bzw. alles was den Zustand der Simulation verändern kann, wenn nicht alle Einflussgrößen wie zum Beispiel das Wetter durch Agenten modelliert werden) den nächsten Zeitpunkt, zu dem sie etwas entscheiden oder zu dem sie den Zustand der Simulation verändern wollen, als Zeitpunkt in den Scheduler ein. Andere Entitäten der Simulation werden dann über die Zustandsänderungen informiert und können als Reaktion darauf neu planen oder einen neuen Zeitpunkt in den Scheduler eintragen. Oft werden Aktivitäten von Agenten in diesem Zusammenhang zyklisch geplant. Das bedeutet, dass ein Agent die Aktivität *Sensor Daten auslesen und interpretieren* zum Beispiel zyklisch alle fünf Sekunden durchführt und dementsprechend alle 5 Sekunden einen Zeitpunkt im Scheduler einträgt. Der Scheduler springt dann jeweils von einem durch eine Aktivität eingetragenen Zeitpunkt zum nächsten.

JADE Continous Multi-Agenten basierte Simulationen lassen sich beispielsweise mit dem Multi-Agenten Framework JADE (Java Agent Development Framework[14]) realisieren, welches durch Tilab (Telecom Italia Lab) entwickelt wurde. Es ist stellt eine Middleware für Agenten zur Verfügung, die dem FIPA Standart entspricht (The Foundation for Intelligent Physical Agents[15]). Das Framework unterstützt die Entwicklung und das Debuggen von Mutli-Agenten-Systemen durch eine Vielzahl grafischer Tools.

Agenten in einem JADE System brauchen eine Ausführungsumgebung. Diese werden Container genannt. Die Container selber existieren in der Agenten Plattform. Die Agentenplattform stellt den Agenten Funktionen wie Kommunikation zur Verfügung. Außerdem überwacht sie die Agenten in deren Lebensspanne (siehe den Aufbau von JADE in Abbildung 2.13). Jede Plattform hat mindestens einen Container, den Main Container. Dieser wird immer zuerst gestartet und enthält bereits zwei spezielle Agenten: das AMS (Agent Management System) und die DF (Directory Facility). Diese Agenten sind notwendig für das gesamte System und überwachen die anderen Agenten auf der Plattform.

Wie ebenfalls in der Abbildung 2.13 zu sehen ist, gibt es zwei Wege in JADE ein verteiltes System aufzusetzen. Der erste Weg ist es, verschiedene Container auf der selben Plattform auf verschiedenen Hosts zu verteilen. Von der Systemlogik her betrachtet existiert dann nur eine Plattform und ein Main Container. Dies führt zu der Möglichkeit Agenten mit ihrem lokalen Namen zu adressieren und somit zu einfacherer Kommunikation. Ein anderer Weg ist es, eine Agenten Plattform auf jedem Host zu installieren. Ein Nachteil dieser Möglichkeit ist die Notwendigkeit von Interplattformkommunikation. Der Vorteil ist, dass die Agenten in realen Systemen eingesetzt werden können. Um miteinander kommunizieren und somit interagieren zu können, ist es dafür nicht einmal nötig, dass alle Agenten in JADE realisiert wurden. Solange alle beteiligten Plattformen und Agenten den FIPA Spezifikationen entsprechen ist eine Kommunikation unter ihnen möglich.

[14] http://jade.tilab.com/
[15] http://www.fipa.org/

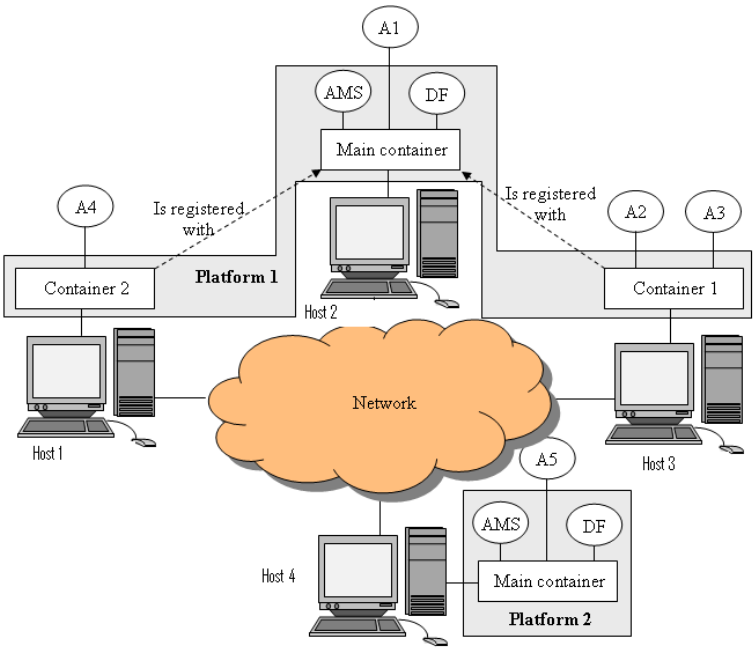


Abbildung 2.13: The JADE Architecture. (Quelle: [Gri13])

Im Rahmen von Untersuchungen über die Vorteile verschiedener Multi Agenten basierter Simulationen wurde ein Kopplung einer MASON Simulation mit einer JADE Simulation durchgeführt. Hierbei konnte gezeigt werden, dass man die Vorteile beider Simulationsarten (Geschwindigkeit bei MASON / DES und Realismus bei JADE / Continous) durch Koppeln beider Systeme für bestimmte Fragen der Intralogistik gleichzeitig nutzen kann. Genauere Ergebnisse wurden in [SSS+13] veröffentlicht.

2.4 Verhandlungen

Als Verhandlung versteht man eine Interaktion "... by which two or more interdependent partys who do not have identical preferences across decision alternatives make joint decisions."(siehe [BC87]) Es geht bei Verhandlungen also darum zu einer Einigung in einer Frage zu finden, bei der die Parteien sich unterscheidende Interessen oder Ansichten vertreten. Hierbei ist die Frage, über die verhandelt wird, nicht festgelegt. In der vorliegenden Arbeit verhandeln die Agenten über Aufträge, Wegstrecken und die Benutzung von Ressourcen.

2.4.1 Klassische Auktionsverfahren

Auktionen sind besondere, genormte Verfahren zur Preisermittlung bei Verkäufen oder Vertragsabschlüssen. Verschiedene Auktionsverfahren werden heutzutage eingesetzt. Mannheims und Oberem heben folgende hervor: ([MO03])

Englische Auktion Bei der englischen Auktion geben die Bieter Gebote ab, die das jeweils aktuell höchste Gebot übertreffen müssen. Hierbei kann gefordert werden, dass das aktuell höchste Gebot stets um eine Mindestdifferenz überboten werden muss. Die

Auktion läuft solange, wie höhere Gebote eingehen. Der Sieger ist derjenige Bieter, der das höchste Gebot abgegeben hat. Er bekommt den Zuschlag für dieses Gebot.

Stille Auktion Bei der Stillen Auktion bieten die Bieter verdeckt. Die Auktion endet zu einem vorgegebenen Zeitpunkt. Anschließend werden die stillen Gebote ausgewertet und das höchste Gebot gewinnt.

Holländische Auktion Bei der holländischen Auktion wird mit einem sehr hohen Anfangsgebot startend der ausgerufene Preis Schritt für Schritt reduziert. Sobald ein Bieter einen Preis akzeptiert endet die Auktion und der Bieter bekommt den Zuspruch für den zuletzt ausgerufenen Preis.

Vickrey-Auktion Die Vickrey-Auktion (benannt nach dem Erfinder und Nobelpreisträger William Vickrey, auch Second-Price-Sealed Auktion)läuft zunächst ab wie die Englische Auktion. Ist die Auktion beendet bekommt auch hier der Höchstbieter den Zuschlag, allerdings muss er nur das zweithöchste Gebot bezahlen. Dieses Verfahren soll dafür sorgen, dass jeder Bieter seine tatsächliche Wertschätzung für das zu versteigernde Produkt bzw. die zu versteigernde Dienstleistung bieten kann. Da er, falls er die Auktion gewinnen sollte, nur das zweithöchste Gebot zahlen muss, was unter seinem Gebot und damit unter seiner Wertschätzung liegt, macht er dennoch einen gefühlten Gewinn.

Zweiseitige Auktion Bei der Zweiseitigen Auktion geben sowohl Nachfrager als auch Anbieter Gebote ab. Passende Gebote werden dann zusammengeführt. Ein prominentes Beispiel für zweiseitige Auktionen ist die Börse.

Klassische Auktionsverfahren bilden oft auch die Grundlage für Verhandlungen in Multi-Agenten-Systemen.

2.4.2 Verhandlungen in Multi-Agenten-Systemen

Verhandlungen in Multi-Agenten-Systemen, auch automatische Verhandlungen genannt, sind Gegenstand der aktuellen Forschung. Die Formulierung automatischer Verhandlungen lässt sich laut Jennings et.al. in drei Teile einteilen: (vgl. [JFL$^+$01])

Verhandlungsprotokolle Verhandlungsprotokolle beschreiben die Regeln der Verhandlung. Sie regeln unter anderem, welche Entitäten an einer Verhandlung teilnehmen dürfen, in welchen Zuständen sich eine Verhandlung befinden kann (wartend auf Angebote, abgeschlossen, abgebrochen) und welche Aktionen die Teilnehmer durchführen, also in der Regel, welche Nachrichten sie zu welchem Zeitpunkt an welchen Teilnehmer schicken können.

Gegenstände der Verhandlung Gibt an über welche Punkte verhandelt wird. Im einfachsten Fall wird über einen einzelnen Wert (zum Beispiel den Preis) verhandelt. In komplexeren Verhandlungen kann aber auch über eine Vielzahl von Punkten gleichzeitig verhandelt werden (Preis, Lieferzeitpunkt, Vertragsstrafen, usw.).

Entscheidungsfindungsmodel der Agenten Beschreibt wie die Agenten unter Berücksichtigung des vorgegebenen Verhandlungsprotokolls und den Gegenständen der Verhandlung versuchen, ihre individuellen Ziele zu erreichen. Die Komplexität dieser Modelle hängt direkt von dem Protokoll und den Gegenständen der Verhandlung ab.

Auch wenn ein Protokoll für eine Verhandlung extrem einfach sein kann (zum Beispiel kann ein Protokoll für eine Holländische Auktion (vgl. 2.4.1) nur aus Angeboten vom Auktionator sowie einer Agebot „Aktzeptiert“-Nachricht von einem Bieter bestehen), haben Verhandlungsprotokolle nach Sandholm eine Reihe von erstrebenswerten Eigenschaften: (vgl. [San99])

Garantierter Erfolg Ein Protokoll garantiert Erfolg, wenn es sicherstellt, dass es immer in einer Einigung endet.

Maximierung des Gesamtnutzens Ein Protokoll, dass dafür sorgt, dass der Gesamtnutzen, also der Nutzen für das System, bei einer Einigung größt möglich ist. Oft ist der Gesamtnutzen als die Summe der Einzelnutzen definiert.

Pareto Optimalität Unter Pareto Optimalität (benannt nach dem italienischen Ökonom und Soziologen Vilfredo Pareto) ist ein Zustand zu verstehen, in dem kein Individuum besser gestellt werden kann, ohne gleichzeitig mindestens ein anderes Individuum schlechter zu stellen. Mit anderen Worten: Wenn eine Einigung nicht pareto optimal ist, dann gibt es eine andere Einigung, die ein Individuum besser stellt ohne die anderen schlechter zu stellen.

Individuelle Vernunft Ein Protokoll ist individuell vernünftig, wenn es sich für die einzelnen Individuen lohnt an einer Verhandlung mit dem gegebenen Protokoll zu beteiligen.

Stabilität Ein Protokoll ist stabil, wenn es alle Beteiligten dazu bringt sich auf eine bestimmte Weise zu verhalten. Eine bekannte Art von Stabilität ist das vom Nobelpreisträger John Nash formulierte Nash-Equilibrium (siehe [Nas50]): Zwei Strategien s und t befinden sich im Nash-Equilibrium wenn t die beste Strategie unter der Annahme ist, dass der Opponent die Strategie s benutzt und umgekehrt.

Einfachheit Ein Protokoll wird einfach genannt wenn es für jeden Beteiligten an einer Verhandlung, die ein solches Protokoll benutzt „offensichtlich“ ist, welche Strategie die optimale ist.

Verteilung Ein Protokoll sollte idealerweise so designed werden, dass es keinen single point of failure gibt und so, dass die notwendige Kommunikation minimiert wird.

Automatische Verhandlungen können als eine verteilte Suche im Raum der möglichen Einigungen angesehen werden. Die Dimension dieses Raums hängt von den Gegenständen der Verhandlung ab. Wird nur über ein Punkt verhandelt ist der Suchraum eindimensional. Die an der Verhandlung eingebundenen Agenten entscheiden über ein vorliegendes Gebot, also ein Objekt im Suchraum, ob es für sie annehmbar ist oder nicht. Sie nehmen dieses Angebot dementsprechend an oder lehnen es ab. Hieraus ergibt sich als notwendige Verhandlungs-Fähigkeiten für die Agenten: vgl. [JFL$^+$01])

- Einen Teil des Angebotssuchraums als annehmbar bestimmen.
- Auf ein Angebot mit Annahme oder Ablehnung reagieren.

Können Agenten nur annehmen oder ablehnen, kann die Suche nach einem für beide (oder alle) Seiten annehmbaren Angebot sehr aufwändig und langwierig sein. Durch die Ablehnung eines Angebots geht nicht hervor, aus welchem Grund das Angebot abgelehnt wurde. Um dieses Problem zu entschärfen können weitere Reaktionen auf ein Angebot zugelassen werden wie zum Beispiel die Kritik an einem Angebot („Ich habe das Angebot abgelehnt weil, ich

den Lieferzeitpunkt nicht einhalten kann.“) oder ein Gegenangebot („Ich würde Ware A zu dem vorgeschlagenen Preis liefern, aber frühestens in zwei Wochen.“).

Die Ermittlung der optimalen Strategie der einzelnen Agenten für die Verhandlung erfolgt laut Jennings et.al. entweder durch Methoden der Spieltheorie oder durch Heuristiken.

Die Spieltheorie beschreibt die Interaktion zwischen egoistischen Agenten. Erstmals so formuliert wurde sie im Jahr 1944 durch von Neumann und Morgenstern (siehe [NM44]). Sie stellt die Frage: Was ist die beste Strategie die unter den gegebenen Umständen anwendbar ist? Laut Jennings et.al. gibt es einige Probleme, die gegen die Nutzbarkeit der reinen Spieltheorie in automatischen Verhandlungen sprechen: (siehe [JFL+01])

Exponentieller Suchraum Der Suchraum aller möglichen Strategien und Interaktionen ist bei komplexeren Verhandlungspunkten oder einem komplexeren Protokoll in der Regel exponentiell wachsend. Die Spieltheorie setzt aber voraus, dass der Suchraum komplett durchsucht wird.

Ordnung der möglichen Ergebnisse Die Spieltheorie setzt voraus, dass sich alle möglichen Auskommen einer Verhandlung nach den Präferenzen bzw. Zielen des Agenten ordnen lassen. Eine solche Ordnung lässt sich in multidimensionalen Suchräumen nicht immer einfach bestimmen.

Auf Grund von diesen Problemen wird die Spieltheorie eher für das Entwickeln von Verhandlungsprotokollen als für das Ermitteln optimaler Strategien eingesetzt.

Heuristiken zur Bestimmung einer guten Strategie sind entweder Approximationen von Techniken aus der Spieltheorie oder die Umsetzung informellerer Verhandlungsstrategien. Beispiele für solche Verfahren finden sich unter anderem bei: Barbuceanu und Lo ([BL00]), Faratin et.al. ([FSJ98]), Kraus und Lehmann ([KL95]) und Sycara ([Syc89]).

Die Nachteile von Heuristiken zur Bestimmung guter Strategien sind laut Jennings insbesondere, dass die entwickelten Strategien in der Regel suboptimal sind (da der Suchraum nicht vollständig exploriert wird) und dass die mittels Heuristiken entwickelten Strategien einer intensiven Evaluation (zum Beispiel durch Simulationen) bedürfen, da ihre Güte sich in der Regel nicht direkt beweisen lässt.

2.5 Graphen und der Dijkstra-Algorithmus

In diesem Abschnitt werden kurz die wichtigsten Definitionen aus der Graphentheorie angegeben, die für diese Arbeit benötigt werden. Anschließend wird der Dijkstra Algorithmus zum Finden kürzester Wege in einem gewichteten Graphen präsentiert, da dieser die Grundlage für viele Routing Algorithmen darstellt und auch die Grundlage des in dieser Arbeit verwendeten Algorithmus ist.

2.5.1 Definitionen aus der Graphentheorie

Die wichtigste Definition der Graphentheorie ist sicher die des Graphen.

Definition 2.3. (ungerichteter) Graph (vgl.[Jun04])
Ein *Graph* G ist ein Paar $G = (V, E)$ bestehend aus einer endlichen Menge $V \neq \emptyset$ und einer Menge $E \subseteq \mathcal{P}_2(V)$. Ein Element $v \in V$ wird als Knoten, ein Element $e = \{a, b\} \in E$ als Kante bezeichnet. v_1 und v_2 aus V heißen **adjazent** wenn es eine Kante $e = \{v_1, v_2\}$ in E gibt. e heißt dann mit v_1 und v_2 **assoziiert**.

Im Themenbereich kürzeste Wege bzw. schnellste Routen hat man es in der Regel mit einer speziellen Art von Graphen, nämlich gerichteten Graphen mit Gewichtsfunktion, zu tun.

Definition 2.4. Gerichteter Graph (*Digraph*) (vgl.[Jun04])
Ein gerichteter *Graph* G ist ein Paar $G = (V, E)$ bestehend aus einer endlichen Menge $V \neq \emptyset$ Knoten und einer Menge $E \subseteq V \times V$ Kanten. Für eine Kante $e = (v_1, v_2)$ nennt man v_1 den **Vorgänger** von v_2 und v_2 den **Nachfolger** von v_1.

Graphen, die Wege, beispielsweise in einer Warenumschlaghalle oder auf einem Logistikhof, repräsentieren, sind i.d.R. gerichtete Graphen. Hierbei stehen die Knoten für Kreuzungen und die Kanten für Wege. Zwei Knoten sind genau dann durch eine Kante verbunden, wenn es einen Weg gibt, der von der durch den ersten Knoten repräsentierten Kreuzung zur durch den zweiten Knoten repräsentierten Kreuzung führt und der auf diesem Teilstück keine weiteren Abbiegemöglichkeiten enthält.

Definition 2.5. Gewichtsfunktion (vgl.[Jun04])
Eine Gewichtsfunktion ist ein Abbildung $w : E \to \mathbb{Z}$ die den Kanten $e \in E$ ein **Gewicht** $w(e)$ zuordnet. Die Kanten eines Graphen mit Gewichtsfunktion werden auch als **gewichtete Kanten** bezeichnet. Das Gewicht einer Kante vom Knoten i zum Knoten j wird oft auch kurz mit w_{ij} bezeichnet.

Auch wenn die Gewichtsfunktion, so wie sie hier definiert wurde, jeder Kante eine ganze Zahl als Gewicht zuordnet, sind die Gewichte im Themenbereich dieser Arbeit, also dem Finden kürzester Routen, alle positiv. Sie stehen entweder für die Distanz zwischen zwei Kreuzungen oder für die benötigte Reisezeit zwischen den beiden Kreuzungen.

Definition 2.6. Pfad (vgl.[Jun04])
Für einen gerichteten Graphen $G = (V, E)$ ist ein Pfad P definiert als eine Folge von Kanten $P = (e_1 \dots e_k)$ für die gilt:

(a) $e_i = \{v_{i-1}, v_i\} \in E$ für $i = 1 \dots k$ $v_{i-1}, v_i \in V$

(b) $e_i \neq e_j \iff i \neq j$

Für einen gewichteten Graphen bezeichnet $w(P) = \sum_{i=1}^{k} w(e_i)$ das **Gewicht des Pfades** P. Weiterhin bezeichnet $d(v, t) = \min_{P \text{ ist Pfad von v nach t}} \{w(P)\}$ den minimalen Abstand (auch Entfernung oder Distanz) der Knoten v und t. Gibt es keinen Pfad von v nach t so wird $d(v, t)$ in der Regel auf unendlich gesetzt.

2.5.2 Der Dijkstra Algorithmus

Der Algorithmus von *Dijkstra* ist ein Markierungsalgorithmus aus der Klasse der Greedy Algorithmen[16], der die kürzesten Wege von einem Startknoten s zu allen anderen Knoten eines Graphen $G = (V, E)$ mit ausschließlich nicht negativen, gewichteten Kanten bestimmt. Dafür werden für alle Knoten j aus V zwei Werte gespeichert: Die Distanz d_j zum Startknoten s und der Vorgänger $pred_j$ des Knotens im kürzesten Weg von s zum Knoten j. Zu Beginn des Algorithmus wird die Distanz von s auf 0 und die von allen anderen Knoten auf unendlich gesetzt, falls es keine direkte Kante von s zu diesem Knoten gibt und auf das Gewicht der Kante, sonst. Nun werden die Knoten in zwei Mengen aufgeteilt: Die Menge S der permanent markierten (settled) Knoten und dem Rest $\overline{S}$. Zu Beginn des Algorithmus besteht S nur aus dem Startknoten s und $\overline{S}$ aus allen anderen Knoten.
Solange nicht alle Knoten permanent markiert wurden, wird in jedem Schritt ein Knoten i aus $\overline{S}$ mit kleinstem, aktuellen Gewicht d_j bestimmt und permanent markiert, also aus der Menge $\overline{S}$ herausgenommen und in die Menge S aufgenommen. Anschließend wird für alle ausgehenden Kanten von j überprüft, ob für die Endpunkte der Kanten gilt, dass das aktuelle Gewicht der Knoten größer ist, als die Summe aus d_j und dem Kantengewicht von j zu diesem Endpunkt (die Kanten werden relaxiert (relaxed)). Ist das der Fall, wird das Gewicht des Endpunktes auf diese Summe gesetzt und j als sein Vorgänger eingetragen.
Am Ende des Algorithmus (wenn also $\overline{S} = \emptyset$ gilt) geben die Gewichte d_j die Länge des kürzesten Weges von s zu j an. Der kürzeste Weg lässt sich rekursiv aus den Vorgängern bestimmen. Die Folge $prec_j, prec_{prec_j}, \ldots$ endet irgendwann bei s.
Für einen Korrektheitsbeweis siehe zum Beispiel [Jun04] oder die Originalveröffentlichung [Dij59].
Der Algorithmus von Dijkstra in Pseudocode ist in Algorithmus 1 gegeben.

Algorithmus 1 Dijkstra

```
1:  S := {s};                          S̄ := V\S;
2:  d_s := 0;                          d_j := w_sj     ∀j ∈ S̄
3:  while S̄ ≠ ∅ do
4:     Bestimme i ∈ S̄ mit d_i = min_{j∈S̄} {d_j};
5:     S := S ⋃ {i};                   S̄ := S̄\{i};
6:     for all Nachfolger j ∈ S̄ von i do
7:        if d_i + w_ij < d_j then
8:           d_j := d_i + w_ij;
9:           pred_j := i;
10:       end if
11:    end for
12: end while
```

Die asymptotische Laufzeit des Algorithmus ist $O(|V|^2)$ falls die Elemente von $\overline{S}$ in einer einfachen Liste gehalten werden. Wird stattdessen eine Datenstruktur benutzt, die das Auffinden des kleinsten Elements aus n Elementen in $log\ n$ garantiert, wie zum Beispiel ein Heap,

[16] Greedy-Algorithmen oder gierige Algorithmen zeichnen sich dadurch aus, dass sie schrittweise den Folgezustand auswählen, der zum Zeitpunkt der Wahl den größten Gewinn bzw. das beste Ergebnis verspricht. Um unter den Folgezuständen eine Auswahl zu treffen, wird oft eine Bewertungsfunktion verwendet. Greedy-Algorithmen sind meist schnell, lösen viele Probleme aber nicht optimal.

beträgt die asymptotische Laufzeit $O(|E| \cdot log|V|)$.
Um nur den kürzesten Weg von einem Startknoten s zu einem Zielknoten t zu bestimmen, kann eine einfache Variante des Algorithmus verwendet werden, die sich nur in Zeile 3 von dem oben vorgestellten Algorithmus unterscheidet. Diese Variante ist in Algorithmus 2 gegeben.

Algorithmus 2 Dijkstra - $s-t$ Variante

1: $S := \{s\}$; $\overline{S} := V \backslash S$;
2: $d_s := 0$; $d_j := w_{sj} \quad \forall j \in \overline{S}$
3: **while** $t \notin S$ **do**
4: Bestimme $i \in \overline{S}$ mit $d_i = \min_{j \in \overline{S}}\{d_j\}$;
5: $S := S \bigcup \{i\}$; $\overline{S} := \overline{S} \backslash \{i\}$;
6: **for all** Nachfolger $j \in \overline{S}$ von i **do**
7: **if** $d_i + w_{ij} < d_j$ **then**
8: $d_j := d_i + w_{ij}$;
9: $pred_j := i$;
10: **end if**
11: **end for**
12: **end while**

Die $s-t$ Variante iteriert nur solange, wie der Zielknoten t nicht permanent markiert wurde. In der asymptotischen Laufzeit hat diese Variante keinen Vorteil, da sie im worst-case genauso viele Iterationen benötigt, wie Algorithmus 1. Wie aber sofort zu sehen ist, ist diese Variante in der Praxis, bei festem s und zufälligem t, amortisiert[17] in etwa doppelt so schnell wie Algorithmus 1, da amortisiert nur halb so viele Schleifeniterationen nötig sind.

[17] Die amortisierte Laufzeitanalyse betrachtet die durchschnittlichen Kosten von Operationen in Folgen. Im Unterschied zur allgemeinen Laufzeitanalyse werden nicht nur die maximalen Kosten der einzelnen Schritte betrachtet, sondern es wird der Worst Case aller Operationen im gesamten Durchlauf des Algorithmus analysiert.

Kapitel 3

Anforderungen und Related Work

Im folgenden Kapitel werden die Anforderungen an die Arbeit und Related Work im Umfeld der Arbeit beschrieben. Die Anforderungen unterteilen sich in Anforderungen an die zu entwickelnde Steuerung und in Anforderungen an die Simulation, mit deren Hilfe die Steuerung evaluiert und die Forschungsfragen beantwortet werden soll.

3.1 Anforderungen an die Steuerung

Ein großer Teil der in dieser Arbeit vorgestellten Konzepte wurden im Projekt FTS entwickelt. Das Projekt wurde, wie schon in der Einleitung beschrieben, forschend vom IPH Hannover sowie vom OFFIS Oldenburg durchgeführt. Der projektbegleitende Ausschuss bestand aus einer Reihe von Industriepartnern aus der FTS Branche. Namentlich waren das:

Götting Hersteller von Datenfunksystemen und Sensoren zur Spurführung von Fahrerlosen Transportfahrzeugen

LinogistiX Anbieter offener IT-Lösungen für Materialfluss und Intralogistik, RFID-Integration und das Retrofitting von Materialflusssteuerungen

MLR Soft Hersteller von von Fahrerlosen Transportsystemen, übernimmt alle wesentliche Punkte von der Planung über die Konzeption und den Bau hin zur Installation

SimPlan Anbieter von Simulationssoftware, individuellen Simulationslösungen sowie Dienstleister im Bereich Simulation

software4production Unter anderem Anbieter von ERP-Systemen sowie von Steuerungssoftware von Industrieanlagen

STILL Anbieter von Gabelstaplern, Wagen und Schleppern sowie Intralogistiksystemen

Weissenburg Hersteller von Fahrerlosen Transportsystemen, übernimmt alle wesentliche Punkte von der Planung über die Konzeption und den Bau hin zur Installation

Das Ziel des Projekts war es eine vollständig dezentrale Steuerung für Fahrerlose Transportsysteme zu entwickeln. Es sollte weiterhin untersucht werden, wie sich die Systemperformanz einer solchen dezentrale Steuerung, also zum Beispiel der Durchsatz, die Durchlaufzeit und die Transportleistung, im Vergleich zu einer zentralen Steuerung darstellt. Die wichtigsten

Anforderungen an die vorliegende Arbeit, die in den nächsten Abschnitten vorgestellt werden, haben sich unmittelbar aus den Projektzielen und den Anregungen der Industriepartner des Projekts ergeben.

3.1.1 Allgemein

Die wichtigste Anforderung ist die Dezentralität der Lösung. Das entwickelte System soll vollständig dezentral sein, also ohne zentrale Komponenten auskommen. Das bedeutet, dass nicht nur die Steuerung der einzelnen Fahrzeuge dezentral erfolgen soll, sondern auch die Auftragsvergabe, die Reservierung von Ressourcen sowie das Berechnen von Routen.

Die zweite Anforderung ist eine gute Systemperformanz. Die entwickelte Lösung soll in den wichtigsten Kennzahlen wie Durchsatz und Transportleistung nicht signifikant schlechter sein, als eine herkömmliche, zentrale Steuerung. Die wichtigste Kennzahl ist laut den Industriepartnern des Projekts die Anzahl an Fahrzeugen, die benötigt wird, um die abgeforderte Transportleistung zu erbringen.

Die dritte Anforderung ist die Anpassbarkeit der Lösung. Im Rahmen des Projekts wurde von den Industriepartnern ein Referenzszenario zur Verfügung gestellt. Bei dem Szenario handelt es sich um eine Anlage zur Getränkeabfüllung. Das Layout besteht aus mehreren Abfüllstationen sowie mehreren Standflächen für die abgefüllten Getränke. Die Schwierigkeiten des Szenarios liegen insbesondere in zwei stark befahrenen, engen Wegen. Die entwickelte Lösung soll nicht auf dieses Szenario hin angepasst werden, sondern so allgemein gehalten werden, dass es sich auch in anderen Szenarien mit anderen Schwierigkeiten einsetzen lässt.

3.1.2 Auftragsvergabe

Es gibt drei wesentliche Anforderungen an die Auftragsvergabe:

- Dezentralität,
- Gewährleistung eines effizienten Systemablaufs,
- Einfache Ermöglichung von Änderungen im Systemverhalten.

Die Dezentralität und die Anforderungen an die Systemperformanz wurden schon in Abschnitt 3.1.1 beschrieben. Das Zuordnen von Transportaufträgen zu zu Transportfahrzeugen, in diesem Fall also die dezentrale Auftragsvergabe spielt natürlich eine wichtige Rolle in der Systemperformanz.

Die dritte Anforderung bezieht sich auf Änderungen an das gewünschte Systemverhalten. Beispielsweise könnte in einer Hochphase, also einer Phase in der sehr viele Aufträge pro Stunde gefahren werden müssen, ein möglichst hoher Durchsatz gefordert sein. In einer Phase in der nur wenige Aufträge pro Stunde gefahren werden müssen, könnte ein möglichst Energie sparendes Systemverhalten gewünscht sein. Eine solche Änderung des gewünschten Systemverhaltens soll durch Änderungen der Auftragsvergabe erreicht werden. Diese Änderung der Auftragsvergabe soll möglichst einfach auch durch einen Endanwender und möglichst unabhängig vom konkreten Szenario erfolgen können.

3.1.3 Routing

Es gibt drei wesentliche Anforderungen an die Auftragsvergabe:

- Dezentralität,
- Gewährleistung eines effizienten Systemablaufs,
- Konfliktvermeidung schon während des Routings.

Die ersten beiden Anforderungen wurden schon in Abschnitt 3.1.1 beschrieben. Das Routing muss zur Erfüllung dieser Anforderungen ebenso beitragen wie beispielsweise die Auftragsvergabe, darum sind sie an dieser Stelle nochmal speziell aufgeführt.

Die dritte Anforderung, die Konfliktvermeidung, meint, dass ein Fahrzeug seine Routen nur so planen soll, dass es beim Abfahren der geplanten Route nicht zu einem Konflikt mit einem anderen Fahrzeug kommt. Ein Konflikt läge hier beispielsweise vor, wenn sich zwei Fahrzeuge in einem Weg entgegenkommen, der nur für ein Fahrzeug breit genug ist. In dieser Situation müsste eins der Fahrzeuge zurücksetzen. Dies hätte zum einen offensichtlich negative Auswirkungen auf die Systemperformanz (das zurücksetzende Fahrzeug braucht länger, um den Auftrag zu erfüllen) und macht zum anderen ein Verfahren zur Konfliktlösung notwendig (Welches Fahrzeug setzt zurück? Nach welchen Kriterien soll entschieden werden?). Wenn alle Fahrzeuge schon beim Planen der Routen Konflikte vermeiden, können diese Probleme umgangen werden.

3.2 Anforderungen an die Simulation

Um die oben aufgestellten Anforderungen an die Steuerung überprüfen zu können, muss eine Simulation entwickelt werden, welche in der Lage ist, diese Überprüfung durchzuführen. Die zentrale Anforderung an die zu entwickelnde Simulation ist dementsprechend, dass die entwickelten Agenten und die entwickelte Steuerung in der Simulation implementierbar sind. Des Weiteren basiert die Überprüfung der Anforderungen an die Steuerung in großen Teilen auf Vergleichen mit einer zentralen Steuerung. Für diese Vergleiche wird üblicherweise auf bestimmte logistische Kennzahlen wie zum Beispiel die Anzahl der benötigten Fahrzeuge, die durchschnittliche Auftragsdauer oder die von den Fahrzeugen zurückgelegte Strecke, zurückgegriffen. Folglich muss die Simulation in der Lage sein, diese Kennzahlen für die dezentrale Steuerung zu bestimmen.

Für den Vergleich mit einer zentralen Steuerung stehen, wie in Kapitel 1 beschrieben, die Kennzahlen sowie weitere Daten eines zentral gesteuerten Fahrerlosen Transportsystems, welches in einer Getränkeabfüllanlage installiert ist, zur Verfügung. Abbildung 3.1 zeigt das Layout dieser Anlage. Die Simulation muss in der Lage sein, das zur Verfügung gestellte Szenario umzusetzen. Für die Umsetzung ist zum einen das Modellieren des Hallenlayouts sowie der Fahrzeuge (Geschwindigkeit, Ladekapazität) als auch die korrekte Abbildung der Auftragseinlastung notwendig. Um weitere Experimente durchführen zu können, sollte die Modellierung eines Szenarios hier allerdings möglichst so einfach gehalten sein, dass es ohne großen Aufwand möglich ist, auch andere Szenarien zu simulieren.

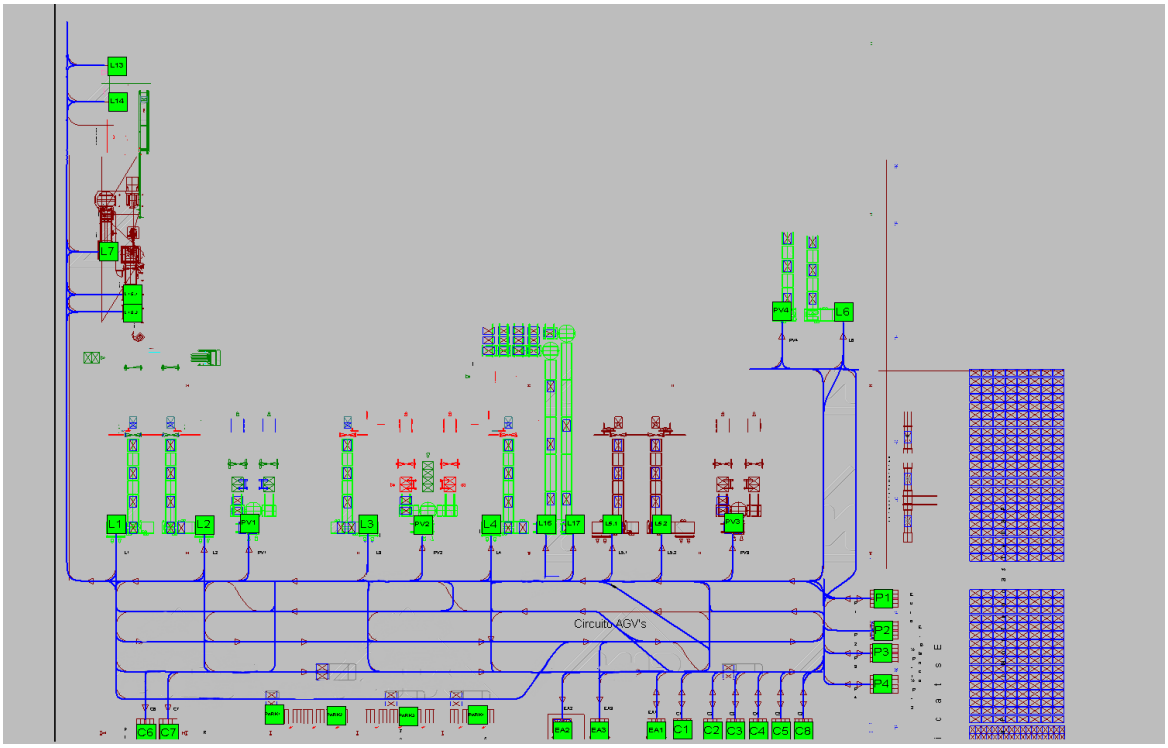


Abbildung 3.1: Layout der Getränkeabfüllanlage

3.3 Related Work

Da die vorliegende Arbeit ein sehr breites Spektrum an Themen über Agententechnologien, Selbststeuerung, Multi-Agenten-Simulation, Intralogistik, dezentraler Auftragsvergabe sowie konfliktvermeidendem Routing abdeckt, ist die Related Work, die die Arbeit betrifft, ebenfalls recht breit aufgefächert. Eine gründliche Beschreibung aller Themen würde daher den Rahmen der Arbeit sprengen. Es wird im folgenden daher auf die wichtigsten Arbeiten im Umfeld dieser Arbeit eingegangen.

3.3.1 Selbststeuerung in der Intralogistik

Dezentrale Steuerung oder auch Selbststeuerung wurde bereits für einige Teilgebiete der Intralogistik konzipiert bzw. auch eingesetzt.

Ein Beispiel hierfür ist die Produktionslogistik. Produktionslogistik beschreibt die internen Materialflüsse, die in einer Produktionsanlage anfallen (siehe zum Beispiel [Jod08]). Scholz-Reiter et.al. haben in [SRWK+04] einen Ansatz für eine teilweise Selbststeuerung eins solchen Systems beschrieben. Die Grundidee wird in Abbildung 3.2 dargestellt. Unter Selbststeuerung werden hier hauptsächlich Informationsflüsse dezentraler Entscheidungsfindung verstanden. Verschiedene Objekte wie Transportgüter, Maschinen oder Fördermittel bekommen direkt die Möglichkeit Informationen auszutauschen und zu verarbeiten. Das Hauptziel ist hier die Realisierung einer höheren Systemrobustheit. Diese wird von Scholz-Reiter et.al. als "[...] die Beibehaltung beziehungsweise Rückkehr in einen stabilen Systemzustand nach Einwirkung einer Klasse von Störungen."definiert. Als Einsatzgebiet ist hier die Kleinserie oder

Einzelteilfertigung zu sehen. Ein Gut sucht sich eigenständig einen Weg zu den jeweiligen Bearbeitungsstationen. Der Aufbau ist hier wie in Abbildung 3.2 abgebildet. In der oberen Ebene werden die Produktionsaufträge ins System eingespeist. Die zweite Ebene stellt ein Kommunikationssystem, beispielsweise WLAN dar. In der dritten Ebene wird dann ein Softwareagent für jedes zu transportierende Gut erzeugt. Dieser Agent kommuniziert mit den Maschinen und Förderanalgen um die benötigten Transporte und Bearbeitungen zugesichert zu bekommen. In der Folge überwacht der Agent die korrekte Abarbeitung dieses Plans. Die eigentliche Maschinen- und Transportsteuerung erfolgt hier weiterhin zentral, wie es in solchen Anlagen heute Stand der Technik ist.

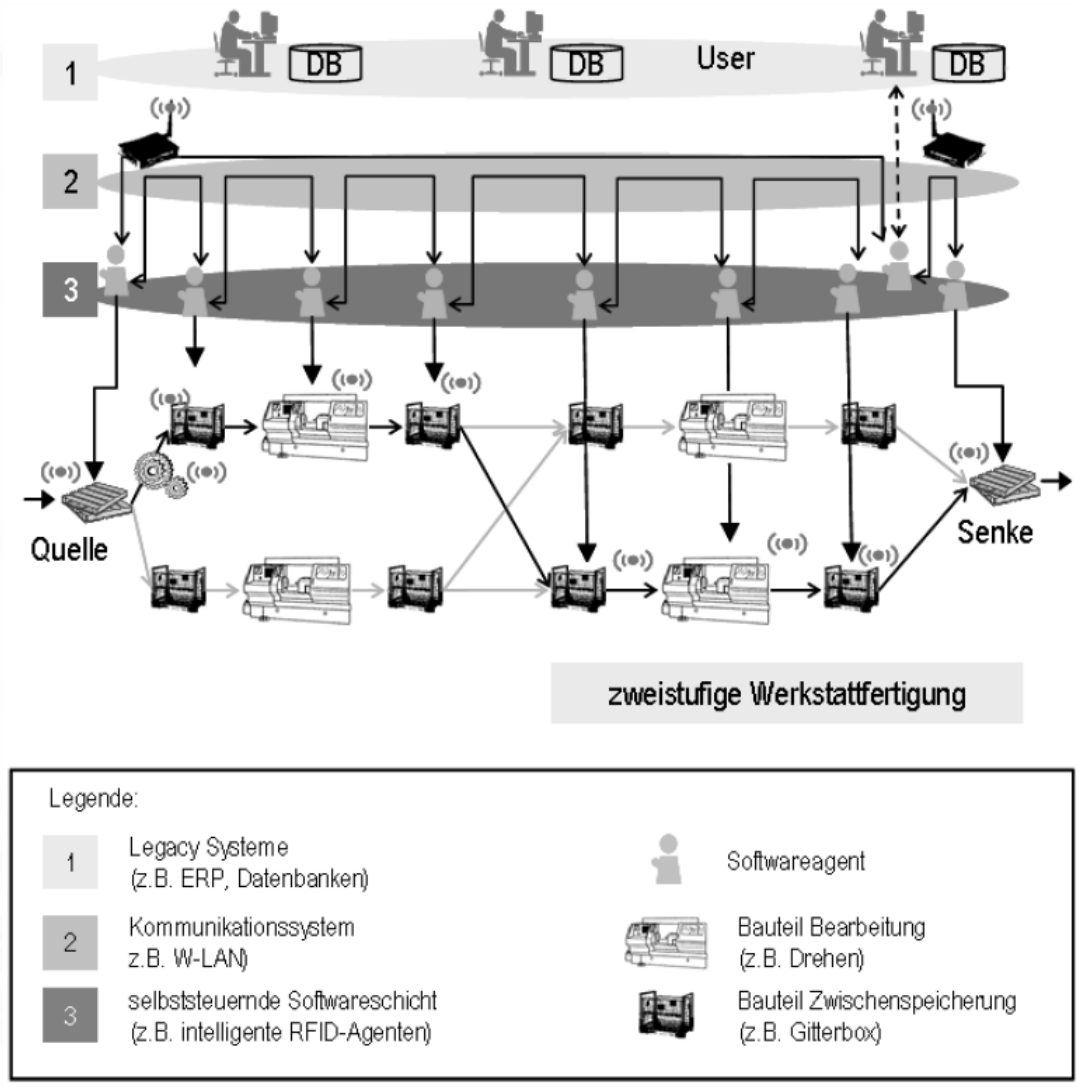


Abbildung 3.2: Grundidee der Selbststeuerung in der Produktionslogistik (Quelle: [SRWK+04]

Ein weiteres Beispiel ist die Selbststeuerung im Supply Chain Management. In [SRH01] wurde ein Ansatz vorgestellt der auf reaktive Agenten als Teil der Datenverarbeitung bei der Steuerung bzw. dem Mangement von Supply Chains setzt. Hier wird die Planung und Steuerung einer Supply Chain in mehreren Schichten betrachtet. Die Schichten sind in Abbildung 3.3 abgebildet. Die oberen Schichten entsprechen längerfristigen Planungen (Programmplanung,

Bereichsplanung und Bereichssteuerung) und werden in dem Modell mit klassischen Verfahren realisiert. Die unterste Schicht, die reaktive Planung, wird durch Multi-Agenten-Systeme realisiert. Dieser Teil der Planung geht auf kurzfristige Verschiebungen im Bereich von Stunden ein. Der Einsatz von Multi-Agenten-Sytemen bietet hier die Möglichkeit nah am Ort der Veränderung zu reagieren und soll hierbei möglichst große Teile des Plans unverändert lassen (stabile Umplanung).

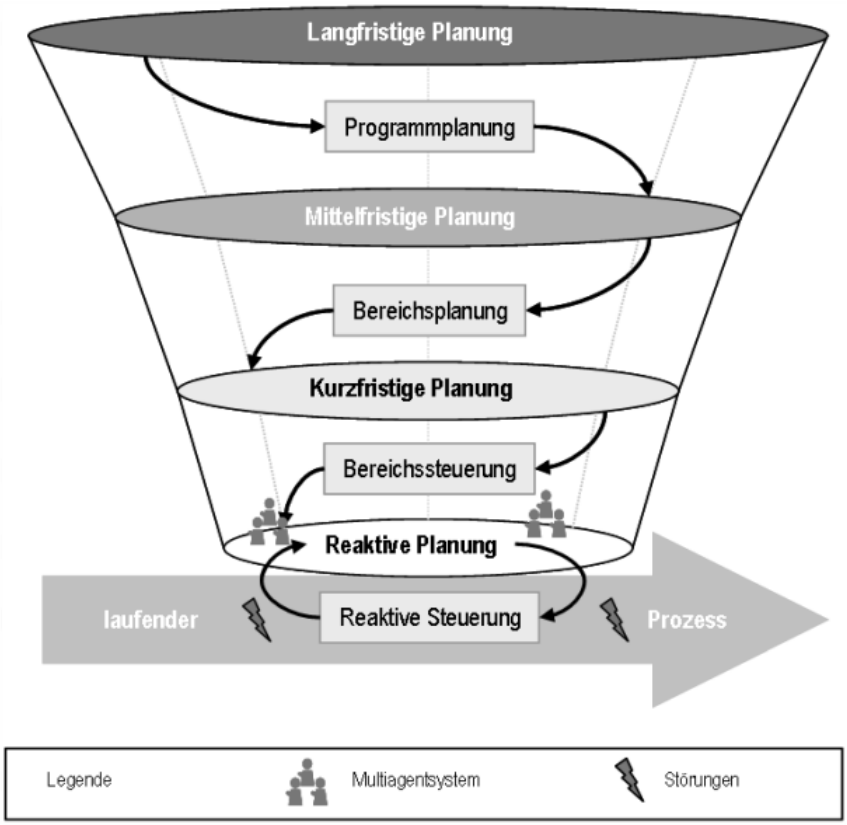


Abbildung 3.3: Reaktive Planung und Steuerung (Quelle: [SRH01]

Ein weiteres Beispiel ist die Steuerung eines Materialflusssystems durch den Einsatz von Agenten. Ein solches System wurde beispielsweise in [DTH10] vorgestellt. Hier wird ein Materialflusssystem als ein Verbund aus Materialflussmodulen (MFM) modelliert. Die Einzelnen Module werden anhand Ihres Ordnungstyps unterschieden. Siehe hierfür Abbildung 3.4. Die erste Zahl der Ordnung gibt den Eingangsgrad, also die Anzahl der MFM, die Transportgüter zum betrachteten MFM schicken können, die zweite Zahl den Ausgangsgrad, also die Zahl der MFM, zu denen das betrachtete MFM schicken kann an.

