Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelorarbeit

Verhaltensbasierte Koordination eines Multi-Roboter-Szenarios mittels BDI-Agenten

Anja Richter

8richter@informatik.uni-hamburg.de
Studiengang Informatik
Matr.-Nr. 6052936

Erstbetreuer: Prof. Dr. Jianwei Zhang
Zweitbetreuer: Dr. Alexander Pokahr

Abstract

Most robot behaviours concentrate on single aspects such as path planning, collision avoidance or the grasping of physical objects as these specific tasks are very complex. This Bachelor thesis aims at modelling a top level behaviour for a multi robot system using software agents following the bdi model (belief-desire-intention). Its practical usage is demonstrated by the implementation of a logistic scenario. A multi agent system is modelled and then implemented by using the agent modelling framework Jadex. This MAS can be transferred to mobile robots an thus offers new application areas for multi robot systems.

Zusammenfassung

Verhaltensmodellierungen in der Robotik behandeln häufig reduzierte Szenarien wie Pfadplanung, Kollisionsvermeidung oder das Greifen eines Ojektes da diese Teilbereiche bereits sehr komplex sind. Diese Bachelorarbeit befasst sich mit der Modellierung eines Verhaltens für ein Multi-Roboter-System auf Basis von BDI-Agenten. Die praktische Umsetzung wird anhand eines logistischen Szenarios gezeigt. Es wird ein Multi-Agenten-System entworfen, und unter Verwendung des Agentenframeworks Jadex implementiert. Dieses MAS soll auf mobile Roboter übertragen werden und zeigt damit neue Anwendungsmöglichkeiten für Multi-Roboter-Systeme.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Roboter sind robuster und leistungsfähiger als Menschen und werden daher seit Jahrzehnten erfolgreich im industriellen Umfeld eingesetzt. Insbesondere bei Aufgaben, die eine hohe Präzision oder Kraft benötigen, sind sie dem Menschen deutlich überlegen und das bei wesentlich höherer Bearbeitungsgeschwindigkeit. Monotone Tätigkeiten zum Beispiel am Fließband, ermüden eine menschliche Arbeitskraft sehr stark. Ein Mensch kann also auch nicht für stetig gleiche Qualität und Geschwindigkeit garantieren. Daher wurden solche Sortier- oder Fertigungsaufgaben zunehmend von Industrierobotern übernommen und konnten Menschen von körperlich anstrengender und monotoner Arbeit entlasten.

Ganz anders verhält es sich mit Aufgaben, die anspruchsvolle kognitive Leistungen erfordern. Dabei gibt es gerade in dem Bereich viele Einsatzmöglichkeiten und Anwendungsbereiche, in denen der Einsatz von Robotern empfehlenswert wäre. Ein mögliches Anwendungsszenario ist der Einsatz von mobilen Robotern bei der Suche von Verletzten in Katastrophengebieten wie beispielsweise im Fall von Fukushima im Jahr 2011. Multi-Roboter-Systeme wären bestens dafür geeignet. Mit unterschiedlichen Sensoren und Aktuatoren ausgestattet, könnten sie verletzte Personen aufspüren und bergen und dadurch verhindern, dass sich menschliche Helfer bei einer Rettungsaktion selbst in Gefahr begeben. Zu diesem Zweck müssen Roboter allerdings nicht nur die technischen Voraussetzungen in Hinblick auf Mobilität, Akku- und Rechenleistung sowie geeignete Sensoren und Aktuatoren verfügen, sondern insbesondere in der Lage sein, sich autonom in dynamischen Umgebungen zurecht zu finden und selbstständig Lösungswege ermitteln können. Industrieroboter existieren seit ca. 1940 und werden seit den 70er Jahren auch in Deutschland in der Fertigungsindustrie eingesetzt. Die Leistungsfähigkeit der Roboter hat sich seither stetig verbessert und auch mobile Roboter verfügen über robuste und hoch performante Hardware.

Unabhängig davon führten die rasanten Fortschritte in der Halbleiterindustrie (Moore‘s Law) auch in der Softwaretechnik zur Entwicklung anspruchsvoller intelligenter Softwareprogramme.

Eine Entwicklungsrichtung beschreibt die Softwareagenten, welche bereits erfolgreich für Simulationen, automatisierte Verhandlungen oder Informationsrecherche zum Beispiel im Internet eingesetzt werden. Aufgrund der Heterogenität im Bereich der Robotik gab es bisher dennoch relativ wenig Versuche Multi-Agenten-Systeme auf mobile Roboter zu portieren. Seit 2011 gibt es das Framework RSAL, das am Fachbereich TAMS der Universität Hamburg entwickelt wurde und die Migration von Multi-Agenten-Systemen auf Multi-Roboter-Systeme unterstützt. Bisher wurden damit lediglich exemplarisch MAS-Systeme implementiert, welche Micro-Agenten des JADEX-Frameworks verwenden. Dies wird in dieser Arbeit zum Anlass genommen, ein Verhalten mittels BDI-Architektur zur Anwendung auf mobilen Multi-Roboter-Systemen zu entwickeln.

1.2 Zielsetzung

Schwerpunkt dieser Arbeit ist die Modellierung eines Verhaltens mittels BDI-Agenten für ein Multi-Roboter-System. Dieses soll anhand eines gängigen logistischen Szenarios evaluiert werden, welches in vielen Domänen der Industrie in dieser oder ähnlicher Form anzufinden ist.

2 Verwandte Arbeiten

Die vorliegende Arbeit verbindet die Wissenschaft Robotik mit der Agententechnologie aus dem Bereich der Verteilten Systeme. Daher werden an dieser Stelle zunächst beide Schwerpunkte gesondert behandelt.

2.1 Robotik

Nach einem Zitat von Mc Kerrow beinhaltet die Robotik

- *den Entwurf, die Herstellung, Regelung und Programmierung von Robotern*
- *die Verwendung bzw. den Nutzen von Robotern als Problemlöser*
- *die Untersuchung von Regelungsprozessen, Sensoren und Algorithmen bei Mensch, Tier und Maschine*
- *die Anwendung und Übertragbarkeit dieser Regelungsprozesse auf den Entwurf von Roboter*

[19]

Er trennt jedoch ganz klar die Konstruktion der Roboter von der Robotik als Wissenschaft.

Diese Arbeit beschäftigt sich mit dem Verhalten von Robotern und ist damit letzterem zuzuordnen. Auf die Konstruktion wird aus diesem Grund nicht näher eingegangen. Für die unterschiedlichsten Anwendungsbereiche sind standardisierte Roboter verfügbar, bei denen die Kinematik oder Multimodalität für bestimmte Zwecke optimiert ist. Da es in dieser Arbeit um eine Verhaltensmodellierung geht, werden vor allem Roboter benötigt, die sich relativ schnell fortbewegen können, um für Evaluierungszwecke eine schnelle Reaktionszeit zu erzwingen. Desweiteren müssen die Roboter über Sensoren für die Lokalisierung im Raum und über genügend Prozessorleistung verfügen, um diese anfallenden Daten verarbeiten zu können. Darüber hinaus muss die Plattform die Verhaltensarchitektur unterstützen, daher sollen Pioneer-Roboter verwendet werden, die im Kapitel 4.2 näher beschrieben werden.

Der aktuelle Forschungsstand in der Robotikwissenschaft wird regelmäßig auf Konferenzen diskutiert, hierbei zählen IROS [1] und ICRA [2] zu den größeren Konferenzen und bieten eine Grundlage zur Evaluierung vergleichbarer Arbeiten.

Bei den eingereichten Arbeiten der letzten 4 Jahre gab es hinsichtlich der Verhaltenssteuerung von Robotersystemen erkennbare Forschungsschwerpunkte bzgl. Selbstlokalisation und Pfadplanung. Dabei geht es weniger um neue Algorithmen, als um die Übertragbarkeit bewährter Ansätze auf neue Einsatzkontexte unterschiedlichster Robotersysteme, wie beispielsweise die Bildung von Formationen und der Lokalisierung im Schwarm oder der Navigation unbemannter Fahrzeuge und Flugobjekte (UAV). Viele Arbeiten behandeln das Greifen von Objekten im Raum. Roboter die in der Lage sind präzise ihre Bewegungen im Raum zu koordinieren, können zukünftig viele neue Einsatzmöglichkeiten im menschlichen Umfeld schaffen.

In dem Betrachtungszeitraum gab es keine erkennbaren Arbeiten, die sich mit Multi-Roboter-Verhalten auf einer höheren Abstraktionsebene beschäftigen.

Dies kann daran liegen, dass nach wie vor die Notwendigkeit an der Optimierung komplexer atomarer Verhalten, wie dem Greifen oder Erkennen von Objekten, der Pfadplanung oder Kollisionsvermeidung besteht. Die Integration dieser Teilbereiche zu einem funktionierenden Gesamtsystem ist in der Regel nicht trivial. Zwar gibt es bereits humanoide Roboter, die in der Lage sind Fussball zu spielen oder auf Zuruf Gegenstände zu holen und ähnliche Aufgaben zu erfüllen, allerdings können diese noch nicht das Optimum aller Teildisziplinen in sich vereinen.
Zwei Beispiele:

- Ein 6-gelenkiger Industrieroboterarm verfügt über genügend Freiheitsgrade für präzises Greifen, sobald jedoch 2 Arme an einem Körper befestigt sind und sich in ihrem Arbeitsraum überlappen, erhöht sich die Komplexität des Greifens enorm, da eine Kollision beider Arme vermieden werden muss und es sich bei dem jeweils anderen Arm um ein dynamisches Hindernis handelt.
- Ein weiteres Beispiel ist der humanoide Gang. Der Forschungswettbewerb RoboCup[3] oder auch Präsentationen von Herstellern mobiler Roboter, wie zum Beispiel Honda[4], lassen eine stetige Entwicklung in dem Bereich erkennen. Dennoch ist diese Fortbe-

wegungsart noch sehr langsam, da die Stabilisierungsmöglichkeiten mit Hilfe entsprechender Sensoren und Regler nicht ausgereift sind. Aus diesem Grund sind die Roboter meist nicht sehr hoch und die Knie sind beim Gehen weit gebeugt, um den Schwerpunkt entsprechend zu verlagern.

Ein weiterer Grund für den Fokus auf Low-Level Verhalten kann auch darauf zurück geführt werden, dass sich im Bereich der Robotik bisher kaum Standards für Betriebssysteme durchgesetzt haben und daher neben den ohnehin komplexen Lösungsansätzen auf der Meta-Ebende auch die Treiberkonfiguration bzw. der Zugriff auf Sensoren und Aktoren jeweils neu zu realisieren ist. Mit den beiden bekanntesten Vertretern ROS [5] und Player/Stage [16], welches mittlerweile in ROS integriert wurde, werden zwar Versuche unternommen, Standards zur Verfügung zu stellen, diese werden jedoch nicht von allen Roboterherstellern verwendet.
Die Pioneer-Roboter, welche in dieser Arbeit zum Einsatz kommen sollen, verwenden die Middleware Player/Stage.

Player / Stage ist ein Software-Projekt von Brian Gerkey, Richard T. Vaughan und Andrew Howard [16], welches die Entwicklung von Software für Multi-Roboter-Plattformen beschleunigen und vereinfachen soll. Es beinhaltet zwei Module, die separat verwendet werden können. Player stellt eine klar definierte und einfache Schnittstelle zum Ansprechen der Sensoren und Aktuatoren vieler gängiger Roboter zur Verfügung und bietet mit Stage eine Simulationsumgebung an, mit deren Hilfe der reale Einsatz gefahrlos getestet werden kann. Stage ist ein umfassend konfigurierbarer Roboter-Simulator in 2,5 D, der viele gängige, in Lehre und Forschung benutzten, Robotertypen unterstützt.

Fazit

Forschungsschwerpunkte der Robotik hinsichtlich Verhaltenssteuerung bewegen sich überwiegend auf der Ebene von atomaren Verhalten, die eine Basis für komplexe Multi-Roboter Szenarien darstellen. Es gibt eine zunehmende Entwicklung von Standards für die Steuerungssoftware oder Middleware von Robotersystemen, wie ROS oder Player/Stage. Damit werden bereits Versuche unternommen, Verhalten unabhängig von der darunter liegenden Hardware modellieren und anschließend auf „beliebigen“ Robotersystemen

auszuführen zu können.

Die Modellierung von intelligentem Verhalten wird häufig getrennt von der Robotik in anderen Wissenschaftszweigen behandelt. Ausgehend von der Frage, wodurch Intelligenz gekennzeichnet ist, hin zu der Frage der Erschaffung einer künstlichen Intelligenz. Zur Eingrenzung dieser Arbeit, wird nachfolgend nur auf den Bereich der Agententechnologie Bezug genommen.

2.2 Agententechnologie

Die Verhaltensmodellierung eines MRS beinhaltet trotz Abstraktion von der Hardware hohe Anforderungen. Die einzelnen Roboter benötigen eine Repräsentation des Weltmodells und Kenntnis über ihre Ziele und möglichen Lösungswege. Desweiteren muss eine Rückkoppelung über die Auswirkungen ihres Handelns bestehen, um die Zielerreichung zu evaluieren. Da Roboter in einer realen Welt agieren, sollten sie möglichst reaktiv sein, um sich angemessen in der dynamischen Umgebung zu verhalten. Darüber hinaus muss die Kommunikation der Roboter untereinander gewährleistet sein, um gemeinschaftliches Verhalten zu koordinieren.
Diese Forderungen eines sowohl proaktiven als auch reaktiven Verhaltens führten unter anderem zu der Entscheidung, das MRS-Verhalten durch Agenten zu realisieren, siehe auch Kapitel 3.2.2.

Insbesondere die Verwendung von BDI-Agenten soll eine menschenähnliche Verhaltensstruktur schaffen, bei der ein Agent eigene Ziele und Absichten verfolgt und dabei stets Kenntnis über seine Umgebung und Handlungsmöglichkeiten besitzt. Die Aktualisierung der Wissensbasis und der Auswahlprozess geeigneter Aktionen sind nicht trivial. Im Bereich der Agententechnologie und Verteilter Systeme wurden in den letzten Jahrzehnten Konstrukte geschaffen, welche die Entwicklung von Agenten in unterschiedlichem Maße unterstützen. Nachfolgend werden die bekannteren Vertreter vorgestellt:

JACK ist eine in Java implementierte Software zur Modellierung von BDI-Agenten für den kommerziellen Einsatz. Es verfügt über eine eigene Entwicklungsumgebung. Aufgrund der Verwendung von Java können die Agenten auf einer Vielzahl verschiedener Plattformen ausgeführt werden [6]. Jack kommt für die Verwendung in dieser Bachelorarbeit nicht in Frage, da es kein frei verfügbares Produkt ist.

JASON ist ein Interpreter basierend auf AgentSpeak [23] und bietet eine frei verfügbare Plattform zur Implementierung von Multi-Agenten-Systemen mit grafischen Entwicklungswerkzeugen an [7]. Die zugrunde liegende Logik-Sprache AgentSpeak wurde 1996 von Anand S. Rao zur Implementierung von BDI-Agenten entwickelt und folgt dem BDI-Konzept. Aufgrund fehlender Plattformunabhängigkeit ist die Verwendung dieser Sprache zur Anwendung auf einem MRS nicht empfehlenswert.

JADE Das Java Agent Development Framework ist eine quelloffene Middleware für agenten-basierte Peer-to-Peer-Anwendungen. Ein Schwerpunkt von JADE ist die Unterstützung des FIPA Standards [8] für die Kommunikation der Agenten untereinander. Es fehlt jedoch eine explizite Abbildung der drei Säulen der BDI-Architektur.

JADEX wurde 2003 an der Universität Hamburg von Lars Braubach und Alexander Pokahr entwickelt und ermöglicht die Implementation von BDI-Agenten unter der Verwendung von Java und XML und unterstützt demnach ebenfalls eine objektorientierte Herangehensweise. Jadex basiert auf dem zuvor genannten Framework JADE und bietet eine API und eine grafische Oberfläche, welche die Entwicklung eines MAS, unter anderem durch geeignete Debugging-Tools, erleichtert [9].

2.3 Schnittstelle zwischen MAS und Robotersystem

Multiagentensysteme (MAS) sind ein Vertreter von Verteilten Systemen.
Darunter versteht man ein System, dessen Hardware- oder Softwarekomponenten auf vernetzten Rechnern verteilt sind und über Nachrichtenaustausch kommunizieren und ihre Aktionen koordinieren [18].

Die Migration des MAS auf die Roboter ist in der Regel nicht trivial. Dies liegt vor allem an der Heterogenität der Software- und Hardwarekomponenten und dem Mangel an Standards der zugrunde liegenden Softwarearchitekturen zum Ansprechen der verschiedenartigen Sensoren und Aktoren.
Für die Integration des MAS auf die Roboterplattform soll daher das Framework RSAL (Robot System Abstraction Layer)[25] als Schnittstelle dienen.

Es bildet eine Abstraktionsschicht für die Verwendung von Jadex und ermöglicht nicht nur den Zugriff auf das angeschlossene MRS sondern bietet darüber hinaus Basisverhal-

ten, wie Pfadplanung und Kollisionsvermeidung an. Dadurch kann sich der Programmierer ausschließlich auf die Modellierung und den Entwurf und des Verhaltens konzentrieren und bei der Implementation auf diese Services zugreifen.

RSAL wurde bereits für die Implementierung von Micro-Agenten in JADEX und einem MRS mit Pioneer Robotern und Player/Stage praktisch erprobt, kann jedoch mit geringfügigen Anpassungen auch für andere MRS (z. Bsp. ROS) verwendet werden.

Zusammenfassung

In dieser Arbeit wird die Modellierung eines MRS-Verhaltens mittels BDI-Agenten untersucht. Diese Aufgabe verknüpft die Robotik mit dem Bereich der Verteilten Systeme, insbesondere der Agententechnologie.

Auf der unteren Ebene befindet sich das MRS, das heißt radgetriebene Roboter vom Typ Pioneer und die Middleware Player/Stage. Es werden radgetriebene Roboter verwendet, da sie sich ausreichend schnell fortbewegen können um die Verhaltensmodellierung zu testen. Sie verfügen über geeignete Sensoren und Aktoren und die notwendige Prozessorleistung zur Berechnung der Daten. Die Gewährleistung erfolgt durch die geeignete Wahl eines Notebooks, welches am Pioneer angebracht wird und diesen steuert.

Das Verhalten wird durch Multiagenten, konkret BDI-Agenten, mit dem Agenten-Framework Jadex realisiert. Jadex erfüllt die notwendigen Voraussetzungen für die Umsetzung der Aufgabenstellung, da es unter anderem die Elemente des BDI-Konzepts, Belief, Desire und Intention sehr gut unterstützt und dank der Verwendung von Java weitestgehend plattformunabhängig ist. Ein weiterer Vorteil ist, dass bereits eine Erweiterung (RSAL) für die Übertragung des MAS-Verhaltens auf die Roboter existiert. RSAL enthält bereits Grundfunktionen zur Kollisionsvermeidung und Pfadplanung.

Daraus ergibt sich die Möglichkeit der Abstraktion von der Roboterhardware, sodass die Implementation der Verhaltensmodellierung direkt auf einer höheren Ebene umgesetzt werden kann.

3 Verhaltensarchitekturen für Roboter

3.1 Notwendigkeit komplexer Roboterverhalten

Es gibt etliche Anwendungsbereiche, in denen eine Automatisierung diskutiert werden kann. Dabei sind stets zwei Fragestellungen zu berücksichtigen: Ist eine Automatisierung technisch umsetzbar und falls ja, ist sie auch sinnvoll?