In der Modellierung von Daniluk und ten Hompel wird nun jedem solchen MFM ein Agent zugewiesen, der eigenständig Entscheidungen treffen kann und der mit den Agenten, die die mit ihm verbundenen MFM steuern, kommunizieren kann. Weiterhin sind die Agenten in der Lage das Ziel eines Transportguts zu identifizieren. Technisch ist das laut den Autoren zum Beispiel durch RFID-Aufkleber und den entsprechenden Lesegeräten zu realisieren. Die Zusammenhänge zwischen den Agenten, den MFM und den Stationen sind in Abbildung 3.5 dargestellt. Der MFM-Graph stellt hier eine unzusammenhängende Menge von MFM dar. Im Beispiel sind das sechs MFM die jeweils grau hinterlegt wurden. Die grau gestrichelten Linien

	Station	Ordnungstyp	Beschreibung
Elementarstation	Quelle	$(0, m)$ mit $m \geq 1$	Erzeugung von Lastobjekten
	Senke	$(n,0)$ mit $n \geq 1$	Entfernung von Lastobjekten aus dem Simulationsmodell
	Bearbeitungs-station	(n,m) mit $n,m \geq 1$	Verknüpfungspunkt Bearbeitungsstation: Ausübung einer Tätigkeit am Lastobjekt
	Lagermittel	(n,m) mit $n,m \geq 1$	Verknüpfungspunkt Lagermittel: Lagerung des Lastobjektes
Transportelement	Verbindungs-element	(1,1)	Verbindung zwischen zwei Fördertechnikabschnitten
	Verzweigung	$(1,m)$ mit $m \geq 2$	Verteilung von einem Eingangsfördergutstrom auf mehrere Ausgangsfördergutströme
	Zusammen-führung	$(n,1)$ mit $n \geq 2$	Zusammenführung von mehreren Eingangsförder-gutströmen zu einem Ausgangsfördergutstrom
	Kreuzung	(n,m) mit $n,m \geq 2$	Verteilung von mehreren Eingangsfördergutströmen auf mehrere Ausgangsfördergutströme

Abbildung 3.4: Elemente eines Materialflussmodells (Quelle: [DTH10]

im MFM-Graph geben die Verbindungen zwischen den MFM an. Aus diesen Verbindungen kann zum einen der Stationengraph und zum anderen der MFM-Strukturgraph abgeleitet werden. Der MFM-Strukturgraph beschreibt hier die Flussbeziehungen zwischen den MFM mit Hilfe gerichteter Kanten. Eine Umwandlung dieser gerichteten Kanten in ungerichtete ergibt dann den Agentenkommunikationsgraph, der angibt, welche Agenten miteinander kommunizieren können. Auf dieser Grundlage lassen sich nun dezentrale Routing-Algorithmen implementieren.

Ein Beispiel das dem Einsatz von Agenten zur Steuerung von Fahrerlosen Transportsystemen vom Aufbau her sehr nahe kommt wurde in Form eines auf Agenten basierten Staplerleitsystems von Göhring und Lorenz in [GL10] vorgestellt. In ihrem Ansatz ist jeder Stapler mit einem mobilen Gerät (zum Beispiel einem Tablet) ausgestattet, auf dem eine Agentenplattform läuft. Weitere Agentenplattformen sind auf Rechnern, die eine Schnittstelle zum System (also zum Beispiel zum Lagerhaus) darstellen sowie auf Rechnern, die als zentrale Instanz die Auftragsvergabe übernehmen, vorhanden. Die Agenten im System sind in Abbildung 3.6 dargestellt. Die Agenten, die vom Terminal-Agent abstammen, repräsentieren bewegliche Entitäten denen Aufträge zugeordnet werden können. Im vorgestellten Fall sind dies Transportaufträge und Kommissionieraufträge. Der Agent Transporteinheit entspricht einem Transportgut. Die Schnittstelle ist für das Ankommen von Transportgütern in das System und das wieder Ausscheiden von Transportgütern aus dem System verantwortlich. Die Manager sind zentrale Agenten. Der Auftragsmanager ist für die Zuweisung und Überwachung von Transportaufträgen zuständig, der Ressourcenmanager für das Zuweisen und Freigeben von Transportaufträgen.

Die Arbeitsweise des agentenbasierten Staplerleitsystems (aSLS) kann wie folgt beschrieben werden: Für alle Transportgüter, die in das System hineinkommen, wird vom Schnittstellen

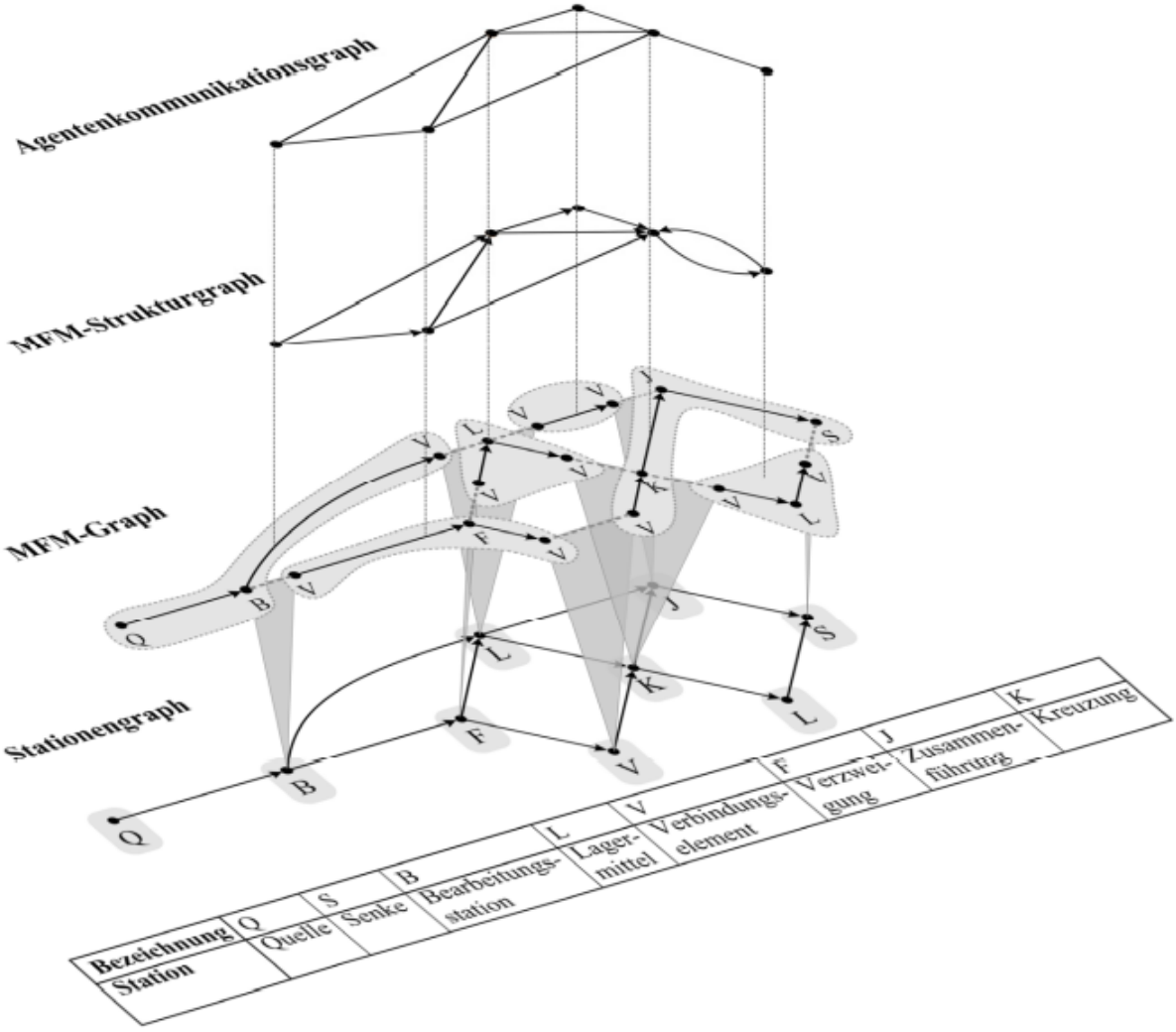


Abbildung 3.5: Darstellung des Zusammenhangs zwischen Stationengraph, MFM-Graph, MFM-Strukturgraph und Agentenkommunikationsgraph (Quelle: [DTH10]

Agent ein Transporteinheit Agent erzeugt. Dieser enthält alle relevanten Informationen über das Transportgut insbesondere auch sein Ziel. Anschließend migriert dieser Agent auf die Plattform, auf der die Manager Agenten laufen. Das bedeutet, dass die erzeugte Software-Instanz nun auf einem anderen Rechner läuft. Der Auftragsmanager sucht nun einen Agenten, der diesen Transportauftrag erledigen kann. Hierfür sucht er den nächstgelegenen Terminal-Agenten. Dieser Agent erhält nun den Auftrag und der entsprechende Transporteinheit Agent migriert auf die Plattform, auf der der entsprechende Terminal-Agent läuft. Einem Terminal Agenten können auch mehrere Aufträge zugeordnet werden oder ein Auftrag, der aus mehreren Unteraufträgen besteht (insbesondere bei Kommissionieraufträgen). Die entsprechenden Terminal-Agenten weisen die Benutzer nun an die Aufträge durchzuführen. Ist ein Auftrag abgeschlossen wird der Transporteinheiten Agent entsprechend aktualisiert (er enthält nun zum Beispiel das Fertigstellungsdatum des Transport und den aktuellen Standort des Transportguts) und anschließend auf die Manager Plattform migriert. Diese wertet den Auftrag aus und schickt das Statusupdate gegebenenfalls weiter zum Schnittstellen Agent der die Informationen an über- oder beigeordnete Systeme, wie zum Beispiel einem Warehouse Management

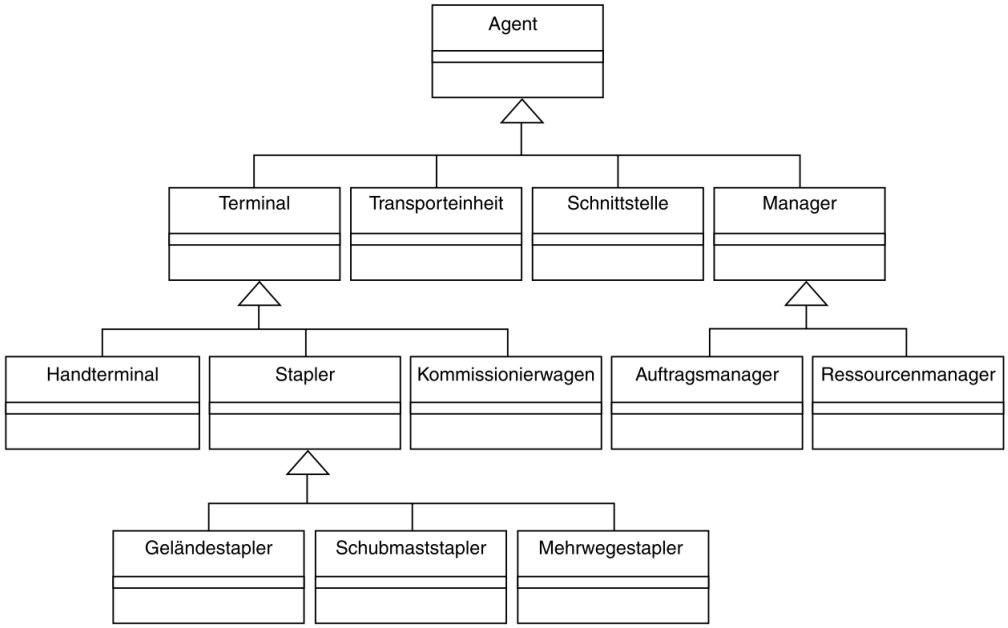


Abbildung 3.6: Klassendiagramm der Agenten im aSLS (Quelle: [GL10]

System (WMS), weiterleiten kann.

Der logische Aufbau dieser Struktur ist in Abbildung 3.7 dargestellt. Die unterste Schicht stellt die physikalische Schicht dar. Hier finden sich neben den Fahrzeugen die Rechner auf denen die zentralen Agenten laufen. Die mittlere Schicht zeigt die Agenten im System sowie die jeweiligen Kommunikationsflüsse. Übergeordnet dazu ist das aSLS, das als zusammenführender Rahmen für die Agenten verstanden werden kann. Als Beispiel für ein beigeordnetes System ist hier ein Warehouse Management System abgebildet.

Als Vorteile gegenüber einem herkömmlichen, zentralen Staplerleitsystem nennen Göhring und Lorenz:

- Standardisierbarkeit
- Skalierbarkeit
- Offline Betriebsmodus
- Simulation
- Terminal-Austauschbarkeit
- Rechenleistungsverteilung

Mit dem Offline Betriebsmodus ist hier gemeint, dass ein Terminal die Aufträge, die ihm zugeordnet sind auch abarbeiten kann, wenn zwischenzeitlich die Kommunikationsverbindung ausfällt (zum Beispiel bei Schwankungen im WLAN Netz oder wenn es in der Halle WLAN-Schatten, also Bereiche ohne Empfang gibt). In einem solchen Fall arbeitet der Agent solange einfach nur die ihm schon zugewiesene Aufträge ab, bis er wieder eine Verbindung zum Auftragsmanager Agenten aufbauen kann oder er keine Aufträge mehr hat. Sobald er wieder mit dem Auftragsmanager kommunizieren kann, übermittelt er alle bis dahin abgearbeiteten

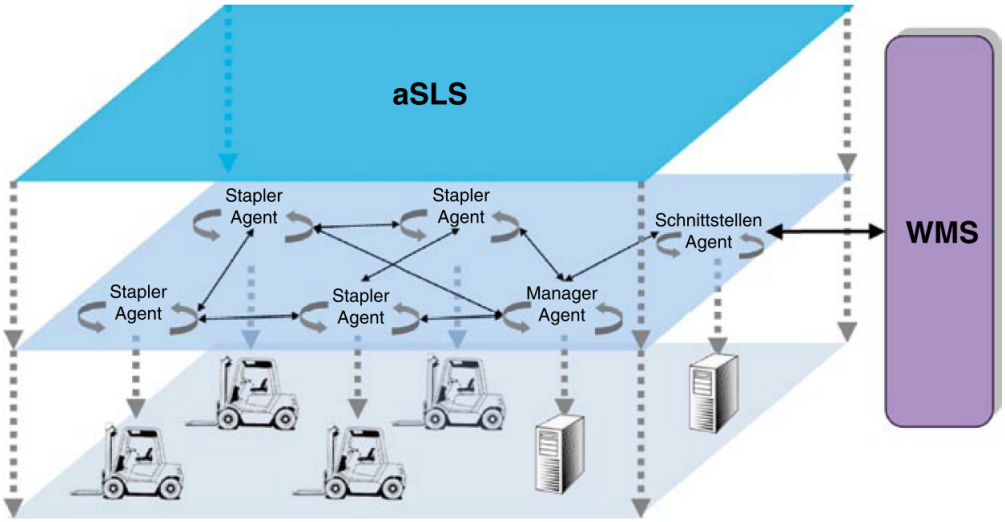


Abbildung 3.7: Logische aSLS-Architektur (Quelle: [GL10]

Transporteinheiten Agenten und erhält gegebenenfalls neue.

Der Schwerpunkt der Arbeit liegt auf der Agentifizierung des Systems und dem Beschreiben der notwendigen Kommunikationsflüsse sowohl innerhalb der aSLS als auch mit über- und beigeordneten Systemen. Der Punkt der effizienten Auftragsvergabe wird hier nicht intensiv behandelt. Der Punkt der Routenplanung spielt im System überhaupt keine Rolle. Die Fahrzeuge im System sind keine Fahrerlosen Transportfahrzeuge sondern Stapler und die Routenplanung wird hier vollständig den menschlichen Fahrern überlassen.

3.3.2 Auftragsvergabe

Die in den Grundlagen vorgestellte dezentrale Auftragsvergabe mit dem klassischen CNET-Protokoll (siehe Abschnitt 2.2) vergibt anfallende Aufträge sofort und endgültig an einen Agenten. Dadurch kann das System nicht flexibel auf Änderungen der Situation reagieren (siehe zum Beispiel [BWHM04]). Solche Änderungen können sich in einem fahrerlosen Transportsystem neben sehr seltenen Ereignissen wie dem Ausfall oder dem Hinzukommen eines Fahrzeugs auch schon durch das Eingehen neuer Aufträge in das System ergeben. Gerade im Fall einer dezentralen Steuerung liegen die abzuarbeitenden Aufträge in der Regel nicht für die ganze Schicht im voraus vor, sondern fallen dynamisch an.

Windelinckx und Strens stellen in [WS04] eine Lösung zur Vergabe von dynamisch anfallende Aufträge an Agenten vor, welches auf reinforced learning basiert. Hier versucht eine zentrale Komponente Muster im Strom der anfallenden Aufträge zu lernen und mit Hilfe des Gelernten Vorhersagen über die in naher Zukunft anfallenden Aufträge zu machen. Die Vergabe von Aufträgen berücksichtigt nicht nur die aktuelle Situation, sondern auch die vorhergesagte nahe Zukunft. Die Auftragsvergabe erfolgt hier vollständig zentral, so dass sich das Verfahren nicht mit den in Abschnitt 3.1 vorgestellten Anforderungen dieser Arbeit vereinbaren lässt.

Eine Klasse von Methoden zur dezentralen Auftragsvergabe, die auf dynamisch anfallende Aufträge reagieren können sind durch die Natur, genauer durch Stigmergie, inspiriert.

Stigmergie ist ein Konzept zur Beschreibung einer besonderen Form der Koordination von Kommunikation in einem dezentral organisierten System, das eine große Anzahl von Individuen umfasst. Dabei kommunizieren die Individuen des Systems nicht unmittelbar, sondern nur indirekt miteinander, indem sie ihre lokale Umgebung modifizieren. Das gemeinsam Erstellte wird gleichsam zum Auslöser von Anschlussaktivitäten und zur allgemeinen Anleitung dafür, wie mit dessen Erstellung fortzufahren ist. Beispielsweise kommunizieren Ameisen bei der Futtersuche indirekt miteinander, indem sie entlang ihrer Straßen Pheromone hinterlassen. Ameisen auf Futtersuche folgen der am stärksten wahrnehmbaren Pheromonspur, da diese wahrscheinlich den kürzesten Weg zu einer Futterquelle weist. Eine Ameisen-Kolonie ist somit ein stigmergisches System. Ein Beispiel für die Ausnutzung dieses Prinzip wurde von Hadeli et.al. vorgestellt (siehe [HVKB04]). In der Arbeit werden Informationen über die Aufträge und deren Prioritäten in der (virtuellen) Umgebung gespeichert (z.b. durch virtuelle Pheromone). Da diese Pheromone in den virtuellen Umgebungsrepräsentationen jedes Agenten gespeichert werden müssen, besteht jedoch ein hoher Kommunikationsaufwand für die Synchronisation dieser Daten bzw. eine hohe Anforderung an die zu installierende Hardware, falls die virtuellen Pheromone an Funkknoten im Layout gespeichert werden sollen.

Ein weiterer Ansatz der auf die Übermittlung von Informationen durch die Umgebung setzt ist der Ansatz der Coordination Fields, vorgestellt durch Mamei et.all. in [MZL04]. Hier werden durch Hosts in der Umgebung oder durch die Hardware, die die Funkkommunikation zur Verfügung stellt, Informationen durch sogenannte computational fields den Agenten zur Verfügung gestellt. Computational fields werden hier als eine Karte von jeder Recheneinheit im Raum zu jedem zu berücksichtigenden Objekt verstanden (siehe zum Beispiel [VDB13]). Beispiele für solche Felder wären der Temperaturverlauf in einem Gebäude, bestimmt durch mehrere Temperaturmesser in einem Gebäude (ein skalares Feld), oder ein Feld für die Bestimmung des kürzesten Weges zu einem bestimmten Ort (ein Vektorfeld). Mamei et.all. setzten diese Felder ein um die Aktionen von mehreren autonomen Agenten zu koordinieren, ohne dass eine direkte Kommunikation zwischen den einzelnen Agenten notwendig wäre. Sie verwenden hierzu unter anderem Felder, die angeben, wie voll ein Bereich gerade ist (wie viele Agenten sich also in einem bestimmten Bereich des Layouts aufhalten) und wie nachgefragt der Bereich gerade ist (im Falle eines FTS also bspw. wie viele Aufträge in dem entsprechenden Bereich gerade anliegen). Um einen solchen Ansatz umzusetzen ist allerdings, ähnlich wie im Ansatz mit Pheromonen, hohe Anforderungen an die Hardware in der Umgebung gestellt.

Boucké et.al. stellten in [BWHM04] einen Ansatz für die dezentrale Vergabe von Aufträgen vor, der auf einem speziellen Verhandlungsprotokoll beruht. Es handelt sich dabei um ein zyklisches Protokoll, das auf Aufgabenenergie und auch Interessenenergie basiert. Jeder erstellten Aufgabe wird einem Auftrags-Agenten zugeordnet und mit einer initialen Aufgabenenergie versehen (basierend zum Beispiel auf der Priorität des Auftrags). Am Anfang eines jeden Zyklus informieren die Auftragsagenten die Fahrzeugagenten über die Aufgabe und deren aktuellen Energielevel. Die Fahrzeugagenten antworten dann mit der Menge Energie, die sie konsumieren wollen (der Interessenenergie). Dieses Interesse beruht auf der Aufgabenenergie (welche mit der Entfernung zum Aufnahmepunkt abnimmt), der Eignung des Fahrzeugs für den Auftrag, der Menge an Energie, die im letzten Zyklus konsumiert wurde und möglichen anderen Faktoren, die das Interesse an der Übernahme des Auftrags verändern können. Fahrzeuge die im Moment keine Ladung haben bewegen sich dabei auf den Aufnahmeort des Auftrags zu, der sie am meisten interessiert (und sind daher beim Start des nächsten Zyklus näher an diesem Aufnahmepunkt). Nach einer festgelegten Zeit wird der aktuelle Zyklus beendet. Die Auftragsagenten bestimmen dann, wie viel Energie verbraucht wurde und erhöhen das

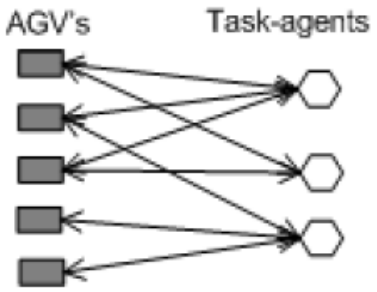


Abbildung 3.8: Many-to-many communication relation (Quelle: [BWHM04]

Energielevel anschließend um einen fixen Betrag (was ein Verhungern der Aufgabe verhindert). Dieses Protokoll ist ein Viele(Auftragsagenten)-zu-Viele(Fahrzeugagenten)-Protokoll (siehe Abbildung 3.8). Boucké et.al. benutzen ihren Ansatz um Probleme, die durch die dezentrale Auftragszuordnung in FTS auftreten können, zu bewältigen: "While the AGV rides towards the pick-up spot (in the third step) many things can happen: new tasks that are better suited for this AGV can show up, e.g. being much closer, more urgent or even a task being on the way to the pick-up spot; AGVs can become unavailable, because of a failure or because they suddenly have to go in maintenance; AGVs better suited for the task, e.g. closer to the pick-up spot, with more energy left in their battery, can become available."(from [BWHM04]).

3.3.3 Routing

Das Routing der Fahrerlosen Transportfahrzeuge, also das Bestimmen der Wege, die diese fahren sollen, gehört zu den wichtigsten Aufgaben eines Faherlosen Transportsystems. Dementsprechend ist die Frage nach guten Routingalgorithmen, die Routen ohne Konflikte für die Fahrzeuge bestimmen können schon lange Teil der Forschung. Eingeführt wurde das Konzept des konfliktfreien, kürzeste Wege FTF Routings 1985 von Broadbent et.al.. Das in [BBPW85] vorgestellte Verfahren benutz den Dijkstra-Algorithmus (siehe Abschnitt 2.5.2) um eine Matrix zu erzeugen, die die Belegungen von Knoten durch die Fahrzeuge repräsentiert. In dieser Matrix werden dann zunächst kürzeste Wege für alle Fahrzeuge berechnet. In einem zweiten Schritt werden dann mögliche Konflikte durch das Verlangsamen bestimmter Fahrzeuge oder Planen alternativer Routen aufgelöst. Es handelt sich also um ein zweistufiges Verfahren. Zuerst wird eine initiale Menge an Routen berechnet und im zweiten Schritt wird diese dann auf Konflikte überprüft, die anschließend aufgelöst werden.

1991 präsentierten Kim und Tanchoco in [KT91] das Konzept des time window graph. Im time window graph repräsentieren die Knoten die freien Zeitfenster und die Kanten die Erreichbarkeiten zwischen den Knoten. Dieser Graph wird aus dem gegebenen Graphen, der das Wegenetz repräsentiert, sowie durch die Information über schon geplante Routen erzeugt. Er ist also dynamisch in dem Sinn, dass er bei jeder Änderung der geplanten Routen neu berechnet werden muss. In diesem time window graph wird dann mit einem an den Dijksta Algorithmus angelehnten Algorithmus ein kürzester Weg bestimmt. Um es den Fahrzeugen zu ermöglichen als Teil ihres Plans zu warten erhält der Graph loops. Das gesamte Verfahren hat eine asymptotische Laufzeit von $o(v^4n^2)$ wobei v der Anzahl der Fahrzeuge entspricht und n der Anzahl der Knoten im gegebenen Graphen. Das Verfahren bestimmt im Gegensatz zu

dem Verfahren von Broadbent et.al. in einem Schritt konfliktfrei Routen. In den „Extensions, variations and operational considerations“gehen Kim und Tanchoco schon auf Fragestellungen des verteilten Planens und des Neuplanens ein. Fragestellungen also, die auch zentraler Teil dieser, über 20 Jahre später erstellten, Arbeit sind.

Möhring et.al. präsentieren in [MKGS04] einen Ansatz, der konfliktfreies FTF Routing mit dynamischem Auftragseingang online ermöglicht. Der vorgestellte Algorithmus verhindert Kollisionen, Deadlocks und Lifelocks schon bei der Planung und berücksichtigt die physikalischen Dimensionen der Fahrzeuge. Dafür werden für jede neue Anfrage (also jeden neu eingegangenen Transportauftrag) als erster Schritt Polygone $P(a)$ für jede Kante berechnet, die die durch darauf fahrende FTF geblockten Bereiche repräsentieren. Nach diesem Preprocessing können die physikalischen Ausmaße der FTF beim Berechnen der kürzesten Wege dann ignoriert werden. Das ursprüngliche Problem reduziert sich damit auf ein Shortest Path Problem With Time-Windows (SPPTW - siehe [San94]). In der allgemein Form ist das SPPTW NP-vollständig (siehe [MKGS04]), Möhring et.al. zeigen aber, dass das SPPTW mit Transitzeiten (inklusive Wartezeiten) als Kostenfunktion in polynomieler Zeit lösbar ist. Das von Möhring et.al. präsentierte Verfahren erlaubt das Berechnen einer konfliktfreihen Route in einem Graphen mit 30.000 Knoten unter Berücksichtigung der physikalischen Ausmaße der FTF auf einem AMD-Athlon 2100+ mit 512 MB RAM in unter einer halben Sekunde und ist damit online fähig. Weiterhin liefert das Verfahren im Vergleich mit dem statischen Verfahren wie es auf dem Container Terminal Altenwerder (CTA) des Hamburger Hafens eingesetzt wird, bessere Resultate (also kürzere, durchschnittliche Strecken).

2007 präsentierten ter Mors et.al. einen Algorithmus, der das Problem des konfliktfreien Routings mit einer asymptotischen Laufzeit von $O(nv\ log(nv)+nv^2)$ löst (siehe [MZ07]). In ihrem Ansatz betrachten sie den Fall, dass Agenten nach dem konfliktfreien Planen ihrer Route die benötigten Streckenabschnitte reservieren und somit die Umstände für die Planung der nach ihnen planenden Agenten verändern. Zur Lösung des Problems bauen Moers et.al. folgendes Modell auf: Eine Menge $\mathcal{A}$ von Agenten, eine Menge $\mathcal{R}$ von Ressourcen (zum Beispiel Wegabschnitte, Fahrstühle oder Batterieladestationen) wobei jede Ressource $r_i \in \mathcal{R}$ eine Kapazität $C(r_i)$ hat (die angibt wie viele Agenten gleichzeitig die Ressource benutzen können), eine Nachfolgerelation $\mathcal{S} \subseteq \mathcal{R} \times \mathcal{R}$ (die angibt ob man von einer Ressource zur anderen gelangen kann) sowie die Werte $D(r_i)$ die angeben, wie lange es dauert die Ressource r_i zu traversieren. Der Plan eines Agenten besteht dann aus einer Sequenz von Ressourcen sowie den Zeitintervallen, in denen der Agent diese besucht: $P = ((r_1, \tau_i), \ldots, (r_n, \tau_n))$ wobei folgende Bedingungen an einen zulässigen Plan gestellt werden:

1. Das Intervall τ_j trifft das Intervall τ_{j+1},
2. $|\tau_j| \geq D(r_j)$,
3. $(r_j, r_{j+1}) \in \mathcal{S}$

Bedingung 1 stellt dabei sicher, dass der Plan zeitlich passend ist, dass der Agent also weder eine Lücke noch eine Doppelbelegung in seinem Plan hat, Bedingung 2 sorgt dafür, dass die Zeit, die der Agent für das Traversieren von Ressource r_j einplant, ausreichend ist und Bedingung 3 sorgt dafür, dass es möglich ist von jeder Ressource im Plan zu der danach geplanten Ressource zu gelangen.

Zu diesen statischen Eigenschaften des Problems kommen durch die Reservierungen der Agenten die dynamischen free time windows hinzu. Diese sind wie folgt definiert: Für eine gegebene

Ressource r_i und eine Menge von Reservierungsintervallen $I = \{\tau_1, \ldots, \tau_m\}$ für die Ressource r_i ist ein free time window auf r_i definiert als ein Intervall $f_{i,v} = [\sigma_{i,v}, \phi_{i,v}]$ so dass:

1. $\forall t \in f_{i,v} : |\{\tau_j \in I | t \in \tau_j\}| < C(r_i)$,

2. $(\phi_{i,v} - \sigma_{i,v}) \geq D(r_i)$.

Bedingung 1 sorgt dafür, dass die Ressource in dem free time window genug freie Kapazität hat, Bedingung 2 sorgt dafür, dass das Zeitintervall groß genug ist um den Agenten ein Traversieren der Ressource zu ermöglichen. Damit ist die Menge der F_i der free time windows auf Ressource r_i ein Vektor $(f_{i,1}, \ldots, f_{i,m})$ von disjunkten Intervallen so dass für alle $j \in [1, \ldots, m-1]$ $f_{i,j}$ vor $f_{i,j+1}$ liegt.

Mit den nun gemachten Definitionen ist es möglich die free time window Erreichbarkeit zu definieren: Gegeben eine Ressource r_i, ein free time window $f_{i,v}$ auf dieser Ressource und ein Zeitpunkt t, dann sagen wir, dass das free time window $f_{i,w}$ auf der Ressource r_j zum Zeitpunkt t von r_i aus erreichbar ist, ausgedrückt durch $f_{j,w} \in \rho(r_i, t)$, wenn:

1. $(r_i, r_j) \in \mathcal{S}$,

2. $t \in (f_{i,v} \bigcap f_{j,w})$,

3. $(t - \sigma_{i,v}) \geq D(r_i)$,

4. $(\phi_{j,w} - t) \geq D(r_j)$.

Die dritte Bedingung sorgt dafür, dass $f_{i,v}$ nicht zum Zeitpunkt t verlassen wird, wenn nicht genug Zeit zur Verfügung steht, um r_i zu traversieren. Die vierte Bedingung sorgt dafür, dass genug Zeit zum traversieren von r_j zur Verfügung steht, wenn es von r_i aus betreten wird. Die Menge der free time windows die von $f_{i,v}$ aus erreichbar sind kann nun durch

$$\rho(f_{i,v}) = \bigcup_{t \in [\sigma_{i,v} + D(r_i), \phi_{i,v}]} \rho(r_i, t)$$

angegeben werden.

Mit den so aufgestellten Definitionen geben ter Mors et.al. ihren context-aware routing Algorithmus wie folgt an:

Algorithmus 3 Context-aware routing.

1: **Pre:** Start Ressource r_s, Ziel Ressource r_d, Startzeit t_s. Q:=$\emptyset$;
2: **Post:** Eingangszeit in Ressource r_d für den kürzesten Pfad von r_s nach r_d
3: **if** $\exists v[f_{s,v} \in F_s | t_s \in f_{s,v}$ **then**
4: $Q \leftarrow \{(r_s, t_s)\}$
5: **end if**
6: **while** $Q \neq \emptyset$ **do**
7: $(r_i, t_i) \leftarrow$argmin$_{(r,t)\in Q} t + D(r))$
8: $Q \leftarrow Q \setminus \{(r_i, t_i)\}$
9: **if** $r_i = r_d$ **then**
10: **return** (r_i, t_i)
11: **end if**
12: **for all** $f_{j,v} \in \rho(r_i, t_i)$ **do**
13: $t_{entry} =$max$(t_i + D(r_i), \sigma_{j,v})$
14: **if** constraintsOK(r_i, r_j, t_{entry}) **then**
15: $Q \leftarrow Q \bigcup (r_j, t_{entry})$
16: $F_j \leftarrow F_j \setminus \{f_{j,v}\}$
17: **end if**
18: **end for**
19: **end while**

Dieser Algorithmus expandiert einen Teilplan indem er überprüft, welche free time windows erreicht werden können.

In Zeile 4 wird zunächst die Liste Q der free time windows, mit der Start Ressource und dem Startzeitpunkt initialisiert. In Zeile 7 wird das Listenelement r_i, t_i mit den kleinsten Kosten $t_i + D(r_i)$ bestimmt. Daher sind die Elemente der offenen Liste nach steigenden Ausgangszeiten sortiert. Um das aktuelle free time window zu expandieren werden in Zeile 12 alle (Ressourcen, free time window) Paare, die in $\rho(r_i, t_i)$ liegen, bestimmt. Die Eingangszeit in ein erreichbares free time window $f_{j,v} = [\sigma_{j,v}, \phi_{j,v}]$ ist entweder die Eingangszeit in die vorherige Ressource r_i plus die Zeit, die zum Traversieren von r_i nötig ist, oder falls $f_{j,v}$ nach $t_i + D(r_i)$ startet, die Startzeit $\sigma_{j,v}$ von $f_{j,v}$. In Zeile 14 wird überprüft, ob alle Szenario spezifischen Bedingungen für einen Übergang erfüllt sind. Wenn das der Fall ist, wird in Zeile 15 die offene Liste um das neue Element erweitert und in Zeile 16 das free time window aus der Liste der free time windows der Ressource r_j entfernt. Der letzte Schritt ist wichtig, damit kein free time window mehr als einmal berücksichtigt wird. (Verschiedene free time windows einer Ressource müssen allerdings gesondert berücksichtigt werden.)

In [MZ07] geben ter Mors et.al. sowohl einen Beweis für die Korrektheit des Algorithmus als auch den Beweis, dass der vorgestellte Algorithmus eine asymptotische Laufzeit von $O(nv\ log(nv) + nv^2)$ hat. Weiterhin konnte durch Experimente gezeigt werden, dass die Reihenfolge, in der die Agenten planen, entscheidend ist. Die Kosten eines durchschnittlichen Plans steigen linear mit der Anzahl der Reservierungen im System. Je später ein Agent plant, desto schlechter ist sein Plan also. Durch diese Beobachtung wird deutlich, dass der in dieser Arbeit verfolgte Ansatz, die Gesamtsystem-Performance durch das Einführen von Verhandlungen in der Planungsphase zu verbessern, vielversprechend scheint.

Neben der Entwicklung fortschrittlicher Algorithmen zum konfliktfreien Routing gibt es auch andere Ansätze. So geben Singh et.al in [SSP09] ein Verfahren an, dass basierend auf der

Beobachtung des Bedarfs an FTF an bestimmten Stellen des Layouts (Lagerplätze, Verarbeitungsmaschinen, usw.) das gesamte Layout in sogenannte exklusive Zonen einteilt. Das Verfahren ist insbesondere für Layouts konzipiert worden, bei denen es keine Ausweichstrecken gibt. Nach der Einteilung in exklusive Zonen wird bei diesem Verfahren jeder dieser Zonen genau ein FTF zugeordnet. Bei der Partitionierung ist zu beachten, dass jede Zone auch von einem FTF erfolgreich versorgt werden kann. Der Bedarf an Transportleistung darf also nicht die Transportleistung des zugeordneten Fahrzeugs übersteigen. Neben dieser notwendigen Bedingung wird versucht die geteilten Streckenabschnitte, also Streckenabschnitte, die mehrere Fahrzeuge zum Erreichen ihrer Zonen benötigen, möglichst klein zu halten. Durch diese möglichst kleinen Überlappungen wird das Konflikt Risiko minimiert. Um Konflikte und Deadlocks schließlich ganz auszuschließen müssen die Fahrzeuge die gemeinsam genutzten Gebiete reservieren. Befindet sich ein anderes Fahrzeug in einem gemeinsam genutzten Bereich, so wartet ein anderes Fahrzeug, das diesen Bereich ebenfalls durchfahren will, so lange, bis der Bereich wieder frei ist. In [SSP09] beschreiben Singh et.al. die erfolgreiche Umsetzung des Verfahrens für eine Produktionshalle, die 18 Maschinen sowie einen Versorgungspunkt hat.

Ein weiteres Verfahren stellen Smolic-Rocak et.al. in [SRBZP10] vor. In diesem Verfahren, das sie Time Windows Based Dynamic Routing nennen, werden offline für alle Zielknotenpaare (also alle Kombinationen möglicher Start- und Endpunkte von Transportaufträgen) eine Menge an kurzen Pfaden, zum Beispiel mit dem Dijkstra Algorithmus (siehe Abschnitt 2.5.2), vorberechnet. Wenn im Betrieb eine neue Mission, also ein neuer Transport- oder Fahrauftrag, generiert wird, wird für diese Pfade überprüft, ob sie zulässig sind. Anschließend wird aus den zulässigen Pfaden der kürzeste ausgewählt und reserviert. Die Überprüfung auf Zulässigkeit arbeitet dabei mit Zeitfenstern. Für jede Kante ergibt sich durch die Länge der Kante und der maximalen Geschwindigkeit der Fahrzeuge eine untere Schranke für das Traversieren. In einem ersten Schritt werden alle Kanten mit dieser unteren Schranke eingesetzt. Der so entstandene Plan wird dann auf Überlappungen überprüft. Gibt es solche wird durch ein Warten lassen der Fahrzeuge auf der letzten Kante vor der ersten Überlappung versucht, diese erste Überlappung zu umgehen. Dieser Prozess wird fortgesetzt, bis alle Überlappungen umgegangen sind (wenn das gelingt ist der Pfad ein zulässiger) oder bis die für das Abfahren des Pfades benötigte Zeit größer ist als die noch für die Erfüllung des Auftrages zur Verfügung stehende Zeit (in diesem Fall ist der Pfad unzulässig). Laut [SRBZP10] wurde dieses Verfahren erfolgreich in sechs europäischen Industrieanlagen installiert.

Kapitel 4

Entwurf

Für die Umsetzung der in Kapitel 3 vorgestellten Anforderungen wurde ein Multi-Agenten-System als Grundlage ausgewählt. Zu den Gründen hierfür zählten unter anderem der erfolgreiche Einsatz von Agenten im Rahmen von Staplerleitsystemen (siehe Abschnitt 3.3.1) sowie der große Grad an Freiheit bei der Entwicklung solcher Systeme und die Vorgaben aus dem Projekt "Dezentrale, agentenbasierte Selbststeuerung von Fahrerlosen Transportsystemen"(siehe Kapitel 1).

Für die dezentrale Steuerung mussten folgende Punkte konzipiert und entwickelt werden:

- Prinzipieller Ablauf,
- Agenten,
- dezentrale Auftragsvergabe,
- dezentrale Routenfindung.

Der prinzipielle Ablauf beschreibt wie die Funktionen eines Fahrerlosen Transportsystems umgesetzt werden sollen. Als Ausgangspunkt zur Beschreibung kann das Anfallen eines neuen Auftrags in das System dienen. Hierbei spielt es für das betrachtete System keine Rolle, wie bzw. woher der Auftrag in das System gelangt. Möglich wären hier beispielsweise ein Warehouse Management bzw. ein Enterprise-Resource-Planning System, produzierende Stationen, die selbstständig Aufträge erzeugen, oder die Einspeisung von Aufträgen per Hand, beispielsweise an einem zentralen Rechner oder dezentral an Terminals oder tragbaren Computern. Der Ablauf, der nach der Einspeisung eines Auftrags in das System angestoßen wird, ist in Abbildung 4.1 dargestellt.

Sobald ein neuer Auftrag im System anfällt (in der Abbildung 4.1 oben links), soll der Auftrag als Auktion ausgeschrieben werden. Auktionen wurden als Technik gewählt, da sie ein Standard-Verfahren für die dezentrale Auftragsvergabe darstellen und es für die notwendige Kommunikation bereits einen FIPA-Standard gibt (FIPA-Contract Net, siehe [FIP01]). Nachdem der Auftrag ausgeschrieben wurde geben die Agenten, die die jeweiligen Fahrerlosen Transportfahrzeuge steuern (FTF-Agenten), ein Angebot ab. Anschließend wird der Auftrag dem Fahrzeug zugewiesen, dessen Agent das beste Angebot abgegeben hat. Dieses Fahrzeug integriert den Auftrag in seinen Plan und fährt ihn schließlich oder gibt ihn ggf. zurück. Die dezentrale Routenfindung ist bereits in den oben beschriebenen Ablauf integriert. Die Angebote, die die Agenten im Schritt oben rechts in der Abbildung 4.1 abgeben, beziehen

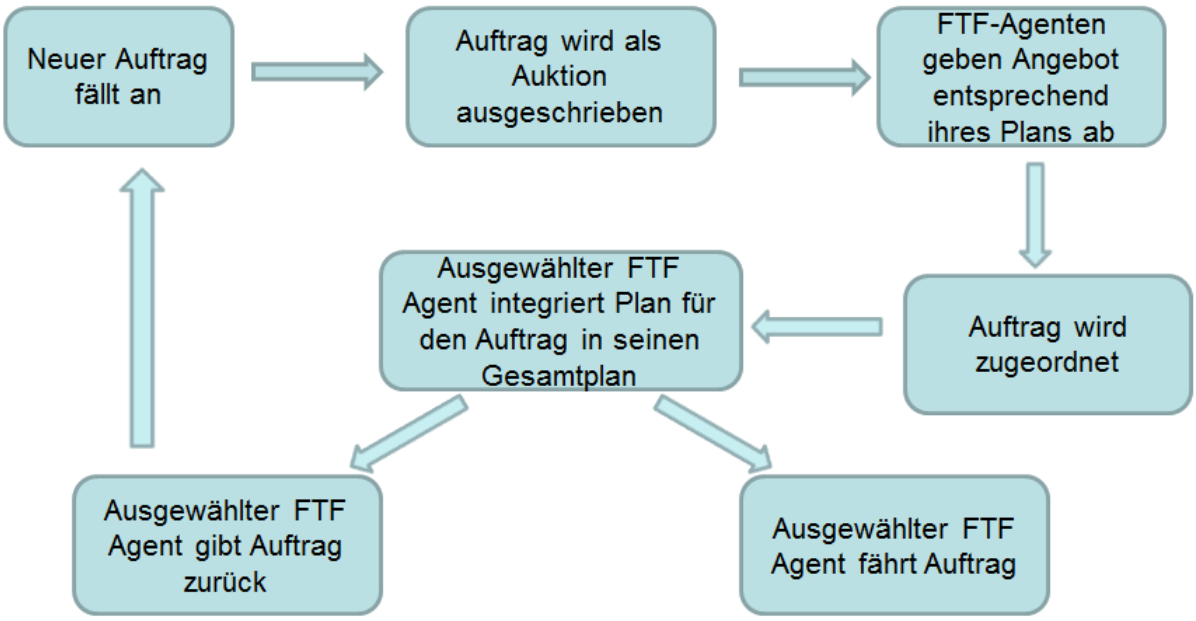


Abbildung 4.1: Prinzipieller Ablauf der Auftragsvergabe und Auftragsabarbeitung

sich immer auf den (Teil-)Plan, den die Agenten für die Abarbeitung des Auftrags entwickeln. Zu diesem Teilplan gehört immer die Anfahrt zum Auftragsstartpunkt und für diese Anfahrt muss der Agent eine Route planen. Das Planen der Routen ist also Teil der Berechnung des Angebots.

In Abbildung 4.2 ist das Konzept für das Gesamtsystem dargestellt. Die Basis des Konzepts ist die Agentenplattform. Diese stellt die erforderlichen Strukturen für ein Multiagentensystem zur Verfügung. Die wichtigsten Komponenten hier sind der Scheduler, der die Simulation taktet und die Agenten verwaltet, der MessageSender, der den Agenten die Möglichkeit der Kommunikation zur Verfügung stellt, und der Graph, der neben Knoten und Kanten auch noch Reservierungen zur Verfügung stellt. Auf dieser Agentenplattform laufen dann mehrere Instanzen des FTF-Agenten (ein Agent pro simuliertes Fahrzeug) sowie ein oder mehrere Instanzen des OrderDisposer. Die FTF-Agenten enthalten die für das jeweilige Fahrzeug notwendigen Informationen wie den aktuellen Plan, die aktuelle Position, den Status der Batterie und die dem Fahrzeug zugeordneten Aufträge. Weiterhin verfügt ein FTF-Agent über Methoden zur Routenplanung, zum Durchführen von Verhandlungen und zur Beteiligung an Auktionen zur Auftragsvergabe. Die OrderDisposer-Agenten verfügen über eine Liste aller Aufträge, die sie vergeben müssen, sowie einer Zuordnung zwischen den bereits vergebenen Aufträgen und den FTF-Agenten an die diese vergeben wurden. Weiterhin verfügen die OrderDisposer-Agenten über Methoden zur Vergabe von Aufträgen und zur Bestimmung von Kennzahlen. Die GUI dient zur Darstellung der Abläufe im System. Sie liest die Position der Agenten sowie der Aufträge aus und bildet diese auf dem statischen Layout des jeweiligen Szenarios ab. Eine Rückkopplung von der GUI in die Agenten findet nicht statt. Die einzelnen Komponenten des Gesamtsystems werden in den folgenden Abschnitten genauer beschrieben.