Die Arbeitsfelder und Tätigkeitsbereiche von Menschen sind überwiegend formalisiert. Das heißt, dass sämtliche Arbeitsabläufe so strukturiert werden, dass sich möglichst viele standardisierte Handlungsabläufe ergeben. Dieser Vorgang ist die notwendige Basis für eine mögliche Automatisierung. Ob dessen Umsetzung sinnvoll ist, ist abhängig von der auszuführenden Tätigkeit und der daran gemessenen Leistungsfähigkeit von Robotersystem und Mensch.
Die Ausführung durch Roboter ist in der Regel schneller, präziser und es kann ein größerer Kraftaufwand betrieben werden im Vergleich zur menschlichen Arbeitsleistung. Dies ist natürlich stark vom Einsatzkontext abhängig und nicht immer wird der gewünschte Erfüllungsgrad erreicht. Dennoch kann häufig eine Optimierung erzielt werden, da Roboter aufgrund einer höheren Materialbeständigkeit meist zuverlässiger und robuster sind und Arbeitsgänge mit gleichbleibender Genauigkeit vollziehen, was einem Unternehmen eine gewisse Planungssicherheit bietet. Diese Aussagen gelten nur bedingt, da dies stark durch die Qualität der verbauten Teile beeinflusst wird. Dennoch ist ein zeitnaher Ersatz in der Regel nur durch einen finanziellen Rahmen beschränkt.
Diese vorteilhaften Eigenschaften werden nutzbar gemacht, indem Roboter bei vielen Aufgaben assistierend wirken und Defizite des Menschen ausgleichen. Beispiele gibt es unter anderem in der Chirurgie. Dort werden Roboter eingesetzt, um beispielsweise anhand der Daten von Voruntersuchungen präzise Löcher am Kopf vorzubohren. In der Fertigung sollen sie besonders schwere wie auch kleinste Bauteile mit hoher Genauigkeit zusammensetzen. Aufgrund der Fortschritte und Erfahrungen in der Robotik gibt es zunehmend eine Verschiebung bei dieser Arbeitsteilung von Mensch und Roboter dahingehend, dass der Mensch dem Roboter assistiert bzw. nur noch eine Überwachungsfunktion

einnimmt. Auch hier gibt es praktische Beispiele wie der Fahrroboter von „Google“[10] der bereits erfolgreich autonom ein Auto im öffentlichen Straßenverkehr lenkt. Während der bisher durchgeführten Fahrten saß stets ein Mensch im Fahrzeug, um in Notsituationen einzugreifen. Auch in der Chirurgie gibt es Bestrahlungsverfahren, bei denen ein Programm zuvor einen Bestrahlungsplan berechnet und anschließend ein Roboter autonom die Bestrahlung durchführt und dabei sowohl das Bestrahlungsgerät, als auch die Position des Operationstisches steuert und somit Lageveränderungen des Patienten ausgleicht um punktgenaue Bestrahlungen vorzunehmen. Ein solcher Vorgang wird lediglich von einem Mediziner im Nebenraum überwacht. Dennoch ist bei all diesen Fortschritten und modernen Einsatzmöglichkeiten eines erkennbar: Es handelt sich stets um feste Verhaltensmuster. Sowohl der Chirurgie- als auch der Fahrroboter vollziehen eine Pfadplanung. Wie bereits erwähnt handelt es sich dabei um eine komplexe Aufgabenstellung, in diesen Fällen sogar mit dynamischen Hindernissen.
Dennoch ist erkennbar, dass der nächste notwendige Sprung in der Robotikforschung darin liegt, Roboter mit der Fähigkeit auszustatten, komplexere Aufgaben in einer dynamischen Umgebung lösen zu können.

Dabei geht es insbesondere um die Fähigkeit autonomer Roboter, folgende Situationen zu erkennen und darauf sinnvoll zu reagieren:

- Eine geplante Aktionen kann durch Veränderungen in der Umgebung nicht mehr ausgeführt werden.
- Es stellt sich plötzlich eine Situation ein, welche die unmittelbare Ausführung eines anderen Plans zur Folge hat.
- Eine vollzogene Aktion hat nicht zur gewünschten Veränderung geführt.
- Ein Ziel kann nur durch das Zusammenwirken verschiedener Aktionen erreicht werden.

Was der Roboter benötigt ist also eine Form von Intelligenz, die es ihm ermöglicht angemessen auf eine dynamische Umgebung zu reagieren und komplexe Ziele zu erreichen.

Fazit

Um Roboter für komplexe Aufgaben autonom einsetzen zu können, kann man an zwei Punkten ansetzen: Zum einen ist die Verwendung eines Multi-Roboter-Systems (MRS) zu nennen.
Ein MRS in dem Roboter mit redundanten Eigenschaften eingesetzt werden, kann einem Ausfall des Systems entgegenwirken, in dem die Aufgabe des ausgefallenen Roboters auf einen anderen übertragen wird. Der Einsatz eines heterogenen MRS bietet darüber hinaus die Möglichkeit durch koordiniertes Handeln gemeinsam Aufgaben zu lösen, zu denen einzelne Roboter nicht in der Lage sind und somit Hardware Defizite (unterschiedliche Endeffektoren oder Sensoren) ausgleichen. In einem Verteilten System kann auch Wissen über die Domäne ausgetauscht werden um unter anderem Messwerte von Sensoren zu verifizieren.

Der zweite Ansatzpunkt ist die Implementierung einer Intelligenz, die sich die Möglichkeiten eines MRS zunutze macht und jeden einzelnen Roboter innerhalb einer dynamischen Umgebung handlungsfähig hält.

3.2 Softwarearchitekturen

Im vorherigen Kapitel wurde festgestellt, dass das MRS Ziele verfolgen und dabei stets auf wichtige Änderungen in der Umgebung geeignet reagieren sollte. Da in einer dynamischen Umgebung unvorhersehbare Situationen eintreten können, die das Ausführen von geplanten Aktionen verhindern können, sollte ein Roboter in der Lage sein, alternative Lösungen zur Zielerreichung in Erwägung zu ziehen.
Diese Aussagen implizieren bereits eine bestimmte Form der Softwarearchitektur, unter anderem dadurch, dass dem Roboter eine Zielverfolgung unterstellt wird. Im diesem Kapitel wird nun ein Software-Paradigma vorgestellt, das den zuvor postulierten Anforderungen grundsätzlich gerecht werden kann.

3.2.1 Agenten

Die Anforderungen an das MRS, die in den vorhergehenden Abschnitten formuliert wurden, sind in einem anderen Kontext bereits vor einigen Jahren in der Softwaretechnik und

im Bereich Verteilter Systeme diskutiert worden. Das Internet und die zunehmende Verfügbarkeit kostengünstiger, leistungsfähiger mobiler Endgeräte prägten in den letzten 10 Jahren einen neuen Begriff, „ubiquitous computing“, der das zunehmende Eindringen von Computersystemen im Alltag der Menschen bezeichnet. Die steigende Komplexität von Software und der Bedarf an allgegenwärtigen, mobilen Computersystemen, die im ständigen Austausch miteinander stehen, brachte die Notwendigkeit eines neuen Software-Paradigmas hervor.

Laut Wooldridge kann man Programme entweder als eine Ansammlung überwiegend passiver Objekte verstehen, die jeweils einen Zustand besitzen, der über Operationen an den Objekten verändert werden kann, oder aber als interagierende semi-autonome Agenten [27].
Wooldridge sieht eine Notwendigkeit in der Erforschung des Agentenansatzes um das Potential verteilter Systeme besser nutzen und ausschöpfen zu können.

Auf der Suche nach einer klaren Begriffsdefinition finden sich in der Literatur verschiedene Definitionen von Softwareagenten, die sich jedoch alle in ihrer Kernaussage ähneln und auf einige grundlegende Eigenschaften reduzieren lassen. Eine treffende Definition für den Agentenbegriff liefern Wooldridge und Jennings:

> *... ein auf Hardware oder (häufiger) auf Software basierendes Computersystem, das über die folgenden Eigenschaften verfügt:*
>
> **Autonomie:** *Agenten operieren ohne direkte menschliche oder andere Einwirkungen und haben eine gewisse Kontrolle über ihre Handlungen und ihren inneren Zustand.*
>
> **Soziale Fähigkeiten:** *Agenten stehen in einer Beziehung mit anderen Agenten (und vielleicht mit Menschen) mittels einer Art Agenten-Sprache.*
>
> **Reaktivität:** *Agenten nehmen ihre Umwelt (welche eine physikalische Welt, ein Benutzer über eine grafische Benutzeroberfläche, eine Reihe anderer Agenten, das Internet oder eine Kombination dieser Möglichkeiten sein kann) wahr und reagieren rechtzeitig auf Veränderungen.*

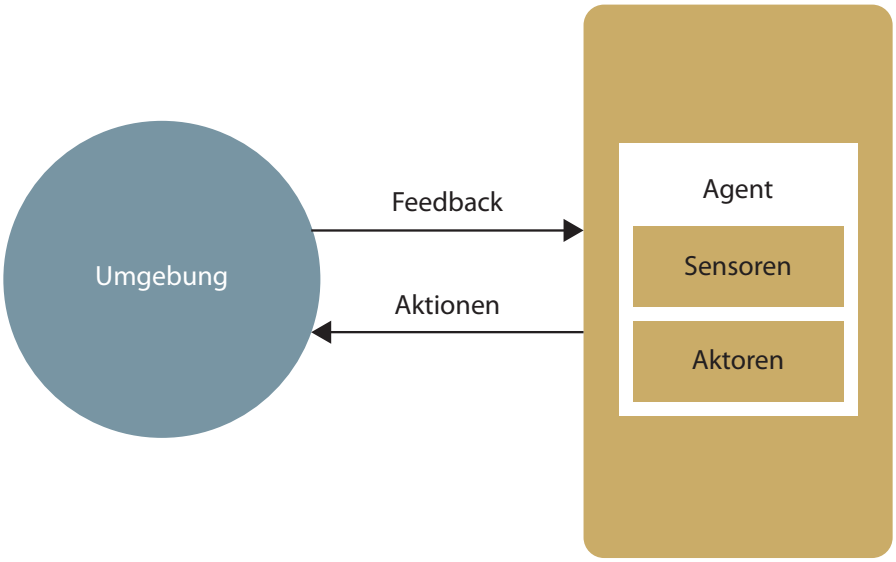

Abbildung 3.1: Diese Abbildung zeigt das Modell eines Agenten. Der Agent interagiert mit seiner Umwelt, indem er mittels seiner Sensoren Veränderungen in der Umgebung wahrnimmt und die Auswirkungen seiner Handlung erfasst. Mittels seiner Aktoren wird absichtlich oder unabsichtlich auf die Umgebung Einfluß genommen. Im Falle reiner Softwareagenten wirken Eingangsparameter und Ausgangswerte von Funktionen analog auf die Laufzeitumgebung des Agenten.

> **Proaktivität:** *Agenten reagieren nicht nur auf ihre Umgebung, sondern sind auch zu zielgerichtetem Verhalten fähig, indem sie Initiativen ergreifen.*
>
> [24]

Die Umgebungen mit denen Agenten interagieren sind häufig Software-Umgebungen, können laut [27] aber auch physikalischer Art sein.
Insofern kann auch das Verhalten von Robotern durch Agenten realisiert werden.
In diesem Fall wird die Umgebung wesentlich dynamischer und die Verwendung analoger Sensoren und Aktoren, zur Interaktion mit der Umwelt, erhöht die Störanfälligkeit des Systems und damit die Wahrscheinlichkeit eines Systemausfalls. In einem MRS kann der Ausfall einzelner Agenten jedoch vorübergehend durch andere Roboter aufgefangen werden.

BDI-Agenten

Die Begriffsdefinition von Agenten legt zunächst Eigenschaften fest, die einen Agenten kennzeichnen. Dies ist noch keine Konkretisierung, wie ein Agent modelliert wer-

den kann. Eine der bekanntesten Möglichkeiten dieses Paradigma umzusetzen sind die BDI-Agenten. Ihr Erfolg geht darauf zurück, dass sie den Gedanken, den menschlichen Entscheidungsfindungsprozess möglichst authentisch abzubilden, sehr direkt umsetzen.
Das BDI Modell geht ursprünglich auf Michael Bratman [14] zurück, einem Psychologie-Professor, der sich mit der Entscheidungsfindung beim Menschen beschäftigte und im Jahr 1987 dieses Modell postulierte.
Der menschliche Entscheidungsfindungsprozess orientiert sich zunächst an individuellen Zielen. In Abhängigkeit der gegebenen Umstände werden Pläne/ Intentionen identifiziert, die zur Erreichung des Gesamtziels notwendig sind. Häufig gibt es hierfür mehrere Möglichkeiten, die einen unterschiedlich hohen Aufwand und Risiko bergen. Bei der Auswahl der auszuführenden Pläne bezieht ein Mensch normalerweise auch die Folgen seines Handels in die Entscheidung ein.

BDI-Agenten basieren auf den drei Komponenten *Belief*, *Desire* und *Intention*.
Unter *Belief* versteht man alle relevanten Fakten einer Domäne. Sie ergeben in ihrer Gesamtheit ein Weltmodell, das fortlaufend aufgrund eingehender Sensordaten validiert oder angepasst wird.
Desires sind die eigentlichen Ziele oder Zustände, die ein Agent erreichen möchte. Wie bereits erwähnt kann man einen gewünschten Zustand häufig auf verschiedene Weise erreichen und es sind auch meist mehrere Teilaufgaben für die Zielerreichung zu bewältigen. Diese Teilaufgaben werden in diesem Kontext auch *Intentions* genannt. Diese Intentionen sind eng mit konkret ausführbaren Aktionen verbunden. Sie können darüber hinaus aber auch Vorbedingungen für eine Planausführung beinhalten. Agenten kennen also ausführbare Lösungsvorschläge für Teilprobleme.

Multi-Agenten-Systeme

Multi-Agenten-Systeme (MAS) ermöglichen das kooperative Verhalten zum gemeinsamen Lösen einer komplexen Aufgabe. Alle Agenten in einem MAS besitzen das gleiche primäre Ziel.

Eine explizite Begriffsdefinition für MAS lässt sich nicht finden. Das liegt unter anderem daran, dass für den darin enthaltenen Agentenbegriff keine eindeutige Definition existiert. Dennoch lässt sich auch hier eine Art Konsens in der Literatur finden. Ein Multiagentensystem besteht demzufolge aus mehreren Agenten, die in der selben Umgebung

agieren und miteinander kommunizieren [27].
In [21] wird diese Aussage konkretisiert und Merkmale genannt, die in einem MAS gelten sollten:

Eigenschaften eines MAS
- Jeder Agent besitzt eine eigene eingeschränkte Weltsicht. Diese äußert sich zum Beispiel in unvollständigen Informationen und Problemlösefähigkeiten.
- Es existiert keine globale Systemkontrolle.
- Die Berechnung erfolgt asynchron.
- Die Daten sind dezentral im System verteilt.

Betrachtet man diese Eigenschaften so stellt man fest, dass es im Kern darum geht, dass auch in einem MAS jeder einzelne Agent autonom handelt. Da dennoch ein gemeinsames Ziel erreicht werden soll, stellt sich die Frage der Kommunikation und Interaktion zwischen den Agenten eines MAS.

Braubach [15] nimmt darauf Bezug und erklärt, dass die Interaktionsformen grundlegend in zwei Arten aufgeteilt werden können, in direkte Kommunikation und indirekte Kommunikation.
Indirekte Kommunikation entsteht laut Braubach dadurch, dass ein Agent durch sein Handeln die Umgebung verändert. Durch diese wahrnehmbare Veränderung werden andere Agenten im System zu einer entsprechenden Reaktion veranlasst.
Eine direkte Kommunikation beinhaltet den direkten Austausch von Daten zwischen Agenten.
Auf Sprechakttheorie (Protokolle, Agent Communication Languages, usw.) wird in dieser Arbeit nicht näher eingegangen. Weitere Informationen finden sich in Wooldridge [27].

Zusammenfassung

Das Agentenparadigma entspricht den Anforderungen die zuvor für die Roboter formuliert wurden. Es stellt sich die Frage, wie dieses Paradigma softwaretechnisch realisiert werden kann. Damit befasst sich der nachfolgende Abschnitt.

3.2.2 Steuerungsarchitekturen

Roboterverhalten bestehen aus den drei Basisfunktionen *sense*, *plan* und *act*. Die Kombination der Ausführungsreihenfolge dieser Basisfunktionen beschreibt jeweils eine spezielle Art der Steuerungsarchitektur.
Murphy [20] unterscheidet drei Architekturansätze für intelligente Roboter. In der Anfangszeit wurde Verhalten vorwiegend hierarchisch modelliert. Mit steigenden Hardwarevoraussetzungen hat man sich jedoch zunehmend davon entfernt. Einen großen Fortschritt beschrieben reaktive Architekturen, die dadurch gekennzeichnet sind, dass sie spontan jeden Eingangswert auf eine Aktion abbilden, ohne eine Planauswahl zu vollziehen. Der hierarchische und reaktive Ansatz unterscheiden sich grundlegend und haben entsprechend stark ausgeprägte Vor- und Nachteile. Um die Vorteile zu vereinen und den Grad an Intelligenz zu verstärken, wurden unterschiedliche hybride Architekturen entwickelt. Hier ist auch der Agentenansatz verortet, er basiert auf eine Erweiterung der deliberativen Architekturen.

Hierarchische/ Deliberative Architekturen

Der hierarchische Ansatz findet seit ca. 1967 Anwendung und ist damit der älteste der vorgestellten Architekturen. Er ist durch die feste Abfolge der „Zustände“ *Sense*, *Plan* und *Act* gekennzeichnet. Das bedeutet, der Agent bzw. Roboter untersucht zunächst die Umgebung und erfasst alle relevanten Fakten (*Sense*). Auf dieser Basis wird nachfolgend ein Plan generiert (*Plan*) und ausgeführt (*Act*). Da ein Abweichen von dieser Reihenfolge oder eine nebenläufige Ausführung nicht vorgesehen sind, birgt dieser Ansatz gleich mehrere Nachteile. Beim Einsatz in einer dynamischen Umgebung besteht eine erhöhte Wahrscheinlichkeit, dass während der „Plan-“ oder „Act-“ Phase Veränderungen eintreten, die eine Unterbrechnung der aktuellen Planausführung erfordern. Das sequenzielle Durchlaufen der drei Zustände verhindert jedoch eine entsprechende Reaktion. Dies führt zu unerwünschtem Verhalten oder groben Fehlern insbesondere in einer realen Umgebung. Zur Verdeutlichung ein kleines Beispiel:

Soll sich ein Roboter kollisionsfrei zu einem Punkt in einem Raum bewegen, dann scannt er zunächst die Umgebung und überprüft, ob in seinem Bewegungsraum Hindernisse vorhanden sind (sense). Steht nach Auswertung der Sensordaten fest, dass keine Hindernisse vorhanden sind, dann wählt der Roboter den entsprechenden Plan aus. Hier zum Beispiel: „Fahre 2 Meter nach vorn.". Wenn nun während der Ausführungsphase

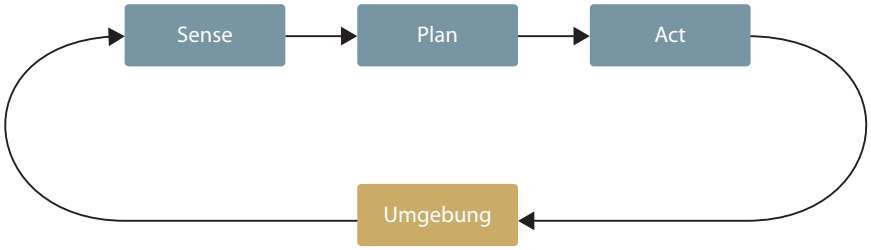

Abbildung 3.2: Hierarchische Architektur von Softwareagenten

des Plans (act), also während der Roboter fährt, ein anderer mobiler Roboter den Weg kreuzt, kommt es zwangsläufig zu einer Kollision, weil der Roboter während seiner Ausführungsphase keine Veränderungen der Umgebung wahrnimmt und demzufolge auch keinen Kollisionsvermeidungsplan ausführen wird.
Dieser Nachteil kann zwar durch ein regelmäßiges „Replan“ ausgeglichen werden, allerdings ist diese Variante sehr Ressourcen aufwendig und lässt das System so träge werden, dass dessen praktische Anwendung nicht tragbar ist.

Diese ungünstige Eigenschaft ist mitverantwortlich dafür, dass der hierarchische Ansatz vermehrt durch die, um 1980 aufkommenden, reaktiven Paradigmen verdrängt wurde.

Planungsstruktur

Die Planungsstruktur beschreibt den Prozess der Planauswahl (*Plan*). Jede Form von Planen basiert auf Fakten oder Annahmen über die Welt. Die hierarchischen Architekturen werden meist mit einer deliberativen Planungsstruktur umgesetzt. In diesem Fall besitzt der Roboter eine explizite Darstellung des Weltmodells. Dabei repräsentieren Symbolmuster die relevanten Fakten der Domäne und ermöglichen so eine Kategorisierung. Auf diesen Mustern können dann Operationen ausgeführt werden um einen ausführbaren Plan zu generieren.

Es folgt ein einfaches, natürlichsprachliches Beispiel:

Datenbasis
Eine Dogge ist ein Hund.
Alle Hunde haben 4 Beine.
...
Schlussfolgerung:
Eine Dogge hat 4 Beine.

Das Schlussfolgern folgt hier logischen Regeln auf Basis der vorhandenen Daten. Sollte die Wissensbasis fehlerhafte Informationen enthalten, werden demzufolge auch die resultierenden Aktionen des Roboters nicht angemessen sein.
Schwierigkeiten bei der Implementation sieht Wooldridge [27] vor allem darin, die Informationen der Umgebung in eine geeignete symbolische Repräsentation zu übersetzen und darin, diese Symbole adäquat und zeitnah zu manipulieren.

Die Wissensbasis wird nach jeder *Sense*-Phase validiert und bei Bedarf angepasst. Aufgrund der festen hierarchischen Abfolge werden in jeder Phase alle Fakten auf Veränderungen überprüft, was sich in der hohen Laufzeitkomplexität dieser Architektur niederschlägt. Einzelne Fakten, die häufige Änderungen erwarten lassen, können nicht nach Bedarf überprüft werden, was ein reaktives Verhalten unmöglich macht.