Im folgenden Abschnitt wird zunächst der Graph, der als Grundlage für die Szenarienmodellierung und für die Routenplanung dient, beschrieben. Anschließend werden in Abschnitt

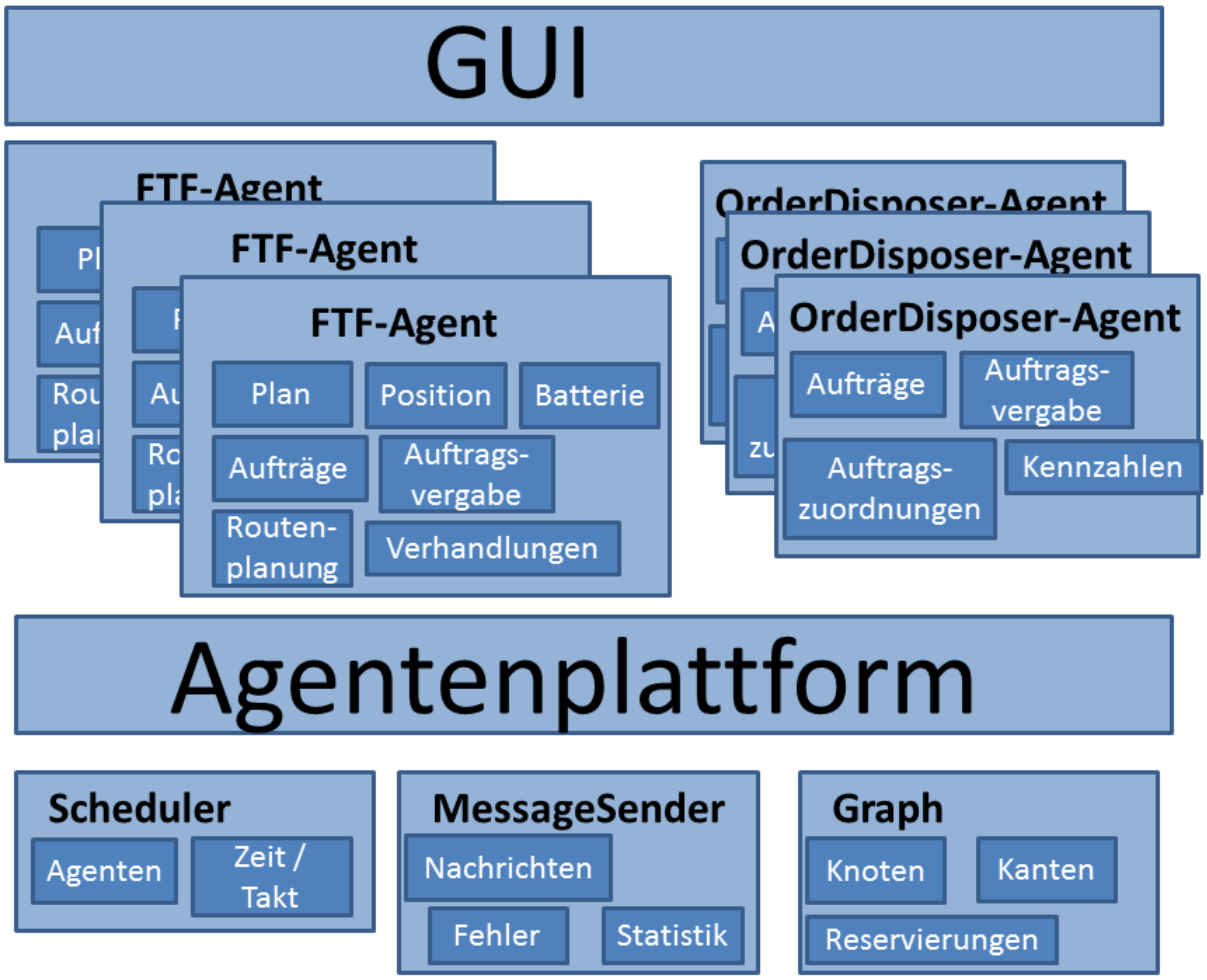
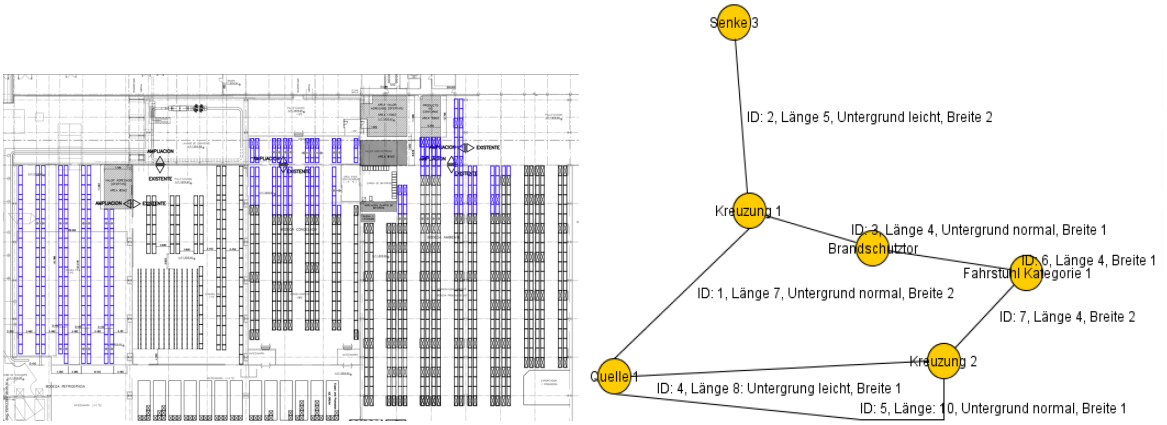


Abbildung 4.2: Darstellung des Gesamtsystems

4.2 Aufträge, Pläne und Aktionen als grundlegende logische Elemente für die Handlungen der Agenten beschrieben. Die konzipierten Agenten werden dann in Abschnitt 4.3 beschrieben. Anschließend werden die für die Umsetzung benötigten Konzepte der Kommunikation zwischen den Agenten (Abschnitt 4.4) und der Konflikt und dessen Erkennung (Abschnitt 4.5) beschrieben. Sehr umfangreich werden dann die beiden Komponenten der entwickelten Steuerung beschrieben, die dezentrale Auftragsvergabe in Abschnitt 4.6 und die dezentrale Routenplanung in Abschnitt 4.7. Am Ende des Kapitels wird der Entwurf der Simulation beschrieben.

4.1 Modellierung des Layouts mit Hilfe von Graphen

Als Grundlage für die Modellierung der Szenarien werden Graphen verwendet. Der Hauptgrund ist die deutlich geringere Komplexität bei der Planung und Wegfindung sowie der geringere Aufwand zum Modellieren von Szenarien. Abbildung 4.3 verdeutlicht das. In Teilabbildung a) ist das Layout einer Warenhalle dargestellt. Das exakte Nachbauen wäre relativ aufwändig. Das Planen sowie Abfahren von Wegen würde eher ins Feld des Robot Motion



(a) Ein komplexes Layout (b) Ein einfacher Graph

Abbildung 4.3: Graphen führen zu geringere Komplexität

Planning gehören, als in das Feld der FTS Steuerung. In Teilabbildung b) ist ein (Teil-)Szenario als Graph dargestellt. Die Knoten stehen für Kreuzungen, Quellen und Senken, die Kanten sind mit zusätzlichen Informationen annotiert. Das Finden von Wegen ist hier in der Regel gleichbedeutend mit dem Finden eines zulässigen kürzesten Weges in einem Graphen, ein Problem das normalerweise weniger berechnungsintensiv als das Robot Motion Planning ist.

Den Vorteilen der Nutzung eines Graphens, also den niedrigeren Berechnungsaufwand, das einfachere Modellieren der Graphen sowie die Ausnutzbarkeit von Methoden aus der Graphentheorie steht das Risiko gegenüber, relevante Eigenschaften des Layouts durch die Abstraktion auf einen Graphen zu verlieren. So kann es im Layout beispielsweise Wege mit verschiedenen Untergründen (einem härteren und einem weniger harten) geben. Die Folge könnte sein, dass beladene Fahrzeuge nur auf den Wegen mit härterem Untergrund fahren können, wohingegen die Wege mit dem weniger harten Untergrund nur von nicht beladenen Fahrzeugen genutzt werden können. Solche Informationen müssen im Graphen mit modelliert und bei den Algorithmen zum Finden von Wegen verwendet werden.

4.1.1 Aufbau des Graphen

Die Grundbestandteile des genutzten Graphen sind, wie bei im Prinzip jedem Graphen, Knoten und Kanten. Dazu kommen noch sogenannte Locks. Diese repräsentieren die Sperrungen auf Knoten und Kanten. Knoten und Kanten sind in der Implementation beide Unterklassen der Klasse Ressource. Locks sind allgemein für Ressourcen implementiert, so dass bei der Verwendung von ihnen für viele Fragestellungen nicht zwischen Knoten und Kanten unterschieden werden muss. Der letzte Teil sind Annotationen, die sowohl Knoten als auch Kanten hinzugefügt werden können.

4.1.1.1 Graph

Die Klasse Graph repräsentiert den kompletten Graphen. Er besteht im wesentlichen aus einer Liste Knoten sowie einer Liste Kanten. Außerdem sind in der Klasse diverse Hilfsfunktionen

implementiert, zum Beispiel für:

- Die Ausgabe aller Knoten mit bestimmten Eigenschaften,
- Die Ausgabe aller Kanten mit bestimmten Eigenschaften,
- Die Ausgabe aller aktuellen Locks,
- Die Ausgabe aller Locks, für die bestimmte Eigenschaften gelten,
- Das Löschen aller Locks mit bestimmten Eigenschaften.

Das Filtern von Knoten und Kanten geschieht hierbei zum Beispiel nach geographischen Relationen (gib mir alle Knoten die in einem Radius von r um den Punkt x,y liegen) oder nach den Sperrungen auf den Ressourcen (gib mir alle Kanten, die mindestens eine Reservierung im Zeitraum $[t_0, t_1]$ haben). Das Filtern von Locks geschieht zum Beispiel nach der Zeit (gib mir alle Locks die im Zeitraum $[t_0, t_1]$ liegen) oder nach den sperrenden Agenten (gib mir alle Locks die den Agenten mit den IDs 1,3 und 7 gehören).

4.1.1.2 Knoten

Knoten stehen im Graph für Kreuzungen sowie wichtige Orte wie Parkplätze, Batterieladestationen, Quellen und Senken. Folgende Daten sind mit jedem Knoten gespeichert:

- x,y Koordinaten,
- Label,
- eindeutige ID,
- Liste der eingehenden Kanten,
- Liste der ausgehenden Kanten,
- Liste der Locks auf dem Knoten,
- Liste der Agenten auf dem Knoten.

Das Label stellt einfach eine Bezeichnung für den Knoten dar und hilft, den Aufbau des Graphen und den Aufbau von Aufträgen lesbarer zu machen. Die Liste der Agenten auf dem Knoten enthält alle Agenten, die zum aktuellen Zeitpunkt auf dem Knoten sind. Es handelt sich also im Gegensatz zu den statischen Listen der ein- und ausgehenden Kanten um eine dynamische Liste.

4.1.1.3 Kanten

Kanten verbinden je exakt zwei Knoten miteinander. Die Kanten im Graph sind gerichtet. Ist ein Weg zwischen zwei Knoten in beide Richtungen befahrbar, so gibt es zwei Kanten zwischen den Knoten, jeweils eine in jede Richtung. Folgende Daten sind mit jeder Kante gespeichert:

- Länge der Kante,
- Label,

- (eindeutige) ID,
- Startknoten,
- Zielknoten
- Liste der Locks auf der Kante,
- Liste der Agenten auf der Kante.

Die Länge der Kante ist in diesem Fall einfach die euklidische Distanz zwischen dem Start- und dem Zielknoten. Soll zum Beispiel modelliert werden, dass ein Fahrzeug auf diesem Wegstück nicht mit maximaler Geschwindigkeit fahren kann, so wird dies durch Annotationen auf der Kante (siehe Abschnitt 4.1.1.5 modelliert und nicht durch ein Verändern der Kantenlänge.

4.1.1.4 Locks

Locks modellieren Reservierungen von Ressourcen (also von Knoten und Kanten). Ein Lock besteht aus folgenden Daten:

- ID des sperrenden Agenten
- Zeitintervall der Reservierung
- Typ

Der Typ kann die Art der Sperrung genauer definieren. So kann beispielsweise zwischen einer Sperrung für eine Fahrt zu einem Parkplatz und zwischen einer Sperrung für einen Auftrag unterschieden werden. Der Typ eines Locks kann so in Verhandlungen berücksichtigt werden.

4.1.1.5 Annotationen

Alle weiteren Eigenschaften des Layouts werden durch Annotationen an Knoten und Kanten modelliert. Zum Beispiel könnte die Eigenschaft *Härte des Untergrunds* durch eine Annotation *Härte* vom Typ boolean modelliert werden. Der Werte *true* würde dann für einen harten, der Wert *false* für einen weichen Untergrund stehen. Eine Modellierung als numerischer Wert um beispielsweise den maximal erlaubten Druck auf den Untergrund abzubilden wäre auch möglich. Ein weiteres Beispiel für eine Annotation wäre die Breite eines Weges. Diese könnte zum Beispiel in Metern angegeben sein oder als Kapazität, also als Angabe, wie viele Fahrzeuge nebeneinander im Weg fahren können.

Eine Annotation im Graph ist also einfach ein Wertetupel $\{Bezeichner, Typ, Wert\}$, wobei der Bezeichner einfach ein String ist (Breite, Härte). Jeder Ressource können beliebig viele solcher Annotationen hinzugefügt werden, die anschließend veränder- und abrufbar sind.

Die Berücksichtigung der mit diesen Annotationen modellierten Sachverhalte muss in den jeweiligen Methoden per Hand erfolgen. Soll ein Algorithmus zur Wegfindung beispielsweise Kanten mit der Annotation $\{Härte, boolean, false\}$ ignorieren, so müssen diese weichen Kanten im Algorithmus ausgenommen werden.

4.2 Aufträge, Aktionen, Pläne

Die Handlung der transportierenden Agenten wird maßgeblich von den ihnen zugewiesenen Aufträgen bestimmt. Für die Erfüllung eines solchen Auftrags konstruiert ein Agent dann einen Plan, der wiederum aus mehreren Aktionen besteht. In den folgenden Abschnitten werden die Aufträge (Abschnitt 4.2.1), Aktionen (Abschnitt 4.2.2) und Pläne (Abschnitt 4.2.3) näher beschrieben.

4.2.1 Aufträge

Ein Auftrag ist eine konkrete Anweisung an einen Agent. Es kann zwischen verschiedenen Arten von Aufträgen unterschieden werden: Transportaufträge, Parkaufträge, Batterieladeaufträge. Jede dieser Auftragsunterklassen kapselt die für ihren Typ zur Durchführung notwendigen Informationen. Zwingend ist eine eindeutige ID (OrderID). Für den Parkauftrag ist ansonsten nur der zugewiesene Parkplatz (angegeben als Knoten im Graphen) notwendig. Für den Batterieladeauftrag ist das die zugewiesene Ladestation (angeben als Knoten im Graphen) sowie die Ladedauer. Für den Transportauftrag ist das zwingend der Startpunkt (die Quelle - angegeben als Knoten im Graphen) sowie der Endpunkt (die Senke - angegeben als Knoten im Graphen). Optional kann ein Transportauftrag noch weitere Daten enthalten, wie den Typen des Transportgutes (falls es unterschiedliche Fahrzeuge im System gibt und nicht alle Fahrzeuge jedes Transportgut transportieren können, oder falls der Transporttyp Auswirkungen auf Größen wie die Lade- und Entladedauer, die maximale Geschwindigkeit des FTF oder ähnliches hat) oder eine Deadline, bis zu der der Auftrag spätestens abgearbeitet sein sollte.

Weiterhin werden für alle Aufträge noch Daten zur späteren Auswertung der Simulation gespeichert. Dies sind die Erstellungszeit, der Auftragsbeginn, der Transportbeginn, das Transportende, das Auftragsende, sowie die zugewiesenen Agenten. Die Erstellungszeit gibt hier an, wann der Auftrag vom OrderDisposer oder dem Agenten selbst (im Fall von Park- und Ladeaufträgen) erstellt wurde. Der Auftragsbeginn gibt die Zeit an, zu der ein Agent tatsächlich mit der Bearbeitung des Auftrags begonnen hat. Diese Zeit kann identisch mit der Erstellungszeit sein, kann aber auch deutlich dahinter liegen, wenn der zugewiesene Agent zwischen der Zuweisung des Auftrags und dem Beginn noch ein oder mehrere Aufträge bearbeitet hat. Die zugewiesenen Agenten sind in der chronologischen Reihenfolge der Zuordnung gespeichert. In den meisten Simulationen ist ein Auftrag allerdings im Laufe der Simulation nur einem (oder keinem) Agenten zugeordnet worden. In einigen Simulationen kann es aber vorkommen, dass ein Agent einen ihm einmal zugeordneten Auftrag wieder zurück gibt und dass dieser dann erneut ausgeschrieben und dadurch einem anderen Agenten zugewiesen wird.

Transportaufträge werden einem Agenten durch die Auftragsvergabe, also durch den OrderDisposer-Agenten (siehe Abschnitt 4.3.2) vergeben, Parkaufträge und Ladeaufträge erzeugt der Agent selbst (Parkaufträge, wenn der Agent idle ist, also nichts zu tun hat und Ladeaufträge, wenn der Batterieladestand des Agenten unter eine festgelegte Schwelle fällt).

4.2.2 Aktionen

Aktionen sind die möglichen Handlungen, die ein Agent ausführen kann. Eine Aktion ist nicht mehr in kleinere Teile zerlegbar. Ein Agent kann über mehrere in einer Liste gespeicherte Aktionen verfügen. Es ist aber immer nur eine Aktion im Moment aktiv (die oberste der Liste). Wenn die aktuelle Aktion beendet wurde, wird die nächste Aktion in der Liste durchgeführt.

Folgende Aktionen werden für die Erfüllung von Transportaufträgen benötigt:

- GoTo Action,
- Load Action,
- UnLoad Action.

Außerdem können noch Aktionen für das Reservieren und Freigeben von Ressourcen notwendig sein:

- RequestResource Action
- FreeResource Action

Falls die Batterie oder der Tank eines Fahrzeugs nicht groß genug sein sollte, damit es ohne Nachladen eine ganze Schicht lang operieren kann, werden Aktionen zum Nachladen notwendig:

- StartRecharge Action,
- StopRecharge Action.

Weiterhin kann noch eine Aktion für das Parken benötigt werden:

- Park Action

Als eine Art Hilfs-Aktion wird außerdem noch eine Wait-Aktion benötigt.

In den folgenden Abschnitten werden die einzelnen Aktionen genauer beschrieben.

4.2.2.1 GoTo Action

Eine GoTo Action fordert den Agenten auf zu einem bestimmten Knoten zu fahren. Der Zielknoten ist Teil der Aktion. Da die Aktionen die kleinsten Einheiten der Logik sind, führt eine GoTo Action immer nur zum nächsten Knoten. Die Aktion *GoTo Knoten x* setzt also voraus, dass der Agent entweder auf einem *Knoten y* ist für den eine Kante Knoten y $\rightarrow$ Knoten x existiert, oder er sich bereits auf einer solchen Kante befindet.

4.2.2.2 LoadAction und UnLoadAction

Die LoadAction beschreibt das Laden eines Transportgutes, die UnLoadAction das Entladen eines Transportgutes. Beide Actions bestehen aus einer (eindeutigen) OrderID, die den zu ladenden oder entladenden Auftrag angibt sowie aus einer Dauer. Diese Dauer kann für jedes Transportgut, also für jeden Auftrag identisch sein oder abhängig von Fahrzeugtyp und / oder Transportguttyp unterschiedlich sein. Wenn ein Agent eine LoadAction oder UnLoadAction durchführt verbringt er die unter Dauer hinterlegte Anzahl an Zeitschritten an seiner aktuellen

Stelle (also einer Quelle oder Senke) und ändert anschließend seinen Beladungszustand zu beladen (im Falle einer LoadAction) oder unbeladen (im Falle einer UnLoadAction).

In der Praxis besteht ein Lade- oder Entladevorgang in der Regel aus mehreren Teilschritten: Der Feinanfahrt an die Station, dem Justieren des Ladegeschirrs, dem Andocken an die Station, der Transportgutaufnahme oder -abgabe, dem Lösen von der Station, der Feinabfahrt und dem Senken des Ladegeschirrs bzw. aus einigen dieser Teilschritte. Da der Schwerpunkt dieser Arbeit auf der Ebene des Planens und nicht auf der Ebene der Maschinensteuerung liegt, wurden all diese Einzelschritte zu einer LoadAction bzw. einer UnLoadAction mit kumulierter Dauer zusammengefasst.

4.2.2.3 RequestRessourceAction und FreeRessourceAction

Diese Aktionen sind für die Reservierung und Freigabe von Ressourcen, die nicht im direkten Zusammenhang mit dem Fahren bzw. dem Planen von Wegen in Zusammenhang stehen, zuständig. Solche Ressourcen können beispielsweise Parkplätze oder Batterieladestationen sein. Wenn eine solche Aktion anliegt, schickt der Agent eine Nachricht an die Stelle, die für die Reservierung dieser Ressourcen zuständig ist (siehe 5.4.2). Erhält der Agent die gewünschte Ressource, fährt er mit dem Plan fort. Erhält er keine der gewünschten Ressource, verwirft er den Plan und fährt mit dem nächsten Auftrag fort (oder wechselt in den idle-Modus, wenn er keinen weiteren Auftrag zu bearbeiten hat).

Mit der FreeRessourceAction gibt ein Agent eine Ressource wieder frei.

4.2.2.4 StartRechargeAction und StopRechargeAction

Mit der StartRechargeAction beginnt ein Agent das Aufladen seiner Batterie. Die StartRechargeAction besteht dabei aus zwei Werten: Der Anzahl der benötigten Simulationsschritte für das Verbinden mit der Ladestation sowie der Laderate pro Zeitschritt. In der vorliegenden Arbeit wurde von einer konstanten Laderate pro Zeitschritt ausgegangen. Komplizierteres Ladeverhalten ließe sich, wenn benötigt, aber auch realisieren.

Mit der StopRechargeAction beendet ein Agent den Ladevorgang. Diese Action gibt an, wie lange es dauert, bis der Agent sich wieder von der Ladestation getrennt hat.

Ein Agent lädt also aus der Folge StartRechargeAction, StopRechargeAction folgende Energie: (StopRechargeAction Startzeitpunkt - StartRechargeAction Startzeitpunkt - Benötigte Anzahl an Simulationsschritten für das Verbinden mit der Ladestation) $\times$ Energie pro Zeitschritt.

4.2.2.5 ParkAction

Die ParkAction ist für das Starten des Parkmodus zuständig. Im Parkmodus führt der Agent keine weiteren Aktionen aus. Das Durchführen der Aktion setzt im Agenten die idle-Flag. Weiterhin meldet sich der Agent beim Scheduler ab. Der Parkmodus (idle-mode) wird nur auf Grund von externen Ereignissen wieder beendet. Das am häufigsten auftretende Beispiel ist der Fall, in dem der Agent eine Auktion gewinnt und ihm dadurch ein neuer Transportauftrag zugewiesen wird.

4.2.2.6 WaitAction

Die WaitAction beschreibt ein zeitlich begrenztes Warten an der aktuellen Position (im Unterschied zur ParkAction, die zeitlich unbegrenzt ist). Die zu wartende Zeit ist Teil der Aktion. Das Warten ist insbesondere bei der Konfliktfreien Planung (siehe Abschnitt 4.7.2) und bei der Kooperativen Planung (siehe Abschnitt 4.7.3) notwendig.

4.2.3 Pläne

Ein Plan ist nun ein eine Liste von Aktionen die nacheinander ausgeführt ein bestimmtes Ziel erfüllen. Zum Beispiel könnte ein Plan, dessen Ziel die Durchführung eines Transportauftrags ist und bei der der Agent auf Knoten 1 startet, wie folgt aussehen:

{GoTo Knoten 3, Wait 3, GoTo Quelle 1, Load, GoTo Knoten 5, Wait 4, GoTo Senke 1, UnLoad }

In diesem Beispiel startet der Agent auf Knoten 1 fährt dann die Kante $_{\text{Knoten1} \rightarrow \text{Knoten3}}$ zu Knoten 3. Dort wartet er drei Zeiteinheiten und fährt anschließend auf der Kante $_{\text{Knoten3} \rightarrow \text{Quelle1}}$ zur Quelle 1. Anschließend lädt er dann das dort vorliegende Transportgut auf und fährt dann zu Knoten 5. Dort wartet er 4 Zeiteinheiten, fährt dann zur Senke 1 und entlädt das Transportgut.

Die Wartezeiten sind im konfliktfreien Routing begründet. Im obigen Beispiel wäre es so, dass die Ressource Kante $_{\text{Knoten3} \rightarrow \text{Knoten5}}$ erst drei Zeiteinheiten nach dem Erreichen von Knoten 3 zur Verfügung steht. Als Konsequenz wartet der Agent drei Zeiteinheiten auf der Vorgänger Ressource. In der Reservierung der Ressourcen muss diese Vorgängerressource dementsprechend drei Einheiten länger reserviert werden (siehe Konflikt freies Routing in Abschnitt 4.7.2).

Sollen die in dieser Arbeit entwickelten Methoden zum kooperativen Planen von Wegen und Aufträgen in Multi-Agenten basierten FTS in die Praxis übertragen werden, kann das Warten auch durch ein Verlangsamen auf der oder den Vorgängerressourcen realisiert werden. Beispielsweise wäre es möglich, anstatt dass der Agent die Vorgängerkante der Länge 3 mit der Geschwindigkeit 1 fährt und anschließend 3 Sekunden wartet möglich, dass er die Vorgängerkante mit der Geschwindigkeit 0.5 fährt. Eine solche Implementierung von Wartezeiten durch verlangsamtes Fahren kann ggf. den Energieverbrauch senken (Anfahrtskosten bzw. Beschleunigungskosten).

Der aktuelle Agentenplan besteht aus der Liste aller aktuell hinterlegten Pläne (der Agent speichert je einen Plan für jeden übernommenen Auftrag).

Die Art, auf die ein Plan komponiert wird, hängt von der Art des zugewiesenen Auftrags ab. Für einen Transportauftrag sind, wie im Beispiel gezeigt, folgende Aktionen notwendig:

1. GoTo Actions und ggf. Wait Actions für die Fahrt zur Quelle,
2. Laden das Transportgut (LoadAction),
3. GoTo Actions und ggf. Wait Actions für die Fahrt zur Senke,
4. Entladen das Transportgut (UnLoadAction).

Für einen Parkauftrag sind das:

1. Reservierung eines Parkplatzes (RequestRessourceAction),
2. GoTo Actions und ggf. Wait Actions für die Fahrt zum Parkplatz,.
3. ParkAction

Für einen Batterieladeauftrag sind das:

1. Reservierung einer Ladestation (RequestRessourceAction),
2. GoTo Actions und ggf. Wait Actions für die Fahrt zur Ladestation,
3. Starten Ladevorgang (StartRechargeAction),
4. Beenden Ladevorgang (StopRechargeAction),
5. Freigeben der Ladestation (FreeRessourceAction).

Wenn ein Plan aktiv geschaltet wird, werden die Aktionen, aus denen der Plan besteht der Action-Liste des Agenten hinzugefügt.

4.3 Agenten

Die dezentrale Steuerung soll durch den Einsatz von Agenten implementiert werden. Agenten bieten sich für eine dezentrale Steuerung von Industrieanlagen wie einem Transportsystem an (siehe zum Beispiel [MM05]). Die Agenten im System sollen hier die logische Planung und Steuerung übernehmen. Die Feinsteuerung der Bewegung, also das konkrete Schalten von Aktuatoren zur Beschleunigung, Lenkung, Gabelsteuerung, kurz das Motion Planing (oder Robot Motion Planung, siehe zum Beispiel [Lat91]), ist nicht Teil der vorliegenden Arbeit. Hier wird davon ausgegangen, dass die Feinsteuerung sowie alle Funktionen der Hardware funktionieren.

Die beiden wichtigsten Agenten, die im Zusammenspiel die Auftragsvergabe und -abwicklung erledigen, sind der BasicFTFAgent und der OrderDisposer die in den folgenden Abschnitten genauer beschrieben werden.

4.3.1 BasicFTFAgent

Der BasicFTF Agent repräsentiert die Basis-Logik, die für die Annahme und Planung von Transportaufträgen notwendig ist. Er enthält folgende Daten:

- Aktuelle Position
- Beladungszustand
- Auftrags-Liste
- Plan-Liste
- Action-Liste
- Kennzahlen

Seine aktuelle Position besteht zum einen aus einem Knoten oder einer Kante sowie dem Fortschritt in Prozent, den er bereits beim Abfahren der Kante gemacht hat sowie aus x/y-Koordinaten. Der Beladungszustand gibt an, ob das Fahrzeug gerade einen Auftrag geladen hat (beladen) oder nicht (unbeladen). Wenn die Liste an Aufträgen, die dem Agenten gerade zugeordnet sind, nicht leer ist, so ist der oberste Auftrag der Liste der gerade aktuelle Auftrag. Für jeden Auftrag in seiner Liste enthält die Plan-Liste den Plan, der für die Erfüllung des jeweiligen Auftrags abgearbeitet werden muss. Die Action-Liste enthält alle elementaren Aktionen, die der Agent für seine Pläne durchführen muss. Neben diesen für die eigentlichen Transportaufgaben relevanten Daten, aktualisiert der Agent am Ende eines jeden Schrittes noch folgende Kennzahlen: Zeitschritte, in denen leer gefahren wurde, Zeitschritte, in denen beladen gefahren wurde, Zeitschritte, in denen Beladen wurde, Zeitschritte, in denen Entladen wurde, Zeitschritte, in denen nichts getan wurde (Parken, idle). In den in dieser Arbeit gemachten Untersuchungen wurden Fahrerlose Transportfahrzeuge betrachtet, die ohne ein Nachladen oder Nachtanken eine komplette Schicht durch operieren konnten. Für Untersuchungen, in denen das nicht der Fall ist, enthält der Agent noch den aktuellen Ladestatus der Batterie bzw. die Restkapazität des Tanks als Kennzahl.

Bei jedem Aufruf überprüft der Agent zunächst, ob er eine Aktion (siehe Abschnitt 4.2) zugewiesen hat oder nicht. Wenn er eine zugewiesen hat, führt er einen Zeitschritt lang die oberste Aktion durch. Falls er dabei eine Aktion beendet, entfernt er sie aus seiner Action-Liste und beginnt mit der nächsten, falls er in diesem Zeitraum noch Zeit dafür hat. Wenn der Agent aktuell keine Aktion zugewiesen hat, verbringt er den aktuellen Zeitraum im idle Modus, er tut also nichts.

Der BasicFTFAgent ist Empfänger von OrderAnnouncement Nachrichten (siehe Abschnitt 4.4.1). Wird dem Agenten eine solche Nachricht zugestellt, berechnet er ein Angebot für den Auftrag (in Abhängigkeit von der benutzten Auftragsvergabe - siehe Abschnitt 4.6) und schickt dieses mit einer Proposal Nachricht (siehe Abschnitt 4.4.2) zurück an den Agenten, der ihm die OrderAnnouncement Nachricht geschickt hat. Um dieses Angebot berechnen zu können, plant der Agent zu diesem Zeitpunkt bereits alle notwendigen Aktionen, um den Plan durchführen zu können, also die Anfahrt zum Startpunkt des Auftrags, das Aufnehmen des Transportguts, die Fahrt zum Endpunkt des Auftrags und das Entladen des Transportguts sowie möglicherweise Wartezeiten, um auf die Pläne anderer Agenten Rücksicht zu nehmen. Falls Verhandlungen um Reservierungen aktiviert sind (siehe Abschnitt 4.7), werden auch diese bereits in diesem Schritt durchgeführt. Dieser Plan wird zusammen mit der OrderAnnouncement Nachricht gespeichert. Weiterhin reserviert der Agent mittels LockChange Nachrichten (siehe Abschnitt 4.4.6) auch schon zu diesem Zeitpunkt alle notwendigen Ressourcen für die Durchführung des Plans.

Erhält der Agent eine ProposalAccepted Nachricht (siehe Abschnitt 4.4.3), schaltet er den für diesen Auftrag hinterlegten Plan aktiv. Hierfür werden die Aktionen, aus denen der Plan besteht, der Action-Liste des Agenten am Ende zugefügt, der Auftrag der Auftrags-Liste hinzugefügt und der Plan der Plan-Liste hinzugefügt.

Erhält der Agent eine ProposalRejected Nachricht (siehe Abschnitt 4.4.4), löscht der Agent den für diesen Auftrag hinterlegten Plan und gibt die Reservierungen, die er für diesen Plan gemacht hat, mittels LockChange Nachrichten (siehe Abschnitt 4.4.6) wieder frei.

Sind Verhandlungen in der Simulation aktiv, so kann der BasicFTFAgent auch LockNegotiation Nachrichten (siehe Abschnitt 4.4.7) erhalten. Nach der Berechnung seiner Antwort schickt

der Agent diese dann an den anfragenden Agenten in Form einer LockNegotiationAnswer Nachricht (siehe 4.4.8).

4.3.2 OrderDisposerAgent

Der OrderDisposerAgent schreibt Transportaufträge aus. Dafür überprüft er in seiner step Methode, ob zum aktuellen Zeitpunkt ein neuer Auftrag ausgeschrieben werden muss. Ist das der Fall, schickt er eine OrderAnnouncement (siehe Abschnitt 4.4.1) Nachricht an alle für den Auftrag in Frage kommenden Agenten. Das können zum Beispiel alle BasicFTFAgenten in der aktuellen Simulation sein oder alle Agenten innerhalb eines bestimmten Radius um den Auftragsstartpunkt. Anschließend wartet der OrderDisposerAgent darauf, dass er von allen angeschriebenen Agenten eine Proposal Nachricht (siehe Abschnitt 4.4.2) erhalten hat. Er wertet die Angebote dann aus und ermittelt das beste Angebot. Anschließend schickt er dem Agenten, der das beste Angebot verschickt hat, eine ProposalAccepted Nachricht (siehe Abschnitt 4.4.3) und allen anderen Agenten, die ein Angebot abgegeben haben, eine ProposalRejected Nachricht (siehe Abschnitt 4.4.4). Außerdem trägt er sowohl in der Order (siehe Abschnitt 4.2.1) und im OrderTracker (siehe Abschnitt 4.8.3) die (eindeutige) ID des Agenten ein, dem der Auftrag zugewiesen wurde. Wurde für den Auftrag kein Angebot eingereicht, wird der Auftrag zu einem späteren Zeitpunkt neu ausgeschrieben. Wie viel später ist ein einstellbarer Parameter.

Die Art und Weise in der die Aufträge in den OrderDisposer eingehen kann sich hier unterscheiden:

- Zufällig erzeugte Aufträge,
- in einer Text-Datei hinterlegte Aufträge,
- in einer Datenbank hinterlegte Aufträge.

In jedem Fall wird die Liste der zu vergebenden Aufträgen zum Simulationsbeginn vollständig eingelesen bzw. erzeugt, so dass alle zu erzeugenden Aufträge von Beginn an vorliegen. Werden im Laufe der Simulation Aufträge von BasicFTFAgenten an den OrderDisposer zurückgegeben, so werden diese Aufträge neu vergeben. Dafür werden sie neu in die Liste der zu vergebenden Aufträge eingefügt.

4.4 Nachrichten

In einem dezentralen System ist die Möglichkeit der Kommunikation zwischen den Entitäten in der Regel unabdingbar, wenn die verschiedenen Entitäten eine gemeinsame Aufgabe bearbeiten sollen. In dem in dieser Arbeit entwickelten, auf Agenten basierenden Lösungsansatz wurde eine Kommunikation, die auf dem Austausch von Nachrichten basiert, entwickelt.

Agenten können hier einem oder mehreren anderen Agenten Nachrichten verschiedener Typen schicken. Die empfangenden Agenten reagieren auf den Erhalt der Nachricht in Abhängigkeit vom Typ der Nachricht.

Die für die Auftragsvergabe notwendigen Nachrichtentypen sind:

- OrderAnnouncement,

- Proposal,
- ProposalAccepted,
- ProposalRejected,
- ReturnOrder.

Neben den Nachrichten für die Auftragsvergabe werden noch Nachrichten für das konfliktfreie Routing und die Verhandlungen zwischen den Agenten benötigt:

- LockChange,
- LockNegotiation,
- LockNegotiationAnswer.

In den folgenden Abschnitten werden die einzelnen Nachrichtenarten kurz beschrieben.

4.4.1 OrderAnnouncement

Mit dieser Nachricht werden Agenten über den Start eine Auktion informiert. Die Nachricht enthält die *Order* die versteigert wird sowie eine Deadline bis zu der Angebote beim Auktionator eingegangen sein müssen um noch berücksichtigt zu werden.

4.4.2 Proposal

Die Transportagenten verschicken Nachrichten vom Typ *Proposal*, um an einer Auktion teilzunehmen. In der Nachricht ist die Order enthalten, auf die sie bieten sowie das Angebot, modelliert durch eine Gleitkommazahl.

4.4.3 ProposalAcceptedMessage

Der Auktionator benachrichtigt den Gewinner einer Auktion mit einer Nachricht vom Typ *ProposalAcceptedMessage* über den Sieg. Die Nachricht enthält einen Verweis auf die *Order* die damit dem Sieger zugewiesen wird.

4.4.4 ProposalRejectedMessage

Die Verlierer einer Auktion werden vom Auktionator mit einer Nachricht vom Typ *ProposalRejectedMessage* über die Niederlage in der entsprechenden Auktion informiert. Die Agenten ignorieren dann die betreffende *Order* in ihren Plänen.

4.4.5 ReturnOrder

Ein Agent kann ein ihm zugewiesenen Auftrag mit einer *ReturnOrder* Nachricht wieder an den Auktionator zurückgeben. In der Nachricht ist ein Verweis auf die entsprechende *Order* enthalten. Das Zurückgeben kann bei den Vergabeverfahren mit Rückgabe notwendig sein (siehe Abschnitte 4.6.2.3 und 4.6.2.4).

4.4.6 LockChangeMessage

Mit dieser Nachricht teilen Agenten anderen Agenten mit, dass sie Ihre Reservierungen geändert haben. Das kann sowohl das Hinzufügen einer neuen Reservierung als auch das Löschen einer vorher gemachten Reservierung bedeuten. Mit der Nachricht werden die betreffende Ressource, der Reservierungszeitraum sowie ob es sich um das Hinzufügen oder das Löschen einer Reservierung hangelt, versendet. Die empfangenden Agenten passen Ihre internen Graphen (siehe Abschnitt 5.1) anhand dieser Nachricht an.

4.4.7 LockNegotiationMessage

Mit dieser Nachricht stoßen Agenten Verhandlungen an. In der Nachricht enthalten sind die Reservierungen, über die verhandelt werden soll, der anfragende Agent sowie das Angebot des anfragenden Agenten. Dieses Angebot kann entweder ein Wert sein, der angibt, wie wertvoll die Reservierung für den anfragenden Agenten ist oder die Route, die der anfragende Agent zu nutzen plant, wenn er alle notwendigen Reservierungen dafür bekommt. Der angefragte Agent berechnet dann, wie wertvoll die Reservierung für ihn ist und lehnt die Freigabe dann entweder ab oder stimmt ihr zu. Die genauen Verfahren, mit denen die angefragten Agenten über die Freigabe einer Reservierung entscheiden ist im Abschnitt 4.7.3 sowie den Unterabschnitten des Abschnitts beschrieben.

4.4.8 LockNegotiationAnswer

Mit dieser Nachricht antwortet ein Agent auf eine LockNegotiationMessage (siehe Abschnitt 4.4.7). In der Nachricht enthalten ist die Reservierung die in der LockNegotiationMessage angefragt wurde sowie die Antwort des Agenten. Diese besteht aus einem einfachen boolean Wert (Zustimmung oder Ablehnung).

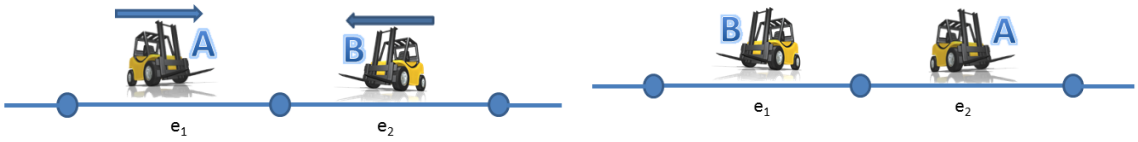
4.5 Konflikte und Konflikterkennung

Auch wenn der Schwerpunkt der Arbeit auf konfliktfreien Planungsmethoden liegt, ist es für den Vergleich mit anderen Verfahren sowie für die Untersuchung von Auswirkungen von Fehlern notwendig, dass die Simulation vorliegende Konflikte erkennt. Hierfür ist der ConflictChecker zuständig. Dieser Untersucht in jedem Simulationsschritt, ob Konflikte vorliegen. Wenn Konflikte identifiziert werden, werden sie zum einen in einer Log-Datei gespeichert, zum anderen kann eine Reaktion in der Simulation ausgelöst werden.

Konflikte entstehen in der Regel beim Fahren sowie beim Benutzen von Ressourcen. Die hier betrachteten Konflikte sind in der Regel ein Überschreiten von Ressourcen-Kapazitäten.

4.5.1 Kapazitätskonflikte

Die Mehrzahl der in den Simulationen verwendeten Ressourcen hat eine beschränkte Kapazität. Das bedeutet nur eine bestimmte Anzahl von Agenten kann die Ressource gleichzeitig nutzen. Parkplätze, Quellen und Senken haben in der Regel eine Kapazität von eins. Sie



(a) Situation zum Zeitpunkt t_1 (b) Situation zum Zeitpunkt $t_2 = t_1+1$

Abbildung 4.4: Head-on Konflikt: Echter Konflikt

können also nur von einem Agenten zur Zeit genutzt werden. Wege, also Kanten im Graphen, können eine höhere Kapazität haben. Eine Kapazität von zwei bedeutet also beispielsweise, dass zwei Agenten die Ressource gleichzeitig benutzen können, drei aber nicht.

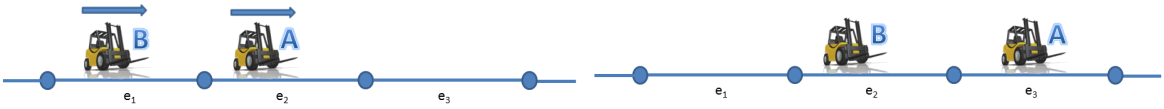
Der ConflictChecker geht in jedem Simulationsschritt alle Ressourcen durch, auf denen sich mindestens ein Agent befindet, und überprüft, ob sich mehr Agent als zulässig auf der Ressource befinden. Ist dies der Fall, hat er einen Konflikt erkannt, speichert diesen und ruft ggf. Reaktionen in der Simulation hervor.

Da Wege, die in beiden Richtungen befahrbar sind, durch zwei Kanten modelliert werden (siehe Abschnitt 4.1), muss die Anzahl der Agenten auf beiden Kanten bei der Überprüfung auf Konflikte berücksichtigt werden. Gleiches gilt, wenn ein Knoten nur eine logische Entität ist (Positionssensor im Boden). Ein Agent, der auf diesem Knoten ist, ist dann für die Ermittlung von Konflikten gleichzeitig auf der Kante, die zu dem Knoten gehört.

4.5.2 Head-on Konflikte

Eine weitere Art von Konflikten sind die so genannten Head-on Konflikte. Diese Konflikte können beim Übergang zwischen zwei Kanten auftreten, wenn sich zwei Fahrzeuge entgegen kommen. Ein solcher Head-on Konflikt ist in Abbildung 4.4 dargestellt. Sowohl die Kante e_1 als auch die Kante e_2 haben eine Kapazität die nicht dafür ausreicht, um Fahrzeug A und Fahrzeug B nebeneinander fahren zu lassen. Die automatische Überprüfung auf Konflikte, wie sie im vorherigen Abschnitt beschrieben ist, würde in der abgebildeten Situation keinen Konflikt erkennen. Sowohl zum Zeitpunkt t_1 (Abbildung 4.4 (a)) als auch zum Zeitpunkt $t_2 = t_1 + 1$ ((Abbildung 4.4 (b)) befindet sich auf beiden Kanten jeweils nur ein Fahrzeug. Die beiden Fahrzeuge sind sich gerade im Übergang zwischen den beiden diskreten Zeitpunkten entgegengekommen. Als erstes berücksichtigt wurden solche Head-on Konflikte von Hatzack und Nebel 2001 (siehe [HN01]) für den Fall von Kanten mit einer Kapazität von 1 und Fahrzeugen mit einer Breite von eins (binärer Fall).

Solche Konflikte müssen von der Simulation erkannt werden können. Eine Möglichkeit diese Art von Konflikten zu erkennen ist es, die Agenten ausschließlich nacheinander handeln zu lassen. Dadurch würde entweder Fahrzeug A zuerst auf Kante e_2 fahren und dort zeitgleich mit Fahrzeug B sein oder Fahrzeug B fährt zuerst auf Kante e_1 und ist dort gleichzeitig mit Fahrzeug A. In beiden Fällen würde der Konflikt erkannt. Dieses Verfahren Head-on Konflikte zu erkennen hat allerdings zwei Nachteile. Zum einen würde es die Möglichkeit nehmen, durch das Parallelisieren der Agenten die Rechenleistung von Multi-Kern Prozessoren und Multi-Prozessoren Rechner auszunutzen, zum anderen würde die in Abbildung 4.5 dargestellte Situation falsch als Head-on Konflikt erkannt werden (false positive), falls zuerst Fahrzeug B



(a) Situation zum Zeitpunkt t_1 (b) Situation zum Zeitpunkt $t_2 = t_1+1$

Abbildung 4.5: Head-on Konflikt: Problem bei der Erkennung durch Reihenfolge

(a) Situation zum Zeitpunkt t_1 (b) Situation zum Zeitpunkt $t_2 = t_1+1$

Abbildung 4.6: Head-on Konflikt: False positive

abgehandelt wird.

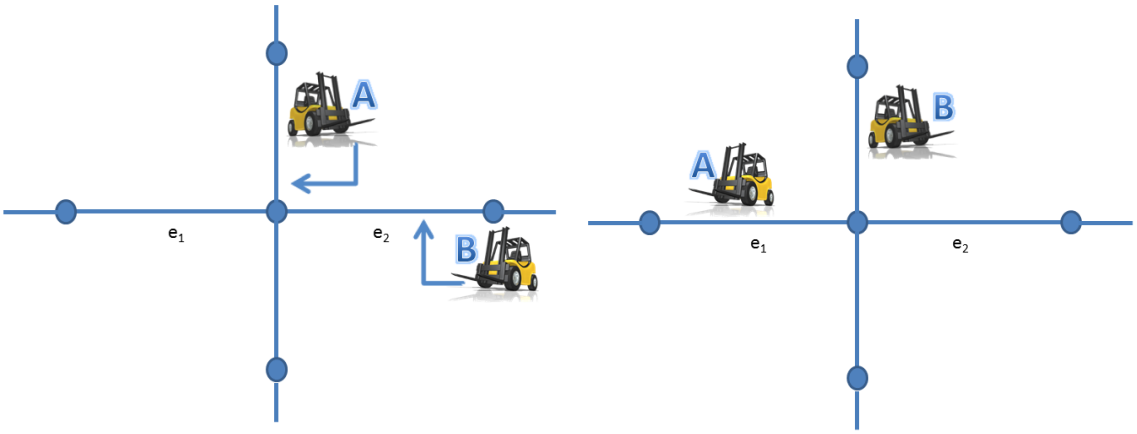
Statt der oben beschriebenen Möglichkeit können Head-on Konflikte auch erkannt werden, in dem neben der aktuellen Situation auch die Situation im vorherigen Zeitschritt berücksichtigt wird und die Situation aus Sicht der beweglichen Agenten betrachtet wird. In [MZ07] beschreiben Mors et.al. wie das Problem durch die Beachtung zweier Zeitpunkte durch nachsehen in einer Hashtabelle in $O(1)$ gelöst werden kann. Ein Head-on Konflikt liegt dann genau vor wenn der Ausdruck 4.1 war ist.

$$A^{t_1}_{\text{Position}} = B^{t_2}_{\text{Position}} \wedge A^{t_2}_{\text{Position}} = B^{t_1}_{\text{Position}} \tag{4.1}$$

Diese Formulierung und der darauf basierende Ansatz der Hashtabelle lässt sich leicht auf den nicht binären Fall übertragen. Sind neben der Position der Fahrzeuge noch die Breite der Fahrzeuge sowie die Kapazität der Kanten gegeben, liegt ein Head-on Konflikt unter Berücksichtigung nicht binärer Kapazitäten genau dann vor, wenn Ausdruck 4.2 wahr ist.