Ist die Umgebung jedoch sehr statisch und vorab bekannt, kann mit dem hierarchischen Ansatz mit relativ geringem Programmieraufwand ein funktionsfähiges Verhalten modelliert werden. In der Blütezeit der hierarchischen Architekturen von 1967 bis ca. 1980 war die Entwicklung in der Halbleiterindustrie noch in einem Stadium, welches die Verwendung dieses Architekturansatzes begründet. Heutzutage gibt es allerdings Multikern-Prozessoren und zahlreiche Frameworks zur Unterstützung von nebenläufigen Architekturen. Damit lässt sich ein wesentlich reaktiveres Verhalten implementieren.

Diese Form der Umsetzung einer Planungskomponente ist auch bei reinen Softwareagenten anzutreffen. Hierbei werden häufig atomare Formeln der Aussagenlogik als Repräsentanten einzelner Fakten gewählt. Daher eignen sich deliberative Agenten unter anderem sehr gut zur Beweisführung von Theoremen.

Reaktive Architekturen

Die Reaktiven Architekturen basieren auf den von Brooks [17] formulierten Ansatz „Reasoning without representation“. Er unternimmt den Versuch, autonome mobile Roboter zu modellieren, die ihre Umgebung wahrnehmen und sowohl spontan als auch angemessen auf Veränderungen darin reagieren ohne jedoch eine interne Repräsentation, also ohne ein Weltmodell, zu besitzen. Dies entspricht in etwa Automatismen beim Menschen, die vom Unterbewusstsein gesteuert werden wie beispielsweise ein Aufschrei in einer Schreckssituation.

Reaktives Verhalten bedeutet also spontan auf Veränderungen in der Umwelt zu reagieren. Dies geschieht sehr direkt, indem der Eingangswert unmittelbar auf eine Aktion abgebildet wird, ohne mögliche Folgen dieser Aktion in die Entscheidung einzubeziehen. Es werden keine Alternativen erwogen. Solch ein System kann relativ einfach ohne großen Aufwand implementiert werden.

$Agent : E \rightarrow A$ | E: Environment, A: Action

Wenn diese Abbildung in einem indizierten Array gespeichert wird, liegt die Laufzeitkomplexität für die Planauswahl bei O(1). Allerdings eignet sich diese Architektur nicht für sehr komplexe Szenarien und kann beim Einsatz in der realen Welt zu gravierenden Fehlern führen. Die 1:1 Abbildung enthält keine Alternativen aus denen Pläne nach Abwägen aller Fakten ausgewählt werden. Daher führen unvorhersehbare Veränderungen häufig zu unangemessenem Verhalten, zum Beispiel genau dann, wenn für ein Ereignis kein geeigneter Output hinterlegt ist. Aufgrund der schnellen Reaktionszeit wird dieser Ansatz häufig für einfache Teilprobleme einer Domäne verwendet, die eine schnelle Reaktionszeit benötigen. Einer der bekanntesten Vertreter der reaktiven Architekturen ist die Subsumption-Architektur.

Hybride deliberative/ reaktive Architekturen

Hybride Architekturen basieren grundsätzlich auf dem reaktiven Ansatz. Die schnelle Reaktionszeit dieser Architektur wollte man beibehalten, sie jedoch eine Planungskomponente ergänzen, um komplexe Problemstellungen autonom von Robotern behandeln lassen zu können. Kern dieser Architektur ist die Feststellung, dass nicht alle Teilauf-

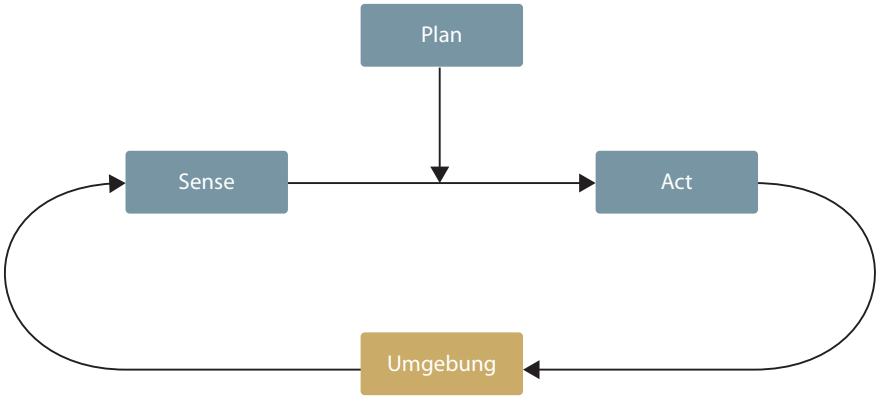

Abbildung 3.3: Hybride deliberative/ reaktive Architektur von Softwareagenten

gaben zur Erreichung eines Ziels ein explizites Planen erfordern. In realen Anwendungsszenarien kann situationsbedingt die Fähigkeit zu einer unmittelbaren Reaktion ebenso wichtig sein, wie die Auswahl eines geeigneten Plans.

Die Planungskomponente beinhaltet die Zerlegung der gegebenen Problemstellung in Teilaufgaben und identifiziert ausführbare Pläne zur Bewältigung dieser Teilaufgaben [20]. Sollte aufgrund der erfassten Sensordaten/ Eingangswerte ein gegebenes Teilproblem identifiziert werden, wird unmittelbar der dafür hinterlegte Plan ausgeführt.
Die Eingangswerte werden also separat an die reaktive Komponente und an die Planungskomponente weitergereicht. Die Planungskomponente aktualisiert daraufhin das Weltmodell und generiert ggf. neue Ziele und ausführbare Pläne.

Das Erfassen, Vorverarbeiten und die Weiterleitung der Sensordaten wird häufig hierarchisch modelliert.

Zusammenfassung

Die Auswahl der Architektur beeinflusst wesentlich den Grad an wahrnehmbarer „Intelligenz“ und den notwendigen Aufwand für die Implementation. Gemessen am Einsatzkontext und gewünschtem Verhalten kann eine eher minimale oder eine aufwendige Architektur gewählt werden.

Während der hierarchische Ansatz von einer oberflächlichen Abstraktion des menschlichen Handelns („ich sehe“, „ich schließe“, „ich handle“) ausgeht, wurde die Entwicklung der reaktiven Verhalten durch Beobachtungen bei Insekten motiviert. Das Ziel ist es jedoch ein System zu implementieren, dass sowohl reaktiv als auch proaktiv ist. Die hybride

Architektur, der das Agentenparadigma zuzuordnen ist, vereint diese beiden konträren Ziele. Um den Grad an Intelligenz zu erhöhen, muss die Funktionsweise der Planungskomponente in den Fokus genommen werden.

3.2.3 Die Implementation von BDI-Agenten

Agenten zu implementieren ist nicht ganz einfach, da zunächst die oben beschriebene Steuerungsarchitektur umgesetzt werden muss. Da die Struktur dieser Software nicht vom Inhalt der ausformulierten Ziele und Aktionen abhängig ist, kann die Implementation von Multi-Agenten-Systemen durch Frameworks unterstützt werden. Eine Form dieser Frameworks sind die PRS-Systeme (Practical Reasoning Systems), zu denen auch Jadex zählt.

Sie unterstützen die Funktionsweise von Practical Reasoning Agents (PRA), einer Klasse Agenten, zu denen auch die BDI-Agenten gehören. Deren ausgeführte Handlungen basieren auf dem Resultat einer Funktion, welche die *Beliefs*, *Desires* und *Intentions* des Agenten einbezieht.
Sie kann wie folgt dargestellt werden:

$$plan : 2^{Bel} \times 2^{Int} \times 2^{Ac} \rightarrow \text{Plan [27]}$$

Diese Funktion beschreibt die Grundstruktur des Entscheidungsfindungsprozesses (Means-Ends-Analyse) und wird in einer Endlosschleife durchlaufen [27]. Zunächst wird die Welt auf Veränderungen überprüft und das interne Weltmodell angepasst. Anschließend entscheidet sich der Agent für eine Intention, die er verfolgen möchte (Deliberation) und ermittelt daraufhin einen Plan, um diese Intention zu erreichen.
Bei der Deliberation wird zunächst überprüft, welche Ziele aufgrund des Weltmodells sinnvoll sind, um dann mittels einer Filterfunktion ein primäres Ziel auszuwählen.

Agentenframeworks wie Jadex stellen ein Rahmenwerk für diesen Prozess und Konzepte zur Kommunikation der Agenten untereinander zur Verfügung.

4 Durchführung einer Verhaltensmodellierung

4.1 Einleitung

In den bisherigen Kapiteln wurde der Einsatz von Softwareagenten zur Modellierung eines intelligenten MRS-Verhaltens vorgeschlagen. In diesem Abschnitt soll nun die praktische Umsetzung einer solchen Modellierung untersucht werden.

Die folgende Grafik zeigt welche Komponenten in dieser Arbeit zusammengeführt werden:

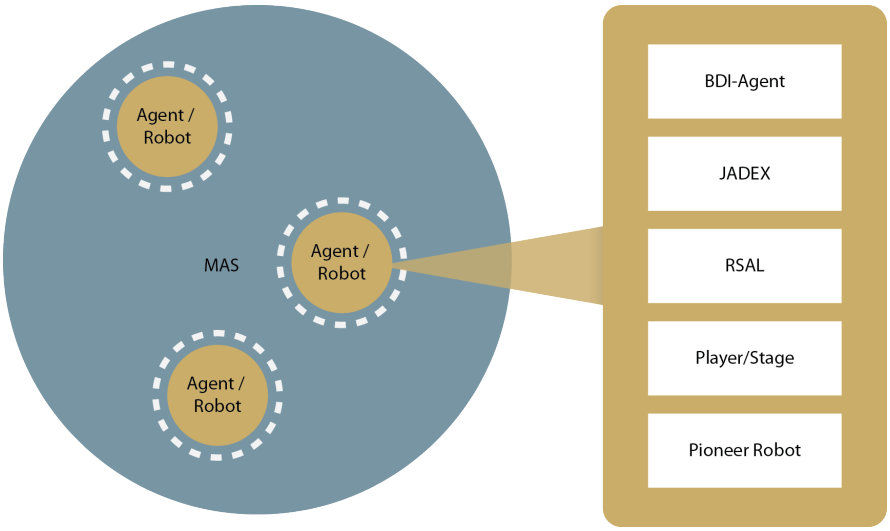

Abbildung 4.1: Diese Grafik veranschaulicht die Teilbereiche, welche in dieser Arbeit zusammengeführt werden. Auf der unteren Ebene befindet sich das MRS, bestehend aus mobilen Robotern vom Typ Pioneer. Das Framework RSAL bietet die Möglichkeit der Abstraktion von der Roboterhardware, sodass die Implementation der Verhaltensmodellierung direkt auf einer höheren Ebene umgesetzt werden kann. Die Implementation erfolgt lediglich gegen die Schnittstellen des Agenten-Frameworks Jadex und der Middleware RSAL.