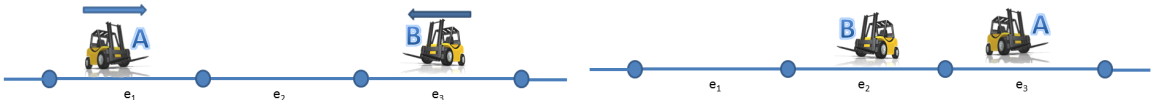
$$\begin{gathered} A^{t_1}_{\text{Position}} = B^{t_2}_{\text{Position}} \wedge A^{t_2}_{\text{Position}} = B^{t_1}_{\text{Position}} \\ \wedge \\ (A_{\text{Breite}} + B_{\text{Breite}} > e_{1_{\text{Kapazität}}} \vee A_{\text{Breite}} + B_{\text{Breite}} > e_{2_{\text{Kapazität}}}) \end{gathered} \tag{4.2}$$

Wenn also die Fahrzeuge A und B ihre Positionen tauschen (erste Hälfte des Ausdrucks) und wenn damit entweder die Kapazität der Kante e_1 oder der Kante e_2 überschritten wird (zweite Hälfte des Ausdrucks). Dieses Verfahren vermeidet das in Abbildung 4.5 dargestellte Problem. Dafür treten andere Probleme auf. Zum einen kann es auch bei diesem Verfahren zu false positives kommen und zwar in Kreisen, die aus zwei Kanten bestehen. Ein solcher Fall ist in Abbildung 4.6 dargestellt. In Wirklichkeit fahren die Fahrzeuge konfliktfrei hintereinander her, der Ausdruck 4.2 wird aber dennoch als wahr ausgewertet.



(a) Situation zum Zeitpunkt t_1 (b) Situation zum Zeitpunkt $t_2 = t_1+1$

Abbildung 4.7: Head-on Konflikt: False negative an Kreuzungen



(a) Situation zum Zeitpunkt t_1 (b) Situation zum Zeitpunkt $t_2 = t_1+1$

Abbildung 4.8: Head-on Konflikt: False negative durch zu hohe Geschwindigkeit

Bei Kreuzungen könnte es zu einem Head-on Konflikt kommen, wenn die Kreuzung eine genauso große (bzw. kleine) Kapazität hat, wie die von ihr abgehenden Kanten. Ein solcher Head-on Konflikt auf einer Kreuzung ist in Abbildung 4.7 dargestellt. Da der Ausdruck 4.2 in dieser Situation als falsch ausgewertet werden würde (die beiden Fahrzeuge haben nicht ihre Position getauscht) kommt es dazu, dass ein realer Konflikt nicht erkannt wird (false negative).

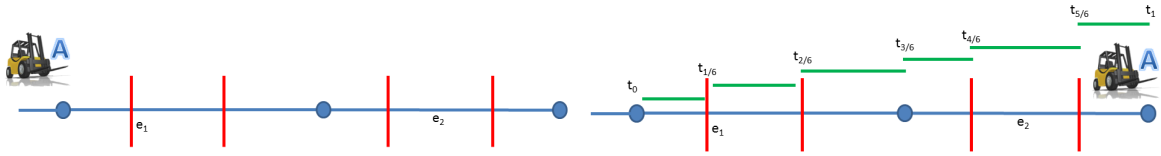
Neben diesen beiden Problemen kommt noch die Einschränkung zum tragen, dass das Verfahren nur funktioniert, wenn die Fahrzeuge pro Fahrschritt höchstens eine Kante weit fahren können. Ist das nicht der Fall kann es neben dem in Abbildung 4.7 dargestellten Fall auch noch zu einem false negative durch zu hohe Geschwindigkeit kommen, wie in Abbildung 4.8 dargestellt. Hier kommen sich die Fahrzeuge A und B zwar in einem Kantenzug e_1, e_2, e_3, der zu geringe Kapazität hat, entgegen, aber dadurch, dass das Fahrzeug A innerhalb eines Zeitschritts zwei Kanten weit gefahren ist, haben Fahrzeug A und B nicht die Position getauscht und somit wird Ausdruck 4.2 als falsch ausgewertet.

Eine weitere Einschränkung ist, dass der Ausruck 4.2 nur für Situationen anwendbar ist, an dem exakt zwei Fahrzeuge beteiligt sind.

Auf Grund dieser Einschränkungen wurde ein Ansatz zur Konflikterkennung implementiert der auf Stigmergie basiert. Der Begriff Stigmergie wurde von Pierre-Paul Grassé als *Stimulation von Arbeiterinnen durch das von ihnen Erschaffene* im Bezug zum beobachteten Verhalten von Termiten definiert (vgl. [Gra59]). Stigmergie wird heute bei vielen Problemen der Informatik, insbesondere bei Optimierungsproblemen ausgenutzt. Der bekannteste Vertreter

(a) Situation zum Zeitpunkt t_1 (b) Situation zum Zeitpunkt $t_2 = t_1+1$

Abbildung 4.9: Bewegung eines Agenten in einem Zeitschritt



(a) Situation zum Zeitpunkt t_1 (b) Situation zum Zeitpunkt $t_2 = t_1+1$

Abbildung 4.10: Auftragen von Pheromonen während der Bewegung

dieser Verfahren dürfte der Ant Colony Optimization Algorithmus von Marco Dorigo sein (vgl. [Dor92]). Bei diesem Algorithmus wird das Verhalten von Ameisen bei der Futtersuche mit Hilfe von künstlichen Pheromonen simuliert.

Für die Konflikterkennung nutzen wir nun ebenfalls eine Markierung der Umgebung ähnlich, wie das bei Pheromonen der Fall ist. Dazu teilen wir die Kanten für den ConflictChecker in Teile die kleiner oder gleich der halben Länge des kürzesten Fahrzeugs sind. In jedem Schritt unterteilt ein Fahrzeug nun seine Bewegung in Teilbewegungen auf diesen ConflictChecker Kanten und markiert diese mit einem Pheromon das nur aus dem Teilzeitschritt besteht, in dem es auf der Teilkante war. Dabei wird jeweils der Startteil der Bewegung als auch der Endteil markiert. Die Abbildungen 4.9 und 4.10 verdeutlichen den Vorgang. Auf Abbildung 4.9 macht der Agent A seine normale Bewegung. Er fährt dabei die beiden Kanten e_1 und e_2 ab, die jeweils eine Länge von 1 haben (der Agent A hat also eine Geschwindigkeit von 2.0). In Abbildung 4.10 ist die gleiche Situation aus Sicht des ConflictCheckers dargestellt. Die beiden Kanten sind jeweils in drei Teile unterteilt (rote, vertikale Linien), der Zeitschritt in 6 Teile. Das Fahrzeug hinterlässt während der Fahrt Pheromone auf den Teilen der Kante (grüne, horizontale Linien) die jeweils den 6 Zeitteilschritten zugeordnet sind. Alle anderen Agenten im System verfahren auf die selbe Weise. Das Pheromon besteht aus den folgenden Daten:

- Ressource auf der das Pheromon ist,
- Teilstück der Kante,
- Zeitintervall.

Das Zeitintervall wird hierbei als Teilintervall des Intervalls [0,1] angegeben.

Nach dem alle aktiv handelnden Agenten ihre Handlungen im aktuellen Simulationsschritt durchgeführt haben, überprüft der ConflictChecker Agent die Pheromone auf Konflikte. Dafür sortiert er zunächst alle Pheromon Einträge nach ihrer Ressourcen ID. Dieser Schritt ist für das Verfahren nicht entscheidend, reduziert aber den Rechenaufwand. Anschließend geht er jedes Pheromon durch und überprüft:

1. Sind andere Pheromone auf dieser Ressource gespeichert?

2. Wenn 1.: Sind andere Pheromone auf diesem Teilstück der Ressource gespeichert?
3. Wenn 2.: Sind andere Pheromone auf diesem Teilstück der Ressource in einem überlappenden Zeitintervall gespeichert?
4. Wenn 3.: Ist die Summe der Pheromone die 3.) erfüllen größer als die Kapazität der Ressource?

Erkennt der ConflictCheckerAgent einen Konflikt speichert er diesen zu Auswertungszwecken. Er kann die am Konflikt beteiligten Agenten auch über den Konflikt informieren.

Die hier vorgestellte Methode erkennt alle vorgestellten Konflikte und erkennt keine unkritischen Situationen fälschlich als Konflikt. Das Verfahren ist aber verhältnismäßig aufwändig, da die komplette Bewegung aller Agenten nach dem sie stattgefunden hat noch ein zweites mal betrachtet werden muss und in diesem zweiten Durchlauf sogar in (möglicherweise deutlich) feineren Unterschritten. Aus diesem Grund wird bei Simulationen mit konfliktfreiem Routing und ohne Fehlereinspeisung irgendeiner Form auf die Konflikterkennung verzichtet.

4.6 Auftragsvergabe

Transportaufträge sind der entscheidende Teil eines jeden Transportsystems. Dies gilt auch für Fahrerlose Transportsysteme, wie sie in dieser Arbeit behandelt werden. Ein Transportauftrag besteht in dieser Arbeit zwingend aus den Größen:

- Erstellungzeitpunkt,
- Startknoten,
- Zielknoten.

Optional können noch die Größen

- Deadline,
- Priorität,

angegeben sein. Der Erstellungszeitpunkt ist hier der Zeitpunkt, zu dem der Auftrag dem System bekannt gemacht wird und zu dem der Auftrag dann auch (zum ersten Mal) vergeben wird.

In diesem Abschnitt wird die Konzeption der Auftragseinlastung sowie der Auftragsvergabe vorgestellt.

4.6.1 Auftragseinlastung und Laststruktur

Zunächst stellt sich die Frage, wie Transportaufträge in das Transportsystem eingebracht werden. Nach [Ull10] gibt es verschiedene Varianten dieser Einbringung:

- Ein übergeordnetes System (zum Beispiel ein Enterprise-Resource-Planning System) kann Transportaufträge über eine Schnittstelle ins System einlasten,
- Stationen, die Wareneingänge oder -ausgänge repräsentieren können anfallende Transportaufträge erzeugen,

- Stationen, die Produktionsmaschinen repräsentieren, können Transportaufträge erzeugen,
- Menschliche Mitarbeiter können Transportaufträge erzeugen.

Da alle hier aufgeführten Entitäten (ERP System, Stationen, Menschen) außerhalb der Grenzen des betrachteten Systems liegen, wird die genaue Art der Auftragseinlastung in der vorliegenden Arbeit nicht unterschieden. Sie stellt sich für das System wie eine Black Box dar. Die Einlastung eines Transportauftrags geschieht also durch die Angabe eines Transportauftrags sowie eines Zeitpunktes, zu dem dieser ins System eingebracht wird.

Die Laststruktur kann entweder als statische Liste oder in Form von Wahrscheinlichkeitsverteilungen, aus denen dann mit Hilfe eines Zufallszahlengenerators eine statische Liste erzeugt wird, angegeben werden. Die Laststruktur liegt für die Untersuchungen immer in Form einer Liste von Transportaufträgen und Einlastungszeiten vor. Die Zeiten werden als Zeiteinheiten nach dem Start des Betrachtungszeitraum (in der Regel also nach Start der Simulation) angegeben.

Tabelle 4.1 zeigt einen Ausschnitt aus einer statischen Auftragseinlastung. Der erste Auftrag soll hier 10 Zeiteinheiten nach Simulationsstart (also zum Beispiel 10 Sekunden oder 10 Minuten) eingelastet werden. Er hat als Start die Quelle_1 und als Ziel die Senke_1. Der Auftrag soll spätestens 120 Zeiteinheiten nach Simulationsstart abgeschlossen sein und er hat das Gewicht (also die Priorität) 1.0.

Zeit	Start	Ziel	Deadline	Weight
10	Quelle_1	Senke_1	120	1.0
20	Quelle_2	Senke_2	150	0.8
30	Quelle_3	Senke_1	150	0.8
40	Quelle_1	Senke_2	160	0.2

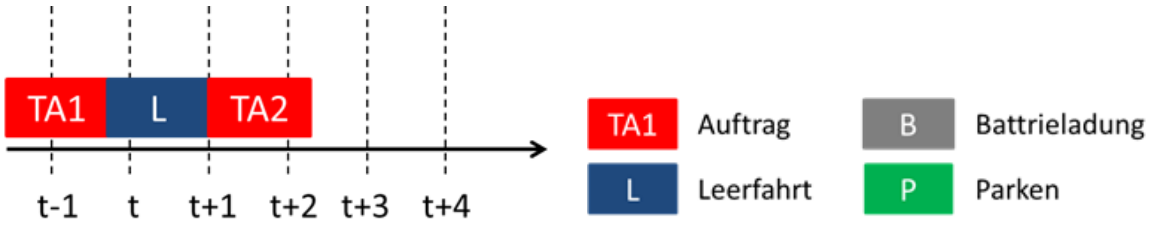
Tabelle 4.1: Beispiel Auftragseinlastung - 5 Spalten

In Szenarien in denen Deadlines oder Prioritäten nicht vorkommen bzw. keine Rolle spielen, können diese bei der Auftragseinlastung weggelassen werden. Die Tabelle enthält dann entsprechend weniger Spalten. Ein Beispiel für eine minimale Einlastung ist in der Tabelle 4.2 gegeben.

Zeit	Start	Ziel
10	Quelle_1	Senke_1
20	Quelle_2	Senke_2
30	Quelle_3	Senke_1
40	Quelle_1	Senke_2

Tabelle 4.2: Beispiel Auftragseinlastung - 3 Spalten

Zur angegebenen Zeit startet dann jeweils das Vergabeverfahren für den Transportauftrag. Die hierfür konzipierten Verfahren werden in den nächsten Abschnitten vorgestellt.



(a) Belegungsplan eines FTF (b) Legende

Abbildung 4.11: Belegungspläne als Gantt-Diagramm

4.6.2 Auftragsvergabe per Auktion

Die Auftragsvergabe in einem Agenten basierten, dezentralen System erfolgt üblicherweise per Auktion. In der vorliegenden Arbeit wurden ebenfalls Auktionen verwendet. In diesem Verfahren werden Transportaufträge, sobald sie ins System eingelastet werden, den Fahrzeugen bzw. den Agenten, die die Fahrzeuge steuern bzw. repräsentieren bekannt gemacht. Diese berechnen darauf hin ihr Angebot in Abhängigkeit ihres aktuellen Zustands. Der aktuelle Zustand eines FTF wird maßgeblich durch seine aktuelle Position und seinen Belegungsplan bestimmt. Der Belegungsplan gibt an, zu welchen Zeiten das Fahrzeug welche Transportaufträge eingeplant hat. Abbildung 4.11 zeigt einen solchen Belegungsplan dargestellt ist in Teil (a) die Belegung als Gantt-Diagramm und in Teil (b) die Legende. Die Legende gilt für alle Belegungspläne in diesem Abschnitt.

Im folgenden werden die konkret konzipierten Auktionsverfahren vorgestellt. In Abschnitt 4.6.2.1 wird ein einfaches Auktionsverfahren vorgestellt, bei dem die Agenten einfach den Zeitpunkt, zu dem sie den ausgeschriebenen Auftrag beginnen können, als Angebot verschicken. In Abschnitt 4.6.2.2 wird das Verfahren um eine parametrierbare Zielfunktion erweitert, so dass auch andere Kriterien wir die gefahrene Strecke in der Angebotsberechnung berücksichtigt werden können. Im in Abschnitt 4.6.2.3 beschriebenen Verfahren haben die Agenten zusätzlich die Möglichkeit ihnen bereits zugewiesenen Aufträge zurückzugeben, falls der gerade ausgeschriebene Auftrag für sie attraktiver ist. Außerdem können die Agenten in diesem Verfahren Aufträge auch in Lücken in ihrem Belegungsplan einplanen. Im letzten vorgestellten Verfahren (Abschnitt 4.6.2.4) wird das Verfahren dann um eine parametrierbare Zielfunktion erweitert.

4.6.2.1 Vergabe per einfacher Auktion

Zunächst wurde ein sehr einfaches Auktionsverfahren konzipiert. Ein Auktionator verschickt bei jedem neu ins System eingelasteten Auftrag an alle FTF die Aufforderung, ein Angebot für diesen Auftrag abzugeben. Erhält ein Agent eine solche Aufforderung wird zwischen zwei Fällen unterschieden: Dem Fall, dass der Agent aktuell keinen Auftrag zugewiesen hat, er also idle ist und dem Fall, dass der Agent bereits einen oder mehrere Aufträge zugewiesen bekommen hat.

Im Fall, dass der Agent aktuell keinen Auftrag zugewiesen bekommen hat, besteht das Angebot aus der Fahrtdauer von seiner aktuellen Position zum Startpunkt des Auftrags plus der aktuellen Zeit, also aus der erwarteten Ankunftszeit am Startort des Auftrags. Die Fahrtdauer hängt hier vom berechneten Weg von der aktuellen Position zum Startort des Auftrags

ab und somit vom verwendeten Routingverfahren. Die konzipierten Routingverfahren sind in Abschnitt 4.7 genauer beschrieben.

Im Fall, dass der Agent bereits einen oder mehrere Aufträge in seinem Belegungsplan hat, besteht das Angebot aus der Fahrtdauer vom Endpunkt des letzten Auftrags in seinem Belegungsplan zum Startpunkt des Auftrags plus dem geplanten Beendigungszeitpunkts des letzten Auftrags in seinem Belegungsplans, also ebenfalls aus der erwarteten Ankunftszeit am Startort des Auftrags. Die Fahrdauer hängt auch hier vom verwendeten Routing Verfahren ab.

In beiden Fällen senden die Agenten anschließend ihr Angebot, also die mögliche Ankunftszeit, als Angebot an den Auktionator zurück. Dieser wählt das beste Angebot, also das mit der frühesten Ankunftszeit am Startpunkt der Auktion aus. Bei Gleichstand wird unter den besten Angeboten ein zufälliges bestimmt. Anschließend teilt der Auktionator dem Gewinner den Auftrag zu und informiert die Verlierer darüber, dass er den Auftrag an ein anderes FTF vergeben hat. Der Kommunikationsablauf zwischen den FTF Agenten und dem Auktionator ist in Abbildung 4.12 dargestellt.

4.6.2.2 Vergabe per einfacher Auktion mit parametrierbarer Zielfunktion

Das Verfahren entspricht bis auf die konkrete Angebotsberechnung dem in Abschnitt 4.6.2.1 vorgestellten Verfahren. Das Angebot entspricht nun aber nicht mehr notwendigerweise dem frühesten Ankunftszeitpunkt am Auftragsstartpunkt. Im hier vorstellten Verfahren wird das Angebot durch das Auswerten einer Zielfunktion ermittelt. Diese Zielfunktion ist in der Regel die Summe verschiedener, gewichteter Parameter. Diese Parameter können zum Beispiel folgende sein:

- Ankunftszeit beim Auftragsstartpunkt,
- Dauer bis zur Beendigung des Auftrags,
- Für den Auftrag zurückzulegende Strecke,
- Für den Auftrag zu benutzende Ressourcen.

Die Komposition einer Funktion aus den Parametern geschieht in Abhängigkeit vom gewünschten Verhalten des Transportsystems. Ist beispielsweise ein schnelles Entleeren der Quellen gewünscht, wird die Ankunftszeit beim Auftragsstartpunkt stark gewichtet. Ist ein Energie effizientes Verhalten gewünscht, wird die zurückzulegende Strecke stark gewichtet. Ein weiteres Beispiel wäre die Benutzung von Fahrstühlen. Im Fall, dass das Angebot nur aus der Zeit besteht müsste aus den beiden folgenden Angeboten das erste ausgewählt werden:

1. Ankunftszeit 11:30:20 Uhr, Fahrstuhl 3 für 60 Sekunden benötigt,
2. Ankunftszeit 11:30:30 Uhr, keine Ressourcen benötigt.

Bei der Benutzung einer parametrierbaren Zielfunktion in der sowohl Zeit als auch Ressourcen berücksichtigt werden, könnte auch Angebot 2 ausgewählt werden. In einem Szenario, in dem die Fahrstühle stark beschränkte Ressourcen darstellen, kann dies zu einem besseren Systemverhalten führen.

Weitere Parameter sind je nach gewünschter globaler Zielfunktion denkbar. Für die Dauer bis zur Beendigung des Auftrags muss der Agent neben der Berechnung des Weges zum Start-

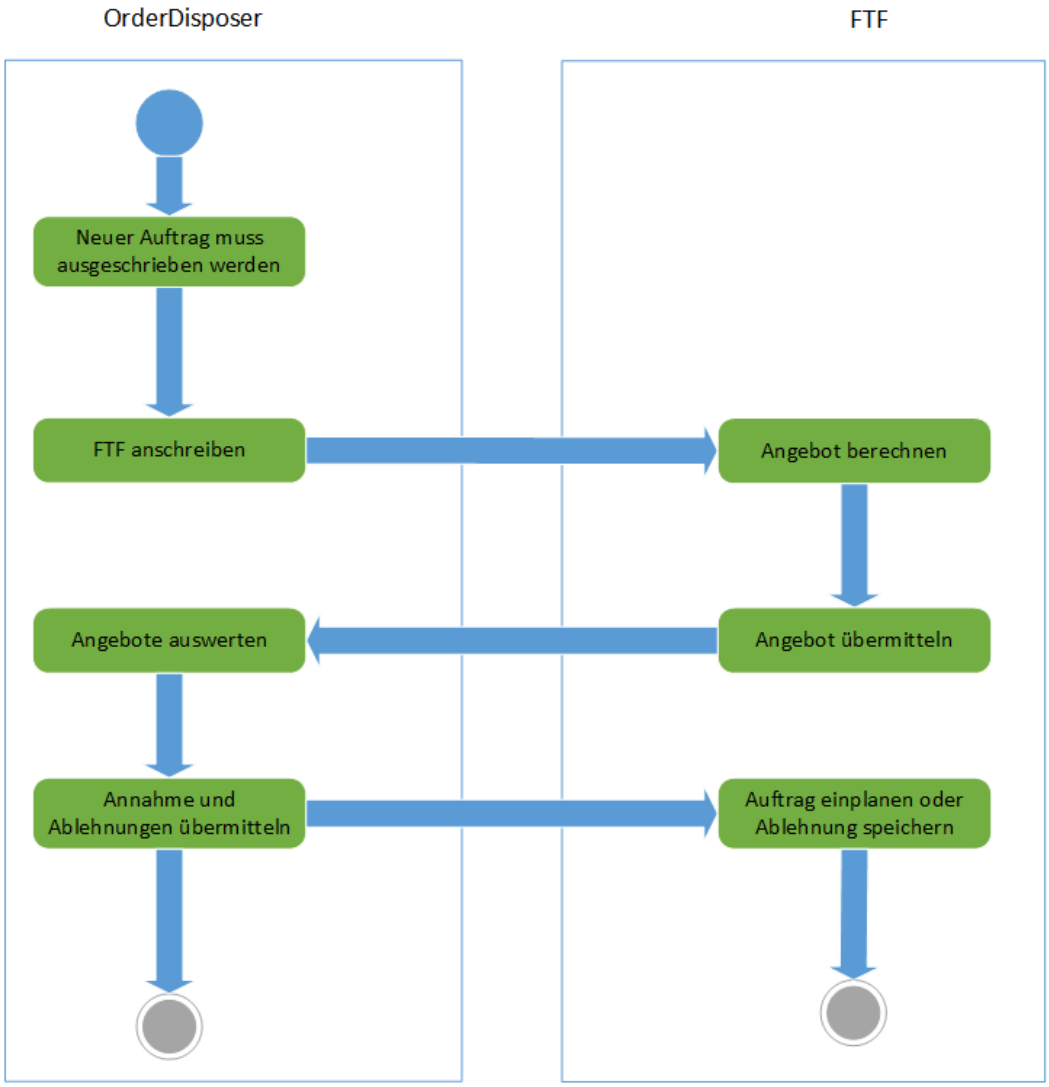


Abbildung 4.12: Kommunikation zwischen dem Auktionator (OrderDisposer) und den FTF

punkt noch die Beladedauer und den Weg vom Start- zum Zielpunkt des Auftrags berechnen (mit dem verwendeten Routingverfahren). Eine seperate Betrachtung von der Ankunftszeit beim Auftragsstartpunkt und der Dauer bis zur Beendigung des Auftrags kann relevant sein, wenn unterschiedliche Fahrzeuge eingesetzt werden (unterschiedliche Geschwindigkeit oder unterschiedliche Beladezeit) und wenn es wichtig ist, dass das Transportgut schnell abgeholt wird (weil die Station zum Beispiel nur einen begrenzten Puffer hat).

Der Ablauf der Auftragsvergabe durch eine Auktion funktioniert bis auf das Berechnen des abzugebenden Angebots identisch mit dem Verfahren, das in Abschnitt 4.6.2.1 beschrieben ist. Der Kommunikationsablauf geschieht dementsprechend wie in Abbildung 4.12 dargestellt.

4.6.2.3 Vergabe mit Einordnung des Auftrags in den aktuellen Belegungsplan und möglicher Rückgabe von Aufträgen

Im Rahmen des Forschungsprojekts "Dezentrale, agentenbasierte Selbststeuerung von Fahrerlosen Transportsystemen"wurde in Zusammenarbeit mit dem IPH Hannover ein weiteres Verfahren für die Auftragsvergabe entwickelt. In diesem Verfahren ist es den FTF möglich Aufträge zurückzugeben und neue Aufträge zwischen schon geplanten Aufträgen einzuplanen. Die Abbildung 4.13 zeigt den aktuellen (oben) und zwei mögliche Belegungspläne eines FTF nach Ausschreibung des TA3 (Transportauftrag 3). Das FTF hat bereits TA1 beendet und ist unterwegs zur Quelle des TA2. Zum Zeitpunkt t wird TA3 ausgeschrieben. Für das FTF ergeben sich zwei Alternativen: Rückgabe des TA2 oder Aufnahme des TA3 erst nach dem TA2. Für das FTF selbst ist die erste Alternative von Vorteil, da dadurch seine Leerfahrtzeit verkürzt wird. Für das gesamte System könnte die Rückgabe des TA2 aber eine Vergrößerung der Gesamtlaufzeit bedeuten, da unklar ist ob und zu welchem Zeitpunkt ein anderes FTF TA2 übernehmen kann.

Um die Performanz des Gesamtsystems nicht zu gefährden, wurde die Regel konzipiert, dass ein angefangener Transportauftrag nicht zurückgegeben werden kann.

Definition 4.1. Angefangener Transportauftrag
Ein Transportauftrag gilt als angefangen sobald ein FTF sich *nah genug* an der Quelle befindet. Die Eigenschaft *nah genug sein* wird durch eine Zeitkonstante σ ausgedrückt und ist in Abbildung 4.14 beispielhaft gezeigt. Ein FTF kann die Entscheidung ob ein Transportauftrag in die Auftragsvergabe einbezogen wird oder nicht, folgendermaßen treffen: Wenn gilt:

$$Ankunft\_Quelle\_Nächster\_TA \text{ - } aktuelle\_Zeit \leq \sigma$$

dann ist der nächste Transportauftrag *angefangen.*

Das σ kann zum Beispiel anhand der Geschwindigkeit eines FTF festgelegt werden. Beträgt die Geschwindigkeit beispielsweise 1,6 m/s, und soll der minimale Abstand zur Quelle. bei dem ein Transportauftrag noch zurückgegeben werden kann, 10 m betragen, so müsste $\sigma =$ 6 sec sein. Weiterhin kann das σ anhand des Layouts festgelegt werden. Gibt es zum Beispiel kurz vor der Quelle noch Abzweigungen oder Wendemöglichkeiten, dann hat das FTF noch die Möglichkeit seinen Kurs zu ändern um zu einer andern Quelle auf einem kürzeren Weg zu gelangen.

Parken und Batterieladen sind spezielle Aufträge und werden hier nicht wie Transportaufträge behandelt:

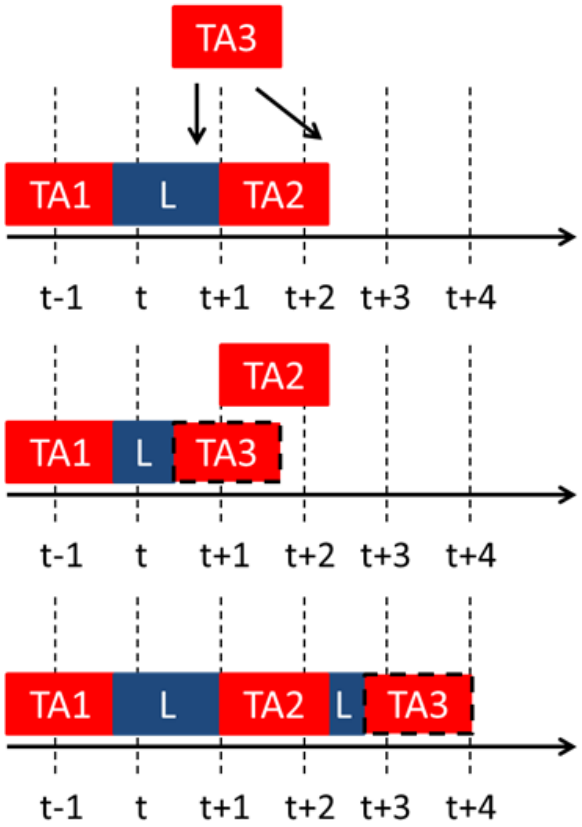


Abbildung 4.13: Beispiel der Vergabe eines Transportauftrags

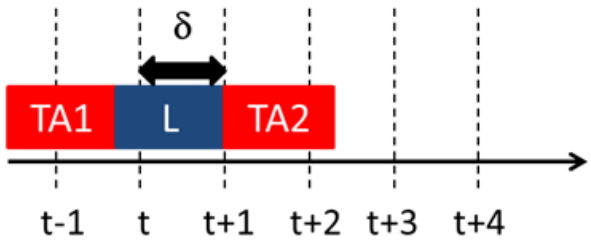


Abbildung 4.14: Ein angefangener Transportauftrag

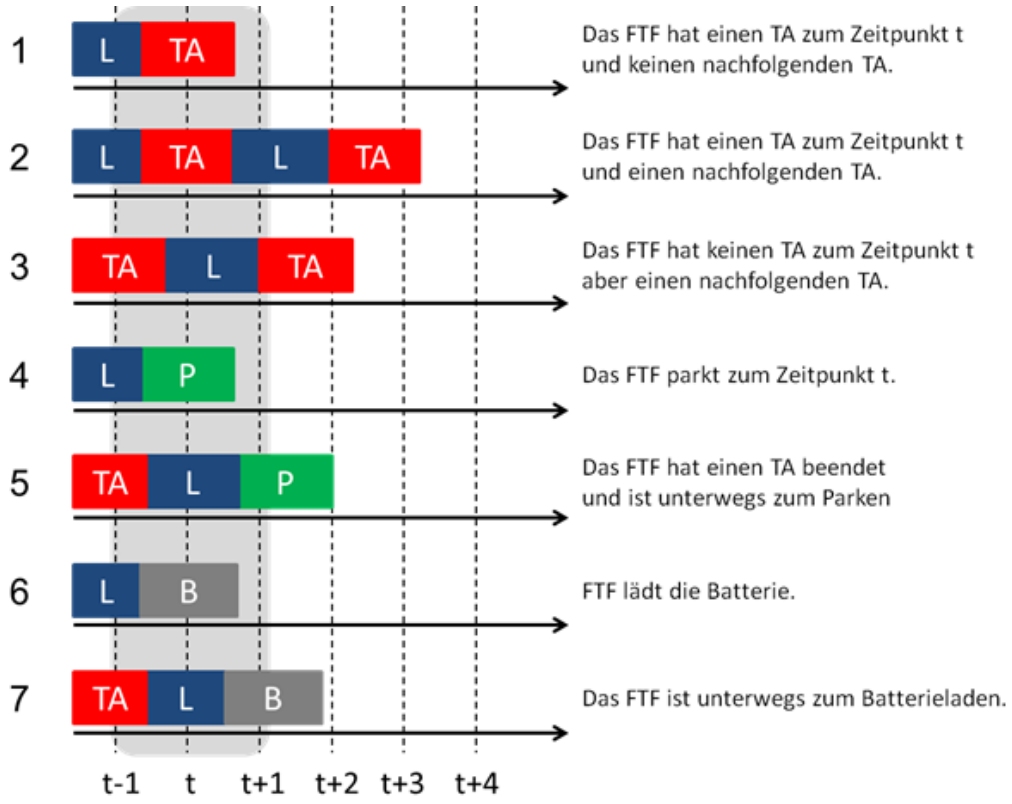


Abbildung 4.15: Belegungsplan: Verschiedene Zustände eines FTF

- Das Parken oder die Fahrt zum Parkplatz dürfen jederzeit für einen neuen Transportauftrag abgebrochen werden.
- Das Laden der Batterie darf nicht abgebrochen werden, solange der gewünschte Ladestand der Batterie erreicht wurde.

Abbildung 4.15 zeigt mögliche Belegungszustände eines FTF zu dem Zeitpunkt t, in dem ein neuer Transportauftrag ausgeschrieben wird. Wir betrachten hier nur die störungsfreien Situationen. Alle FTF sind also einsatzbereit.

Wie schon erwähnt, ist Batterieladen ein spezieller Auftrag, der nicht unterbrochen wird. Die Fälle 6 und 7 entsprechen deshalb den Fällen 1 und 2. Das Parken darf jederzeit unterbrochen werden. Die Fälle 4 und 5 können deshalb auf den Fall 3 reduziert werden. Es bleiben also drei mögliche Zustände für FTF, die für die Transportauftragsvergabe relevant sind:

1. Das FTF führt gerade ein Transportauftrag aus. Nach dem Transportauftrag ist das FTF frei verfügbar (siehe Parkstrategien)
2. Das FTF führt gerade ein Transportauftrag aus und hat bereits einen weiteren Transportauftrag verplant.
3. Das FTF ist unterwegs zu einem Parkplatz oder zur Quelle des nächsten Transportauftrag.

Aus den Zuständen 1-3 ist ersichtlich, dass lediglich die Frage ob bereits ein weiterer Transportauftrag eingeplant ist Auswirkung auf die Angebotsberechnung hat (Abbildung 4.16). Ist

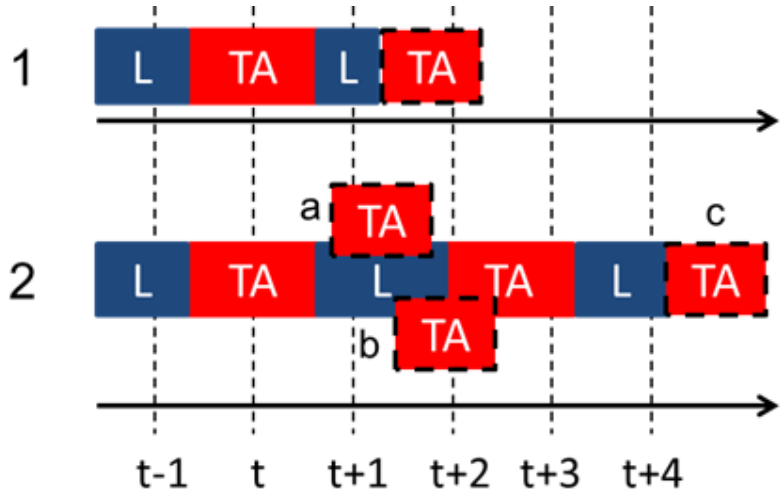


Abbildung 4.16: Belegungsplan: Möglichkeiten der Einordnung

kein weiterer Transportauftrag eingeplant, so hängt das Angebot des FTF lediglich von seiner Anfahrtszeit zu der Quelle des neuen Transportauftrags ab. Ist ein weiterer Transportauftrag eingeplant, so gibt es die folgenden Möglichkeiten:

- Der neue Transportauftrag kann noch vor dem bereits eingeplanten Transportauftrag in den Belegungsplan integriert werden (2a). In diesem Fall wird die Leerfahrtzeit des FTF minimiert und der Makespan nicht erhöht. Sowohl das FTF als auch das gesamte FTS profitieren also von dieser Planung.
- Der neue Transportauftrag kann nicht vor dem bereits eingeplanten Transportauftrag in den Belegungsplan integriert werden. Hier gibt es die Möglichkeit den verplanten Transportauftrag zurückzugeben (2b) oder den neuen Transportauftrag erst am Ende in den Belegungsplan zu integrieren (2c).

Im Fall (2c) hängt das Angebot wie im Fall 1 nur von der Anfahrtszeit ab. Im Fall (2b) muss die Rückgabe des schon eingeplanten Transportauftrag berücksichtigt werden. Die Leerfahrtzeit wird zwar minimiert. Es ist aber unklar ob die Rückgabe zu einer Erhöhung des Makespans und somit zur Verschlechterung der Systemeffizienz führt.

Die Rückgabe eines Auftrags erfolgt durch eine Benachrichtigung an den Auktionator, der den Auftrag ursprünglich ausgeschrieben hatte. Dieser startet darauf hin eine erneute Auktion um den Auftrag ein weiteres mal zuzuordnen.

Das Angebot eines FTF wird nun wie folgt berechnet:

$$Angebot = \alpha \times \frac{1}{AnfahrtsZeit} - \beta \times Rückgabe + \gamma \times ReduzierteLeerfahrtundStillstandszeit$$

Mit folgenden Variablen:

Kehrwert der Anfahrtszeit Je kleiner die Anfahrtszeit ist, desto höher wird das Angebot. Falls die Anfahrtszeit gleich Null ist, wird der Ausdruck als 2α ausgewertet.

Rückgabe Ein binärer Wert: JA (=1) oder NEIN (=0). Eine notwendige Rückgabe reduziert das Angebot.

Reduzierte Leerfahrt und Stillstandszeit Lässt sich der Transportauftrag in eine Lücke des Belegungsplans integrieren, so führt die Verminderung der Leerfahrtzeit und / oder der Stillstandszeit zu einer Erhöhung des Angebots.

Die Parameter α, β, und γ gewichten die drei Variablen. Sie sind frei in Abhängigkeit vom Szenario, der Auftragslast und dem gewünschten Systemverhalten zu bestimmen.

Der Kommunikationsablauf erweitert sich durch die Möglichkeit der Rückgabe wie in Abbildung 4.17 dargestellt.

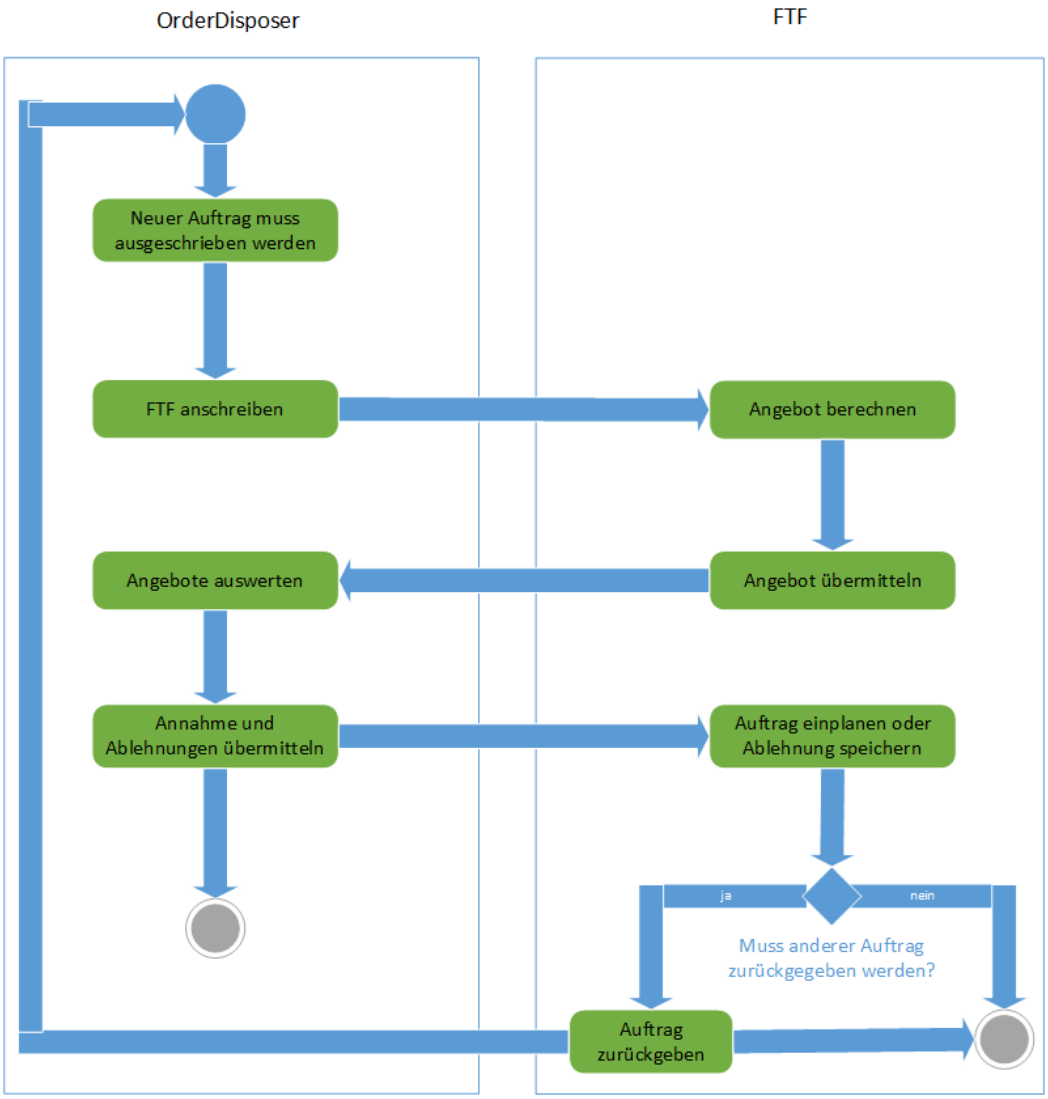


Abbildung 4.17: Kommunikation zwischen Auktionator (OrderDisposer) und den FTF (Auktion mit der Möglichkeit der Auftragsrückgabe

Der anfängliche Ablauf ist hier derselbe wie bei der Kommunikation im Falle der einfachen Auktion (siehe 4.6.2.1). Wenn das FTF, das das erfolgreiche Angebot abgegeben hat in Folge des Zuschlags einen Auftrag zurückgeben muss, so teilt es dies am Ende des Ab-

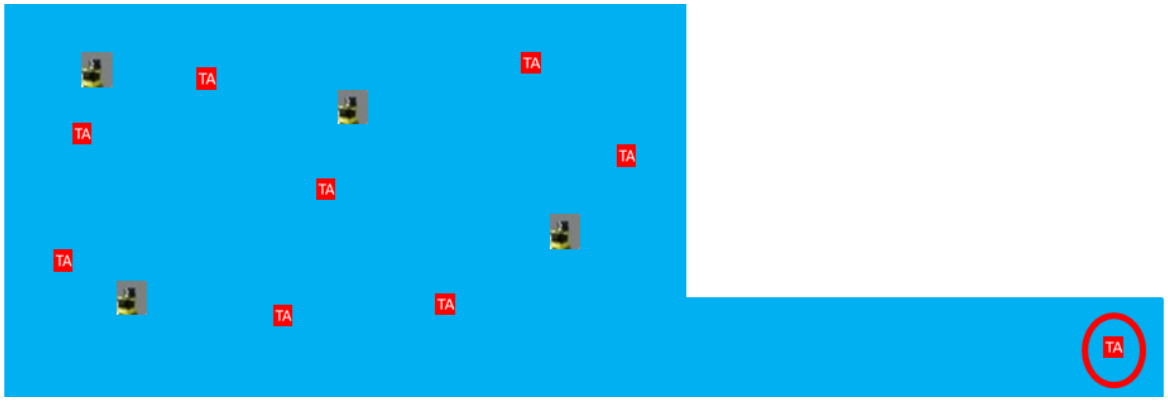


Abbildung 4.18: Starvation Beispiel

laufs dem Auktionator (OrderDisposer) mit. Dieser startet dann eine neue Auktion, um den zurückgegebenen Auftrag zuzuordnen.

4.6.2.4 Vergabe mit Einordnung des Auftrags in den aktuellen Belegungsplan und möglicher Rückgabe von Aufträgen sowie parametrierbarer Zielfunktion

Ähnlich wie die Vergabe per einfacher Auktion (siehe Abschnitt 4.6.2.1) durch die Verwendung einer parametrierbaren Zielfunktion erweitert werden kann (siehe Abschnitt 4.6.2.2), kann auch die Vergabe mit Einordnung des Auftrags und der Möglichkeit der Rückgabe eines Auftrags durch eine parametrierbare Zielfunktion erweitert werden.

Bedingt durch die Möglichkeit der Rückgabe eines Auftrags kann es bei bestimmten Szenarien leicht zum Liegenbleiben eines Auftrags kommen (Starvation). Betrachten wir zum Beispiel das angedeutete Szenario in Abbildung 4.18: Bleibt das Auftragsaufkommen auf einem ähnlichen Niveau wie in der abgebildeten Situation (9 Transportaufträge auf 4 FTF), wird bei einer Auftragsvergabe wie in Abschnitt 4.6.2.3 beschrieben der rot umkreiste Transportauftrag in der unteren rechten Ecke wahrscheinlich liegenbleiben. Egal welches der FTF die Auktion zur Vergabe dieses Transportauftrags gewinnt, wenn der nächste Auftrag ausgeschrieben wird, wird er diesen wahrscheinlich zurückgeben (weil das FTF einen sehr großen Leerfahrtanteil zur Erfüllung benötigt). Lösen ließe sich das Problem natürlich durch einen sehr hohen Wert für β (Malus für die Rückgabe eines Auftrags - siehe Abschnitt 4.6.2.3) beim Berechnen des Angebots. Das würde aber dafür sorgen, dass der Vorteil der Möglichkeit einer Rückgabe eines Auftrags (eben die Reduzierung des Leerfahrtanteils) so gut wie nie zum Tragen kommt.

Um das Problem des Liegenbleibens eines Transportauftrags zu umgehen wurde die Auftragsberechnung um die Berücksichtigung der dynamischen Priorität eines Transportauftrags erweitert.

Definition 4.2. Dynamische Priorität eines Transportauftrags

$$Priorität_{dynamisch} = \zeta \times Platzmangel + \eta \times Dringlichkeit + \iota \times Starvation + \theta \times Priorität_{statisch}$$

mit:

$$Platzmangel = \frac{1}{Puffergröße - Anzahl\_Wartender\_Transportaufträge}$$

$$Dringlichkeit = \frac{Ausführungszeit}{Deadline\text{-}Aktuelle\_Zeit}$$

$$Starvation = \frac{1}{Rückgabe\_Maximum \text{ - } Anzahl\_Rückgaben}$$

$$Priorität_{statisch} = w$$

Der Platzmangel beschreibt also, wie viele Transportaufträge an der jeweiligen Station noch zwischengespeichert werden können. Das ist zum Beispiel in Szenarien relevant, wo die Transportgüter an produzierenden Stationen abgeholt werden müssen. Oft ist es so, dass an der Station nur wenige fertiggestellte Güter zwischen gelagert werden können. Ist der Zwischenspeicher voll, stellt die Station die Produktion ein. Um das zu verhindern soll ein Auftrag von einer Station mit (fast) vollem Zwischenlager höher gewichtet werden als ein Auftrag von einer Station mit ausreichen Kapazität im Zwischenlager.

Die Dringlichkeit gibt an, wie viel Zeit noch zum Ausführen des Auftrags bis zu seiner Deadline vorhanden ist. Die Ausführungszeit bezeichnet dabei die komplette Zeit, die das FTF für die Ausführung des Auftrags benötigt (Zeit für vorher eingpelante Aufträge, Anfahrt zum Startort, Beladung, Anfahrt zum Zielort und Entladung). Die Deadline d ist ein Wert der dem Auftrag bei der Einlastung zugewiesen wurde.

Starvation gibt an, wie oft der Auftrag schon im Verhältnis zur maximal zulässigen Anzahl an Rückgaben zurückgegeben wurde. Die maximal zulässige Anzahl ist ein Parameter der ebenso wie ζ, η, θ und ι festgelegt werden muss. Die Angabe als Kehrwert dient der Normalisierung des Wertes. Die Normalisierung wird hier bevorzugt, da diese das Verhältnis der Parameter so direkt das Verhältnis der realen Gewichtung widerspiegelt (alle Einflussgrößen sind normalisiert).

Die Festlegung der Parameter ζ, η, θ und ι geschieht in Abhängigkeit des Szenarios, der Auftragslast und dem gewünschten Zielverhalten des Systems.

Wenn es im betrachteten Szenario keine Deadline und / oder keine begrenzten Lagerkapazitäten und / oder keine statische Priorität gibt, können die entsprechenden Parameter auf 0 gesetzt werden. Gibt es enge Deadlines oder stark begrenzte Lagerkapazitäten kann der Wert Starvation redundant sein, da eine wiederholte Rückgabe eines Transportauftrag dessen Dringlichkeit bzw. den Platzmangel deutlich erhöhen dürfte. Sollte die separate Betrachtung in diesem Fall nicht nötig sein, kann der Parameter ι auf 0 gesetzt werden.

Das Auftragsangebot wird nun anstatt mit dem Ausdruck:

$$Angebot = \alpha \times \frac{1}{AnfahrtsZeit} - \beta \times Rückgabe + \gamma \times ReduzierteLeerfahrtundStillstandszeit$$

mit dem Ausdruck:

$$Angebot = \alpha \times \frac{1}{AnfahrtsZeit} - Priorität_{dynamisch} \times Rückgabe + \gamma \times ReduzierteLeerfahrtundStillstandszeit$$

bestimmt. Die für die Auswertung notwendigen Parameter α, β, γ, ζ, η, θ und ι sind in Abhängigkeit vom Szenario, der Auftragslast und dem gewünschten Systemverhalten festzulegen. Die Parameter lassen auch durch Suchverfahren oder automatisches Lernen bestimmen.

4.7 Routing

Das Routing, also das Planen einer Route vom Startpunkt zum Zielpunkt muss, im Falle einer dezentralen Steuerung, jedes Fahrzeug bzw. jeder Agent für sich selber übernehmen. Die Grundlage hierfür bilden in der Regel Graphen.

Die Güte der bestimmten Routen hängt von bestimmten Eigenschaften ab:

- Der Länge der Route,
- Der benötigten Fahrzeit,
- Der Konfliktfreiheit.

Die benötigte Fahrzeit kann eine Eigenschaft sein, die nicht direkt aus der Länge der Route folgt, wenn zum Beispiel bestimmte Streckenabschnitte nur mit begrenzter Geschwindigkeit befahren werden können (Kurven oder Steigungen). Konfliktfreiheit bezeichnet die Eigenschaft, dass die geplante Route nicht mit geplanten Routen anderer Fahrzeuge in Konflikt steht, es also beispielsweise beim Abfahren der Routen zu keinen Kollisionen kommt. Weitere Eigenschaften können bei speziellen Szenarien eine Rolle spielen. Beispiele für weitere Eigenschaften wären Kosten (Maut) oder virtuelle Kosten. Mit virtuellen Kosten ließen sich beispielsweise Engpässe eines Layout simulieren: Stark frequentierte Streckenabschnitte haben hohe virtuelle Kosten, schwach frequentierte Streckenabschnitte haben niedrige Kosten. Eine kostengünstige Route zu berechnen kann dann zu einer geringeren Staugefahr in den stark frequentierten Bereichen des Layouts führen.

Grundsätzlich kann man drei Arten von dezentraler Routenplanung in Systemen mit mehreren Teilnehmern und gemeinsamen Ziel unterscheiden:

- Uninformierte Planung,
- Informierte, konfliktvermeidende Planung,
- Kooperative Planung.

4.7.1 Uninformierte Planung

Bei der uninformierten Planung weiß das planende Fahrzeug bzw. der planende Agent nichts von den Plänen der anderen Systemteilnehmer. Er kann deren Pläne folglich nicht in seine eigenen Pläne integrieren. Es kann hier einerseits zwischen völlig uninformierter Planung, in der der planende Agent keinerlei Informationen über die Pläne der anderen Agenten hat und uninformierter Planung, in der der Agent gewisse, auf Wahrscheinlichkeiten basierte Vermutungen über die Pläne der anderen Agenten anstellen kann, andererseits unterschieden werden. Im ersten Fall berechnet der Agent einfach die Route, die die beste Güte nach den Kriterien Länge, Dauer und ggf. Kosten hat. Das kann zum Beispiel mit dem klassischen

Abbildung 4.19: Konflikt durch sich entgegenkommende Fahrzeuge

Dijkstra Algorithmus (siehe 2.5.2) geschehen. Die Kantengewichte sind hier die Summe aus Kosten, Länge und Dauer in der gewünschten Gewichtung.

Im zweiten Fall kann der Agent Wahrscheinlichkeiten für die Pläne der anderen Systemteilnehmer in seine Planung integrieren. Ein Beispiel dafür wären die schon beschriebenen virtuellen Kosten. Das Bestimmen dieser virtuellen Kosten kann durch eine statische Layout-Analyse erfolgen. Hierfür wird jede Kante mit virtuellen Kosten von 0 initialisiert. Dann wird der kürzeste Weg zwischen jedem Start- und Zielknotenpaar berechnet. Für alle Kanten dieses Weges werden anschließend die virtuellen Kosten um eins erhöht. Dieses Verfahren kann noch mit Hilfe der Aufragseinlastung gewichtet werden, wenn diese bekannt ist. Eine weitere Möglichkeit wäre es, das System die virtuellen Kosten lernen zu lassen. Hierbei beobachtet das System alle Fahrzeuge und erhöht die virtuellen Kosten einer Kante in jedem Zeitschritt um die Anzahl der Fahrzeuge auf der Kante. Hierbei kann das Gewicht auch aus einem begrenzten Zeitraum bestimmt werden (nur die letzte Stunde, nur der letzte Tag wird berücksichtigt). Die Berücksichtigung der virtuellen Kosten bei der Planung der Route kann so zu Entzerrungen im Layout führen. Die stark frequentierten Bereiche werden weniger stark frequentiert und die Gefahr von Konflikten oder Staus nimmt ab.

In beiden oben beschriebenen Fällen kann es aber zu Konflikten kommen. Der am häufigsten auftretende Konflikt ist das sich Entgegenkommen zweier Fahrzeuge auf einem Wegabschnitt, der nicht breit genug ist, als dass beide Fahrzeuge aneinander vorbei passen (siehe Abbildung 4.19).

Weitere mögliche Konflikte können durch das gleichzeitige Verplanen beschränkter Ressourcen (Fahrstühle, Stationen) entstehen. Die Folge ist, dass der Einsatz von uninformierter Planung zwingend eine Konfliktlösungsstrategie benötigt. So muss es also klare Regeln geben, nach denen die Fahrzeuge bzw. die Agenten einen vorliegenden Konflikt auflösen. Im Beispiel des Entgegenkommen im engen Gang (Abbildung 4.19) könnte das zum Beispiel sein:

- Das Fahrzeug welches die kürzere Strecke zurücksetzen muss, setzt zurück,
- Das Fahrzeug welches den wichtigeren / dringenderen Auftrag fährt, setzt zurück,
- Das Fahrzeug was in einen weniger stark frequentierten Bereich zurücksetzen müsste, setzt zurück.

Weitere Kriterien sind, wie auch eine parametrisierte Kombination aus mehreren Kriterien, denkbar.

Durch die Notwendigkeit einer Kollisionsauflösung steigt die Komplexität des Verfahrens. Die Berechnung einer kürzesten Route mit Hilfe des Dijkstra Algorithmus ist bekanntlich ja nur $\mathcal{O}(n \cdot \log n + m)$ (wobei n die Anzahl der Knoten im Graphen und m die Anzahl der Kanten im Graphen angibt). Weitere Beschleunigungen sind zum Beispiel durch den A* Algorithmus (siehe [HNR68]) oder bei wenigen Start- Zielknotenpaaren sogar durch Vorberechnung aller Routen und Nachschlagen in einer lookup-table möglich. Das Auflösen von Konflikten kann

hier komplexer sein, als das Berechnen der Routen. Durch den Zeitaufwand, der das Auflösen der Konflikte hervorruft (das Zurücksetzen eines Fahrzeugs zum Beispiel) sinkt außerdem die System Performanz.

4.7.2 Informierte / Konfliktvermeidende Planung

Bei der informierten Planung wissen die Agenten über die Pläne der anderen Agenten Bescheid, wenn sie eine Route planen. Sie können somit die Pläne der anderen Agenten in ihren Planungen berücksichtigen. Da die Agenten in der vorliegenden Arbeit an einem gemeinsamen Ziel arbeiten (dem Warentransport in einer Anlage) und also keine egoistischen Ziele verfolgen, berücksichtigen sie die Pläne der anderen Agenten nicht nur, sie respektieren sie sogar. Es handelt sich also nicht nur um eine informierte Planung sondern um eine konfliktvermeidende Planung.

Als Grundlage für das konfliktvermeidende Routing wurde in dieser Arbeit der free-time-window Algorithmus von ter Mors et.al. verwendet (siehe Algorithmus 3 auf Seite 55). Dieser Algorithmus liefert eine unter den vorliegenden Bedingungen (also insbesondere unter den vorliegenden Reservierungen) eine optimale, in der Regel also eine kürzeste Route. Nach Durchlaufen des Algorithmus muss die resultierenden Route reserviert werden. Dies geschieht in der dezentralen Umsetzung durch Nachrichten. Außerdem muss die Ausgabe des Algorithmus, also die Liste aus Ressourcen-Eingangszeiten-paaren in Aktionen für den Agenten umgerechnet werden. Das Verfahren zur Bestimmung der Route in Aktionen ist also zweistufig:

Algorithmus 4 actionTFW

1: gegeben: Graph mit Reservierungen G, *start*, *ziel*, t_0
2: $tfw(start, ziel, t_0, G) \rightarrow L_R$
3: $transform(L_R, t_0) \rightarrow L_A$
4: **return** L_A

Als erstes (Zeile 2) wird der time-free-window Algorithmus wie in Abschnitt 3 angegeben durchgeführt. *start* gibt hier die Start- *ziel* die Zielressource und t_0 den Startzeitpunkt an. Die Angabe des Startzeitpunktes ist hier relevant, da die Reservierungen im Graphen für bestimmte Zeitintervalle gelten und weil eine Planung nicht nur für den aktuellen Zeitpunkt sondern auch für einen zukünftigen Zeitpunkt durchgeführt werden kann. Der Algorithmus ermittelt eine kürzeste, konfliktfreie (also Reservierungen respektierende) Route in Form einer Liste von Ressourcen und Eingangszeiten (L_R). Eine solche Liste könnte zum Beispiel wie folgt aussehen:

$$\{r_1, t_1\}, \{r_2, t_2\}, \{r_3, t_3\}, \ldots, \{r_n, t_n\}$$

wobei r_i die i-te Ressource angibt und t_i den Zeitpunkt in dem diese Ressource im Plan betreten wird. Die folgende Teilliste:

$$\{Kante_1, 20\}, \{Kante_2, 45\}$$

bedeutet also, dass der Agent (bzw. das Fahrzeug) die Kante $Kante_1$ zum Zeitpunkt 20 betritt und die Kante $Kante_2$ zum Zeitpunkt 45. Daraus folgt, dass der Agent sich 25 Zeiteinheiten auf der Kante $Kante_1$ aufhält. Wenn die kürzeste Traversierungszeit für die Kante und das Fahrzeug kürzer ist (wenn das Fahrzeug bei maximaler Geschwindigkeit also weniger Zeit als diese 25 Zeiteinheiten benötigt), dann heißt das praktisch, dass das Fahrzeug auf der Kante

$Kante_1$ eine Pause einlegt oder die Kante mit einer geringeren Geschwindigkeit befährt. Ob diese Differenz zwischen kürzest möglicher und tatsächlicher Traversierungszeit durch Warten oder durch ein Verlangsamen realisiert wird, ist jedoch nur für eine technische Umsetzung, nicht jedoch für die Planung relevant.

Die letzte Eingangszeit (im obigen Beispiel t_n) gibt gleichzeitig die Ankunftszeit im Ziel an. Die Differenz aus der letzten Eingangszeit t_n und dem Startzeitpunkt der Planung t_0 entspricht der Dauer der Route. Ist die erste Eingangszeit t_1 größer als der Startzeitpunkt der Planung t_0, so muss der Plan mit einer Warte-Aktion beginnen.

Die Umrechnung der Ressource-Eintrittszeit-Paar Liste in eine Liste von Aktionen geschieht wie folgt:

Algorithmus 5 transform

```
1: gegeben: L_R und t_0 sowie D(r) für alle r
2: initialisiere L_A
3: t_t = t_0
4: r_t = null mit D(null) = 0
5: for all {r, t} in L_R do
6:    if t > t_t + D(r_t) then
7:       L_A.append(Warte-Aktion(t − (t_t + D(r))))
8:    end if
9:    L_A.apppend(GoTo-Aktion(r))
10:   t_t = t
11:   r_t = r
12: end for
13: return L_A
```

Gegeben ist hier die Liste der Ressourcen-Eingangszeit-Paare L_R und der Startzeitpunkt t_0. Außerdem liegen die minimalen Traversierungsdauern $D(r)$ für alle Ressourcen $r \in R$ vor. Die Liste der Aktionen L_A wird als leere Liste initialisiert (Zeile 2). Die Hilfsvariable t_t wird mit t_0 initialisiert (Zeile 3). Sie steht für die aktuelle Planungszeit. Die Hilfsvariable r_t wird mit *null* initialisiert. Sie steht für die zuletzt berücksichtigte Ressource. Dann wird für jedes Element der Liste L_R überprüft, ob die Eingangszeit in die Ressource größer ist, als die aktuelle Planungszeit plus der minimalen Traversierungszeit der Ressource (Zeile 6). Ist das der Fall, wird eine Warte-Aktion mit der entsprechenden Differenz zu der Liste L_A hinzugefügt (Zeile 6). In jedem Fall wird eine GoTo-Aktion mit der aktuellen Ressource als Ziel zur Liste L_A hinzugefügt (Zeile 9) und die aktuelle Planungszeit auf die Eingangszeit der Ressource gesetzt (Zeile 10) sowie die zuletzt berücksichtigte Ressource auf die aktuelle gesetzt (Zeile 11). Am Ende der for-Schleife liegt dann die Liste der Aktionen vor.

Beispiel

Betrachten wir folgendes Beispiel:

$$t_0 = 1$$
$$L_R = \{\{r_1, 5\},\ \{r_2, 15\},\ \{r_3, 30\},\ \{r_4, 40\}\}$$
$$D(r_1) = 8,\ (D(r_2) = 10,\ D(r_3) = 10$$

Der $transform$ Aufruf generiert aus der Liste L_R, der Planstartzeit $t_0 = 1$ und den minimalen Traversierungsdauern folgende Aktionen Liste L_A:

{{Wait (6)}, {GoTo (r_2)}, {Wait (5)}, {GoTo (r_3)}, {GoTo (r_4)}}

Der Plan zur Durchführung eines Transport- Park- oder Ladeauftrags ist dann eine Kombination aus Teilplänen.

Die Erstellung eines Plans zur Durchführung eines Transportauftrags ist in Algorithmus 6 gegeben:

Algorithmus 6 composeTransportPlan

1: gegeben: Transportauftrag o, Graph mit Reservierungen G, Planstartzeitpunkt t_0,
2: Planstartressource r_0
3: $L_{A_1} = transform(actionTFW(G, r_0, o.s, t_0), t_0)$
4: $L_{A_2} = \{Load\}$
5: $L_{A_3} = transform(actionTFW(G, o.s, o.t, t_0 + dauer(L_{A_1}) + dauer(L_{A_2}), t_0 + dauer(L_{A_1}) + dauer(L_{A_2}))$
6: $L_{A_4} = \{UnLoad\}$
7: $L_A = L_{A_1}.append(L_{A_2}).append(L_{A_3}).append(L_{A_4})$
8: **return** L_A

Der Plan (in Form einer Liste von Aktionen) zur Durchführung eines Transportauftrags ist also die Aneinanderreihung von vier Aktionslisten. Zunächst wird die Liste der Aktionen bestimmt, die nötig ist, um zum Startort des Transportauftrags zu gelangen (Zeile 3). Anschließend wird das Laden des Transportguts eingeplant (Zeile 4). Als dritter Schritt wird der Weg vom Startort des Transportauftrags zum Zielort bestimmt (Zeile 5). Hierfür ist es auf Grund der zeitlichen Natur der Reservierungen nötig, dass die Planung nicht vom aktuellen Zeitpunkt bzw. von Planstartzeitpunkt ausgeht sondern von dem Zeitpunkt, zu dem diese Route starten wird. Dieser Zeitpunkt entspricht dem Planstartzeitpunkt plus der Dauer der beiden Teilpläne für die erste Route und das Beladen. Der vierte Teilplan ist für das Entladen des Transportguts notwendig (Zeile 6). Zurückgegeben wird schließlich der komplette Plan.

Beispiel
Beispiel eines vollständigen Transportplans:

{{Wait (4)}, {GoTo (r_2)}, {Wait (5)}, {GoTo (r_3)}, {GoTo (r_4)}, {Load}, {GoTo(r_5)}, {GoTo(r_7)}, {UnLoad}}

Andere Pläne werden auf ähnliche Weise komponiert. Als Beispiel der Plan für das Laden der Batterie.

Algorithmus 7 composeChargingPlan

1: gegeben: Zugewiesene Ladestation r_l, Graph mit Reservierungen G,
2: Planstartzeitpunkt t_0, Planstartressource r_0
3: $L_{A_1} = transform(actionTFW(G, r_0, r_l, t_0), t_0)$
4: $L_{A_2} = \{StartRecharge\}$
5: $L_{A_3} = \{StopRecharge\}$
6: $L_A = L_{A_1}.append(L_{A_2}).append(L_{A_3})$
7: **return** L_A

Hier wurde davon ausgegangen, dass bereits eine Ladestation zugewiesen wurde. Ist das nicht der Fall, müsste die Komposition mit einer RequestRessource-Aktion begonnen werden.

Abschließend muss der Agent einen gemachten Plan reservieren, dass heißt er muss die Ressourcen (Kanten, Knoten, Stationen, usw.) die er für die Durchführung seines Plans benötigt mit Sperren (locks) versehen. Ein Lock besteht dabei aus folgenden Daten:

- Startzeit,
- Endzeit,
- Ressource,
- Typ,
- ID des sperrenden Agenten.

Der Typ entspricht im Prinzip der Aktion, für die Sperre benötigt wird. Ein lock wird direkt mit der entsprechenden Ressource gespeichert. Die Ressource gehört dennoch zu den Daten des Locks, da diese Informationen auch an andere Agenten übertragen werden müssen.

Der Algorithmus zum Bestimmten der Locks ist in Algorithmus 8 gegeben.

Algorithmus 8 calculateLocks

gegeben: Zu reservierender Plan L_A, Graph G, $D(r)$ für alle r, t_0, r_0
$R = \emptyset$
$t_t = t_0$
$r_t = r_0$
for all Aktion $a \in L_A$ **do**
  if a is Wait-Aktion **then**
    $R.append(newLock(t_t, t_t + dauer(a), r_t, wait))$
    $t_t = t_t + dauer(a)$
  else if a is Load-Aktion **then**
    $R.append(newLock(t_t, t_t + dauer(a), r_t, load))$
    $t_t = t_t + dauer(a)$
  else if a is UnLoad-Aktion **then**
    $R.append(newLock(t_t, t_t + dauer(a), r_t, unload))$
    $t_t = t_t + dauer(a)$
  else if a is StartRecharge-Aktion **then**
    $R.append(newLock(t_t, t_t + dauer(a), r_t, recharge))$
    $t_t = t_t + dauer(a)$
  else if a is StopRecharge-Aktion **then**
    $R.append(newLock(t_t, t_t + dauer(a), r_t, recharge))$
    $t_t = t_t + dauer(a)$
  else if a is Park-Aktion **then**
    $R.append(newLock(t_t, \infty, r_t, idle))$
  else if a is GoTo-Aktion **then**
    $R.append(newLock(t_t, t_t + dauer(a), r_t, goto))$
    $t_t = t_t + dauer(a)$
    $r_t = (GoTo)a.getTarget()$
  end if
end for
return R

Hier entspricht t_t immer der aktuellen Zeit im Plan und r_t der Ressource in der sich der Plan gerade befindet. Mit $dauer(a)$ wird die Dauer, die die gerade betrachtete Aktion hat, angegeben. Bei einer GoTo-Aktion entspricht die Dauer der minimalen Traversierungsdauer (es gilt also $dauer(a) = D(r)$). Die Dauer einer Load oder UnLoad Action kann vom Fahrzeug und oder vom Transportauftrag abhängen oder ein fester Parameter sein. Die Dauer einer StartRecharge-Aktion wird in der Regel vom Ladestand der Batterie abhängen. Der Algorithmus geht alle vorliegenden Aktionen durch (Zeile 5) und erstellt aus jeder Aktion eine passende Reservierung (Zeilen 7,10,13,16,19,22,24). Anschließend modifiziert er die aktuelle Planzeit (Zeilen 8,11,14,17,20,25). Eine Ausnahme hiervon ist die Park-Aktion. Da Parken im entwickelten System immer bedeutet, dass der Agent keine weiteren Aufträge hat, ist Parken immer die letzte Aktion in einem Agentenplan. Außerdem ist für das Parken kein Ende absehbar (das Parken wird erst beendet, wenn dem Agent ein neuer Transportauftrag zugewiesen wird). Daher wird das Ende der Parkreservierung auf ∞ gesetzt. Die Parkreservierung ist also eine Dauerreservierung. Bei der Bearbeitung einer GoTo-Aktion muss noch die aktuelle Ressource auf das Ziel der GoTo-Aktion gesetzt werden (Zeile 26). Es wird davon ausgegangen, dass die RequestRessource Aktion und die FreeRessource Aktion keine Zeit benötigen. Daher muss für diese beiden Aktionen keine Ressource reserviert werden und daher werden die beiden Aktions-Typen im Algorithmus übersprungen. Der Algorithmus liefert eine Liste, aller Reservierungen für den übergebenen Plan zurück.

4.7.3 Kooperative Planung

Das im vorherigen Abschnitt vorgestellte konfliktfreie Routing ermittelt zwar die optimale Route unter Berücksichtigung aller vorliegenden Reservierungen, dass bedeutet aber nicht, dass das Gesamtsystem Verhalten optimal ist. Ein Beispiel für ein nicht optimales Gesamtsystemverhalten kann aus der Situation, die in Abbildung 4.20 dargestellt ist abgeleitet werden.

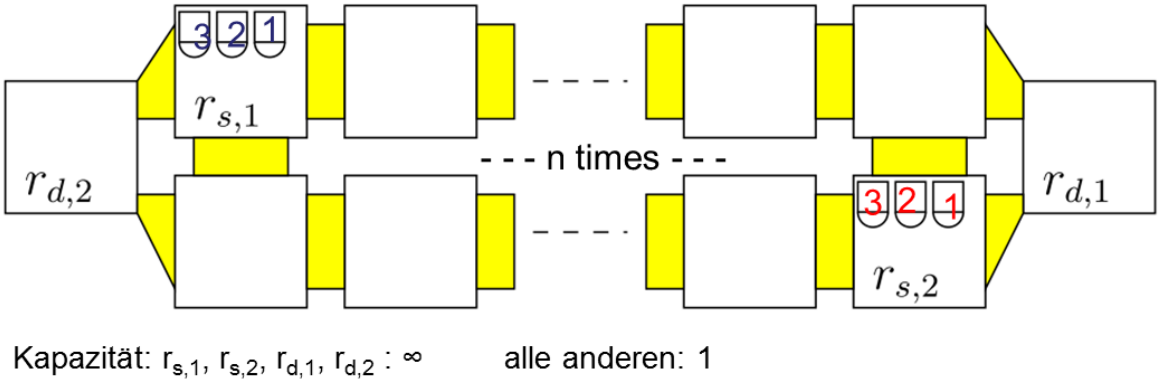


Abbildung 4.20: Beispiel für nicht optimales Gesamtsystem Verhalten [nach: [Mor11]]

In dem Beispiel befinden sich zum Startzeitpunkt drei Agenten im Knoten $r_{s,1}$ (schwarze 1,2 und 3) sowie drei Agenten im Knoten $r_{s,2}$ (rote 1,2 und 3). Das Ziel der drei schwarzen Agenten ist der Knoten $r_{d,1}$, das Ziel der drei roten Agenten der Knoten $r_{d,2}$. Auf den Knoten $r_{s,1}$,$r_{s,2}$,$r_{d,1}$ und $r_{d,2}$ können beliebig viele Agenten zur gleichen Zeit sein. Auf allen anderen Knoten und Kanten kann jeweils nur ein Agent zur gleichen Zeit sein. Das Layout besteht im wesentlichen aus zwei Ketten mit jeweils n Knoten. Wir nennen diese Ketten im folgenden die obere Kette und die untere Kette.

Gehen wir nun davon aus, dass zu erst die schwarzen Agenten ihre Routen planen (in der Reihenfolge 1,2,3) und anschließend die roten Agenten (ebenfalls in der Reihenfolge (1,2,3). Weiterhin seien alle Kantenlängen (Kanten sind in der Abbildung in gelb dargestellt) und die Geschwindigkeit aller Fahrzeuge genau 1. Zunächst plant der schwarze Agent 1. Er plant den direkten Weg der oberen Kette. Insgesamt hat sein Plan so eine Dauer von n Zeiteinheiten. Der schwarze Agent 2 könnte nun entweder eine Zeiteinheit auf dem Knoten $r_{s,1}$ warten und anschließend den selben Weg nehmen oder er könnte im ersten Schritt zur unteren Kette wechseln und anschließend die untere Kette benutzen. In beiden Fällen ergibt sich eine Dauer von n+1 Zeiteinheiten. Der Einfachheit halber nehmen wir an, dass der Agent eine Zeiteinheit wartet und anschließend die obere Kette nimmt. Der dritte Agent könnte nun zwei Zeiteinheiten auf $r_{s,1}$ warten und anschließend den selben Weg nehmen, wie die ersten beiden Agent was zu einer Dauer von n+2 Zeiteinheiten führt oder er nimmt die untere Kette, was zu einer Dauer von n+1 Zeiteinheiten führt. Da die Agenten beim konfliktfreien Routing wie hier vorgestellt immer die kürzeste, konfliktfreie Route nehmen, nimmt der Agent die untere Kette. Der rote Agent 1 muss nun n+1 Zeiteinheiten auf $r_{s,2}$ warten, da die obere Kette bis zum Zeitpunkt n+1 und die untere Kette bis zum Zeitpunkt n+2 reserviert ist. Nach dem Warten reserviert er die obere Kette. Sein Plan hat also eine Dauer von 2n+1 Zeiteinheiten. Analoge Überlegungen für die roten Agenten 1 und 2 ergeben Dauern von je 2n+2 Zeiteinheiten für die Routen der beiden Agenten. Insgesamt haben die 6 Pläne der Agenten so eine Dauer von 9n+7 Zeiteinheiten.

Würden die schwarzen Agenten jeweils oben planen ergäben sich folgende Pläne: Der schwarze Agent 1 plant wie oben, Dauer n. Der schwarze Agent 2 wartet eine Zeiteinheit und folgt dem ersten Agent anschließend, Dauer n+1. Der schwarze Agent 3 wartet zwei Zeiteinheiten und folgt anschließend den anderen beiden Agenten, Dauer n+2. Die roten Agent planen analog auf der unteren Kette. Insgesamt haben die 6 Pläne so eine Dauer von 6n+6 Zeiteinheiten, was eine deutlich kürzerer Gesamtdauer darstellt.

Global betrachtet handelt es sich bei dem Problem um das multi vehicle, multi targets Problem, also das Problem für mehrere Fahrzeuge Routen zu verschiedenen Zielen zu finden, so dass die Summe aller Routen möglichst klein ist. Dieses Problem ist NP-vollständig (für eine genaue Definition des Problems und einen Beweis zur NP-Vollständigkeit siehe zum Beispiel [Mor10]). Um das globale Systemverhalten zu verbessern wurde im Rahmen dieser Arbeit eine kooperative Planung konzipiert.

Bei der kooperativen Planung wissen die Agenten nicht nur über die Pläne der anderen Agenten Bescheid, sie können außerdem versuchen diese zu beeinflussen. Kooperativ meint hier, dass die Agenten ein gemeinsames Ziel verfolgen. In der vorliegenden Arbeit ist das eine bestmögliche Systemperformanz. Das hat zur Folge, dass ein Agent einen anderen nur dann vom Abweichen von seinen Plänen überzeugen will, wenn das zu Vorteilen für das Gesamtsystem führt. Davon ausgehend, dass der vorliegende Plan eines Agent einen für diesen bestmöglichen darstellt, bedeutet das also, dass der Nachteil für den nachgebenden Agenten kleiner sein muss, als der Vorteil für den nachfragenden Agenten. Die Bewertung hängt hier natürlich an der gewählten Zielfunktion.

In der vorliegenden Arbeit interagieren die Agenten untereinander in den Bereichen Auftragszuordnung und Streckenplanung. Die Auftragszuordnung wurde im Abschnitt 4.6 im Detail beschrieben. Der Austausch über die Streckenplanung erfolgt in der vorliegenden Arbeit über Reservierungen (locks - siehe Abschnitt 4.7.2). Agenten erkennen Konflikte zwischen ihren idealen Routen und den Plänen anderer Agenten über die Reservierungen auf dem Graphen.

Der Versuch, einen Agenten zu einer Änderung seines Plans zu bewegen bedeutet somit also den Agenten zur Veränderung bzw. Rücknahme von von ihm gemachten Reservierungen zu überzeugen.

Im folgenden soll der allgemeine Ansatz für Verhandlungen um Reservierungen sowie zwei Beispielverhandlungen beschrieben werden.

4.7.3.1 Allgemeines Verhandlungsmodell

Mit Verhandlungen sollen Agenten in die Lage versetzt werden, bereits getätigte Reservierungen von anderen Agenten zu übernehmen um damit ihre eigene Route zu verbessern. Agenten, die ihm Rahmen von Verhandlungen Reservierungen abgeben müssen an dieser Stelle eine neue Route berechnen, die ohne die abgegebenen Reservierungen auskommt. Insgesamt soll durch solche Verhandlungen das Gesamtsystem Verhalten verbessert werden. Wie bereits im letzten Abschnitt angegeben, ist das multi vehicle, multi target Problem NP-vollständig. Das bedeutet, dass auch mit der Hilfe von Verhandlungen das Erreichen eines optimalen Gesamtsystemverhaltens ein NP-vollständiges Problem ist. Im Allgemeinen lässt sich mit realistischem Rechenaufwand das Gesamtsystemverhalten mit der Hilfe von Verhandlungen also nur verbessern, nicht aber ein optimales Verhalten erreichen.

Um ein optimales Systemverhalten mit Hilfe von Verhandlungen zu erreichen, müsste ein Agent bei jeder Routenplanung Verhandlungen starten, wenn die kürzeste konfliktfreie Route länger ist, als die kürzeste Route ohne Berücksichtigung von Reservierungen. Liegen auf der kürzesten Route Reservierungen von mehreren Agenten, so müsste der planende Agent Verhandlungen mit mehreren Agenten aufnehmen. Die Agenten, mit denen Verhandlungen aufgenommen werden müssen im Rahmen der Verhandlungen alternative Routen berechnen, um zu ermitteln wie hoch die Auswirkungen des Abgebens von Reservierungen für sie sind. Im Zuge dieser Bestimmung von alternativen Routen müssen auch diese Agenten gegebenenfalls Verhandlungen beginnen. Sollte der ursprünglich planende Agent die kürzeste Route nicht bekommen weil ein oder mehrere angefragte Agenten nicht auf die entsprechenden Reservierungen verzichten, so müsste der Agent versuchen andere Routen zu finden, die kürzer (bzw. besser nach der verwendeten Zielfunktion) sind als die beste konfliktfreie Route. Insgesamt würde sich so ein sehr aufwändiges, möglicherweise zyklisches Verhandlungssystem ergeben, welches letztlich zum beschriebenen NP-vollständigen Problem führt.

Stattdessen wurde in dieser Arbeit ein Verhandlungsmodell entwickelt bei dem eine Verhandlung nur gestartet wird, wenn bestimmte Kriterien erfüllt sind. Hierdurch soll der Aufwand für das Verhandeln begrenzt werden. Gleichzeitig wird hierdurch natürlich auch der Effekt von Verhandlungen begrenzt. Insgesamt soll gezeigt werden, dass es mit der Hilfe von Verhandlungen prinzipiell möglich ist, das Gesamtsystemverhalten zu verbessern.

In den nächsten Abschnitten werden beispielhaft zwei Verhandlungen beschrieben. Zur Beschreibung gehören jeweils die Kriterien für den Verhandlungsstart, die Kriterien, die erfüllt sein müssen, damit die angefragten Agenten die entsprechenden Reservierungen abgeben sowie Details zur Implementierung.

4.7.3.2 Idle-Lock Verhandlungen

idle-lock Verhandlungen

Die Kriterien für den Verhandlungsstart sind:

- Alle Reservierungen, die die kürzere bzw. bessere Route blockieren, sind Reservierungen für Parkplätze bzw. für Fahrten zu Parkplätzen,
- Die kürzeste konfliktfreie Route ist mindestens um den Faktor ϵ_{IL} größer als die Route, für die die Verhandlung gestartet werden soll.

Angefragte Agenten geben Reservierungen unter folgenden Bedingungen auf:

- Es existiert eine alternative, konfliktfreie Route.

Bei den ersten Untersuchungen mit dem konfliktfreien Routing (Abschnitt 4.7.2) im Szenario der Getränkeabfüllanlage (siehe Experimente - Abschnitt 6.1) fiel auf, dass die Fahrzeuge einen relevanten Teil der Zeit mit der Fahrt zu Parkplätzen verbracht haben. Diese Fahrten machten folglich auch einen nicht irrelevanten Teil der gefahrenen Gesamtstrecke aus. Das Parken ist aber notwendig, um das Blockieren von Stationen zu verhindern. Andernfalls würden die Fahrzeuge die Stationen, an denen ihr letzter Auftrag endete sowie ggf. die Zufahrten zu diesen für andere Fahrzeuge unerreichbar machen.

Um das Problem anzugehen wurde die idle-lock Verhandlung konzipiert. Hierbei fahren die Agenten nicht zu einem Parkplatz, wenn sie Ihren letzten Auftrag ausgeführt haben, sondern bleiben an ihrer aktuellen Position stehen, wenn für diese keine weiteren Reservierungen in der Zukunft vorliegen. Sie reservieren dann diesen Ort mit einem dauerhaften idle-lock (siehe Algorithmus 8 auf Seite 93). Gibt es für ihre aktuelle Position in der Zukunft liegende Reservierungen, suchen sie die nächstgelegenen Position, für die es keine in der Zukunft liegenden Reservierungen gibt, fahren diese an und bleiben dort stehen. Sie reservieren dann die angefahrene Postion mit einem dauerhaften idle-lock. Ist das Szenario dicht befahren, kann es hier in letzter Konsequenz durchaus dazu kommen, dass die nächstgelegene Position ohne in der Zukunft liegende Reservierungen ein Parkplatz ist, das Fahrzeug am Ende seines letzten Plans also doch zu einem Parkplatz fährt.

Die Routenplanung eines Agenten ist in den Algorithmen 9 und 10 beschrieben.

Algorithmus 9 idleLockRouting

1: gegeben: Graph mit Reservierungen G, $start$, $ziel$, t_0, ϵ
2: $tfw(start, ziel, t_0, G) \rightarrow L_R$
3: build $G' = G$ ohne $idle$-Reservierungen
4: $tfw(start, ziel, t_0, G') \rightarrow L'_R$
5: **if** dauer$(L_R) > \epsilon_{IL}$·dauer(L'_R) **then**
6: **if** $tryGetIdleLocks(G, ID, L'_R)$ **then**
7: $transform(L'_R, t_0) \rightarrow L_A$
8: **else**
9: $transform(L_R, t_0) \rightarrow L_A$
10: **end if**
11: **end if**
12: **return** L_A

Algorithmus 10 tryGetIdleLocks

```
1: gegeben: Graph mit Reservierungen G, ID des anfragenden Agenten, L_R,
2:          Map< ID, Agent > aller Agenten im System
3: for all idle lock Reservierung l in G die in Konflikt mit L_R stehen do
4:   if not owenerOf_l.canHaveLock(l) then
5:     return false
6:   end if
7: end for
8: return true
```

Wenn ein Agent im idle-lock Verfahren eine Route plant, plant er zunächst eine konfliktfreie Route wie in Abschnitt 4.7.2 beschrieben (Algorithmus 9 - Zeile 2). Als zweites berechnet er eine konfliktfreie Route, bei der er alle Reservierungen (locks) mit dem Typ *idle* ignoriert (Algorithmus 9 - Zeile 4). Anschließend vergleicht er die Ergebnisse der beiden Routen anhand der gegebenen Güte-Funktion (in der Regel also nach der Dauer der Route). Ist die Route, in der die *idle*-Reservierungen ignoriert werden, um einen vorgegebenen Faktor ϵ_{IL} besser (Algorithmus 9 - Zeile 5), so fordert der Agent die Inhaber dieser *idle*-Reservierungen auf, diese freizugeben. Mit dieser Aufforderung zusammen verschickt er vorläufige Reservierungen für die von ihm geplante Route (Algorithmus 9 - Zeile 6 + Algorithmus 10). Bekommt er von allen beteiligten Agenten die benötigten idle-locks (liefert der Algorithmus 10 also true zurück), transformiert er die zweite, kürzere Route und gibt diese zurück (Algorithmus 9 - Zeile 7+12). Bekommt er nicht alle Freigaben, transformiert er die erste Route und gibt sie zurück (Algorithmus 9 - Zeile 9+12).

Im Algorithmus 10 werden zunächst alle relevanten idle-locks bestimmt, also genau diejenigen die im Konflikt mit der gewünschten Route L_R stehen (Zeile 3). Für jede dieser Reservierungen wird dann überprüft, ob der Inhaber diese freigeben würde (Zeile 4). Ist das für eine Reservierung nicht der Fall, wird der Vorgang abgebrochen und false zurückgegeben. Erfolgt die Freigabe für jede Reservierung, wird true zurückgegeben.

Die Anfrage an die Inhaber der Reservierungen (Algorithmus 10 - Zeile 4) erfolgt über Nachrichten (zur Konzeption der Nachrichten siehe Abschnitt 4.3, zur Implementierung der Nachrichten Abschnitt 4.4 sowie Unterabschnitte). Jeder Agent, dem eine oder mehrere relevante idle-Reservierung gehört, wird mit einer LockNegotiation-Nachricht angeschrieben und um Freigabe dieser Reservierung gebeten. In der Nachricht enthalten ist die geplante Route des anfragenden Agenten. Da idle-Reservierungen nur von Agenten gemacht werden, die aktuell keinen Auftrag zugeordnet haben, versuchen diese Agenten, allen Anfragen nach Freigabe von idle-Reservierungen zu entsprechen. Der einzige Grund, eine solche Anfrage abzulehnen ist der, dass sie keine konfliktfreie Route zu einem anderen, möglichen Parkplatz (also einer Ressource im Graphen, für die keine Reservierung vorliegt) finden können. Erhält ein Agent also eine Anfrage zur Freigabe einer idle-Reservierung erzeugt er einen temporären Graphen G_{temp}, der aus dem aktuellen Systemgraphen G sowie aus allen Reservierungen die sich aus der geplanten Route L_R des anfragenden Agenten entsteht. In diesem Graphen G_{temp} versucht der Agent dann mittels des free-time-window Algorithmus eine konfliktfreie Route zu einer nahegelegenen Ressource ohne Reservierungen zu finden. Gelingt ihm das, reserviert er diese Route und gibt die angefragte idle-Reservierungen frei. Gelingt ihm das nicht, lehnt er die Anfrage um Freigabe ab.

Das Vorgehen ist in Algorithmus 11 dargestellt. Der Algorithmus wird bei Erhalt einer

LockNegoation-Nachricht, die die Aufforderung der Freigabe einer idle-Reservierung erhält, angestoßen.

Algorithmus 11 tryToFree

```
1: gegeben: Graph mit Reservierungen G, geplante Route des anfragenden Agenten L_R,
2:          Postion des Agenten r_0, aktuelle Zeit t_0
3: bestimme G_temp = G + alle Reservierungen die sich aus L_R ergeben
4: bestimme Liste F aller Ressourcen ohne Reservierungen in G_temp
5: sortiere F ansteigend nach der Entfernung zu r_0
6: for all r ∈ F do
7:   if L_A=actionTFW(G_temp, r_0, r, t_0) ≠ ∅ then
8:     reserviere L_A und setzte L_A aktiv
9:     return true
10:  end if
11: end for
12: return false
```

Das Reservieren des neuen Plans (Zeile 8) erfolgt durch das Berechnen der notwendigen Reservierungen (siehe Algorithmus 8) sowie dem Verschicken der Reservierungen in Form von Nachrichten an die anderen Agenten.

4.7.3.3 Single Blocker Verhandlungen

Single Blocker Verhandlungen

Die Kriterien für den Verhandlungsstart sind:

- Alle Reservierungen, die die kürzere bzw. bessere Route blockieren, stammen von einem einzelnen Agenten,
- Die kürzeste konfliktfreie Route ist mindestens um den Faktor ϵ_{SB} größer als die kürzere bzw. bessre Route, die nur durch einen Agenten blockiert ist.

Angefragte Agenten geben Reservierungen unter folgenden Bedingungen auf:

- Es existiert eine alternative, konfliktfreie Route, die für den Agenten zu einer Verschlechterung führt die kleiner ist als der mögliche Nutzen des anfragenden Agenten.

Eine naheliegende Verhandlungsart ist diejenige, an der nur zwei Agenten beteiligt sind. Bei dieser Verhandlung ist der Aufwand auf Grund der begrenzten Anzahl durchzuführender Routenplanungen überschaubar. Die entsprechende Routenplanung ist in Algorithmus 12 angegeben.

Zunächst wird die konfliktfreie Route berechnet und in L_R gespeichert (Zeile 3). Anschließend wird die Liste der Agenten im System durchgegangen und je eine Route im Graphen, der die Reservierungen des jeweiligen Agenten nicht enthält, geplant und in einer Map gespeichert(Zeile 5-8). Die Map wird dann fallend nach den Dauern bzw. den Kosten der so ermittelten Routen sortiert (Zeile 9). Anschließend werden Verhandlungen durchgeführt. Beginnend mit der besten Route wird, falls die Route mindestens um den Faktor ϵ_{SB} besser ist

Algorithmus 12 singleBlockerRouting

```
1: gegeben: Graph mit Reservierungen G, start, ziel, t_0, ϵ
2:          Map< ID, Agent > map (alle Agenten im System)
3: tfw(start, ziel, t_0, G) → L_R
4: v:=Vector< Route, Agent >
5: for all Agent a in map do
6:   build G' = G ohne Reservierungen von a
7:   v.add{tfw(start, ziel, t_0, G'), a}
8: end for
9: sortiere v absteigend nach der Dauer (den Kosten) der geplanten Route
10: for all {L_R^a, a} in v do
11:   if dauer(L_R) > ϵ_SB·dauer(L_R^a) then
12:     if a.tryGetSBLocks(G, ID, L_R^a,(dauer(L_R)−dauer(L_R^a)) then
13:       transform(L_R^a, t_0) → L_A
14:       end for
15:     end if
16:   end if
17: end for
18: if L_A==null then
19:   transform(L_R, t_0) → L_A
20: end if
21: return L_A
```

als L_R, der entsprechende Agent angefragt, ob er die für die Route notwendigen Reservierungen frei gibt (Zeile 12). Wenn das der Fall ist, wird die entsprechende Route transformiert und die for-Schleife beendet, andernfalls wird mit der nächsten Route auf die selbe Weise verfahren. Konnte keine bessere Route verhandelt werden, wird die ursprünglich berechnete Route transformiert (Zeile 19). Am Ende wird die festgelegte Route zurückgegeben (Zeile 21).

Der angefragte Agent überprüft mit dem in Algorithmus 13 angegebenen Verfahren, ob er die angefragten Reservierungen aufgibt. Hierfür berechnet er den Graphen der aus dem aktuellen Graphen und den Reservierungen, die der anfragende Agent für L_R benötigt besteht (Zeile 5). Er überprüft anschließend für alle ihm zugeordneten Aufträge, wie viel schlechter die Pläne zur Erfüllung dieser Aufträge durch die Abgabe der angefragten Reservierungen werden würden (Zeile 8). Weiterhin überprüft er, ob er durch die Abgabe einen Auftrag nicht mehr erfüllen kann (Zeile 9). Wenn das der Fall sein sollte, so lehnt er die Anfrage sofort ab (Zeile 10). Wenn kein Auftrag durch die Abgabe unerfüllbar wird, überprüft der Agent, ob die Summe der Verschlechterungen, die sich für die ihm zugeordneten Pläne ergibt kleiner ist, als der Gewinn, der sich für den anfragenden Agenten ergibt (Zeile 13). Ist das der Fall, gibt er die angefragten Reservierungen frei und teilt das dem anfragenden Agenten mit (Zeile 14 und 15). Andernfalls verweigert er die Freigabe (Zeile 17).

4.7.3.4 Verhandlungen Ausblick

Das in den vorherigen Abschnitten vorgestellte Konzept für Verhandlungen sowie die beiden Beispiele zeigen eine Möglichkeit, wie sich Verhandlungen für die Verbesserung des Gesamt-

Algorithmus 13 tryGetSBLocks

```
1: gegeben: Graph mit Reservierungen G, ID des anfragenden Agenten, L_R,
2:          Verbesserung für anfragenden Agenten, Vector< Auftrag, Plan > v aller
3:          zugewiesener Aufträge und der entsprechenden Pläne sortiert nach Startzeiten
4: loss=0
5: build G' = G plus die Reservierungen die sich aus L_R ergeben
6: for all {Auftrag, Plan} in v do
7:   Berechne neuen Plan für den Auftrag in G'
8:   loss+= Unterschied alter zu neuer Plan
9:   if Neuer Plan erfüllt Bedingungen an Auftrag nicht then
10:    return false
11:  end if
12: end for
13: if loss < Verbesserung für anfragenden Agenten then
14:   Freigabe der für L_R notwendigen Reservierungen
15:   return true
16: end if
17: return false
```

systemverhaltens eines dezentral gesteuerten Fahrerlosen Transportsystems einsetzen lassen.