In dem realen Anwendungsszenario wird das MRS durch Pioneer Roboter vertreten,

auf denen die Agentensoftware ausgeführt wird. In diesem Fall entspricht die Rolle eines Roboters im Wesentlichen der Entität eines Agenten. Wobei dies nicht zwangsläufig so umgesetzt werden muss, da die Modellierung eines Agenten sich nicht an real existierenden Objekten orientiert, sondern mit der Kopplung von Eigenschaften und Fähigkeiten verbunden ist.

Das Verhalten soll sowohl in der Simulationsumgebung von Stage, als auch in einem realen Anwendungsszenario getestet werden. Es gibt also verschiedene Umgebungen, in denen sich das Verhalten bewähren soll, die bei der Implementation ebenso berücksichtigt werden müssen, wie die Eigenschaften der Roboter. Daher wird zunächst etwas näher auf diese Aspekte eingegangen.

4.2 Beschreibung der Evaluationsplattform

Die Hardware der Roboter stellt Programmierer typischerweise vor große Herausforderungen. Die besondere Bauart nimmt Einfluss auf die Platzierung der einzelnen Komponenten. Einige dieser Komponenten, wie die verschiedenartigen Sensoren oder die Kamera, benötigen eine korrekte Ausrichtung und sollten bei Erschütterungen nicht verrutschen. Zur Gewährleistung der Mobilität werden mobile Roboter ausschließlich mit einem Akku betrieben statt über ein Netzteil und auch die Kühlung des Prozessors ist in dem oft kleinen Gehäuse nicht ganz einfach zu lösen. Derzeit werden noch relativ selten Mehrkernprozessoren verbaut. Daher spielt bereits die Auswahl der verwendeten Roboter eine große Rolle. In diesem Fall sollen Pioneer Roboter benutzt werden, da die Agentensoftware relativ viel Prozessorleistung erfordert. Die Roboter verfügen je nach Bauart über einen integrierten Computer, können alternativ aber mit einem beliebigen Notebook gesteuert werden, das auf der Plattform oberhalb des Roboters befestigt wird und ist mit gängigen Betriebssystemen wie Windows oder einer Linux Distribution kompatibel.

Bei den verwendeten Pioneer Robotern handelt es sich um mobile radgetriebene Roboter vom Typ Pioneer 2DX und 3AT der Firma Adept MobileRobots [11]. Beide Arten sind mit jeweils 8 bzw. 16 Ultraschall-Sensoren und je einem Laser Sensor ausgestattet. Diese Roboter werden häufig in der Forschung und Lehre eingesetzt. Aufgrund ihrer Flexibilität und Erweiterbarkeit werden sie überwiegend für Aufgaben eingesetzt, die autonome Navigation, Pfadplanung, Selbstlokalisation oder Interaktion mit anderen Objekten erfordern. Man kann sie um Endeffektoren, wie beispielsweise einen Greifer, ergänzen um

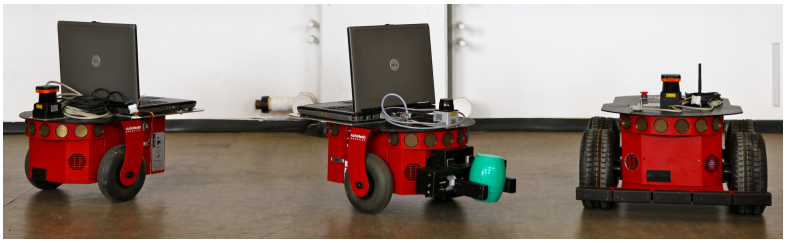

Abbildung 4.2: Diese Abbildung zeigt mobile Roboter des Typs Pioneer, welche für den Indoor-Einsatz vorgesehen und unterschiedlich ausgestattet sind. Im vorderen Bereich, unterhalb der Plattform, sind die Ultraschallsensoren erkennbar. Oberhalb der Plattform sind ein Laptop, auf dem das Programm läuft sowie ein Laserscanner befestigt. Die Roboter können unter anderem mit einem Greifer ausgestattet werden, um Objekte zu transportieren. Abbildung: ©Sebastian Rockel, 2011

Objekte im Raum greifen und transportieren zu können. Die Ausfertigungen 2DX und 3AT unterscheiden sich wie folgt:

Der **Pioneer 2DX** verfügt über 3 Räder mit geringem Profil und ist für den Einsatz im Indoor Bereich gedacht. Der Antrieb ermöglicht eine Fortbewegungsgeschwindigkeit von bis zu 1,6 m pro Sekunde. Die 3 Batterien sorgen für eine Laufzeit von bis zu 8 Stunden.
Der **Pioneer 3AT** hingegen ist für den Einsatz im Gelände gedacht und besitzt daher 4 Räder mit tiefem Profil. Er kann sich mit einer Geschwindigkeit von bis zu 0,8 m pro Sekunde fortbewegen.

Zur Ansteuerung der Sensoren und Aktuatoren benötigt man gerätespezifische Treiber. Die Pioneer Roboter bieten eine Schnittstelle zum Ansprechen des On-Board Microcontroller, die mit C oder auch C++ implementiert werden kann, um Funktionen auf einer höheren Abstraktionsebene entwickeln zu können. Bei der Schnittstelle handelt es sich um P2OS, einem Treiber des Projekts Player/Stage [16], für das gleichnamige Betriebssystem der Pioneer-Roboter.

4.3 Umgebung

Die Umgebung ist einer der wichtigsten Aspekte beim Entwurf eines Agentenszenarios. Agenten werden als Teil der Umgebung modelliert und interagieren mit ihr bzw. den darin befindlichen Objekten. Laut Russel u. Norvig [26] können Umgebungen aufgrund

folgender Eigenschaften klassifiziert werden:

Beobachtbarkeit Die Umgebung ist beobachtbar, wenn sie vollständig einsehbar bzw. zugänglich ist und jederzeit alle notwendigen Informationen über die Umgebung verfügbar sind. Die reale Welt oder das Internet, eine der typischen Umgebungen für reine Softwareagenten, sind beispielsweise nicht vollständig beobachtbar.

Determinismus In einer deterministischen Umgebung führt jede Aktion eines Agenten garantiert zu einem bestimmten Folgezustand. In einer physikalischen Umgebung kann dieser Zustand im Grunde nie erreicht werden. Gerade diese Eigenschaft begründet jedoch den Einsatz von Agenten.

Dynamik Werden Veränderungen in einer Umgebung allein durch das Handeln eines Agenten hervorgerufen, so handelt es sich um eine statische, andernfalls um eine dynamische Umgebung.

Zeitliche Abhängigkeit Die Ausführung von Aktionen sowie die Erfassung der Eingangswerte erfolgt entweder zeitdiskret oder kontinuierlich (reale Welt).

Gemessen an diesen Kriterien stellt die reale Welt die komplexeste Umgebung dar. Sie ist nicht vollständig beobachtbar, nicht-deterministisch, dynamisch und kontinuierlich.

Die Beschaffenheit der Umgebung bestimmt maßgeblich die Systemanforderungen, hinsichtlich verfügbarer Kommunikationswege und die mögliche Fortbewegungsart bei Multi-Roboter-Systemen.

Das behandelte Szenario beschränkt sich auf einen Weltausschnitt. Es hat sich angeboten, dafür die Räumlichkeiten des Arbeitsbereichs TAMS am Informatikum der Universität Hamburg zu verwenden. Eine grafische Darstellung dieser Umgebung ist aus mehreren Gründen vorgesehen. Zum einen bietet die Visualisierung eine Möglichkeit das Verhalten der gesamten Agenten auch dann zu beobachten, wenn sich einzelne physikalische Entitäten (Roboter) in unterschiedlichen Räumen befinden. Zum anderen steht nur eine begrenzte Anzahl dieser Roboter zur Verfügung, sodass das Verhalten in einer skalierten Konfiguration nur in einer Simulationsumgebung visuell beobachtet werden kann.

Diese Arbeit sieht mehrere Konfigurationen des Szenarios vor, eine vollständige Simulation, eine Mixed Reality Anwendung und eine Durchführung des Szenarios in der realen Umgebung.

4.3.1 Virtuelle-/ Simulations-Umgebung

Für die realitätsnahe Simulation des Multi-Agenten-Systems muss die Welt möglichst genau und vollständig abgebildet werden. Dazu gehören nicht nur die statischen Eigenschaften wie Wände, Möbel und freie Wege, sondern auch alle Akteuere (Roboter, Fabriken) und die Ressourcen/ Rohstoffe. Eine Darstellung dieser Objekte in 2D bzw. 2,5D wird durch das Framework Stage [16] ermöglicht. Abbildung 4.3 zeigt eine zweidimensionale Darstellung des Arbeitsbereichs TAMS der Universität Hamburg. Die Simulation bietet gegenüber der realen Welt Vorteile hinsichtlich der Skalierbarkeit des Szenarios, Abstraktion von der Hardware und gefahrloses Testen, was jedoch bei der Übertragbarkeit des Systems für den Einsatz in der realen Welt berücksichtigt werden muss. Die virtuelle Umgebung ist darüber hinaus zeitdiskret und in Teilbereichen deterministisch. Da keine Sensordaten erfasst und vorverarbeitet werden müssen, entfällt die Bearbeitungszeit dafür und die Wahrscheinlichkeit unbeabsichtigt fehlender Eingangswerte sinkt.

4.3.2 Mixed Reality

Sie ist weitaus komplexer als die virtuelle Umgebung. Sie dient hier als Erweiterung der realen Welt um virtuelle Objekte (Avatare), mit dem Zweck, gezielt Situationen herbeizuführen und steuern zu können, wie zum Beispiel eine zehnfache Anzahl Roboter. So können die Auswirkungen beim Einsatz in der realen Welt und die Funktionalität des Systems realitätsnah geprüft und dennoch gewünschte Bedingungen bewusst erzeugt werden. Es können also zu jedem Zeitpunkt gezielt Situationen, die getestet werden sollen, hervorgerufen werden. Das bedeutet auch, dass diese zu Evaluierungszwecken beliebig oft reproduziert werden können. Dennoch ist das Szenario in eine reale Umgebung eingebettet und birgt somit die Gefahr, dass sich die Umgebung spontan verändert oder beteiligte Komponenten ausfallen. Damit erhält man eine realitätsnahe Testumgebung.

4.3.3 Reale Umgebung

In der realen Welt haben wir es mit einer hoch dynamischen Umgebung zu tun. Es können jederzeit unvorhergesehene Veränderungen in der Umgebung eintreten, die Auswirkungen auf das gewünschte Verhalten des Agenten haben. Dies können entweder Situationen

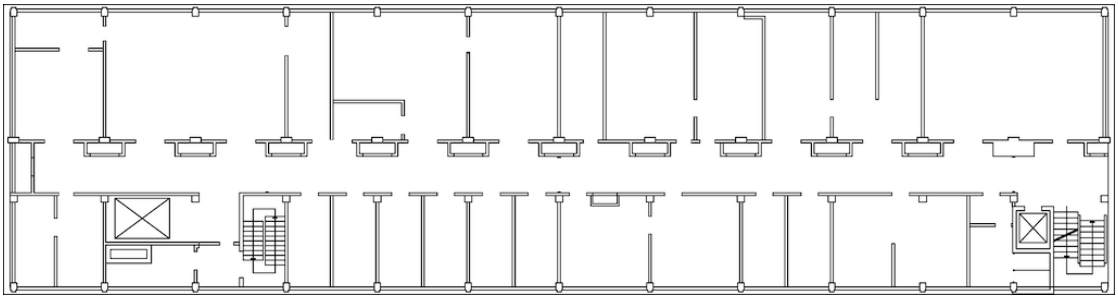

Abbildung 4.3: Diese Abbildung zeigt eine grafische Darstellung des Arbeitsbereichs TAMS der UNI Hamburg. Sie kann um Objekte ergänzt werden, die bei der Navigation berücksichtigt werden sollten, wie Tische und Schränke. Zusätzlich werden auch virtuell existierende Objekte, wie die Fabriken oder zusätzliche Roboter visualisiert.

sein, die die Ausführung eines Plans verhindern, den Anstoß eines neuen Ziels zur Folge haben oder die Ausführung alternativer Pläne ermöglichen. Die reale Umgebung stellt große Anforderungen an das gesamte System. Die Software soll flexibel sein um sich in allen eintretenden Situationen zu bewähren und die Hardware muss sehr robust sein, da es in dieser Umgebung zu Kollisionen und Erschütterungen kommen kann. Die reale Welt ist komplex, sehr dynamisch und ggf. unbekannt, weshalb bei mobilen Robotern der Arbeitsraum unter Umständen immer wieder neu erfasst und das Weltmodell entsprechend angepasst werden muss.

4.4 Szenario

Beschreibung des Szenarios

Autonome Roboter sollen verschiedenartige Rohstoffe innerhalb einer begrenzten Umgebung finden und in endlicher Zeit einem Bestimmungsort (Produktionsanlage) zuführen.

Die simulierten Produktionsanlagen stellen spontan Rohstoffanforderungen. Eine solche Rohstoffanforderung beinhaltet neben der Art des angeforderten Rohstoffs auch eine Deadline für die Lieferung. Ist diese Deadline verstrichen, entsteht ein Produktionsausfall.

Ziel ist es, die Ausfallzeiten möglichst gering zu halten. Um den Rohstoffanforderungen nachzukommen müssen die Roboter vorübergehend unterschiedliche Rollen (Explorer, Collector) wahrnehmen und sich untereinander abstimmen. Das angestrebte Verhalten

soll dazu führen, dass Roboter Leerlaufzeiten nutzen um ggf. die Umgebung zu erkunden und Rohstoffvorkommen zu kartografieren oder ein Zwischenlager anzulegen.
Das Szenario wird zum Zwecke der Messbarkeit in jedem Durchlauf mit festen Anforderungsplänen der Produktionsstätten durchgeführt. Diese sind den Robotern jedoch nicht bekannt.

Die Roboter (Lieferanten) möchten möglichst viele Ausschreibungen bedienen. Dazu muss der gewünschte Rohstoff zunächst gefunden werden um ihn dann zur Produktionsanlage (Fabrik) zu bringen. An dieser Stelle ist die Verwendung eines homogenen MAS sinnvoll. Die Roboter sollen die Umgebung explorieren und gefundene Rohstoffe erfassen.

Ein Rohstoffgenerator sorgt für eine konstante Anzahl verfügbarer Rohstoffe in der Umgebung. Die Startkonfiguration des Szenarios initialisiert eine feste Anzahl verschiedenartiger Rohstoffe und verteilt diese an zufälligen und den Robotern unbekannten Positionen in der Umgebung. Die explorierenden Roboter erkunden die Umgebung und erfassen die Art, Anzahl und Position der gefundenen Rohstoffe und stellen damit dieses Wissen dem gesamten MRS zur Verfügung. Dieses Wissen stellt eine notwendige Bedingung zur Annahme von Ausschreibungen zur Verfügung. Auf Basis dieser Fakten über die Welt, sind die Roboter in der Lage, verbindliche Zusagen zu treffen und Aufträge nicht nur zeitnah zu erfüllen, sondern auch ihren Durchsatz zu steigern. Um die Lieferung innerhalb einer Deadline vollziehen zu können und ggf. mehrere parallele Ausschreibungen fristgerecht zu erfüllen, soll derjenige Roboter den Auftrag ausführen, der bei Auftragsvergabe den kürzesten Weg zwischen aktueller Position, Rohstoff und Bestimmungsort befahren kann. Zusätzlich soll die Pfadplanung kürzeste Wege bevorzugen und Kollisionen vermeiden.

Im Falle, dass mehrere Produktionsstätten zeitgleich Rohstoffe anfordern, sollen die Roboter gemäß „earliest deadline first“ die Produktionsstätten beliefern oder sich doppelte Wege sparen, indem sie ihre Kapazität und Kardinalität kennen und Waren austauschen.

4.5 Modellierung des Verhaltens

Der Entwurf eines MAS erfordert ein methodisches Vogehen für die Gestaltung und Umsetzung des Entwurfs eines Zielsystems. Der Entwurf dient dazu das gewünschte Systemverhalten ausgehend vom Szenario darzustellen. Damit werden die konkreten Anfor-

derungen deutlich und Agententypen können identifiziert werden.

Das gewünschte Verhalten wird nachfolgend unter Berücksichtigung der Rahmenbedingungen modelliert. Wie zuvor erwähnt, werden dazu BDI-Agenten entworfen, die dass Szenario bewältigen. Daher muss zunächst aus dem gegebenen Szenario das gewünschte Verhalten abgeleitet und Agententypen identifiziert werden. Desweiteren muss das Verhalten der einzelnen Agententypen zerlegt bzw. abstrahiert werden und den 3 Kernelementen eines BDI-Agenten, nämlich Belief, Desire und Intention zugeordnet werden. Es stellt sich die Frage nach einer geordneten Vorgehensweise bzw. nach Modellierungswerkzeugen, die den Entwurf eines MAS unterstützen. In der Softwaretechnik gibt es eine Vielzahl verschiedener Entwurfsparadigmen und zugehöriger Werkzeuge. In der objektorientierten Softwareentwicklung werden häufig UML-Diagramme (Unified Modeling Language), oder auch Petrinetze, zum Beispiel zur Modellierung und Verifikation nebenläufiger Programme, verwendet. Die meisten dieser Modellierungswerkzeuge können auch zur Beschreibung des Verhaltens von Agenten verwendet werden. Bei dem Versuch das gewünschte Verhalten mittels eines Petrinetzes zu modellieren, stellte sich jedoch heraus, dass es zwar eine gute Methode ist, das sichtbare Verhalten einzelner Agenten zu veranschaulichen, sich jedoch nicht dazu eignet, das gewünschte Verhalten des gesamten MAS darzustellen. Da die Eigenschaften eines MAS und insbesondere die Zerlegung in Belief, Desire und Intention nicht erkennbar sind, bietet eine Modellierung allein mittels UML keine ausreichende Grundlage für spätere Architekturentscheidungen. Um diesen Anforderungen gerecht zu werden benötigt man geeignete Methoden und Entwicklungswerkzeuge, die den Entwurf von Softwareagenten geeignet unterstützen.

Lars Braubach befasst sich in seiner Dissertation, „Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme “ [15], mit Methoden und Entwicklungswerkzeugen zur Modellierung von Agenten und kommt zu dem Schluss, dass die Auswahl einer geeigneten Entwicklungsmethode auch auf das verwendete Framework abgestimmt sein sollte. Das verwendete Modellierungswerkzeug sollte die Konzepte des Frameworks wiederspiegeln. Auf die Frage nach einer geeigneten Methode, zur Verhaltensmodellierung von BDI-Agenten auf der Basis von JADEX, gibt Brauchbach ebenfalls eine Antwort. Er untersuchte eine Auswahl an gängigen Entwicklungsansätzen, nämlich Prometheus, Gaia, PASSI und Tropos. Die Untersuchungskriterien waren zum einen die Abbildung der Agentenarchitektur sowohl aus interner als auch sozialer Sicht und die Integrationsfähigkeit der Methoden und Frameworks. Die interne Architektur beinhaltet Punkte wie die Abbildung der Kernelemente Beliefs, Goals, Plans, Events und weitere.

Mit sozialer Architektur sind die Konzepte der Interaktion der Agenten eines Systems, insbesondere die Verwendung von Protokollen, Rollen und der Umgebung, gemeint. Diese Untersuchung ergab, dass keiner der von Braubach betrachteten Entwicklungprozesse für BDI-Agenten den Anforderungen, in Bezug auf die Verwendung von JADEX, vollständig gerecht wird. Dennoch schnitten Prometheus und Tropos bei dieser Untersuchung mit dem besten Gesamtergebnis ab. In Abhängigkeit der Aufgabenstellung sollten diese Methoden lt. Braubach dennoch den aufgabenspezifischen Gegebenheiten angepasst werden.

Diese Arbeit folgt diesem Ergebnis und verwendet die Tropos-Methode [22] zur Modellierung des MAS. Alternative Empfehlungen, wie zum Beispiel die von Bernhard Bauer [13], welcher AUML (Agent-UML), eine Erweiterung von UML, für die Modellierung eines MAS vorschlägt, finden zusätzlich in Tropos Verwendung, allerdings erst in späteren Entwicklungsphasen des Systems.

Der Modellierungsansatz Tropos setzt bereits auf der Meta-Ebene des Szenarios an. Ausgehend von der Betrachtung des Einsatzkontextes und aller beteiligten Akteure, werden verschiedene Phasen sukzessive durchlaufen und das Verhaltensmodell stetig erweitert, bis ein Konzept entstanden ist, aus dem die Implementation abgeleitet werden kann. Während der einzelnen Phasen werden zunehmend die BDI-Konzepte und Agententypen des Zielsystems in die bestehende Ausformulierung des gewünschten Verhaltens eingearbeitet.