Weitere Verhandlungen beziehungsweise Erweiterungen der vorgestellten Verhandlungen sowie die Kombination mehrerer Verhandlungen sind hier natürlich möglich. Beispielsweise werden bei der Single-Blocker Verhandlung die Verhandlungsoptionen nach dem Gewinn für den anfragenden Agenten sortiert durchgeführt und die erste Möglichkeit (also die erste Verhandlung, in der der angefragte Agent seine Reservierungen aufgibt) angenommen. Denkbar wäre hier, dass stattdessen die Verhandlungsoption gesucht wird, bei der die Differenz zwischen dem Gewinn für den anfragenden Agenten und die Einbußen für den abgebenden Agenten möglichst groß ist.

4.8 Simulation

Um die entwickelten Ansätze zur dezentralen Steuerung untersuchen und mit einer zentralen Steuerung vergleichen zu können, wurde eine Simulation konzipiert, die in der Lage ist, die Konzepte umzusetzen. In den folgenden Abschnitten wird diese Simulation kurz beschrieben.

4.8.1 Scheduler und Agenten

Kern der konzipierten Simulation sind die Agenten und der Scheduler. Der Scheduler taktet die Simulation und steuert die Agenten an. Die Länge des Zeitintervalls, die ein Simulationstakt abbilden soll ist frei einstellbar, für die Simulationen der vorliegenden Arbeit entspricht ein Takt allerdings immer einer Sekunde. In jedem Takt ruft der Scheduler alle Agenten einmal auf, welche dann gemäß ihrer step-Methode handeln. Die Agenten sind in drei Level unterteilt (siehe Abbildung 4.21). Im ersten Level ist der bzw. sind die OrderDisposer Agenten, im zweiten Level die BasicFTF-Agenten und im dritten Level diverse Hilfsagenten. Zunächst werden alle Agenten im ersten Level, dann die Agenten im zweiten und zum Schluss

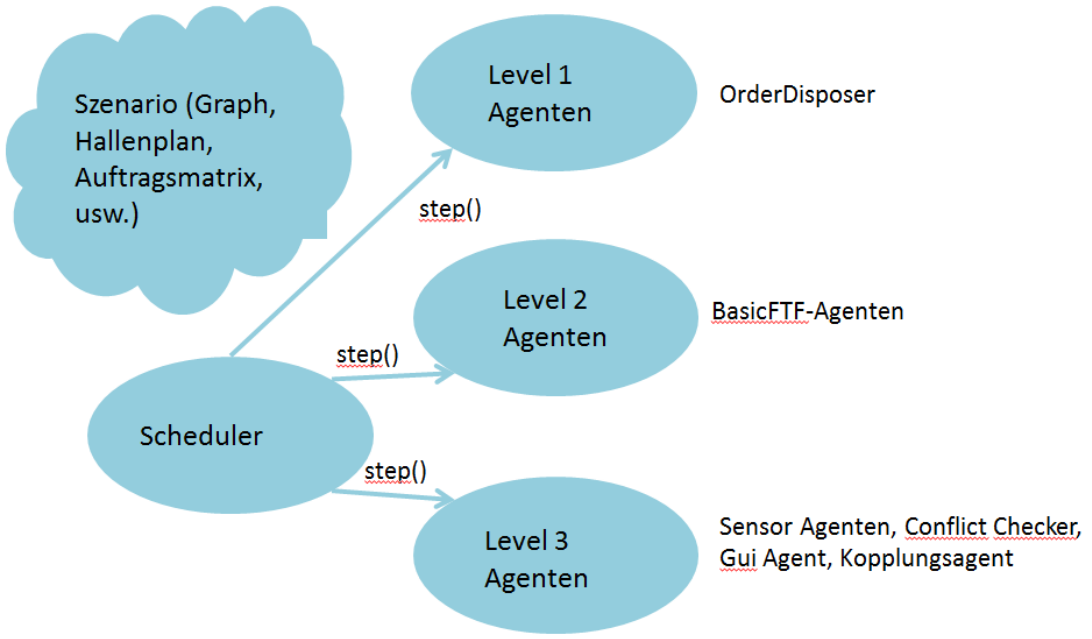


Abbildung 4.21: Scheduler

die Agenten im dritten Level angesprochen. Der Grund hierfür ist, dass die Angebotsvergabe den aktuellen Stand aller Agenten berücksichtigen soll (was nicht der Fall wäre, wenn einige BasicFTF-Agenten schon vor der Auftragsvergabe und einige erste danach handeln würden) und dass die Hilfsagenten, die vor allem für die Kopplung mit anderen Systemen oder für die Visualisierung zuständig sind, den vollständigen Schritt als Basis haben.

Die Agenten sind Klassen, die das *Agenten*-Interface implementieren. Die einzig zwingende Vorgabe ist hier, dass die Agenten eine step-Methode implementieren, die vom Scheduler aufgerufen werden kann.

Wird die step-Methode eines Agenten vom Scheduler aufgerufen handelt er gemäß seines aktuellen Plans (siehe die Abschnitte 4.2 und 4.3).

Insgesamt simuliert der Scheduler auf diese Weise eine voreingestellte Anzahl an Takten um so die gewünschte Zeitspanne zu simulieren. Zum Beispiele würden zur Simulation von 4 Stunden und einer Taktdauer von einer Sekunde 14.400 Takte simuliert werden (4 Stunden = 4*60*60=14.400 Sekunden).

4.8.2 Graph

Der Graph wurde als Menge von Knoten und gerichteten Kanten implementiert, wobei jede Kante jeweils einen Verweis auf ihren Start- und ihren Zielknoten hat und jeder Knoten jeweils Verweise auf alle von ihm ausgehende Kanten hat. Knoten und Kanten haben den gemeinsamen Obertyp Ressource, was notwendig für die Umsetzung des Free-Time-Window Algorithmus war (siehe Abschnitt 4.7.2). Weiterhin sind im Graph Methoden hinterlegt um alle auf den Knoten und Kanten hinterlegten Reservierungen zu erhalten, Reservierungen von

speziellen Kanten oder Knoten zu erhalten oder alle Reservierungen zu löschen, deren Ende vor einem bestimmten Zeitpunkt liegen.

Neben den Verweisen auf die ausgehenden Kanten hat ein Knoten noch eine eindeutige ID, sowie x/y Koordinaten. Optional kann er weitere Annotationen erhalten. Eine Kante hat ebenfalls eine eindeutige ID, sowie einen Geschwindigkeitsfaktor. Letztere wird benutzt um die minimale Travesierungsdauer der Kante zu bestimmen. Für diese gilt in Abhängigkeit von der Geschwindigkeit eines Agenten:

$$\begin{gathered}\text{Minimale Travesierungsdauer} \\ = \\ \text{Euklidische Distanz Startknoten zu Zielknoten} \times \text{Geschwindigkeitsfaktor} \\ \div \text{Geschwindigkeit des Agenten}\end{gathered}$$

Reservierungen enthalten jeweils die Ressource, die sie reservieren, ein Start- und einen Endzeitpunkt für die Reservierung, den Agenten, dem die Reservierung gehört sowie einen Typ. Der Typ wird genutzt um Reservierungen für die Fahrten zu Parkplätzen anderes zu behandeln, als Reservierungen für die Fahrten im Rahmen von Aufträgen (siehe Abschnitt 4.7.3).

4.8.3 Auftragseinlastung und Auftragsauswertung

Die Auftragseinlastung in die Simulation kann wie in Abschnitt 4.3.2 beschrieben auf verschiedene Weisen erfolgen. Die für den jeweiligen Simulationsdurchlauf gültige Auftragsliste wird vor Beginn der eigentlichen Simulation von dem OrderDisposer eingelesen und sortiert nach Erzeugungsdatum in einer Liste gespeichert. Neben dieser Liste speichert der OrderDisposer jeweils den Zeitpunkt, zu dem der nächste Auftrag erzeugt werden muss. In jedem Takt überprüft er in seiner step-Methode, ob zu dem aktuellen Zeitpunkt ein oder mehrere Aufträge ausgeschrieben werden müssen und startet dann für jeden auszuschreibenden Auftrag eine Auktion. Nach dem Eingang der Angebote, vergibt er die Auktion an den Gewinner, und informiert die Verlierer. Sollten im Rahmen der Vergabe ein oder mehrere andere Aufträge zurückgegeben worden sein (siehe Abschnitt 4.6.2.3), werden diese sofort neu ausgeschrieben.

Für die Auftragsauswertung wurde eine zentrale Instanz, der OrderTracker entworfen welcher alle erstellten Aufträge verwaltet. Der OrderTracker übernimmt hier jedoch keine steuernden Funktion innerhalb der Simulation. Er ist für die Auswertung am Ende der Simulation zuständig. Er berechnet dann folgenden Kennzahlen:

- Durchschnittliche und maximale Auftragsdauer,
- durchschnittliche und maximale Transportdauer,
- durchschnittliche und maximale Zeit zwischen Auftragsvergabe und Auftragsstart,
- Anzahl und Quote der nicht vergebenen Aufträge,
- Anzahl und Quote der mehrfach vergebenen Aufträge,
- Anzahl und Quote der Aufträge, die nach ihrer Deadline beendet wurden.

Diese Daten können nach Ablauf der Simulation für die Auswertung der Simulationsergebnisse abgerufen werden (siehe Abschnitt 5.8.2). Weiterhin stellt der OrderTracker eine Liste aller Aufträge die gerade auf eine Beladung warten zur Verfügung. Diese wird vom Gui-Agenten (siehe 5.4.3) genutzt um die auf Abholung wartenden Aufträge darzustellen.

4.8.4 Kommunikation

Für die Simulation wurde ein Nachrichten-System konzipiert welches zum einen die in einer realen Implementierung benötigten Nachrichten abbildet zum anderen aber auch das gezielte Einbringen von Störungen oder Verzögerungen sowie statistische Auswertungen der zu versendenden Nachrichten ermöglicht. Letztere Eigenschaft kann für Experimente zur Bestimmung der Robustheit des Systems bei Kommunikationsproblemen hilfreich sein. Die in Abschnitt 4.4 vorgestellten Nachrichtentypen (OrderAnnouncement, Proposal, ProposalAccepted, ProposalRejected, ReturnOrder, LockChange, LockNegotiation und LockNegotiationAnswer) finden sich alle auch als Klassen in der Simulation wieder. Sie erben alle von der gemeinsamen Oberklasse Message. Diese Oberklasse und weitere konzipierte Klassen für die Simulation der Kommunikation werden in den nächsten Abschnitten beschrieben.

4.8.4.1 Message

Die Klasse *Message* ist die Oberklasse aller Nachrichten. Sie kapselt den Sender einer Nachricht und stellt diesen zur Verfügung. Während der Simulation werden allerdings nur Unterklassen von *Message* versandt.

4.8.4.2 MessageReceiver

Alle Klassen, die die oben vorgestellten Nachrichten empfangen können sollen, müssen das Interface *MessageReceiver* implementieren. Dieses Interface gibt nur eine Methode (*receiveMessage*) vor, dient aber gleichzeitig zur Unterscheidung von kommunizierenden Agenten (die, dieses Interface implementieren) und den nicht kommunizierenden.

4.8.4.3 MessageSender

Jede Kommunikation in der Simulation läuft über den *MessageSender* anstatt direkt zwischen den beteiligten Agenten. Diese Klasse stellt dafür Methoden für direkte 1-zu-1 Kommunikation, Broadcast-Nachrichten (Nachrichten die an alle kommunizierenden Agenten gehen) sowie gefilterte Broadcast-Nachrichten (Nachrichten die an alle kommunizierenden Agenten eines bestimmten Typs gehen, zum Beispiel an alle FTF-Agenten) zur Verfügung. Diese zentrale Kommunikationsinstanz der Simulation hat keine Entsprechung in der Realität, bietet aber einige Vorteile. So ist es zum Beispiel relativ einfach möglich verschiedene Verzögerungsmodelle wie zum Beispiel konstante Verzögerung, gleichverteilte Verzögerung oder Poisson-verteilte Verzögerung zu implementieren. Diese Verzögerungen beim Zustellen der Nachrichten müssen in diesem Fall einfach in die entsprechenden Funktionen des *MessageSender* eingebaut werden. Ähnliche Vorteile bieten sich wenn Nachrichten unter bestimmten Umständen (zum

Beispiel wenn sich der Empfänger in einem Funkloch befindet) gar nicht zugestellt werden können. Auch hier lassen sich die Kriterien an einer Stelle kapseln.

4.8.4.4 MessageLogger

Der optionale *MessageLogger* dient zur statistischen Analyse der versandten Nachrichten. Wenn eine solche Analyse gewünscht wird, wird er an den *MessageSender* gekoppelt und protokolliert dann die Größe und den Zeitpunkt des Versendens einer jeden Nachricht. Nach dem Simulationslauf stellt die Klasse dann Statistiken über die versandten Nachrichten zur Verfügung. Dazu gehören die Anzahl der versandten Nachrichten, ggf. die durchschnittliche Verzögerung, die Ausfallrate oder auch den maximalen Kommunikationsfluss (in Kilobyte) in bestimmten Zeitintervallen. Letzteres ermöglicht Aussagen über die benötigte Kommunikationsinfrastruktur oder, andersherum, darüber, ob die vorhandene Kommunikationsinfrastruktur in der Lage ist, das durch das System verursachte Kommunikationsaufkommen zu bewerkstelligen.

4.8.5 Szenarien

Szenarien kapseln alle Informationen, die für das Durchführen einer Simulation notwendig sind. Das sind:

- Der Simulationsgraph,
- die Agenten,
- die notwendigen Daten für die Aufträge,
- die Laufzeit,
- Parameter.

Der Simulationsgraph ist der vollständige Graph mit allen Informationen wie in Abschnitt 4.1.1 beschrieben.

Die Agenten werden im Szenario mit allen notwendigen Parametern (zum Beispiel Geschwindigkeit bei FTF-Agenten) erstellt und mit einem Startzustand versehen. Dieser besteht zum einen aus einer Startposition zum andern aus bereits in der Liste des Agenten vorhandenen Actions (siehe Abschnitt 4.2.2).

Die notwendigen Daten für die Aufträge hängen von der Art des oder der verwendeten OrderDisposerAgent(en) ab (siehe Abschnitt 4.3.2). Es kann sich dabei um eine Liste von Aufträgen, um Wahrscheinlichkeitsverteilungen oder um Zugangsdaten zu einer Datenbank mit Aufträgen handeln.

Die Laufzeit gibt an, welcher Zeitraum simuliert werden soll. Die Angabe erfolgt in der Anzahl der Simulationsschritte. Die simulierte Zeit ergibt sich somit aus der Multiplikation der Laufzeit mit der Größe eines Simulationsschrittes. Beispielsweise bedeutet eine Angabe von 43.200 bei einer Schrittgröße von einer Sekunde, dass ein Zeitraum von 12 Stunden (43.000 / 3.600) simuliert werden soll.

Die Parameter entsprechen einer ausformulierten Instanz der Konfigurationsdatei (siehe Abschnitt 5.8.1).

Zusammenfassend lässt sich sagen, dass alle für die Simulation notwendigen Daten in einem Szenario gekapselt formuliert werden.

Kapitel 5

Implementation

Die Implementation der in Kapitel 4 beschriebenen Lösungsansätze sowie der konzipierten Simulation erfolgte in JAVA. Die Implementierung ist dabei sehr eng am beschriebenen Entwurf, so dass eine genaue Beschreibung jeder implementierten Klasse nicht notwendig erscheint. Stattdessen wird im folgenden nur auf einige wichtige Details der Implementierung eingegangen. Eine Übersicht der erstellten Pakete findet sich in Abbildung 5.1. In den Paketen finden sich jeweils folgende Implementierungen:

strings Strings die zur Kopplung mit der JADE-Simulation verwendet werden (siehe Abschnitt. 5.4.4).

scenarios Abgespeicherte Szenarien, enthalten die notwendigen Graphen.

negotiations Implementierung eines abstrakten Verhandlungskonzepts sowie der idle-Lock Verhandlung.

mlrRelated Klassen für das Einlesen von Daten aus der zentralen Referenz.

gui Implementierung der 2D-Visualisierung.

conflictFreeRouting Implementierung des Free-Time-Window Algorithmus.

tools Hilfsklassen für IO und Justierung.

configuration Parameter für die Steuerung der Simulation.

plans_actions Aktionen und Pläne.

graphen Implementierung des Graphen sowie der Methoden zur Knoten und Kantenauswahl.

messages Implementierung der Kommunikation.

simple_scheduler_and_agents Implementierung des Scheduler und der Agenten.

orders Implementierung der Order und des OrderTrackers.

distributionRV Separate Implementierung der Auftragsvergabe.

locks Implementierung der Reservierungen.

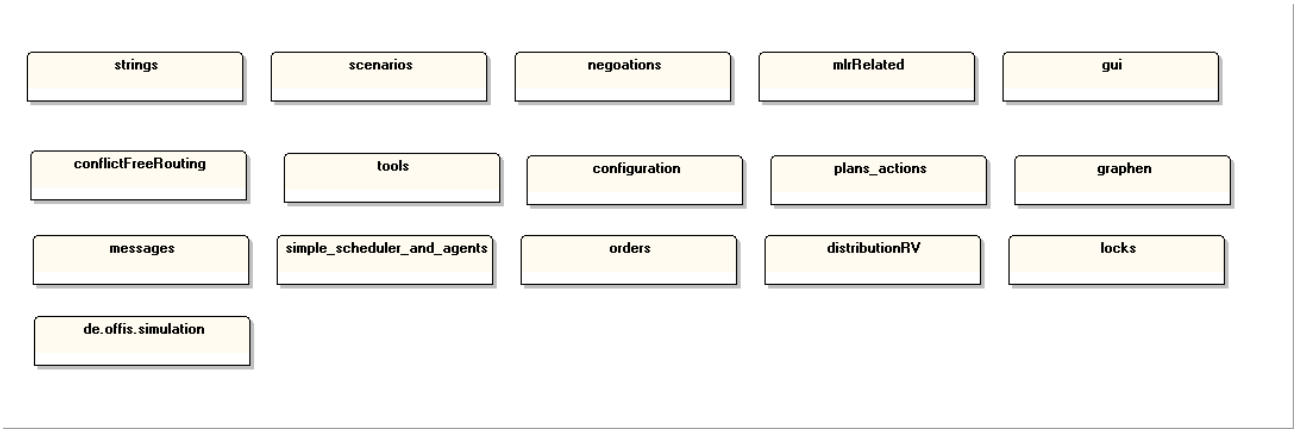


Abbildung 5.1: Paket Übersicht

de.offis.simulation Klassen zur Kopplung mit der 3D-Visualisierung (siehe Abschnitt 5.4.5).

5.1 Graphen und interne Graphen

Die Implementierung der Graphen ist wie in dem Entwurf beschrieben geschehen (siehe Abschnitt 4.1). Der Graph selber besteht aus einer ArrayList aller Knoten sowie einer ArrayList aller Kanten. Die Knoten und Kanten wiederum haben die im Entwurf beschriebenen Eigenschaften. Des Weiteren erhält der Graph einige Funktionen zum Suchen von bestimmten Knoten oder Kanten, zum Suchen und Hinzufügen von Reservierungen sowie zur Ausgabe des Graphen. Reservierungen wurden ebenfalls wie im Entwurf beschrieben implementiert.

Neben dem zentralen Graphen der Simulation können die Agenten noch einen eigenen, internen Graphen haben. Dies kann notwendig sein, wenn die Agenten ein sich unterscheidendes bzw. ein vom wirklichen Ist-Zustand abweichendes Weltbild haben. Sollen zum Beispiel die Auswirkungen von Fehlern oder Limitationen in der Kommunikation untersucht werden, so kann es vorkommen, dass die Agenten nicht alle LockChangeMessage Nachrichten (siehe Abschnitt 4.4.6) mitbekommen und daher ein vom Ist-Zustand abweichendes Bild über den aktuellen Stand der Reservierungen im System haben. Da diese Reservierungen aber direkt an den Knoten und Kanten und somit im Graphen gespeichert werden (siehe Abschnitt 4.1.1.4), muss der Graph, mit dem die Agenten arbeiten vom Simulationsgraphen abweichen. Um dies umzusetzen arbeiten die Agenten in einer solchen Simulation mit internen Graphen. Hier speichern sie dann beispielsweise nur die Reservierungen, die sie mitbekommen. In Experimenten, in denen die Agenten kein vom Ist-Stand abweichendes Weltbild haben können, wird auf die Benutzung von internen Graphen zu Gunsten der Rechenzeit abgesehen.

Um das Modellieren der Layouts zu vereinfachen wurden zwei Tools entwickelt. EnvEdit (Environment Editor) und GraphMaker. Mit Hilfe dieser beiden Tools ist es mit relativ geringem Aufwand möglich, Layouts für die Simulation zu modellieren.

EnvEdit kann dafür genutzt werden, komplexe Umgebungen mittels einer grafischen Oberfläche zu modellieren. Komplexe Umgebungen sind hier Umgebungen, in denen sehr viele Ressourcen durch Agenten gesteuert werden. Das Tool ermöglicht das Modellieren solcher

Umgebungen durch einfaches Drag & Drop von solchen Agenten in den Raum. Auch eine Überlagerung mehrere Agenten und das getrennte oder gleichzeitige Bearbeiten der Agenteneigenschaften ist mit EnvEdit möglich. Das Tool wurde genauer in [SBB+12] beschrieben.

GraphMaker ermöglicht das Modellieren des Graphen per Drag & Drop. Hierfür wird zunächst ein Bild des zu modellierenden Layouts als Hintergrund geladen. Ein solches Bild kann zum Beispiel die Architektenzeichnung einer Warenhalle oder die Darstellung einer anderen Simulation sein. In diesem Bild können dann durch Mausklicks Knoten gesetzt und mit Kanten verbunden werden. Durch die Angabe der Koordinaten sowie Ausmaße des Hintergrundbildes berechnet das Programm automatisch die x,y-Positionen der Knoten sowie die Längen der Kanten. Außerdem ist es möglich sowohl einzelnen Knoten oder Kanten als auch einer beliebig großen Anzahl Knoten oder Kanten gleichzeitig beliebige Annotationen hinzuzufügen. Weitere Features von GraphMaker sind:

- Hervorheben von Knoten und Kanten mit bestimmten Eigenschaften,
- Automatisches Aufteilen von Kanten in mehrere, kürzere Teilkanten,
- Automatisches Erstellen einer Versionshistorie,
- Laden und Speichern des Layouts,
- Automatischer Export in eine Java Datei.

Das Aufteilen von Kanten in kürzere Teilkanten kann erwünscht sein, wenn es sehr lange aber schmale Kanten im Graphen gibt. Wenn eine solche Kante von einem Agenten reserviert wird, kann in der Zeit, in der die Reservierung gültig ist, kein anderer Agent auf der Kante fahren, selbst wenn es eigentlich möglich wäre, dass die beiden Agenten hintereinander in die selbe Richtung fahren. Um dieses Problem zu umgehen, kann die Kante in mehrere, kürzere Kanten aufgeteilt werden. Dies kann mit GraphMaker am Ende der Modellierung automatisch geschehen. Hierzu gibt man eine maximal zulässige Kantenlänge an und alle Kanten, die länger als diese Länge sind, werden in mehrere Kanten aufgeteilt, so dass alle Kanten kürzer als diese Länge sind. Hierfür werden sogenannte Hilfsknoten eingeführt. Aus einer Kante der Form

$$Knoten1 - Kante1 -> Knoten2$$

wird so zum Beispiel

$$Knoten1 - Kante1_1 -> Knoten1 - 2_1 ->$$

$$Kante1_2 -> Knoten1 - 2_2 -> Kante1_3 -> Knoten2$$

Hierbei übernehmen die neuen Kanten sowie alle Hilfsknoten, im Beispiel also $Kante1_1, Kante1_2, Kante1_3, Knoten1 - 2_1$ und $Knoten1 - 2_2$ alle Annotationen der alten Kante, im Beispiel also von Kante $Kante1$. Somit bleiben die relevanten Eigenschaften für die Algorithmen (wie zum Beispiel die Härte des Untergrunds) vollständig erhalten.

Die Oberfläche von GraphMaker ist in Abbildung 5.2 dargestellt.

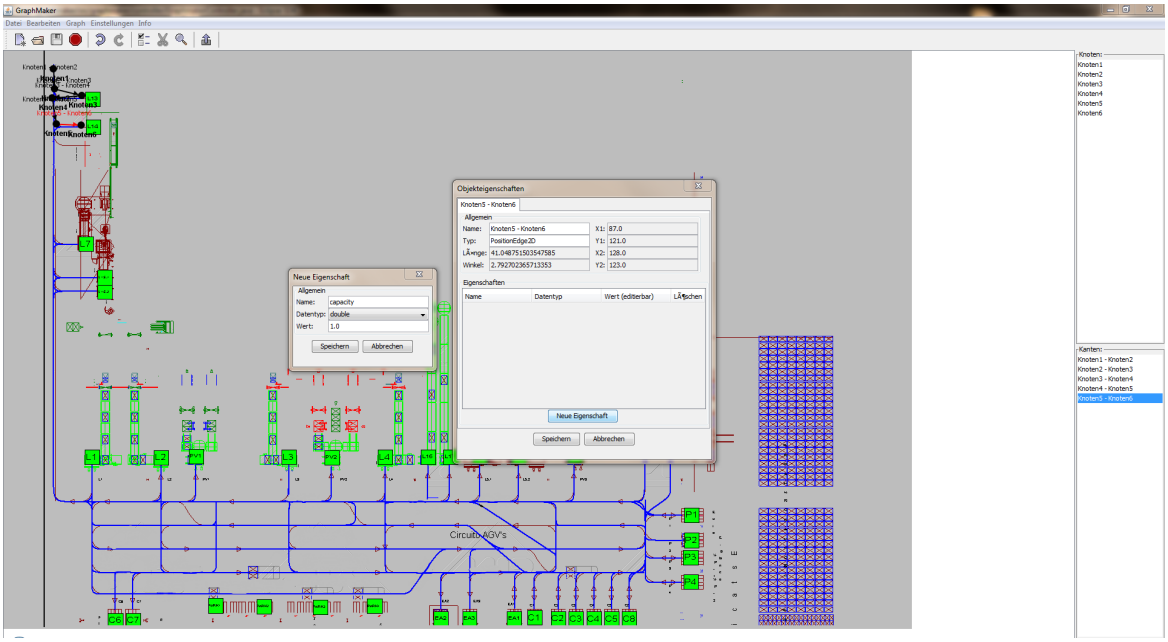


Abbildung 5.2: Oberfläche von GraphMaker

5.2 Aufträge, Aktionen, Pläne

Aufträge wurden als Objekte implementiert, welche die Daten Erstellungszeitpunkt, Start, Ziel, den zugewiesenen Agenten, ggf. die Deadline, die Priorität, den Start- und den Endzeitpunkt der Beladung, der Entladung und des Transports sowie eine eindeutige ID enthalten. Alle erstellten Order werden im OrderTracker gespeichert. Nach einem Simulationsdurchlauf kann dieser die die Aufträge betreffenden Kennzahlen berechnen.

Die in Abschnitt 4.2.2 vorgestellten Aktionen wurden als Unterklassen der abstrakten Klasse Action implementiert. Jede Unterklasse enthält die für die jeweilige Aktion notwendigen Daten. Für die GoTo-Action ist das der Zielknoten, für die Load- und UnLoad-Action der zu ladende bzw. entladende Auftrag sowie die Lade- bzw. Entladedauer, für die RequestRessource-Action und die FreeRessource-Action die jeweilige Ressource und für die Penalty-Action die Wartezeit.

Pläne sind in der Implementierung ArrayListen von Aktionen. Es wird jeweils eine ArrayListe pro vom Agenten geplanten Auftrag gespeichert.

5.3 Scheduler

Die zentrale Komponente der Simulation ist der Scheduler. Dieser ist für die Simulation der Zeit und für die Taktung der Agenten zuständig. Die Simulation springt dabei jeweils von einem konkreten Zeitpunkt zum nächsten (siehe Abschnit 2.3.1.1). In jedem dieser Zeitpunkte ruft der Scheduler dann alle bei ihm eingetragenen Agenten auf. Diese führen dann die Handlungen aus, die sie in dem Zeitraum seit dem letzten Aufruf durchführen können.

Die Agenten sind in verschiedenen Ebenen im Scheduler gespeichert. Diese Ebenen werden

vom Scheduler hintereinander aufgerufen. Ein Agent aus Ebene zwei wird also erst aufgerufen, wenn bereits alle Agenten aus Ebene eins aufgerufen wurden. Alle mit der Steuerung im Zusammenhang stehenden Agenten (BasicFTFAgent, PositionFTFAgent, OrderDisposer) sind in Ebene eins. Die Agenten in Ebene zwei werten die Handlungen dieser Agenten aus (GUI-Agent, JADECouplingAgent, 3DCouplingAgent, ConflictCheckerAgent) und sollen daher erst aufgerufen werden, wenn die Agenten aus Ebene eins bereits alle aufgerufen wurden.

Die Speicherung der Agenten erfolgt in ArrayLists. Die Laufzeit der Simulation wird als Anzahl zu simulierender Schritte sowie als Länge eines Schrittes zum Simulationsstart übergeben. In der Simulation ruft der Scheduler dann in jedem Schritt die Step-Methode von jedem hinterlegten Agenten einmal auf.

Weiterhin sind folgende Informationen beim Scheduler hinterlegt und von anderen Objekten der Simulation abrufbar:

- Die aktuelle Simulationszeit,
- Auflistung aller Agenten,
- Der Simulationsgraph,
- Ggf. Ressourcenverwaltung.

Die aktuelle Simulationszeit wird als die Anzahl der bisher simulierten Zeitpunkte angegeben. Eine Umrechnung in eine Zeitangabe in der Form von Tagen, Stunden, Minuten, Sekunden oder ähnlichem kann aber ebenfalls erfolgen. Bei den in Kapitel 6 durchgeführten Simulationen wird jeweils der Zeitraum einer Sekunde innerhalb eines Zeitschrittes simuliert.

Die Auflistung aller Agenten ist als Liste abrufbar. Weiterhin können Teillisten mittels Reflection abgerufen werden. So ist es zum Beispiel möglich, alle Agenten abzurufen, die das Interface MessageReceiver implementiert haben, also alle Agenten, denen man eine Nachricht schicken kann. Oder es können alle Agenten, die Unterklasse des Agenten BasicFTFAgent sind, abgerufen werden, also alle Agenten, die ein FTF repräsentieren.

Weiterhin kann der globale Simulationsgraph abgerufen werden (siehe Abschnitt 4.1).

Falls es im Szenario Ressourcen gibt, die durch andere Komponenten als Agenten verwaltet werden, wie zum Beispiel Park- oder Batterieladepätze oder Fahrstuhlreservierungen, dann bietet der Scheduler ebenfalls Zugriff auf die Komponenten, die diese Ressourcen verwalten.

5.4 Agenten

Jede Klasse, die das Interface Agent implementiert, wird vom Scheduler als Agent angesehen. Neben einigen Methoden, die Zugriff auf Daten wie einer eindeutigen ID sowie einem (nicht notwendigerweise eindeutigen) Label gewähren, ist die entscheidende, durch das Interface vorgegebene Methode die step Methode. Diese wird zu jedem Simulationszeitpunkt vom Scheduler aufgerufen. Nach dem Aufruf der Methode überprüft der Agent seinen Zustand, den seiner Umgebung sowie sein aktuelles Ziel und handelt dann anhand dieser Begebenheiten.

Die zentralen Agenten, BasicFTF-Agent und OrderDisposer wurden wie in den Abschnitten 4.3.1 und 4.3.2 beschrieben implementiert.

Neben diesen zentralen Agenten für die Vergabe, Verwaltung und Durchführung der Transportaufträge enthält die Simulation noch einige andere Agenten. Diese Agenten haben keine Entsprechung in der Realität und sind in der Regel keine Agenten nach der Definition von Wooldridge (siehe Abschnitt 2.3.1). Die jeweiligen Funktionen wurden aus Gründen der einfachen Realisierbarkeit ebenfalls als Agent implementiert. Die wichtigsten dieser Agenten sind der ConflictChecker Agent, der Ressource Agent, der Gui Agent, der JADECoupling Agent sowie der 3DCoupling Agent. Diese Agenten werden in den folgenden Abschnitten beschrieben.

5.4.1 ConflictCheckerAgent

Der ConflictCheckerAgent ist für das Erkennen von Konflikten zuständig. Das Konzept hinter der und die Implementierung der Konflikterkennung ist im Detail in Abschnitt 4.5 beschrieben.

5.4.2 RessourceAgent

Der Ressource Agent modelliert Ressourcen in der Umgebung die über eine eigene Steuerung verfügen. Dies können zum Beispiel Rampen, Fahrstühle, Brandschutztore oder auch Quellen und Senken die die Anfahrt der Fahrzeuge mit kontrollieren bzw. steuern sein. Diese Agenten sind zuständig für die Vergabe und Verwaltung von Rechten zur Nutzung der durch sie kontrollierten Ressourcen. Die genauen Entscheidungskriterien der Agent hängen stark vom jeweiligen Szenario ab und müssen daher für jedes Szenario per Hand programmiert werden.

5.4.3 GuiAgent

Der GUI Agent übernimmt eine einfache Visualisierung der Simulation. Hierfür werden dem Agenten ein Bild des Layouts sowie Bilder der dynamischen Objekte (FTF und Aufträge) übergeben. Bei jedem Aufruf der step Methode zeichnet der Agent dann den aktuellen Zustand der Simulation. Hierfür liest er die Positionen der Agenten aus der Agenten Liste des Shedulers (siehe 5.3) und die Zustände der Aufträge aus dem OrderTracker (siehe 4.8.3). Die Visualisierung durch den GUI Agenten ist in Abbildung 5.3 dargestellt.

Die Visualisierung durch den GUI-Agenten ist sozusagen eine Übersicht über die Geschehnisse in der Simulation. Sie hilft außerdem dabei bestimmte Vorgänge zu erkennen.

5.4.4 JADECoupling Agent

Die in diesem Kapitel vorgestellte Simulation kann mit einer JADE Simulation gekoppelt werden, die im Rahmen des Forschungsprojekts „Dezentrale, agentenbasierte Selbststeuerung von Fahrerlosen Transportsystemen (FTS)“, in dem große Teile dieser Arbeit entstanden sind, vom IPH[1] entwickelt wurde. Im gekoppelten Zustand übernimmt die hier vorgestellte Simulation die Simulation der Umgebung und des Fahrverhaltens, die JADE Simulation die Berechnung der Agenten-Logik. Wenn die Simulation im gekoppelten Zustand läuft, simuliert

[1] Institut für Integrierte Produktion Hannover - http://www.iph-hannover.de/

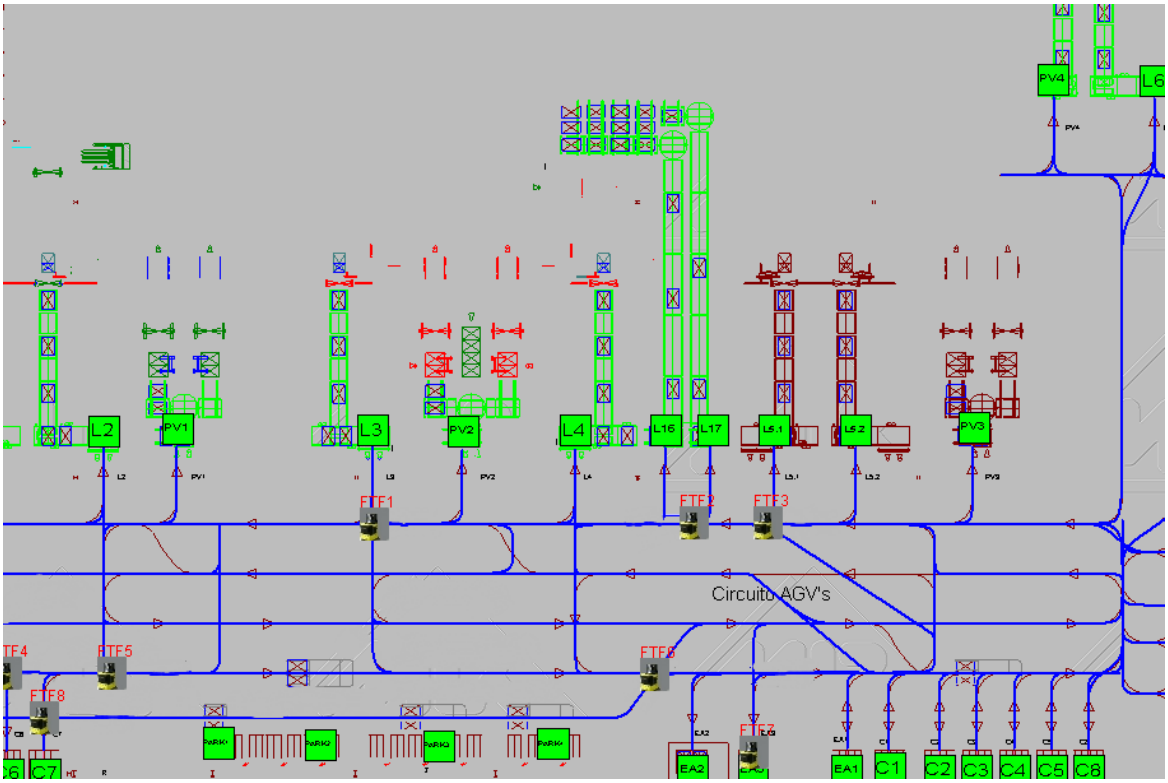


Abbildung 5.3: Visualisierung durch den GUI-Agenten

die Simulation zunächst einen Zeitschritt. Dann schickt die Simulation den aktuellen Ist-Stand der Simulation an die JADE Simulation. Diese verarbeitet diese Informationen und schickt dann Befehle an die Simulation. Der Vorgang ist in Abbildung 5.4 dargestellt.

Die Kopplung wurde implementiert um zu evaluieren, ob die in dieser Arbeit vorgestellten Stuerungskonzepte auch auf realen Agenten umsetzbar sind, ob diese also die notwendige Rechenleistung aufbringen können und ob die notwendige Kommunikation innerhalb eines WLAN-Netzes durchführbar ist. Der genaue Aufbau der Kopplung und die Auswirkungen einer solchen Kopplung auf die Rechenzeit, die für einen Simulationsdurchlauf benötigt wird, werden in [SSS+13] ausführlich beschrieben.

Wenn die Simulation mit der JADE Simulation gekoppelt wird, ist der JADECouplingAgent für die Kommunikation mit der JADE Simulation zuständig. Alle anderen, für die JADE Simulation relevanten Agenten teilen diesem wesentliche Änderungen ihres Zustands mit. Dies sind zum Beispiel:

- Erreichen eines Knotens,
- Erreichen des Ziels,
- Start des Ladevorgangs,
- Ende des Ladevorgangs,
- Start des Entladevorgangs,
- Ende des Entladevorgangs.

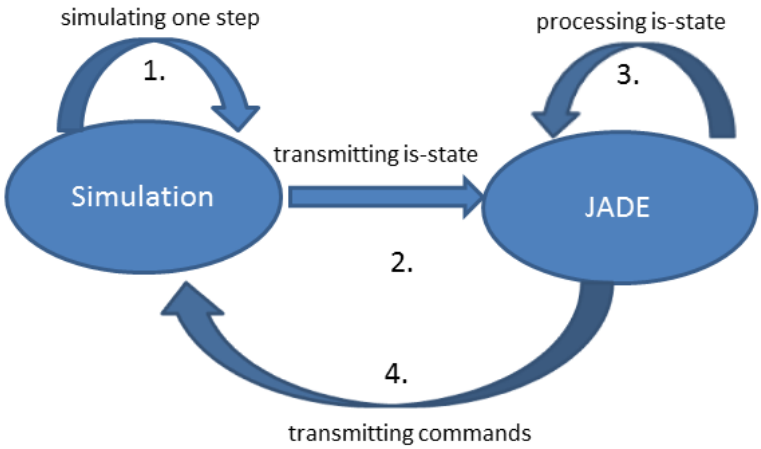


Abbildung 5.4: Kopplung der Simulation mit einer JADE Simulation

Der JADECouplingAgent schickt diese dann per Socket an einen Kopplungsagenten auf JADE Seite. Umgekehrt schickt der Kopplungsagent aus der JADE Simulation Befehle an den JADECouplingAgent.

5.4.5 3DCouplingAgent

Im Rahmen des Projekts CogniLog[2], in dem die entwickelte Simulation ebenfalls zum Einsatz kam (siehe Abschnitt 6.3.1, wurde eine 3D-Visualisierung implementiert. Diese stellt die Vorgänge in der Simulation in 3D da (siehe Abbildung 5.5). Für die Kopplung mit der Visualisierung ist der 3DCouplingAgent zuständig. Dieser sammelt beim Aufruf seiner step Methode alle notwendigen Informationen (Position der Agenten und der Aufträge) ein und schickt diese dann an die Visualisierung. Die Aufbereitung der Daten erfolgt hier durch das Auslesen aller relevanten Positionen aus der Agenten-Liste des Schedulers sowie aus der Order-Liste des OrderTrackers. Die Kommunikation zwischen der Simulation und der 3D-Visualisierung erfolgt über die objektorientierte Remote Procedure Middleware ICE[3].

5.5 Nachrichten

Die verschiedenen Nachrichten wurden, wie in Abschnitt 4.8.4 beschrieben, als Unterklasse der Klasse Message implementiert. Jedes Nachrichtenobjekt enthält einen Verweis auf den Sender der Nachricht. Des Weiteren enthält jede Nachricht die notwendigen Informationen. Für die Nachrichtentypen OrderAnnouncement, ProposalAccepted, ProposalRejected und ReturnOrder ist das der entsprechende Auftrag. Für den Nachrichtentyp Proposal ist das der entsprechende Auftrag und das Angebot. Für den Nachrichtentyp LockChange ist das eine Reservierung, für die LockNegoation eine Reservierung, ein Typ und ein Wert und für die LockNegoationAnswer ein Lock und der Wert true oder false.

Das Interface MessageReceiver sowie die Klassen MessageSender und MessageLogger wurden wie im Entwurf beschrieben implementiert (siehe Abschnitt 4.8.4).

[2] www.cognilog.de
[3] Internet Communications Engine - http://www.zeroc.com/

Abbildung 5.5: 3D Visualisierung der Simulation

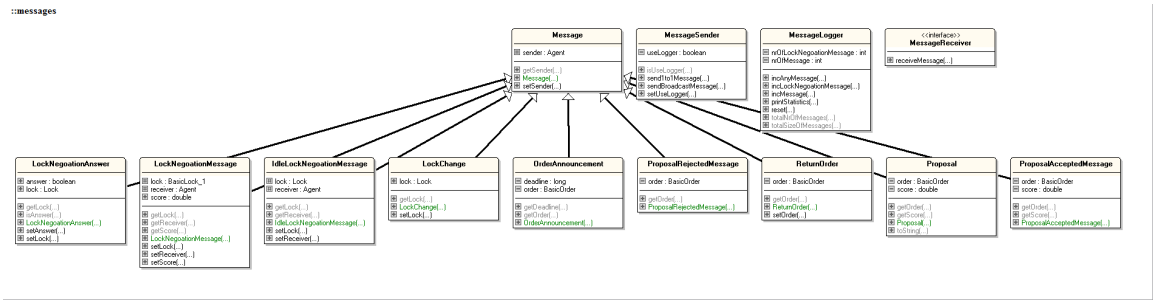


Abbildung 5.6: Darstellung des Packages *messages*

Der Aufbau des dafür zuständigen Paketes ist in Abbildung 5.6 dargestellt.

5.6 Auftragsvergabe

Die Auftragsvergabe ist in den BasicFTF und den OrderDisposer Agenten integriert worden. Die OrderDisposer Agenten erhalten zu Beginn der Simulation eine Liste von auszuschreibenden Aufträgen sowie den jeweiligen Ausschreibungszeitpunkten. Wenn ein Auftrag zurückgegeben wird, wird dieser der Liste hinzugefügt, wobei er als neuen Ausschreibungszeitpunkt den aktuellen Zeitpunkt zuzüglich eines einstellbaren Zuschlages erhält. In jedem Simulationszeitpunkt, in dem ein Auftrag ausgeschrieben wird, schickt er eine OrderAnnouncement Nachricht an die BasicFTF Agenten. Diese führen das eingestellte Verfahren zur Auftragsvergabe durch (siehe Abschnitt 4.6). Während der Berechnung eines Angebots wird der dabei erstellte Plan vom Agenten als temporär markiert gespeichert. Anschließend schickt der BasicFTF Agent das Angebot an den OrderDisposer zurück. Dieser wählt das beste An-

gebot aus und schickt entsprechend ProposalRejected Nachrichten an die Verlierer und eine ProposalAccepted Nachricht an den Gewinner der Auktion. Die Verlierer löschen daraufhin den als temporär markierten Plan, den sie für dieses Angebot gemacht haben. Der Gewinner entfernt die temporär-Markierung für den entsprechenden Plan und fügt diesen Plan seinem Gesamtplan hinzu.

5.7 Verhandlungen

Zur Implementierung von Verhandlungen wurde ein Verhandlung-Interface verwendet. Das Interface verlangt von einer Verhandlung die Implementierung der folgenden drei Methoden:

- checkCriterias
- negotiate
- reserveNewPlan

Der Aufruf der Methode checkCriterias überprüft ob die Kriterien für die entsprechende Verhandlung erfüllt sind, im Fall der Idle-lock Verhandlung als Beispiel also, ob alle Reservierungen, die die optimale Route blockieren, idle-Locks sind (siehe Abschnitt 4.7.3.2). Mit der Methode negotiate wird die entsprechende Verhandlung durchgeführt und mit reserveNewPlan wird der neue Plan reserviert, sofern die Verhandlung erfolgreich war. Durch die Verwendung eines Interfaces ist die nachträgliche Implementierung von weiteren Verhandlungen möglich.

Neben dem oben beschriebenen Interface wurde die Methode negotiate im BasicFTFAgent implementiert. Diese Methode wird im Rahmen der Angebotsberechnung nach dem Erstellen eines initialen Plans aufgerufen. Innerhalb der Methode werden alle in der aktuellen Simulation eingesetzten Verhandlungen abgerufen und nacheinander durchgeführt. Zur Durchführung werden jeweils die drei im oben beschriebenen Interface beschriebenen Methoden checkCriterias, negotiate (falls checkCriterias true zurück geliefert hat) sowie reserveNewPlan (falls negotiate eine neue Route zurück geliefert hat) aufgerufen.

Die Auswahl, welche Verhandlungen prinzipiell in der Simulation durchgeführt werden sollen, geschieht vor dem Start der Simulation durch das Hinzufügen von Klassen, die das Verhandlungs-Interface implementieren zu einer ArrayList.

5.8 Sonstiges

Neben den für die Umsetzung der beschriebenen Steuerung notwendigen Teile der Simulation enthält die Simulation noch Klassen und Methoden zur Einstellung von Parametern und zur Auswertung.

5.8.1 Konfiguration

Die Konfiguration der Simulation geschieht zu einem großen Teil mit Hilfe von Parametern. Diese Parameter sind in einer Konfigurationsdatei gesammelt. Enthalten sind unter anderem Parameter für:

- Die Lade und -Entladezeit,
- die Geschwindigkeitsprofile der Fahrzeuge,
- Optionen zur Visualisierung,
- Angaben, ob die Simulation mit der 3D-Visualisierung oder der JADE-Simulation gekoppelt werden soll und wenn ja, mit welchen Verbindungsdaten,
- Welche Algorithmen zur Routenfindung, zur Auftragsvergabe und zur Kooperation verwendet werden sollen,
- Die Parameter für die zu verwendeten Algorithmen.
- Welche Ausgaben in welcher Form gespeichert werden sollen,
- Welche Ausgaben die Simulation zur Konsole ausgeben soll-

Eine Instanz dieser Konfigurationsklasse muss für jede Simulation vorliegen. Sie ist Teil des zu simulierenden Szenarios (siehe Abschnitt 4.8.5).

5.8.2 Auswertung

Die Auswertung berechnet nach dem Durchlauf der Simulation die zu untersuchenden Kennzahlen. Hierfür fragt sie nach dem Ende der Simulation die Rohdaten über die Aufträge vom OrderTracker (siehe Abschnitt 4.8.3) sowie Rohdaten von den einzelnen Agenten ab. Die Rohdaten, die von den FTF Agenten abgefragt werden können enthalten zum Beispiel die Anzahl der Zeitschritte, die der Agent idle war, die Anzahl der Zeitschritte, die der Agent beladen gefahren ist, die Anzahl der Zeitschritte, die der Agent unbeladen gefahren ist, die gefahren Strecke mit Last, die gefahren Strecke ohne Last usw.. Aus diesen Rohdaten bestimmt die Auswertung Kennzahlen wie:

- Die von allen Agenten gefahrene Strecke,
- Die durchschnittliche Auftragsdauer,
- Den Transportdurchsatz,
- Die durchschnittliche idle Zeit der Fahrzeuge,
- Die maximale Zeit bis zum Auftragsstart.

Welche Kennzahlen genau ermittelt und gespeichert werden sollen, kann in der Auswertung eingestellt werden. Weiterhin kann eine Funktion hinterlegt werden, die aus den ermittelten Kennzahlen einen einzelnen Wert berechnet. Dieser Wert kann für die automatische Bewertung eines Simulationsergebnisses genutzt werden.

Kapitel 6

Evaluation

In Kapitel 1 und 3 wurden drei Fragen herausgearbeitet, die mit der vorliegenden Arbeit beantwortet werden sollten:

1. Wie schneidet eine dezentrale Steuerung in Bezug auf die benötigte Anzahl an Fahrzeugen, die durchschnittliche Auftragsdauer, die zurückgelegte Strecke und den Leerfahrtanteil im Vergleich zu einer zentralen Steuerung ab?
2. Kann eine dezentrale Steuerung flexibler auf sich ändernde Anforderungen an das Systemverhalten reagieren als eine zentrale Steuerung?
3. Kann eine dezentrale Steuerung einfacher auf eine Erweiterung bzw. Veränderung des Szenarios angepasst werden als eine zentrale Steuerung?

In diesem Kapitel wird nun untersucht, zu welchen Antworten auf diese drei Fragen das beschriebene (siehe Kapitel 4) und implementierte (siehe Kapitel 5) Konzept führt.

Der Aufbau des Kapitels orientiert sich an den drei Fragen. In Abschnitt 6.1 wird die erste Frage betrachtet. Hierfür wird die entwickelte Steuerung mit einer zentralen Steuerung, die von einem Industriepartner zur Verfügung gestellt wurde, verglichen. Als Basis für die durchgeführten Experimente dient das Szenario einer Getränkeabfüllanlage. In Abschnitt 6.2 wird die zweite Frage betrachtet. Hierfür dient erneut das Szenario der Getränkeabfüllanlage als Basis. Für die Experimente wird die Auftragseinlastung auf Aufträge mit verschiedenen Prioritäten und Deadlines erweitert. Anschließend wird mit verschiedenen Parameter-Konfigurationen untersucht, in wie weit die Auftragsvergabe zu verschiedenem Systemverhalten führen kann und ob das System sich somit flexibel an die gerade vorliegende Situation bzw. an die sich ändernden Anforderungen der Betreiber anpassen lässt. In Abschnitt 6.3 wird die dritte Frage untersucht. Für diese Untersuchung diente eine Warenumschlaghalle, die mit Gabelstaplern betrieben wurde, als Basis.

6.1 Performanz

Im Rahmen des über die AIF (Allianz Industrie Forschung[1]) im Rahmen des Programms zur Förderung der Industriellen Gemeinschaftsforschung und -entwicklung (IGF) vom Bun-

[1] www.aif.de

desministerium für Wirtschaft und Technologie aufgrund eines Beschlusses des Deutschen Bundestages geförderten Vorhabens 17237 N: Dezentrale, agentenbasierte Selbststeuerung von Fahrerlosen Transportsystemen (FTS) wurden reale Daten über ein Fahrerloses Transportsystem zur Verfügung gestellt. Dieses wurde zentral mit einem etablierten (aber nicht veröffentlichten) Verfahren gesteuert. Für dieses Szenario lagen folgende Daten vor:

- Das Layout der Anlage,
- Die FTF Daten (Geschwindigkeit, Lade- und Entladezeiten),
- Die Auftragseinlastung,
- Die Kennzahlen der Anlage mit der zentralen Steuerung.

Das Layout der Anlage wurde bereits in Abbildung 3.1 dargestellt. Die wichtigen Stationen (Quellen, Senken und Parplätze) sind durch grüne Boxen dargestellt. Die Wege, auf denen die FTF fahren können, durch blaue Linien. Die Wege haben jeweils eine Kapazität von eins. Zwei Fahrzeuge können sich also nicht auf einem Weg entgegenkommen. Auch kann kein Fahrzeug ein anderes auf einem Weg überholen. Die Anlage ist eine Getränkeabfüllanlage. Mehrere Stationen produzieren Getränke, die als Einzelpaletten von FTF abgeholt werden. Die Außenmaße der Halle sind ca. 80 Meter * 125 Meter. Der Simulationsgraph für dieses Layout (siehe Abschnitt 4.1.1.1) besteht aus 189 Knoten, von denen 32 Knoten Stationen und 4 Knoten Parkplätze sind, sowie aus 472 Kanten. Die übrigen Knoten sind Kreuzungen.

Im System wird nur ein Typ von FTF eingesetzt. Die FTF fahren mit einer Geschwindigkeit von bis zu 1.30 Metern pro Sekunde. Das Laden- und Entladen dauert jeweils 61 Sekunden. Die Batterien der Fahrzeuge sind so ausgelegt, dass sie eine komplette Schicht lang ohne Nachladen operieren können. Ein Nachladen findet daher nicht statt.

Im System werden durchschnittlich 52 Aufträge pro Stunde durchgeführt. Diese Aufträge stehen in Form einer Liste mit den Angaben Zeit, Start und Ziel zur Verfügung. Im System haben die Aufträge weder eine Deadline noch eine Priorität noch liegen die aktuellen Zustände der Zwischenlager an den Stationen vor. Diese Werte konnten daher bei diesem Experiment nicht berücksichtigt werden.

Folgende Kennzahlen liegen zum FTS vor:

- Durchschnittliche Auftragsdauer,
- Fahrstrecke der FTF,
- Leefahrtanteil der FTF in Prozent,
- Durchschnittliche Dauer bis zum Transportstart,
- Maximale Dauer bis zum Transportstart.

Die Dauer bis zum Transportstart (sowohl die durchschnittliche wie auch die maximale) gibt jeweils den Zeitraum an, der zwischen der Ausschreibung und der Abholung des Auftrags vergeht.

Das System wurde mit 8 Fahrzeugen betrieben. Ziel der Experimente war es hier, herauszufinden ob eine dezentrale Steuerung der Fahrzeuge mit der selben oder sogar einer niedrigeren Anzahl an Fahrzeugen auskommt.

Das Szenario wurde mit den in Tabelle 6.1 angegebenen Konfigurationen dezentraler Steuerung getestet. Konfiguration 1 wurde hierbei nur zum Vergleich ebenfalls simuliert. Das Auktionsverfahren mit Rückgabe und Einordnung wurde mit den Parametern $\alpha = 1,0$ (Kehrwert der Anfahrtszeit), $\beta = 0.5$ (Rückgabe) und $\gamma = 0.5$ (Reduzierung der Leerfahrt) durchgeführt (siehe Abschnitt 4.6.2.3).

Es wurde eine Schicht der Länge 8 Stunden simuliert. Die zentrale Referenz lieferte folgende Kennzahlen:

Erledigte Aufträge: 416.0

Durchschnittliche Auftragsdauer: 371,1 Sekunden

Gesamtstrecke aller FTF: 72.000 Meter

Leerfahrtanteil: 56,77 %

Durchschnittliche Dauer bis zum Transportstart: 72,3 Sekunden

Maximale Dauer bis zum Transportstart: 242, 0 Sekunden

Innerhalb der 8 Stunden wurden insgesamt 416 Aufträge in das System eingelastet. Die zentrale Referenz erledigt also alle eingelasteten Aufträge erfolgreich. Die erzielten Ergebnisse mit den in Tabelle 6.1 aufgeführten Konfigurationen sind in den Tabellen 6.2 und 6.3 dargestellt.

Konfigration	Anzahl eingesetzter Fahrzeuge	Verwendetes Routingverfahren	Verwendetes Auftrags-vergabeverfahren
1	8	Nicht konfliktvermeidend	Einfaches Auktionsverfahren
2	8	Konfliktvermeidend ohne Verhandlungen	Einfaches Auktionsverfahren
3	7	Konfliktvermeidend ohne Verhandlungen	Einfaches Auktionsverfahren
4	6	Konfliktvermeidend ohne Verhandlungen	Einfaches Auktionsverfahren
5	5	Konfliktvermeidend ohne Verhandlungen	Einfaches Auktionsverfahren
6	8	Konfliktvermeidend mit idle lock Verhandlungen	Einfaches Auktionsverfahren
7	7	Konfliktvermeidend mit idle lock Verhandlungen	Einfaches Auktionsverfahren
8	6	Konfliktvermeidend mit idle lock Verhandlungen	Einfaches Auktionsverfahren
9	5	Konfliktvermeidend mit idle lock Verhandlungen	Einfaches Auktionsverfahren
10	8	Konfliktvermeidend ohne Verhandlungen	Auktionsverfahren mit Rückgabe und Einordnung
11	7	Konfliktvermeidend ohne Verhandlungen	Auktionsverfahren mit Rückgabe und Einordnung
12	6	Konfliktvermeidend ohne Verhandlungen	Auktionsverfahren mit Rückgabe und Einordnung
13	5	Konfliktvermeidend ohne Verhandlungen	Auktionsverfahren mit Rückgabe und Einordnung
14	8	Konfliktvermeidend mit idle lock Verhandlungen	Auktionsverfahren mit Rückgabe und Einordnung
15	7	Konfliktvermeidend mit idle lock Verhandlungen	Auktionsverfahren mit Rückgabe und Einordnung
16	6	Konfliktvermeidend mit idle lock Verhandlungen	Auktionsverfahren mit Rückgabe und Einordnung
17	5	Konfliktvermeidend mit idle lock Verhandlungen	Auktionsverfahren mit Rückgabe und Einordnung

Tabelle 6.1: Performanz Experiment: Konfigurationen

Nr.	1	2	3	4	5	6	7	8	9
Erledigte Aufträge	416,0	416,0	416,0	416,0	404,8	416,0	416,0	416,0	408,2
Durchschnittliche Auftragsdauer	521,3 s	278,1 s	287,6 s	300,3 s	379,2 s	273,5 s	281,1 s	293,2 s	369,5 s
Gesamtstrecke aller FTF	71.936 m	66.112 m	63.688 m	63.552 m	57.888 m	62.352 m	61.352 m	60.722 m	57.622 m
Leerfahrtanteil	57,2 %	52,2 %	52,0 %	51,9 %	51,8 %	52,1 %	51,9 %	51,8 %	51,7 %
Durchschnittliche Dauer bis zum Transportstart	81,5 s	57,7 s	70,2 s	83,8 s	172,4 s	54,1 s	70,1 s	81,0 s	174,2 s
Maximale Dauer bis zum Transportstart	245,3 s	237,7 s	240,4 s	250,7 s	439,7 s	241,6 s	240,5 s	249,7 s	427,4 s

Tabelle 6.2: Performanz Experiment: Ergebnisse 1

Nr.	10	11	12	13	14	15	16	17
Erledigte Aufträge	416,0	416,0	416,0	404,2	416,0	416,0	416,0	407,6
Durchschnittliche Auftragsdauer	277,1 s	286,2 s	299,1 s	377,1 s	273,5 s	280,9 s	292,9 s	369,3 s
Gesamtstrecke aller FTF	66.013 m	63.222 m	63.177 m	56.922 m	62.284 m	61.172 m	60.324 m	57.288 m
Leerfahrtanteil	51,9 %	51,9 %	51,7 %	51,7 %	51,9 %	51,6 %	51,5 %	51,5 %
Durchschnittliche Dauer bis zum Transportstart	57,9 s	70,5 s	84,2 s	174,2 s	54,1 s	70,0 s	80,5 s	173,7 s
Maximale Dauer bis zum Transportstart	238,1 s	255,4 s	266,1 s	440,1 s	238,7 s	239,1 s	249,4 s	424,8 s

Tabelle 6.3: Performanz Experiment: Ergebnisse 2

Die Konfigurationen mit nur 5 Fahrzeugen (5,9,13 und 17)sind nicht in der Lage gewesen, alle 416 in der Schicht anfallenden Aufträge abzuarbeiten. Sie sind hier nur der Vollständigkeit halber aufgeführt.

Die Konfigurationen 2,3 und 4, die konfliktvermeidendes Routing und das Einfache Auktionsverfahren einsetzten erledigen alle 416 eingelasteten Aufträge. Die durchschnittliche Auftragsdauer, die durchschnittliche Dauer bis zum Transportstart sowie der Leerfahrtanteil sind dabei jeweils besser als die zentrale Referenz. Bei der Konfiguration mit 8 Fahrzeugen (Konfiguration 2) sinkt die durchschnittliche Auftragsdauer um ca. 25%, die durchschnittliche Dauer bis zum Transportstart um ca. 20%, der Leerfahrtanteil um ca. 8% und die insgesamt zurückgelegte Strecke um ca. 8%.

Wird die Fahrzeuganzahl reduziert (Konfigurationen 3 und 4) steigen die durchschnittliche Auftragsdauer (um 3,4% bzw. 8,0%) sowie die durchschnittliche Dauert bis zum Transportstart (um 21,7% bzw. um 45%) sowie die maximale Dauer bis zum Transportstart (um 1,1% bzw. um 5,5%). Der Grund dafür ist, dass die Agenten häufiger bereits einen Auftrag fahren, wenn sie einen neuen Auftrag zugewiesen bekommen habe. Das erhöht Wartezeit und Dauer für diesen Auftrag. Die Gesamtfahrstrecke sinkt hingegen durch die Reduzierung (um 3,7% bzw. um 3,9%). Dies liegt zum einen daran, dass die Agenten häufiger bereits einen Auftrag fahren, wenn ihnen ein neuer Auftrag zugewiesen wird. Dadurch fahren die Fahrzeuge seltener einen Parkauftrag. Außerdem ist die Halle weniger dicht befahren, je weniger Fahrzeuge in ihr operieren. Das hat zur Folge, dass durch das konfliktvermeidende Routing weniger Umwege, um die Reservierungen anderer Fahrzeuge zu respektieren, eingeplant werden müssen.

Der Wechsel vom einfachen konfliktvermeidenden Routing zum Routing mit idle-lock Verhandlungen (Konfigurationen 6,7,8 und 9 führt zu einer weiteren Verbesserung der Kennzahlen. Bei 8 Fahrzeugen (Konfigurationen 2 und 6) sinkt die durchschnittliche Auftragsdauer um 1,8%, die durchschnittliche Dauert bis zum Transportstart um 6,3%, und die Gesamtstrecke um 5,7%. Diese Änderungen lassen sich durch die nicht mehr notwendigen Fahrten zu den Parkplätzen erklären. Die Einsparung in der Gesamtstrecke als Folge der eingesparten Parkfahrten erschließt sich hier sofort. Die Einsparungen in der durchschnittlichen Auftragsdauer und der durchschnittlichen Wartezeit hängt in diesem Fall stark am Layout des Szenarios. In der Halle sind alle vier Parkplätze am südlichen Rand positioniert (das dritte, vierte, fünfte und sechste grüne Kästchen von links unten). Die Aufträge hingegen starten fast alle in den nördlichen Teilen der Halle. Die Anfahrt zu einem Auftrag ist also in der Regel höher, wenn das FTF von einem Parkplatz aus starten muss, als wenn es von einer Position nahe des Endes seines letzten Auftrags aus starten kann. Die Reduzierung auf 7, 6 und 5 Fahrzeuge verändert die Kennzahlen auf ähnliche Weise wie das beim konfliktvermeidenden Routing ohne Verhandlungen der Fall ist (Konfigurationen 2-4).

Die Konfigurationen mit konfliktvermeidendem Routing und dem Auktionsverfahren mit Rückgabe und Einordnung (Konfigurationen 10-13) liefern nahezu identische Kennzahlen wie die Konfigurationen mit konfliktvermeidenden Routing ohne Verhandlungen und dem einfachen Auktionsverfahren (Konfigurationen 2-4). Der Grund dafür dürfte in diesem Fall an der Auftragseinlastung liegen. Tabelle 6.4 zeigt stellt die ersten 26 Aufträge da, die in das System eingelastet werden.

Der Zeitpunkt ist hier jeweils als Sekunden nach Simulationsstart zu verstehen. Start und Ziel gibt jeweils eine Station im Layout und damit einen Knoten im Graphen der Simulation an.

Auftrag ID	Zeitpunkt	Start	Ziel
0	40	L2	EA3
1	83	L6	EA3
2	126	L6	EA1
3	169	L3	EA1
4	212	L1	EA3
5	255	L4	EA2
6	298	C1	L16
7	341	L3	EA1
8	384	L5.2	EA2
9	426	L3	C7
10	471	L5.1	EA3
11	514	L6	EA3
12	557	L6	EA3
13	600	C3	L15.2
14	643	L4	EA3
15	686	L4	EA2
16	729	L1	EA1
17	772	P1	PV4
18	816	L4	EA1
19	859	L6	EA3
20	902	C4	L15.2
21	945	L5.2	EA3
22	988	L2	EA1
23	1074	C2	L15.2
24	1161	L6	EA3
25	1333	C3	L16

Tabelle 6.4: Performanz Experiment: Auszug aus der Auftragseinlastung

Betrachten wir nun die zeitlichen Unterschiede zwischen den Aufträgen 0 und 8 (=344 Sekunden) 1 und 9 (=343 Sekunden) 2 und 10 (=345 Sekunden) und im allgemeinen zwischen zwei Aufträgen deren ID sich um 8 unterscheiden und betrachten die durchschnittliche Auftragsdauer in Konfiguration 2 (278,1 Sekunden) so stellen wir fest das die Zeitspanne, die zwischen zwei Aufträgen, deren ID sich um 8 oder mehr unterscheiden, größer als die durchschnittliche Auftragsdauer ist. Wenn wir nun davon ausgehen, dass in der Regel ein Fahrzeug, dass aktuell keinen Auftrag zugeteilt hat, ein besseres Angebot abgeben kann, als ein Fahrzeug, das gerade einen Auftrag fährt, so werden die ersten 8 Aufträge in der Regel auf die 8 zur Verfügung stehenden Fahrzeuge verteilt. Das hat zur Folge, dass das Fahrzeug, was den ersten Auftrag (ID =0) zugewiesen bekommen hat, in der Regel diesen schon beendet hat, wenn der neunte Auftrag (ID=8) ausgeschrieben wird. Nach den Zahlen wird in der Regel sogar schon ein zweites Fahrzeug seinen ersten Auftrag beendet haben, wenn der neunte Auftrag ausgeschrieben wird. Das hat zur Folge, dass in dem betrachteten Szenario bei der gegebenen Auftragseinlastung nur sehr selten ein Auftrag an ein Fahrzeug vergeben wird, dass aktuell schon einen Auftrag zugewiesen bekommen hat. Damit können die theoretischen Vorteile der möglichen Einordnung eines neuen und Rückgabe eines alten Auftrags in dem hier vorgestellten Szenario nicht zum Tragen kommen.

Die gleiche Beobachtung gilt auch für die Konfigurationen mit idle-lock Verhandlungen und dem Auktionsverfahren mit Rückgabe und Einordnung (Konfigurationen 14-17). Die Kennzahlen dieser Konfigurationen unterscheiden sich nur marginal von denen mit idle-lock Verhandlungen und Vergabe per einfachem Auktionsverfahren (Konfigurationen 6-9).

6.2 Flexibilität

Für die Untersuchung, in wie weit die konzipierte und implementierte dezentrale Steuerung flexibel auf verschiedene Auftragseinlastungen und sich ändernde Zielvorgaben reagieren kann, wurde das Szenario der Getränkeabfüllanlage aus dem vorherigen Abschnitt als Grundlage verwendet. In diesem Szenario wurden mehrere Simulationsläufe mit verschiedenen Auftragseinlastungen und verschiedenen Parametersets für die dezentrale Auftragsvergabe durchgeführt und die ermittelten Resultate untersucht.

Im Abschnitt 6.2.1 werden verschiedenen Auftragseinlastungen beziehungsweise deren Erzeugung sowie die verschiedenen verwendeten Parametersets beschrieben. Im Abschnitt 6.2.2 werden die Ergebnisse der Experimente beschrieben. Im Abschnitt 6.2.3 wird ein kurzes Fazit zu den Untersuchungen zur Flexibilität gezogen.

6.2.1 Auftragseinlastung

Für Aussagen zu den Wirkungen der entwickelten Auftragsvergabe ist es notwendig, dass die Aufträge über Deadlines verfügen und die Stationen über Ausgangspuffer. Da diese Informationen in der von den Industriepartnern zur Verfügung gestellten Simulation nicht vorlagen, mussten für die Experimente entsprechenden Aufträge erzeugt werden. Hierfür wurde ein Zufallsgenerator implementiert, der Aufträge nach bestimmten Vorgaben erzeugt. Für die Erzeugung eines Auftrags sind hier die folgenden Größen erforderlich:

- Start- und Zielknoten,
- Ausschreibungszeitpunkt,
- Deadline.

Für die Auswahl von Start- und Zielknoten wurde die Verteilung der zur Verfügung gestellten Simulation aus dem vorherigen Abschnitt verwendet. Der Ausschreibungszeitpunkt wurde gleichverteilt im Intervall [15+letzter Ausschreibungszeitpunkt, 25+letzter Ausschreibungszeitpunkt] gewählt. Für die Deadline wurden zwei Varianten verwendet:

- Weiche Deadlines,
- Harte Deadlines.

Zur Bestimmung der Deadlines wurden im Vorfeld die kürzesten Routen zwischen jedem vorkommenden Start- / Zielknotenpaar und daraus jeweils die kürzeste Fahrtzeit bestimmt (im leeren Graph). Aus der kürzesten Fahrzeit wurde durch Addition der Lade- und Entladedauer die kürzeste Auftragsdurchführungszeit berechnet. Bei der Variante mit weichen Deadlines wurde die Deadline dann gleichverteilt auf einen Wert aus dem Intervall [2 * kürzeste Auftragsdurchführungszeit + Ausschreibungszeitpunkt, 4 * kürzeste Auftragsdurchführungszeit

+ Ausschreibungszeitpunkt] gesetzt. Bei der Variante mit harten Deadlines wurde die Deadline gleichverteilt auf einen Wert aus dem Intervall [1,5 * kürzeste Auftragsdurchführungszeit + Ausschreibungszeitpunkt, 2,5 * kürzeste Auftragsdurchführungszeit + Ausschreibungszeitpunkt] gesetzt.

Für die Ausgangspuffer wurden die Werte 0 und 2 gewählt. Die Anzahl der maximal erlaubten Rückgaben (siehe Besprechung der Starvation in Abschnitt 4.6.2.4) kann entweder als Teil der Auftragseinlastung oder als Teil der Vergabeparamter verstanden wissen. Der Übersicht halber wird der Wert hier der Auftragseinlastung zugeordnet. Als Werte wurden 1 und 5 gewählt.

Insgesamt ergeben sich so die folgenden Konfigurationen für die Auftragserstellung:

Konfiguration	Deadlines	Ausgangspuffergröße	Maximale Rückgabeanzahl
1	weich	0	1
2	weich	0	5
3	weich	2	1
4	weich	2	5
5	hart	0	1
6	hart	0	5
7	hart	2	1
8	hart	2	5

Tabelle 6.5: Flexibilität Experimente: Konfigurationen der Auftragserstellung

Für die Experimente wurden folgende Parametersets für die Auftragsvergabe gewählt:

Konfiguration	α	γ	ζ	η	ι
1	0,33	0,33	0,11	0,11	0,11
2	0,66	0,16	0,055	0,055	0,055
3	0,25	0,42	0,11	0,11	0,11
4	0,25	0,25	0,3	0,1	0,1
5	0,25	0,25	0,1	0,3	0,1
6	0,25	0,25	0,1	0,1	0,3

Tabelle 6.6: Flexibilität Experimente: Parameterkonfigurationen für die Auftragsvergabe

Wobei die einzelnen Parameter wie in Abschnitt 4.6.2.4 beschrieben folgende Größen gewichten:

α: Auftragsdauer,

γ: Reduzierung der Leerfahrt- bzw. Stillstandzeit in Folge einer Auftragsrückgabe,

ζ: Platzmangel an der Station,

η: Dringlichkeit in Abhängigkeit von der Deadline des Auftrags,

ι: Starvation in Abhängigkeit von der maximal zulässigen Anzahl von Rückgaben eines Auftrags und der aktuellen Anzahl der Rückgaben.

Bei der Konfiguration 1 werden Auftragsdauer, Reduzierung der Leerfahrt- bzw. Stillstandzeit und der Malus für eine Rückgabe gleich stark gewichtet. Bei der Konfiguration 2 ist

die Auftragsdauer die wichtigste Größe, bei Konfiguration 3 die Reduzierung der Leerfahrt- bzw. Stillstandzeit, und bei den Konfigurationen 4,5 und 6 der Malus für die Rückgabe. Bei der Konfiguration 4 ist hier der Platzmangel die wichtigste Größe, bei Konfiguration 5 die Dringlichkeit und bei Konfiguration 6 die Starvation.

Andere Parametersets sind natürlich möglich, aber die hier ausgewählten sechs repräsentieren eine Gleichgewichtung sowie jeweils ein Übergewicht eines jeden Faktors und sollten somit geeignet sein, um die prinzipiellen Auswirkungen der Parameter auf die Auftragsvergabe und damit auf die Systemperformanz aufzuzeigen.

6.2.2 Ergebnisse

Durch die 8 Konfigurationen bei der Auftragserstellung und die 6 verschiedenen Parametersets ergeben sich insgesamt 48 verschiedene Experimente. Die Ergebnisse dieser Experimente sind in den Tabellen 6.7 und 6.8 dargestellt. Die durchschnittliche Auftragsdauer und die maximale Wartezeit ist hier in Sekunden angegeben. Die maximale Wartezeit gibt die längste Zeitspanne zwischen der Erzeugung und den Beginn eines Auftrags an. Die Angaben "Deadline gerissen"gib an, wie viel Prozent der Aufträge erst nach Ablauf ihrer Deadline beendet wurden, die Angabe "Puffer voll"gibt an wie viel Prozent der Zeit mindestens ein Stationspuffer voll war, als der nächste Auftrag hätte ausgeschrieben werden sollen und die Angabe Leerfahrtanteil gibt an, wie viel Prozent der gefahrenen Strecken die FTF leer gefahren sind.

Alle Experimente wurden mit konfliktvermeidendem Routing ohne Verhandlungen durchgeführt.

Auftrags Konfiguration	Parameter Konfiguration	Durchschnittliche Auftragsdauer	Deadline gerissen	Puffer voll	Maximale Wartezeit	Leerfahrtanteil
1	1	299,3s	0,0%	6,9%	265,8s	51,8%
1	2	287,1s	0,0%	9,6%	259,1s	52,4%
1	3	305,7s	0,0%	11,1%	295,1s	51,4%
1	4	301,7s	0,0%	0,0%	263,1s	53,7%
1	5	300,8s	0,0%	7,3%	270,8s	52,9%
1	6	301,4s	0,0%	7,2%	271,9s	53,1%
2	1	299,1s	0,0%	6,7%	266,9s	51,7%
2	2	287,4s	0,0%	9,2%	260,7s	52,3%
2	3	305,5s	0,0%	10,8%	322,2s	51,4%
2	4	301,9s	0,0%	0,0%	268,1s	53,5%
2	5	300,6s	0,0%	7,1%	273,4	52,7%
2	6	301,8s	0,0%	7,1%	272,1s	52,5%
3	1	299,0s	0,0%	0,0%	267,1s	51,7%
3	2	286,8s	0,0%	0,0%	261,7s	52,5%
3	3	305,1s	0,0%	0,0%	318,2s	51,4%
3	4	301,0s	0,0%	0,0%	264,2s	53,2%
3	5	299,9s	0,0%	0,0%	275,1s	52,7%
3	6	301,1s	0,0%	0,0%	276,2s	52,9%
4	1	298,4s	0,0%	0,0%	268,2s	51,7%
4	2	286,4s	0,0%	0,0%	260,7s	52,1%
4	3	304,7s	0,0%	0,0%	355,7s	51,3%
4	4	300,8s	0,0%	0,0%	265,2s	53,1%
4	5	299,6s	0,0%	0,0%	277,2s	52,5%
4	6	300,8s	0,0%	0,0%	273,4s	52,3%

Tabelle 6.7: Flexibilität Experimente: Ergebnisse - 1

Auftrags Konfiguration	Parameter Konfiguration	Durchschnittliche Auftragsdauer	Deadline gerissen	Puffer voll	Maximale Wartezeit	Leerfahrtanteil
5	1	299,7s	2,1%	7,1%	264,9s	52,1%
5	2	288,0s	2,7%	9,9%	259,0s	52,7%
5	3	305,9s	4,1%	11,7%	292,0s	51,8%
5	4	302,0s	1,3%	0,4%	264,0s	53,9%
5	5	301,3s	0,0%	7,7%	260,9s	55,7%
5	6	301,6s	1,2%	7,3%	270,8s	53,8%
6	1	299,9s	1,9%	6,9%	265,7s	52,2%
6	2	287,9s	2,5%	9,5%	260,1s	52,6%
6	3	305,8s	3,9%	11,0%	312,7s	51,6%
6	4	302,2s	1,1%	0,0%	267,4s	55,9%
6	5	300,9s	0,0%	7,4%	263,4s	55,6%
6	6	301,9s	0,8%	7,2%	271,8s	53,7%
7	1	299,7s	1,9%	0,0%	266,7s	52,2%
7	2	287,1s	2,6%	0,0%	261,4s	52,5%
7	3	305,5s	3,8%	0,0%	311,2s	51,5%
7	4	301,5s	1,0%	0,0%	263,9s	53,9%
7	5	300,6s	0,0%	0,0%	265,1s	55,6%
7	6	301,8s	0,9%	0,0%	273,8s	53,4%
8	1	298,7s	1,8%	0,0%	267,1s	51,9%
8	2	286,7s	2,4%	0,0%	259,9s	52,5%
8	3	305,0s	3,6%	0,0%	321,2s	51,7%
8	4	301,2s	0,9%	0,0%	264,7s	53,4%
8	5	300,0s	0,0%	0,0%	267,8s	55,4%
8	6	301,3s	0,7%	0,0%	272,9s	53,0%

Tabelle 6.8: Flexibilität Experimente: Ergebnisse - 2

6.2.3 Fazit

Die in den Tabellen 6.7 und 6.8 präsentierten Ergebnisse, zeigen, dass die Wahl der Auftragsvergabeparameter durchaus einen Einfluss auf die erzielten Kennzahlen haben. Beispielsweise erzielte der Schwerpunkt auf das Einhalten der Deadlines (Parameter Konfiguration 5) bei den Auftragslasten mit harten Deadlines (Auftragskonfigurationen 5-8) durchgängig 0% gerissene Deadlines. Ähnliches gilt für die Puffer bei den Auftragskonfigurationen mit kleinen Stationspuffern (1,2,5,6) und der Vergabekonfiguration mit dem Schwerpunkt auf die Puffer (Parameter Konfiguration 4). Der Schwerpunkt auf die Rückgabe (Parameter Konfiguration 3) führt im Vergleich zu den anderen Parameter Konfigurationen bei gleicher Auftragslast zur besseren Ausnutzung der Fahrzeuge (kleinerer Leerfahrtanteil), allerdings auch zu einer schlechteren durchschnittlichen Auftragsdauer.

Insgesamt sind die Unterschiede in den Kennzahlen bei den durchgeführten 48 Experimenten in Abhängigkeit von den benutzten Auftragsvergabeparameter erkennbar, aber nicht sehr groß. Es kann dennoch festgehalten werden, dass die Auswahl der Parameter das Systemverhalten in die gewünschte Richtung lenken kann. Die Auswahl passender Parameter in Abhängigkeit vom gewünschten Systemverhalten ist ein Problem, was im Rahmen dieser Arbeit nicht gelöst wird. Möglichkeiten dafür dieses Problem zu lösen wären beispielsweise maschinelles Lernen oder Optmierungsalgorithmen.

6.3 Erweiterbarkeit

Die Untersuchung wie einfach das in dieser Arbeit entwickelte Konzept erweiterbar ist, wurde im Rahmen des Projekts CogniLog durchgeführt. Im folgenden Abschnitt wird daher eine kurze Einführung in das Projekt Cognilog gegeben. Im Abschnitt 6.3.2 wird dann beschrieben, welche Erweiterungen und Änderungen an der Simulation und an der entwickelten Steuerung notwendig waren um die Fragestellung des Projekts zu beantworten.

6.3.1 Kurzeinführung in das Projekt Cognilog

Das Projekt Cognilog beschreibt sich selbst wie folgt:

> Das Ziel des Forschungsprojekts CogniLog ist die durch Informationstechnik unterstützte Vernetzung von einzelnen Fördermodulen zu einem automatisierten, kognitiven Logistiknetzwerk. Das so entstehende Netzwerk reduziert die Komplexität der Organisation, Steuerung und Überwachung der eingesetzten Fördermodule und erhöht den Automatisierungsgrad.
>
> CogniLog reagiert autark auf Änderungen innerhalb der aktuellen Materialfluss-Konfiguration, etwa durch neue Auftragskombinationen, es steuert situativ, optimiert und behebt nicht vorhersagbare Störungen im intralogistischen Ablauf.
>
> Mit dem Projekt CogniLog wird eine Technologie für dezentral selbstkonfigurierende Logistiksysteme konzipiert. Die wissenschaftlichen Kernentwicklungen von CogniLog bestehen in

- der Entwicklung und Integration der notwendigen Schnittstellentechnologien zur Verbindung der verschiedenen Fördermodule,
- der Optimierung der Einsatzflexibilität der Fördermodule, so dass diese beliebig kombinierbar und für alle in der Intralogistik auftretenden Aufgabenstellung einsetzbar sind,
- der Integration geeigneter Planungs- und Optimierungsmethoden aus dem Bereich der reaktiven Planung, neuen Ansätzen zur Integration von Steuerungselektronik und dezentraler Intelligenz in die Fördermodule und der zugehörigen Hardware-Software-Infrastruktur,
- der Entwicklung einer geeigneten IT-Architektur für zentrale und dezentrale Komponenten in einem offenen System, der notwendigen Kopplung heterogener Netze (IEEE 802.11, Profibus, CAN-Bus, Ethernet, PowerLine u.a.) zur Integration in bestehende Strukturen,
- der Abstraktion der Hardware- und Netzwerktechnik durch eine Middleware,
- der Entwicklung neuartiger Einsatz-Optimierungs-Strategien für dezentral gesteuerte und intermittierend agierende Transportmittel.

Das Projekt adressiert die innerbetriebliche Materialflusstechnik als einen Kernaspekt der Intralogistik. Durch die höhere Flexibilität steigt die Nachhaltigkeit von Investitionen in Fördersysteme. Im Rahmen der Forschungsarbeiten erarbeitetes Wissen soll hier im Land genutzt werden, um Arbeitsplätze zu schaffen und die Wirtschaftskraft zu steigern.

Das Forschungsprojekt CogniLog wird mit Mitteln des europäischen Strukturfonds für regionale Entwicklung (EFRE) gefördert. Das Ziel des EFRE ist die Stärkung der wirtschaftlichen und sozialen Kohäsion in der Europäischen Union durch Abbau der Ungleichheiten zwischen den einzelnen Regionen. Durch das Projekt CogniLog wird der Aufbau innovativer logistischer Infrastrukturen im "Verkehrsdurchgangsland"Niedersachsen unterstützt.

Projektstart von CogniLog war der 1. Oktober 2008. Das Projekt hat eine Gesamtlaufzeit von 5 Jahren. (www.cognilog.de - Aufgerufen am 14.12.2013)

Das Projekt wurde durchgeführt vom Institut für Transport- und Automatisierungstechnik der Leibniz-Universität Hannover (ITA), vom OFFIS Institut für Informatik e. V. und von der Hochschule Osnabrück.

Zwei wesentliche Ergebnisse des Projekts waren zum einen Hardware zum anderen Software die zusammen einen schnell und einfach rekonfigurierbaren Stetigförderer ermöglichen sollte.

Auf Harwareseite wurde vom ITA Hannover ein Cognitive Conveyor (CoCo) entwickelt. Kern des CoCos ist ein hochfunktionaler Intralogistik-Knoten der mittels kleinskaliger Module eine Möglichkeit bietet, Weichen einfach durch Zusammenstellen bereits vorhandener Förderbandelemente zu realisieren. Auf einer ebenen Fläche sind eine Vielzahl frei schwenkbarer Rollen untergebracht, diese können in ihrer Neigung verstellt werden und beeinflussen somit den Weg, den ein Fördergut auf dem CoCo nimmt. Abbildung 6.1 zeigt eine solche Rolle.

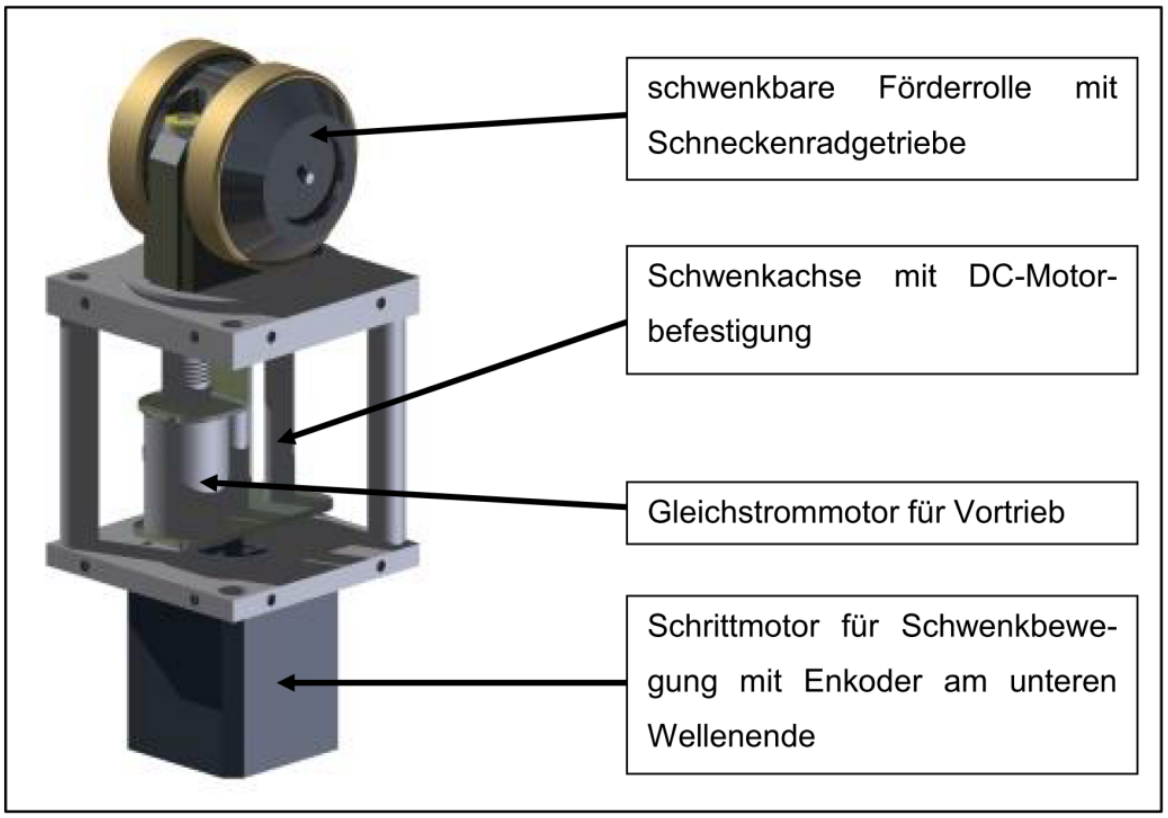


Abbildung 6.1: Kleinskaliges, modulares Stetigförderermodul[Quelle: CogniLog Abschlussbericht]

Abbildung 6.2 zeigt schematisch eine Fördermatrix bestehend aus einer großen Anzahl solcher Module. Durch die verschiedenen Einstellungen der einzelnen Module kann der Weg und die Drehung eines Transportgutes bestimmt werden. Mit dieser Technik ist es möglich, dass ein bestehender Stetigfördereraufbau umgebaut werden kann, ohne dass dafür hochspezialisiertes Personal oder ein größerer zeitlicher Aufwand notwendig wäre. Es wird an jeder Abzweigung ein CoCo aufgebaut, welches herkömmliche Förderelemente verbindet. Ausführliche Beschreibungen zum CoCo finden sich zum Beispiel in [KRSO13], [KO13] oder [VHR+12].

Auf Softwareseite wurde eine Steuerung entwickelt, die das Rekonfigurieren der Stetigförderer ohne aufwändige Neuprogrammierung ermöglichen soll. Die Ansteuerung von Stetigförderern erfolgt üblicherweise durch Industrie-PCs auf denen eine statische, speziell für den Aufbau angepasste Steuerung läuft. Eine solche Steuerung würde für jede physikalische Rekonfiguration ein Anpassen oder sogar ein neu Schreiben der Steuerungssoftware erforderlich machen. Die im Projekt vom OFFIS entwickelte Steuerung basiert auf auf Microcontrollern laufenden Agenten. Hierbei ist jedem Element des Stetigförderers ein Microcontroller zugeordnet, auf dem ein Agent das Element verwaltet. Bei einer Rekonfiguration tauschen sich nun diese Agenten aus, erkennen die neue Konfiguration und justieren die ihnen zugeordneten, physikalischen Stetigfördererelemente (Geschwindigkeit, Ein- und Ausschleuseverhalten, usw.) eigenständig, so dass der Stetigförderer direkt in der neuen Konfiguration betriebsbereit ist (siehe zum Beispiel [SH13]).

Im Projekt ging es weiterhin darum, ein dynamisches Intralogistiksystem zu entwerfen, das sowohl aus Stetig- als auch aus Unstetigförderern besteht. Die Unstetigförderer werden im

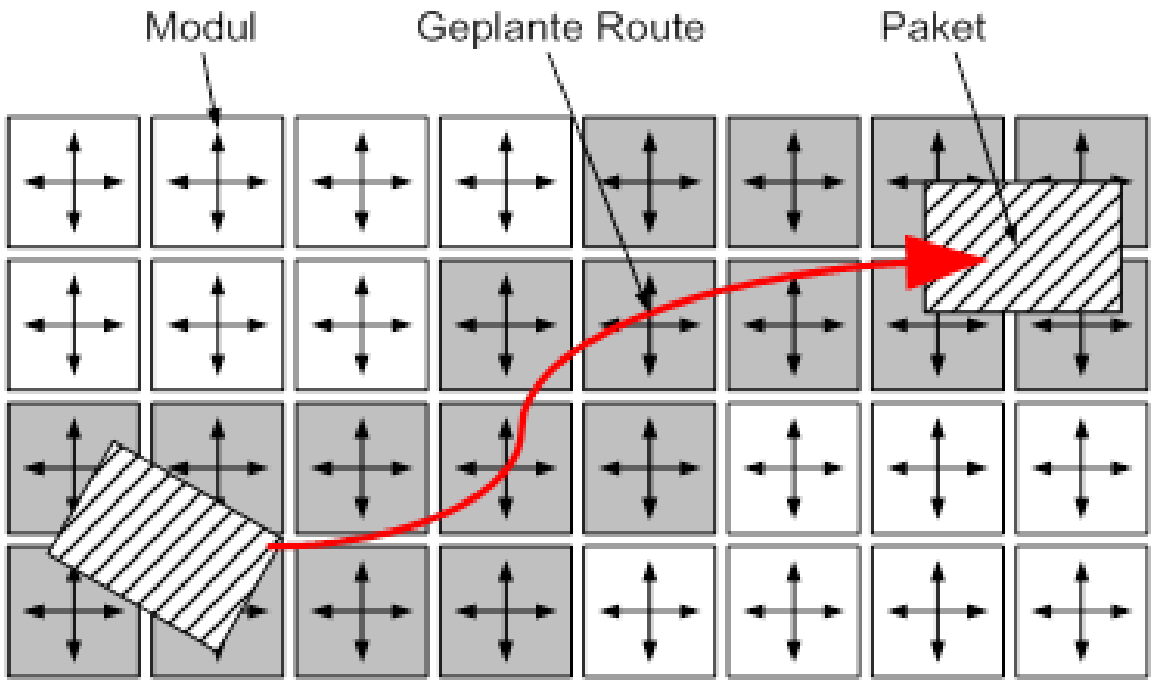


Abbildung 6.2: Schema der Fördermatrix: eine große Anzahl von Modulen wird zu einer Fläche verbunden, die in der Lage ist, Aufgaben wie das Fördern, Drehen oder Puffern von Paketen zu bewerkstelligen.[Quelle: CogniLog Abschlussbericht]

Projekt von menschlichen Fahrern gesteuert (Gabelstapler). Diesen soll zur Unterstützung ihrer Arbeit ein Agenten basiertes Navigationssystem zur Verfügung gestellt werden (siehe zum Beispiel [SH12]). Dieses System soll sowohl für die Vergabe von Aufträgen genutzt werden, als auch für das Festlegen von konfliktfreien Routen. Damit gelten für das Agenten-basierte Navigationssystem die gleichen Vorgaben, wie für die in dieser Arbeit entwickelte dezentrale FTS-Steuerung. Für die in diesem Abschnitt beschriebenen Experimente wurden daher die in dieser Arbeit entwickelten Verfahren zur dezentralen Steuerung für die Navigationsgeräte verwendet. Wenn wir, wie in dieser Arbeit der Fall, davon ausgehen, dass die FTF ohne Fehler in der Hardware funktionieren und wenn wir weiterhin davon ausgehen, dass sich die menschlichen Fahrer der Gabelstapler exakt an die Anweisungen des Navigationsgeräts halten, dann ist das Verhalten von FTF und Gabelstapler (bei identischen Fahreigenschaften wie Geschwindigkeit) gleich.

Das entwickelte Gesamtsystem sollte mit Hilfe einer Simulation evaluiert werden. Als Szenario wurde hierfür eine Crossdocking Halle eines Industriepartners gewählt. Das Layout der Halle ist in Abbildung 6.3 schematisch dargestellt. Die Innenmaße der Halle sind in etwa 100 Meter mal 40 Meter. Auf der oberen Seite gibt es 11 Wareneingangstore und auf der unteren Seite 11 Warenausgangstore. Das Dach wird von mehreren Säulen getragen, welche sowohl für die Gabelstapler als auch für mögliche Förderbandaufbauten Hindernisse darstellen. Im Mittel sind in dieser Halle 152 Transportaufträge pro Stunde durchzuführen. Im realen Betrieb wird in dieser Halle ausschließlich mit Gabelstaplern gearbeitet.

In der Simulation sollte untersucht werden, welche Auswirkungen der Einsatz eines rekonfigurierbaren Stetigförderers in der Halle auf die Anzahl der benötigten Unstetigförderer (Gabelstapler) hat. Insgesamt gibt es bei 11 Eingangs- und 11 Ausgangstoren 11*11=121

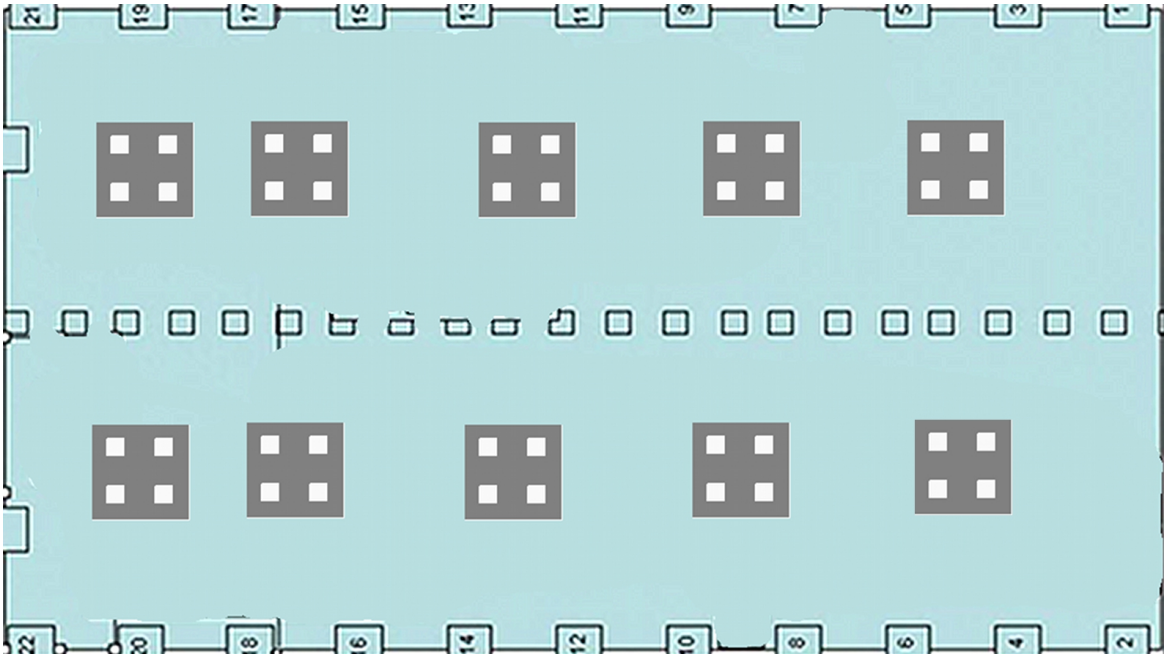


Abbildung 6.3: Die Crossdocking Halle

verschiedene Start/Ziel-Paare für Transportaufträge. Bei 157 Aufträgen in der Stunde, liegt der Erwartungswert bei 157/121≈1,3 Transportaufträgen pro Start/Ziel-Paar und Stunde. Bei einer solchen Gleichverteilung wäre der Einsatz eines rekonfigurierbaren Stetigförderers damit wahrscheinlich wenig hilfreich. In Wirklichkeit treten in der Halle aber immer wieder zeitlich begrenzte Phasen auf, in denen bestimmte Start/Ziel-Paare die deutliche Mehrheit der Transportaufträge darstellen. Das ist zum Beispiel immer dann der Fall, wenn gerade ein LKW an einem Wareneingangstor entladen und dessen Ladung auf zwei an Ausgangstoren stehende LKW aufgeteilt wird, oder wenn ein LKW an einem Warenausgangstor mit Ladung von zwei an Wareneingangstoren stehenden LKW beladen wird. Während dieser Be- und Entladungen könnte ein Stetigförderer, der diese drei Tore verbindet, eine positive Auswirkung auf die Gesamtperformanz haben.

Für die Experimente sollten drei Verteilungen berücksichtigt werden:

- Gleichverteilt,
- Schwach ausgeprägte Spitzen,
- Stark ausgeprägte Spitzen.

Bei der gleichverteilten Lastverteilung ist jedes Start/Ziel Paar gleich wahrscheinlich, d.h. für jedes Start/Ziel Paar gilt, dass die Wahrscheinlichkeit, dass ein Auftrag das entsprechende Paar hat 1/121 ist. Bei der Lastverteilung mit schwach ausgeprägten Spitzen wurden im simulierten Zeitraum vier Phasen von je 20 Minuten, in denen die Aufträge mit einer Wahrscheinlichkeit von 25% zwischen einem Starttor und zwei Zieltoren oder zwischen zwei Starttoren und einem Zieltor transportiert werden müssen, simuliert. Die übrigen 75% der Aufträge in diesen Phasen sind gleichverteilt. In der restlichen Zeit sind alle Aufträge gleichverteilt. Bei der Lastverteilung mit stark ausgeprägten Spitzen wurden im simulierten Zeitraum vier Phasen von je 20 Minuten, in denen die Aufträge mit einer Wahrscheinlichkeit von 50% zwi-

schen einem Starttor und zwei Zieltoren oder zwischen zwei Starttoren und einem Zieltor transportiert werden müssen, simuliert. Die übrigen 50% der Aufträge in diesen Phasen sind gleichverteilt. In der restlichen Zeit sind alle Aufträge gleichverteilt.

Neben der Auftragsverteilung ist die Umbaudauer des rekonfigurierbaren Stetigförderers eine wichtige Größe. Die Konfigurationen werden nach Abschluss einer Spitze umgebaut. Die Konfiguration wechselt dann von der Konfiguration, die die letzte Spitze bedient, zu der Konfiguration, die die kommende Spitze bedient. Es liegen noch keine empirischen Daten für die erwartete Umbaudauer vor, daher wurden die Experimente mit drei verschiedenen Umbaudauern durchgeführt. Die ausgewählten Umbaudauern sind:

- 10 Minuten
- 15 Minuten
- 20 Minuten

Die reale Umbauzeit sollte in der Praxis ebenfalls in diesem Bereich liegen. Während des Umbaus sind die entsprechenden Bereiche der Halle für Gabelstapler gesperrt.

Unterschieden werden sollte bei den Experimenten weiterhin, ob der Stetigförderer zu einer baulichen Trennung der Halle führt oder nicht. Abbildung 6.4 zeigt eine solche Trennung. Der Stetigförderer teilt die Halle in einen linken und einen rechten Bereich. Diese bauliche Trennung hat zwei entscheidende Folgen für den Betrieb der Unstetigförderer. Zum einen werden die Unstetigförderer in einem Bereich der Halle „eingesperrt“ . Das bedeutet, dass zum Beispiel Gabelstapler, die beim Aufbau des Stetigfördereres im linken Teil der Halle stehen, auch nur im linken Teil der Halle operieren können, solange der Stetigförderer in dieser Konfiguration aufgebaut ist. Zum anderen werden alle Aufträge, die von der einen in die andere Hälfte der Halle transportiert werden, zurückgehalten. Ein Transport über die Trennung durch den Stetigförderer hinweg ist nicht möglich. Solche Aufträge müssen also auf einen späteren Zeitpunkt verschoben werden.

Um die Auswirkungen einer solchen baulichen Trennung besser einschätzen zu können, wurden die Experimente auch ohne eine bauliche Trennung durchgeführt. In diesem Fall ist es den Unstetigförderen möglich, vom linken in den rechten Teil der Halle zu wechseln. Realisiert werden kann das zum Beispiel durch die Möglichkeit, außen an der Halle entlang zu fahren, durch eine Lücke zwischen dem Stetigförderer und den Toren oder durch eine Brücke im Stetigförderer. Im letzten Fall ist ein Abschnitt des Stetigförderers soweit erhöht, dass die Unstetigförderer darunter hindurch fahren können. Wie die Aufhebung der baulichen Trennung realisiert wird, wurde in den durchgeführten Experimenten nicht separat aufgeschlüsselt.

Die Experimente sollten mit einer erweiterten Version der in dieser Arbeit entwickelten Simulation sowie der in dieser Arbeit entwickelten Steuerung durchgeführt werden. Damit sollte neben der oben dargestellten Fragestellung des Projekts Cognilog ebenfalls die Frage beantwortet werden, wie einfach die dezentrale Steuerung erweiterbar ist.

6.3.2 Notwendige Erweiterungen

Um die von Cognilog benötigten Experimente durchführen zu können, mussten die in dieser Arbeit entwickelte Simulation und die in dieser Arbeit entwickelte Steuerung um folgende Punkte erweitert werden:

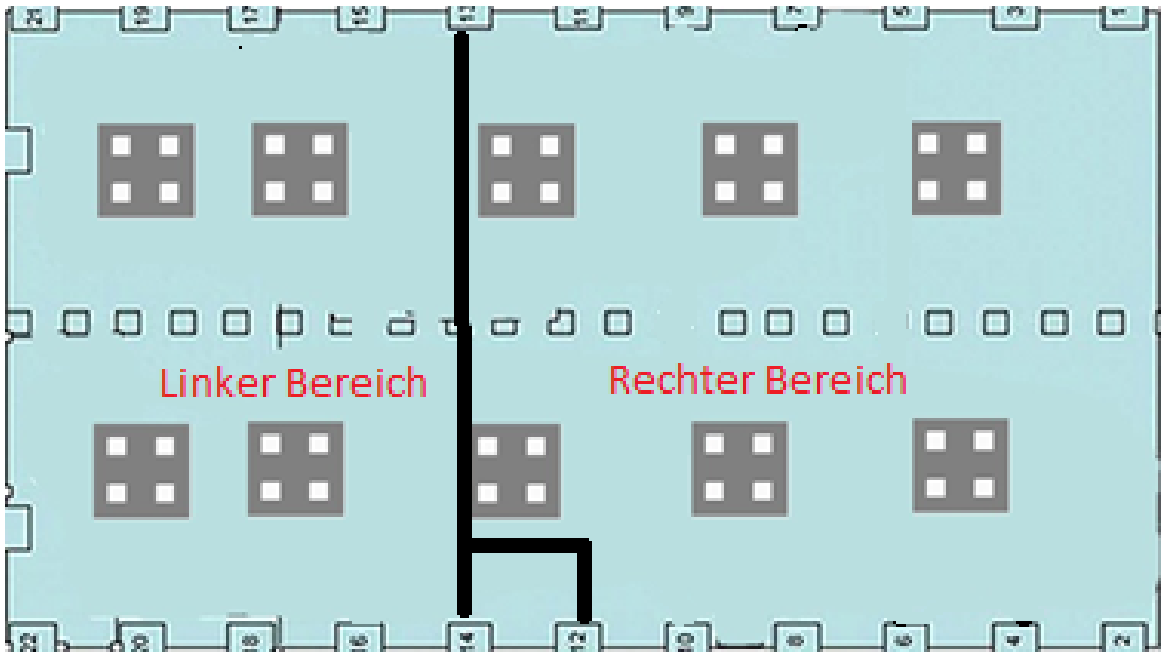


Abbildung 6.4: Bauliche Trennung der Halle

- Ein Modell für Stetigförderer und deren Steuerung,
- ein Modell für die Rekonfiguration der Stetigförderer,
- eine erweiterte Auftragsvergabe.

Stetigförderer und deren Steuerung
Bei dem Modell für die die Stetigförderer und deren Steuerung ging es an dieser Stelle nicht darum, die genaue Funktionsweise der Hardware, also wie genau welcher Aktor angesprochen werden muss, zu simulieren. Stattdessen musste das Verhalten, dass das System insgesamt zeigt, simuliert werden. Das Funktionieren der Hardware und deren Steuerung wurde wie im Falle der FTF als gegeben angenommen. Ein Stetigförderer wird in der Simulation durch eine Liste von Start- und Zielknoten sowie der benötigten Transportzeit zwischen diesen beschrieben. Abbildung 6.5 zeigt den Aufbau eines Förderbands, das den Knoten 13 (oben) mit den beiden Knoten 14 (unten links) und 12 (unten rechts) verbindet. Bei einer Transportzeit von 30 Zeiteinheiten von Knoten 13 zu Knoten 14 und einer Transportzeit von 40 Sekunden von Knoten 13 zu Knoten 12 würde das Förderband durch die beiden Tupel {Knoten13, Knoten14, 30} und {Knoten13, Knoten12, 40} beschrieben.

In der Simulation kann in jedem Schritt ein neues Transportgut auf das Förderband auf einem Startknoten platziert werden. Dieses wird dann gleichmäßig zum Zielknoten transportiert. Bei einer Platzierung eines Transportgutes auf den Förderbandabschnitt der durch das Tupel {Knoten13, Knoten14, 30} repräsentiert wird, bedeutet das, dass das Transportgut in jedem Takt um ein dreißigstel der Gesamtstrecke zwischen Knoten 13 und Knoten 14 bewegt wird und 30 Zeiteinheiten nachdem es auf Knoten 13 eingebracht wurde den Zielknoten 14 erreicht. Da die innere Logik des Systems nicht modelliert und simuliert wird, ist dieses Verhalten ausreichend. Im realen System gibt es sicher Beschleunigungs- und Abbremsphasen, so dass das Transportgut hier nicht in jedem Takt eine konstante Distanz weit bewegt wird.

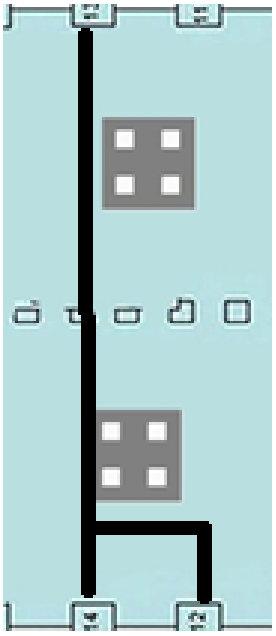


Abbildung 6.5: Beispiel eines Förderbandes

Die Einbettung in das in dieser Arbeit vorgestellte System geschieht über einen Agenten, der das Förderband repräsentiert. Dieser Conveyor-Agent kann, ähnlich wie der BasicFTF Agent (siehe Abschnitt 4.3.1) Angebote auf OrderAnnouncement abgeben. Das Angebot besteht bei ihm immer aus der Transportzeit. Außerdem kann er natürlich nur Angebote abgeben für Aufträge deren Start- und Zielknoten mit Start- und Zielknoten des Förderbandes, welches er repräsentiert, übereinstimmen.

Rekonfiguration der Stetigförderer
Die Rekonfiguration der Stetigförderer geschieht in der Realität durch das Umpositionieren und neu Zusammenfügen der einzelnen Komponenten der Stetigförderer. Für die Ausführungslogik ist das jedoch nicht relevant. Hier kommt es nur darauf an, zu wissen zu welchen Zeitpunkten welche Konfigurationen einsatzbereit sind. Hierfür wurde ein Agent konzipiert, der diese Informationen kapselt. Im sogenannten ConveyorManager-Agenten werden alle verfügbaren Konfigurationen der Stetigförderer als Conveyor-Agent gespeichert. Neben den Agenten gibt es ein Zeitprofil, das die Änderungen der Konfiguration zeitlich auflistet. Hierdurch kann der Agent Anfragen der Form „Gib mir alle aktuell aktiven Conveyor-Agenten“, „Gib mir alle zum Zeitpunkt x aktiven Conveyor-Agenten“ oder „Gib mir alle Conveyor-Agenten die zwischen den Zeitpunkten x und y aktiv sind“ beantworten.

Die Frage, zu welchen Zeitpunkten welche Konfiguration im Einsatz sein sollte, sprich, das Ermitteln der besten oder möglichst guter Umbauzeiten, ist für die Effizienz und den Nutzen der Lösung entscheidend. Sie ist aber nicht Frage der vorliegender Arbeit. Daher wurde in der vorliegenden Arbeit davon ausgegangen, dass die Daten, zu welchem Zeitpunkt welche Konfigurationen aktiv sind, schon zum Simulationsstart vorliegen.

Erweiterte Auftragsvergabe
Neben den beschriebenen neuen Agenten Conveyor-Agent und ConveyorManager-Agent musste die Auftragsvergabe erweitert werden. Als Grundlage hierfür wurde die Auftragsvergabe wie im Abschnitt 4.6 beschrieben verwendet.

Die erweiterte Auftragsvergabe geschieht in zwei Phasen. In der ersten Phase wird der Auftrag wie in Abschnitt 4.6 beschrieben ausgeschrieben. Das beste Angebot wird als Referenz benutzt um den Auftrag anschließend auch an die Stetigförderer auszuschreiben. Hierfür wird zunächst aus dem besten FTF Angebot die maximale Zeit ermittelt, die ein Angebot eines Stetigförderers bis zur Beendigung des Auftrags benötigen kann, damit das Angebot noch besser als das beste Angebot der FTF-Agenten ist. Anschließend fragt der OrderDisposer beim Conveyor-Manager alle Conveyor-Agenten ab, die zwischen dem aktuellen Zeitpunkt und dem aktuellen Zeitpunkt plus der ermittelten maximalen Zeit aktiv sind. Den so ermittelten Conveyor-Agenten schickt er dann ebenfalls das OrderAnnouncement. Die Conveyor-Agenten überprüfen dann, ob sie den Auftrag ausführen können, ob also der Start- und Zielknoten des Auftrags mit Start- und Zielknoten ihrer Konfiguration übereinstimmen, und schicken, wenn das der Fall ist, ein Angebot zurück. Im letzten Schritt vergleicht der OrderDisposer dann das beste Angebot der FTF mit dem besten Angebot der Stetigförderer und vergibt den Auftrag an das bessere der beiden Angebote.

Die Angebotsberechnung beim Stetigförderer basiert ausschließlich auf der Zeit. Wird für die Angebotsberechnung der FTF ein anderes Verfahren verwendet, werden die entsprechenden anderen Terme, wie beispielsweise die benötigten Ressourcen, bei den Angeboten der Stetigförderer auf 0 gesetzt.

Die Frage der Rekonfigurationsplanung, also wann welche Konfiguration aktiv sein sollte, wann und nach welchen Kriterien ein Umbau starten sollte und wie sich diese Kriterien aus den Informationen über den bevorstehenden (oder auch den vergangenen) Warenflüssen ermitteln lassen, bietet sicher noch einiges an Forschungspotential. In diesem Zusammenhang kann eine andere Modellierung der Auftragsvergabe notwendig oder sinnvoll werden.

Zusammenfassend war für die Erweiterung der dezentralen Steuerung eines fahrerlosen Transportsystems zu einem System bestehend aus fahrerlosen Transportfahrzeugen und rekonfigurierbaren Stetigförderern also die Erweiterung des OrderDisposers sowie die Implementierung eines einfachen Conveyor-Agenten sowie eines ConveyorManger-Agenten notwendig.

6.3.3 Kurze Zusammenfassung der Ergebnisse

Es wurden insgesamt 21 Experimente durchgeführt. Die einzelnen Parameter sind der Tabelle 6.9 zu entnehmen. Die Experimente 1, 2 und 3 dienen zum Vergleich. In diesen Experimenten wird die Auftragslast ausschließlich von Unstetigförderern bewältigt. Dies ist in der Warenhalle, die als Grundlage für die Experimente verwendet wurde, auch der Fall. Weiterhin wurden Experimente mit Stetigförderern jeweils mit und ohne baulicher Trennung, mit Umbauphasen von 10, 15 und 20 Minuten Dauer sowie mit gleichverteilter Auftragslast, Auftragslast mit schwach ausgeprägten Spitzen und Auftragslast mit stark ausgeprägten Spitzen durchgeführt. In Tabelle 6.10 sind die Ergebnisse aller Experimente aufgeführt.

Auffallend ist, dass es zu einer signifikanten Verschlechterung der benötigten Unstetigförderer und der Durchlaufzeit kommt, wenn die Benutzung rekonfigurierbarer Stetigförderer zu einer baulichen Trennung der Halle führt (Experimente 4-12). Der Bedarf der Unstetigförderer steigt von 7-9 auf 12-15, die durchschnittliche Durchlaufzeit auf den Unstetigförderern von 166 – 214 Sekunden auf 360 – 410 Sekunden. In den Experimenten, in denen es zu keiner baulichen Trennung der Halle kommt (Experimente 13-21) hängt der Nutzen der rekonfigurierbaren Stetigförderer von der Auftragslast ab. Bei einer Gleichverteilung der Aufträge können keine

Nr.	Einsatz von Stetigförderern	Bauliche Trennung	Umbaudauer	Last-Verteilung
1	Nein	-	-	Gleichverteilt
2	Nein	-	-	Schwach ausgeprägte Spitzen
3	Nein	-	-	Stark ausgeprägte Spitzen
4	Ja	Ja	10	Gleichverteilt
5	Ja	Ja	10	Schwach ausgeprägte Spitzen
6	Ja	Ja	10	Stark ausgeprägte Spitzen
7	Ja	Ja	15	Gleichverteilt
8	Ja	Ja	15	Schwach ausgeprägte Spitzen
9	Ja	Ja	15	Stark ausgeprägte Spitzen
10	Ja	Ja	20	Gleichverteilt
11	Ja	Ja	20	Schwach ausgeprägte Spitzen
12	Ja	Ja	20	Stark ausgeprägte Spitzen
13	Ja	Nein	10	Gleichverteilt
14	Ja	Nein	10	Schwach ausgeprägte Spitzen
15	Ja	Nein	10	Stark ausgeprägte Spitzen
16	Ja	Nein	15	Gleichverteilt
17	Ja	Nein	15	Schwach ausgeprägte Spitzen
18	Ja	Nein	15	Stark ausgeprägte Spitzen
19	Ja	Nein	20	Gleichverteilt
20	Ja	Nein	20	Schwach ausgeprägte Spitzen
21	Ja	Nein	20	Stark ausgeprägte Spitzen

Tabelle 6.9: CogniLog Experimente: Konfigurationen

Nr.	Benötigte Unstetigförderer	Aufträge pro Stunde auf Stetigförderer	Aufträge pro Stunde auf Unstetigförderern	Ø Durchlaufdauer auf Stetigförderer	Ø Durchlaufdauer auf Unstetigförderern
1	7	-	152	-	166 s
2	9	-	152	-	213 s
3	9	-	152	-	214 s
4	15	2	150	50 s	360 s
5	14	23	129	45 s	390 s
6	12	49	103	46 s	410 s
7	15	2	150	50 s	360 s
8	14	23	129	45 s	390 s
9	12	49	103	46 s	410 s
10	15	2	150	55 s	405 s
11	14	23	129	49 s	363 s
12	13	49	103	48 s	412 s
13	7	2	150	50 s	168 s
14	7	23	129	45 s	195 s
15	5	49	103	46 s	174 s
16	7	2	150	50 s	168 s
17	7	23	129	45 s	195 s
18	5	49	103	46 s	174 s
19	7	2	150	55 s	168 s
20	8	23	129	49 s	195 s
21	6	49	103	48 s	174 s

Tabelle 6.10: CogniLog Experimente: Ergebnisse

Unstetigförderer eingespart werden. Bei schwach oder stark ausgeprägten Spitzen können ein oder zwei Unstetigförderer eingespart werden. Die verschiedenen Umbaudauern von 10,15 und 20 Minuten haben keinen oder nur einen sehr kleinen, vernachlässigbaren Einfluss auf die erzielten Ergebnisse.

Durch die durchgeführten Experimente konnte gezeigt werden, dass der Einsatz rekonfigurierbarer Stetigförderer in einer Warenumschlaghalle ergänzend zum Einsatz von Unstetigförderern nur zu einem Einsparpotential führen kann, wenn der Aufbau des Stetigförderers nicht zu einer baulichen Trennung der Halle führt und wenn die Auftragslast phasenweise ausgeprägte Spitzen aufweist. Sind diese Bedingungen erfüllt, kann die Anzahl der benötigten Unstetigförderer um bis zu 22% (2 von 9) reduziert werden.

Die Ergebnisse der Untersuchungen haben gezeigt, dass eine bauliche Trennung der Halle durch den Stetigförderer zu einer relativ schlechten Performanz des Gesamtsystems führt. Ein möglicher Lösungsansatz wäre hier beispielsweise die Möglichkeit einzuräumen einen Transportauftrag sowohl von FTF als auch von Stetigförderern transportieren zu lassen. So könnte ein FTF den Auftrag vom Eingangstor zum Förderband bringen, dieses transportiert ihn zum anderen Ende der Halle wo es von einem zweiten FTF aufgeladen und zum Ziel transportiert wird. Neben der technischen Aufgabenstellung FTF das Be- und Entladen vom Stetigförderer zu ermöglichen wäre hierfür ein neuer Ansatz für die Auftragsvergabe notwendig.

Kapitel 7

Fazit und Ausblick

Die Evaluationen der entwickelten Konzepte haben gezeigt, dass eine gute Systemperformanz von Fahrerlosen Transportsystemen erreichbar ist, wenn diese dezentral gesteuert werden. Ebenso ist es möglich eine dezentrale Steuerung um weitere Konzepte zu erweitern, wie am Beispiel der dynamisch rekonfigurierbaren Stetigförderer gezeigt wurde. Durch eine geeignete Wahl der Parameter für die Auftragsvergabe kann das Systemverhalten an sich ändernde Vorgaben angepasst werden.

Insgesamt konnte durch die entwickelten und implementierten Ansätze somit gezeigt werden, dass eine vollständig dezentrale Steuerung von Fahrerlosen Transportsystemen möglich ist.

Wie bei vielen dezentralen Ansätzen ist es auch bei dem in dieser Arbeit vorstellten Ansatz so, dass die entwickelten Ansätze auch in einer zentralen, Agenten basierten Steuerung eingesetzt werden können. Es ist ein spannende Frage, ob bei einer solchen zentralen Steuerung alle in dieser Arbeit vorgestellten Konzepte übernommen werden sollten oder ob beispielsweise eine andere Form der Auftragsvergabe zu besseren Ergebnissen führen würde. Das Zulassen einer solchen Mischform eröffnet in jedem Fall ein breites Spektrum an Möglichkeiten für die Kombination der in dieser Arbeit vorgestellten Konzepte mit zentralen Ansätzen beispielsweise aus dem Bereich des Scheduling.

Ein Faktor für die gute Systemperformanz war die Einführung von Verhandlungen als Heuristik zur Verbesserung des konfliktfreien Routings. In dieser Arbeit wurde zum einen ein allgemeines Modell für diese Verhandlungen vorgestellt, als auch zwei konkrete Verhandlungen beschrieben. Hiervon ausgehend stellen sich nun viele weitere Fragen. Als erstes natürlich, was für Verhandlungen noch eingesetzt werden können. Hier spielt die Balance zwischen benötigtem Aufwand für die notwendigen Berechnungen sowie dem Nutzen für die Systemperformanz eine wichtige Rolle. Mit einer größeren Zahl von Verhandlungen stellt sich weiterhin die Frage, wie diese kombiniert werden können oder welche Verhandlungen hintereinander und in welcher Reihenfolge durchgeführt werden können.

Wenn auf die vollständige Dezentralität, die Kernanforderung der vorliegenden Arbeit war, verzichtet wird, kann weiterhin eine Erweiterung des Agentenansatzes zu einem holonischem System untersucht werden. Eine naheliegende Möglichkeit wäre hier der Fall, in dem bisher Verhandlungen zum Einsatz kamen: Wenn ein Agent beim Berechnen seiner Route feststellt, dass die für ihn optimale Route durch Reservierungen gesperrt ist, ermittelt er alle Agenten, die Reservierungen auf dieser Route haben. Diese Agenten schließen sich dann zu einem

Holon zusammen, welches für alle an diesem Zusammenschluss beteiligten Agenten Routen ermittelt, die den Plänen der Agenten genügen und die zusammen eine möglichst gute Lösung für das Gesamtsystem darstellen.

Insgesamt zeigt die vorliegende Arbeit, dass die dezentrale Steuerung von Fahrerlosen Transportsystemen effizient möglich ist und dass Erweiterungen und Kombinationen mit zentralen Techniken zu einer weiteren Verbesserung der resultierenden Systemperformanz führen können.

Literaturverzeichnis

[BBPW85] Broadbent, A.J. ; Besant, C.B. ; Premi, S.K. ; Walker, S.P.: Free ranging AGV systems: promises, problems and pathways. In: *Proc. 2nd Int. Conf. on Automated Materials Handling* (1985), S. 221–237

[BC87] Bazerman, Max H. ; Carroll, John S.: Negotiator cognition. In: *Research In Organizational Behavior* 9 (1987), 247–288. `http://dspace.mit.edu/handle/1721.1/48577`

[BL00] Barbuceanu, M. ; Lo, W.: A multi-attribute utility theoretic negotiation architecture for electronic commerce. In: *Proceedings of 4th Int. Conf. on Autonomous Agents* (2000), S. 239–247

[BWG+06] Becker, M. ; Wenning, B.-L. ; Görg, C. ; Gehrke, J.D. ; Lorenz, M. ; Herzog, O: Agent-based and discrete event simulation of autonomous logistic processes. In: *Proceedings of the 20th European Conference on Modelling and Simulation.*, 2006

[BWHM04] Boucké, Nelis ; Weyns, Danny ; Holvoet, Tom ; Mertens, Koenraad: Decentralized Allocation of Tasks with Delayed Commencement. In: *In Proceedings of European Workshop on Multiagent Systems* (2004)

[BZW97] Brenner, Walter ; Zarnekow, Rüdiger ; Wittig, Hartmut: *Intelligente Softwareagenten: Grundlagen und Anwendungen (German Edition).* Springer, 1997

[Dev10] Devine, Pat: *Democracy and Economic Planning.* Polity, 2010

[Dij59] Dijkstra, E.W.: A Note on Two Problems in Connecting with Graphs. In: *Numerische Mathematik* 1 (1959), S. 269 – 271

[Dor92] Dorigo, Marco: Optimization, learning and natural algorithms. In: *Ph. D. Thesis, Politecnico di Milano, Italy* (1992)

[DTH10] Daniluk, D. ; Ten Hompel, M.: Modellierung von multiagentenbasierten Materialflusssystemen für die verteilte Simulation. In: *Integrationsaspekte der Simulation: Technik, Organisation und Personal*, 2010

[Ech11] Echelmeyer, Wolfgang: Automatisierung in der Logistik - Chancen und Herausforderungen bei der Vollautomatisierung in der Intralogisitk. In: *Forschungsmagazin, Reutlingen Research Institute* (2011), S. 7–8

[Fer98] Ferber, Jacques: *Multi-agent systems : an introduction to distributed artificial intelligence.* Harlow : Addison-Wesley, 1998. – ISBN 0201360489

[FIP01] FIPA ; FIPA (Hrsg.): *FIPA Contract Net Interaction Protocol Specification.* FIPA, 2001. http://www.fipa.org/specs/fipa00029/

[FSJ98] Faratin, P. ; Sierra, C. ; Jennings, N.R.: Negotiation Decision Functions for Autonomous Agents. In: *Int. Journal of Robotics and Autonomous Systems* 4 (1998), S. 159–182

[GG05] Graudina, V. ; Grundspenkis, J.: Technologies and Multi-Agent System Architectures for Transportation and Logistics Support: An Overview. In: *Proc. CompSysTech' 2005* (2005)

[GH10] Günthner, Willibald A. (Hrsg.) ; Hompel, Michael (Hrsg.): *Internet der Dinge in der Intralogistik (VDI-Buch).* Springer, 2010

[GH13] Galloway, B. ; Hancke, G.P.: Introduction to Industrial Control Networks. In: *Communications Surveys Tutorials, IEEE* 15 (2013), Nr. 2, S. 860–880

[GL10] Göhring, S. ; Lorenz, T.: Agentenbasierte Staplerleitsysteme. In: Günthner, W.A. (Hrsg.) ; Ten Hompel, M. (Hrsg.): *Internet der Dinge in der Intralogistik (VDI-Buch).* Springer, 2010

[GLWB06] Gehrke, J. D. ; Lorenz, M. ; Wenning, B.-L. ; Becker, M.: Integration of two approaches for simulation of autonomous logistic processes. In: *Tagungsband zur 12. ASIM Fachtagung*, 2006

[Gra59] Grassé, Pierre-Paul: La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie:essai d'interprétation du comportement des termites constructeurs. In: *Insectes Sociaux* 6 (1959), S. 41–83

[Gri13] Grimshaw, David: *JADE Administration Tutorial.* Version: 2013. http://jade.tilab.com/doc/tutorials/JADEAdmin/index.html

[Hei06] Heinecker, M.: *Methodik zur Gestaltung und Bewertung wandelbarer Matrialflusssysteme*, TU München, Diss., 2006

[Hei11] Heiserich, Gerd: *Kooperative adaptive Ablaufsteuerung für innerbetriebliche Materialflusssysteme.* Garbsen : PZH Produktionstechnisches Zentrum, 2011. – ISBN 9783943104028

[HN01] Hatzack, W. ; Nebel, B.: The operational traffic problem: Computational complexity and solutions. In: *Proceedings of the 6th European Conference on Planning* (2001)

[HNR68] Hart, P.E. ; Nilsson, N.J. ; Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *Systems Science and Cybernetics, IEEE Transactions on* 4 (1968), Nr. 2, S. 100–107. http://dx.doi.org/10.1109/TSSC.1968.300136. – DOI 10.1109/TSSC.1968.300136. – ISSN 0536–1567

[HVKB04] Hadeli ; Valckenaers, Paul ; Kollingbaum, Martin ; Brussel, Hendrik V.: Multi-agent coordination and control using stigmergy. In: *Computers in Industry* 53 (2004), Nr. 1, S. 75 – 96

[JFL+01] Jennings, N.R. ; Faratin, P. ; Lomuscio, A.R. ; Parsons, S. ; Sierra, C. ; Wooldridge, M.: Automated negotiation: prospects, methods and challenges. In: *Int. Journal of Group Decision and Negotiation* (2001), 1–30. `http://www.springerlink.com/index/JV77718M36178836.pdf`

[JL08] Jedermann, Reiner ; Lang, Walter: The benefits of embedded intelligence - tasks and applications for ubiquitous computing in logistics. In: *In Proceedings of the 1st International Conference on the Internet of Things (IOT*, 2008, S. 105–122

[Jod08] Jodlbauer, Herbert: *Produktionsoptimierung: Wertschaffende sowie kundenorientierte Planung und Steuerung (Springers Kurzlehrbücher der Wirtschaftswissenschaften) (German Edition).* Springer, 2008

[Jun04] Jungnickel, Dieter: *Graphs, Networks and Algorithms (Algorithms and Computation in Mathematics).* Springer Verlag, 2004. – ISBN 3540219056

[Jun09] Jungmann, Uta: Fahrerlose Transportsysteme - Vom Lastenschlepper zur Minibar. In: *FAZ vom 23.04.2009* (2009)

[KG13] Kugler, Wilfried ; Gehlich, Dirk: Einsatz von Agentensystemen in der Intralogistik. In: Göhner, Peter (Hrsg.): *Agentensysteme in der Automatisierungstechnik.* Springer Berlin Heidelberg, 2013, S. 113–128

[KL95] Kraus, S. ; Lehmann, D.: Designing and building an automated negotiation agent. In: *Computational Intelligence* 11 (1995), S. 132–171

[KO13] Krühn, T. ; Overmeyer, L.: Deadlock Prevention for a Cognitive Conveyor with Overlapping Intersections. In: *In Proceedings of the Distributed Intelligent Systems and Technologies Workshop*, 2013, S. 57–66

[KRSO13] Krühn, T. ; Radosavac, M. ; Shchekutin, N. ; Overmeyer, L.: Decentralized and Dynamic Routing for a Cognitive Conveyor. In: *In Proceedings of the International Conference on Advanced Intelligent Mechatronics (AIM)*, 2013, S. 436–441

[KT91] Kim, C.W. ; Tanchoco, J.M.A.: Conflict-free shortest-time bidirectional AGV routeing. In: *International Journal of Production Research* 29 (1991), Nr. 12, 2377–2391. `http://cat.inist.fr/?aModele=afficheN&cpsidt=5080594`

[Lat91] Latombe, Jean-Claude: *Robot Motion Planning.* Norwell, MA, USA : Kluwer Academic Publishers, 1991. – ISBN 079239206X

[LCrPS04] Luke, Sean ; Cioffi-revilla, Claudio ; Panait, Liviu ; Sullivan, Keith: MASON: A new multi-agent simulation toolkit. In: *Proceedings of the 2004 SwarmFest Workshop*, 2004

[Loo04] Loomis, Mildred J.: *Decentralism: Where It Came From–Where Is It Going?* Black Rose Books, 2004

[LTSK06] Langer, H ; Timm, I J. ; Schönberger, J ; Kopfer: H.: Integration von Software-Agenten und Soft-Computing-Methoden für die Transportplanung. In: *Hrsg.): Softwareagenten und Soft Computing im Geschäftsprozessmanagement.*

Innovative Methoden und Werkzeuge zur Gestaltung, Steuerung und Kontrolle von Geschäftsprozessen in Dienstleistung, Verwaltung und Industrie, Cuvillier Verlag, 2006, S. 39–51

[Mar06] Martin, H.: *Transport- und Lagerlogistik - Planung, Struktur Steuerung und Kosten von Systemen in der Intralogisitk. 6. vollst. überarb. Aufl.* Wiesbaden : Vieweg, 2006

[Mat08] Matloff, Norm: Introduction to discrete-event simulation and the simpy language. In: *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2 (2008), S. 2009

[MHH08] Mes, Martijn ; Heijden, Matthieu van d. ; Hillegersberg, Jos van: Design choices for agent-based control of AGVs in the dough making process. In: *Decision Support Systems* 44 (2008), Nr. 4, 983–999. http://doc.utwente.nl/79107/

[MKGS04] Möhring, R.H. ; Köhler, E. ; Gawrilow, E. ; Stenzel, B.: *Conflict-free real-time AGV routing.* http://www.springerlink.com/index/G455J4572W315870.pdf. Version: 2004

[MM05] Marik, V. ; McFarlane, D.: Industrial adoption of agent-based technologies. In: *Intelligent Systems, IEEE* 20 (2005), Nr. 1, S. 27–35

[MO03] Mannheims, Hildegard ; Oberem, Perter: *Versteigerung: Zur Kulturgeschichte Der Dinge Aus Zweiter Hand. Ein Forschungsbericht (Beitrage Zur Volkskultur).* Waxmann Verlag Gmbh, 2003. – ISBN 3830912803

[Mor10] Mors, Adriaan W.: *The world according to MARP.* The Netherlands, Delft University of Technology, Diss., 2010

[Mor11] Mors, Adriaan ter: Evaluating heuristics for prioritizing context-aware route planning agents. In: *Proceedings of the 8th IEEE International Conference on Networking, Sensing, and Control* (2011), April, 127–132. http://dx.doi.org/10.1109/ICNSC.2011.5874899. – DOI 10.1109/ICNSC.2011.5874899. ISBN 978–1–4244–9570–2

[MZ07] Mors, AW ter ; Zutt, J.: Context-aware logistic routing and scheduling. In: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling* (2007), 328–335. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Context-Aware+Logistic+Routing+and+Scheduling#0

[MZL04] Mamei, Marco ; Zambonelli, Franco ; Leonardi, Letizia: Co-Fields: A Physically Inspired Approach to Distributed. In: *IEEE Pervasive Computing* 3 (2004)

[Nas50] Nash, J.: *Non-cooperative games*, Princeton University, Diss., 1950. http://www.jstor.org/stable/10.2307/1969529

[NK12] Nebel, Bernhard ; Kleiner, Alexander: Multi-Agenten-Systeme in der Intralogistik. In: *IEE - Elektrische Automatisierung + Antriebstechnik* (2012), Nr. 4, S. 48–53

[NM44] Neumann, John V. ; Morgenstern, Oskar: *Theory of Games and Econo-*

mic Behavior. Princeton University Press, 1944 `http://jmvidal.cse.sc.edu/library/neumann44a.pdf`. – ISBN 0691119937

[O'C98] O'Connor, M.L.: *Carrier-phase differential GPS for automatic control of land vehicles*, Stanford University, Diss., 1998

[Rob04] Robinson, Stewart: *Simulation: the practice of model development and use*. Wiley, 2004

[San94] Sancho, N.G.F.: Shortest Path Problems with Time Windows on Nodes and Arcs. In: *Journal of Mathematical Analysis and Applications* 186 (1994), Nr. 3, 643 - 648. `http://dx.doi.org/10.1006/jmaa.1994.1324`. – DOI 10.1006/jmaa.1994.1324. – ISSN 0022–247X

[San99] Sandholm, T.W.: Distributed rational decision making. In: Weiss, G. (Hrsg.): *Multiagent Systems*. MIT Press, 1999, S. 201–258

[SBB+12] Schwarz, Christoph ; Busch, Jan ; Beth, Carsten ; Sauer, Jürgen ; Hahn, Axel;: EnvEdit – A Graphical Environment Editor. In: *Proceedings of the 4th International Conference on Agents and Artifical Intelligence – Volume 2* INSTICC, 2012

[Sch07] Schmidt, Vivien A.: *Democratizing France: The Political and Administrative History of Decentralization*. Cambridge University Press, 2007

[SH12] Schwarz, Christoph ; Hahn, Axel: Modeling Individual Decision Makers In A Multi Agent Warehouse Simulation. In: *Proceedings of the 10th International Industrial Simulation Conference*, 2012

[SH13] Stasch, A. ; Hahn, A.: Towards a Multi-agent Platform for Cyber-Physical Systems Based on Low-Power Microcontroller for Automated Intralogistics - A Minimized Embedded Solution for the Internet of Things in Intralogistical Environments. In: *In Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2013

[SHS12] Schwarz, Christoph ; Hahn, Axel ; Sauer, Jürgen: Multi Agent Simulation Of A Warehouse. In: *Proceedings of the 2012 International Conference on Logistics and Maritime Systems (LOGMS)*, 2012

[Smi80] Smith, R. G.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. In: *IEEE Trans. Comput.* 29 (1980), Nr. 12, S. 1104–1113

[SRBZP10] Smolic-Rocak, Nenad ; Bogdan, Stjepan ; Zdenko, Kovacic ; Petrovic, Tamara: Time windows based dynamic routing in multi-AGV systems. In: *IEEE Transactions on Automation Science and Engeneering* 7 (2010), Nr. 1, 151–155. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4907246`

[SRFR+05] Scholz-Reiter, B. ; Freitag, M. ; Rekersbrink, H. ; Wenning, B.L. ; Gorldt, C. ; Echelmeyer, W.: Auf dem Weg zur Selbststeuerung in der Logistik - Grundlagenforschung und Praxisprojekte. In: *Begleitband zur 11. Magdeburger Logistiktagung* (2005), S. 166–180

[SRH01] Scholz-Reiter, B. ; Höhns, Hartmut: Reaktive Planung und Steuerung von logistischen Prozessen mit Multiagentensystemen (MAS). In: *Industrie Management* 17 (2001), Nr. 6

[SRWK+04] Scholz-Reiter, B. ; Windt, K. ; Kolditz, J. ; Böse, F. ; Hildebrandt, T. ; Philipp, T. ; Höhns, H.: New Concepts of Modelling and Evaluating Autonomous Logistic Processes. In: *In Proceedings of the IFAC-MIM 2004*, 2004

[SSP09] Singh, Namita ; Sarngadharan, P. V. ; Pal, Prabir K.: AGV scheduling for automated material distribution: a case study. In: *Journal of Intelligent Manufacturing* 22 (2009), Juli, Nr. 2, 219–228. `http://dx.doi.org/10.1007/s10845-009-0283-9`. – DOI 10.1007/s10845–009–0283–9. – ISSN 0956–5515

[SSS+13] Schwarz, Christoph ; Schachmanow, Jurij ; Sauer, Jürgen ; Overmeyer, Ludger ; Ullmann, Georg: Coupling Of A Discrete Event Multi Agent Simulation With A Real Time Multi Agent Simulation. In: *Proceedings of the 11th International Industrial Simulation Conference*, 2013

[SSTW09] Sauer, Jürgen ; Schumann, René ; Timm, Ingo J. ; Wenzel, Sigrid: Planung und Simulation in logistischen Anwendungen. In: Fischer, Stefan (Hrsg.) ; Maehle, Erik (Hrsg.) ; Reischuk, Rüdiger (Hrsg.): *Informatik 2009: Im Fokus das Leben*. Lübeck : Gesellschaft für Informatik, 2009, S. 462–464

[Syc89] Sycara, K.: Multi-agent compromise via negotiation. In: *Distributed Artifical Intelligence II* (1989), S. 119–139

[Tra07] Trautmann, Andreas: Multiagentensysteme im Internet der Dinge — Konzepte und Realisierung. In: Bullinger, Hans-Jörg (Hrsg.) ; Hompel, Michael (Hrsg.): *Internet der Dinge*. Springer Berlin Heidelberg, 2007, S. 281–294

[Ull10] Ullrich, Günter: *Fahrerlose Transportsysteme: Eine Fibel - mit Praxisanwendungen - zur Technik - für die Planung (Fortschritte der Robotik)*. Vieweg+Teubner Verlag, 2010

[Ull11] Ullrich, G.: Quo vadis FTS? - Fahrerlose Transportsysteme in Vision und Wirklichkeit. In: *Hebezeuge Fördermittel* 10 (2011), S. 562–565

[VDB13] Viroli, Mirko ; Damiani, Ferruccio ; Beal, Jacob: A Calculus of Computational Fields. In: *ESOCC Workshops*, 2013, S. 114–128

[VHR+12] Ventz, K. ; Hachicha, M. B. ; Radosavac, M. ; Krühn, T. ; Overmeyer, L.: Aufbau hochfunktionaler Intralogistik-Knoten mittels kleinskaliger Module als Cognitive Conveyor. In: *In Proceedings of the 8. Fachkolloquium der Wissenschaftlichen Gesellschaft für Technische Logistik e.V. (WGTL)*, 2012, S. 19–36

[WHG06] Wilke, M. ; Heinecker, M. ; Günthner, W.A.: *Modulare Materialflusssysteme für wandelbare Fabrikstrukturen - Forschungsbericht*. TU München, 2006

[Wil13] Wildner, Christian: Robustheit – eine Anforderung an intralogistische Systeme. In: *Logistics Journal* 2013 (2013), Nr. 10

[Woo02] Wooldridge, Michael: Intelligent agents: The key concepts. In: *in Proceedings*

of the 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001 on Multi-Agent-Systems and Applications IISelected Revised Papers, ser. LNAI, Springer-Verlag, 2002, S. 3–43

[WS04] Windelinckx, N. ; Strens, M.: Dynamic scheduling of multiple agents for a heterogeneous task stream. In: *Fourth Symposium on Adaptive Agents and Multi-Agent Systems - AISB* (2004)

[Zwi04] Zwintzscher, Olaf: *Software-Komponenten im Überblick: Einführung, Klassifizierung und Vergleich von JavaBeans, .Net, EJBs, Corba, UML 2*. 1., Aufl. W3l, 2004. – ISBN 9783937137605