Konkret werden die Anforderungsanalyse (Frühe und Späte Anforderungsanalysephase), die Systemspezifikation (Architekturdesignphase und Detaillierte Designphase) und die Implementationsphase nacheinander durchlaufen [22].

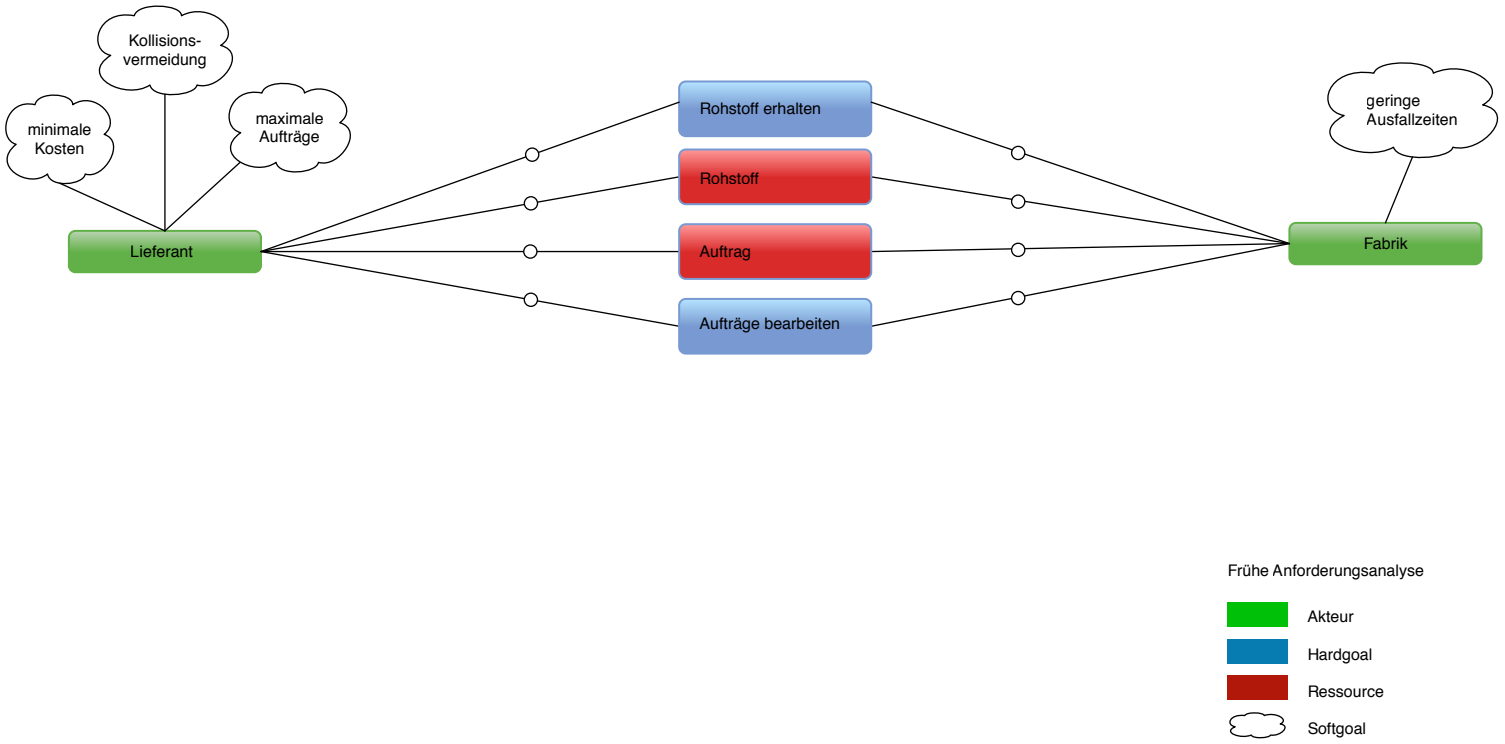

Abbildung 4.4: Frühe Anforderungsanalysephase: Diese Darstellung veranschaulicht die identifizierten Akteure und deren Abhängigkeiten in Bezug auf das beschriebene Szenario. Zur besseren Übersichtlichkeit wird in dieser Abbildung nur die Wechselbeziehung zwischen Lieferant und Fabrik dargestellt.

4.5.1 Frühe Anforderungsanalysephase

In der frühen Anforderungsanalysephase wird bewusst vom Zielsystem abstrahiert und zunächst lediglich die Akteure und ihre Abhängigkeiten im Szenario identifiziert und in Beziehung zueinander gestellt. Jeder Akteur verfolgt Ziele zu dessen Erfüllung er auch von der Kooperation anderer Akteure abhängig ist. Neben diesen primären Zielen (Hardgoals) verfolgen die Akteure auch Interessen, die den Erfüllungsgrad eines solchen Ziels bestimmen.
In dem betrachteten Szenario sind primär drei Arten von Akteuren beteiligt. Die Lieferanten, die Fabriken und die Umwelt, in Form eines Rohstoffgenerators.

Die **Fabriken** fordern in regelmäßigen Abständen Rohstoffe an, die sie bis zu einem bestimmten Zeitpunkt benötigen. Es wird angenommen, dass eine Lagerung der Rohstoffe vor Ort aus finanziellen oder logistischen Gründen nicht vorgesehen ist. Eine Rohstofflieferung nach Ablauf der Deadline führt zu Ausfallzeiten. Die Fabriken sind also zur Einhaltung ihrer Ziele sehr stark an ein funktionierendes MRS gebunden und stehen somit in direkter Verbindung mit den Lieferanten, da sie an minimale Ausfallzeiten interessiert sind.

Die **Lieferanten** möchten möglichst viele Aufträge annehmen und fristgerecht liefern. Eine Gewinnmaximierung kann durch eine Zeitersparnis erzielt werden, indem die Rohstoffsuche optimiert wird. Mit der Gewinnmaximierung wird ein Ziel postuliert, das eine Nebenbedingung für das eigentlich verfolgte Ziel der Auftragsbearbeitung darstellt und dessen Erfüllungsgrad nicht von anderen Akteurentypen abhängig ist. Ein weiteres so genanntes weiches Ziel (Softgoal) der Lieferanten ist die Kollisionsvermeidung, damit soll der Ausfall von Robotern vermieden werden.

Um Fabriken beliefern zu können, müssen zunächst Rohstoffe gefunden werden. Daher besteht eine Wechselbeziehung zwischen den Lieferanten und ihrer Umwelt. Die Umwelt ist kein Akteur im eigentlichen Sinne. Sie kann in einem real existierenden Szenario einem Großhandel oder verteiltem Lager entsprechen, so wie auch die einzelnen Roboter jeweils einem Lastwagen einer Spedition entsprechen könnten. In diesem konkreten Fall ist die Modellierung der Umwelt für die Bewertung des Szenarios notwendig. Es soll zu Evaluierungszwecken eine konstante Anzahl verfügbarer Rohstoffe in der Umgebung zur Verfügung gestellt werden. Ein Rohstoffüberschuß würde dazu führen, dass die Roboter

nicht suchen müssen. Eine zu geringe Anzahl Rohstoffe führt zwangsläufig zu einer Unterversorgung der Fabriken, die nicht der Leistungsfähigkeit des MRS geschuldet ist. Zu diesem Zweck wird die Umwelt als eigenständiger Akteur, in Form eines **Rohstoffgenerators** modelliert, mit dem Ziel Rohstoffe an zufälligen Positionen der Umgebung zu erzeugen und eine konstante Anzahl verfügbarer Rohstoffe zu gewährleisten.
Es gibt zwei Ressourcen in dem Anwendungsszenario, zum einen sind das die Rohstoffe und zum anderen die Aufträge, von denen möglichst viele vergeben und korrekt ausgeführt werden sollen. Eine Auftragsannahme erfolgt nur, wenn der korrekte Rohstoff geliefert wurde.

4.5.2 Späte Anforderungsanalysephase

In der späten Anforderungsanalyse wird das entsprechende Zielsystem definiert, welches die funktionalen Anforderungen der Akteure koordiniert und stellvertretend ausführt.

Das bedeutet zunächst, das die Hardgoals mittels konkreter Planstrukturen im System umgesetzt werden, was die Ausführung von Handlungen oder die Formulierung von Unterzielen zur Folge hat. Da nach Möglichkeit auch die Einhaltung der Softgoals der jeweiligen Akteure gewährleistet sein soll, werden diese ebenfalls dem Zielsystem übertragen.
Um Aufträge bearbeiten zu können, müssen mehrere Bedingungen erfüllt sein. Es müssen zunächst Aufträge vorliegen aber auch die notwendigen Rohstoffe verfügbar sein, um die Fabriken zu beliefern. Diese Voraussetzungen werden in dieser Entwurfsphase als eigenständige Ziele formuliert. Falls der gewünschte Rohstoff vorhanden ist und Roboter verfügbar sind, können Aufträge angenommen werden. Das heißt aber auch, dass kontinuierlich Rohstoffe gefunden werden müssen. Verfügbare Lieferanten sollten sich um Aufträge bewerben oder kenntlich machen, dass sie für die Ausführung von Aufträgen zur Verfügung stehen. Ein Lieferant ist nicht verfügbar, wenn er bereits einen Auftrag ausführt oder, im Falle eines Szenarios in der realen Umgebung, ein Roboter beschädigt ist.
Die Fabriken wollen Rohstoffe erhalten, dazu müssen sie die Warenanforderung ausschreiben. Die Ausschreibung des Auftrags beinhaltet die Angabe einer Deadline für den Erhalt der Lieferung und die Art des gewünschten Rohstoffs. Bei Eintreffen der Lieferung muss geprüft werden, ob es sich um den angeforderten Rohstoff handelt. Bei Ablauf der Deadline ist die Ausfallzeit zu erfassen.

An den Rohstoffgenerator werden die funktionalen Aufgaben der Umgebung delegiert. Aus Gründen der objektiven Bewertbarkeit des MAS-Verhaltens wurde, wie bereits erwähnt, festgelegt, dass stets nur eine konstante Anzahl an Rohstoffen in der Umgebung vorhanden ist. Sobald ein Rohstoff erzeugt wurde, ist dieser verfügbar. Erst bei der Warenannahme durch die Fabrik bzw. dem Verbrauch eines Rohstoffs ist dieser nicht mehr existent. Daher besteht zwischen den Fabriken und dem Rohstoffgenerator eine Beziehung, die den Rohstoffgenerator zur Erzeugung eines neuen Rohstoffs veranlasst.

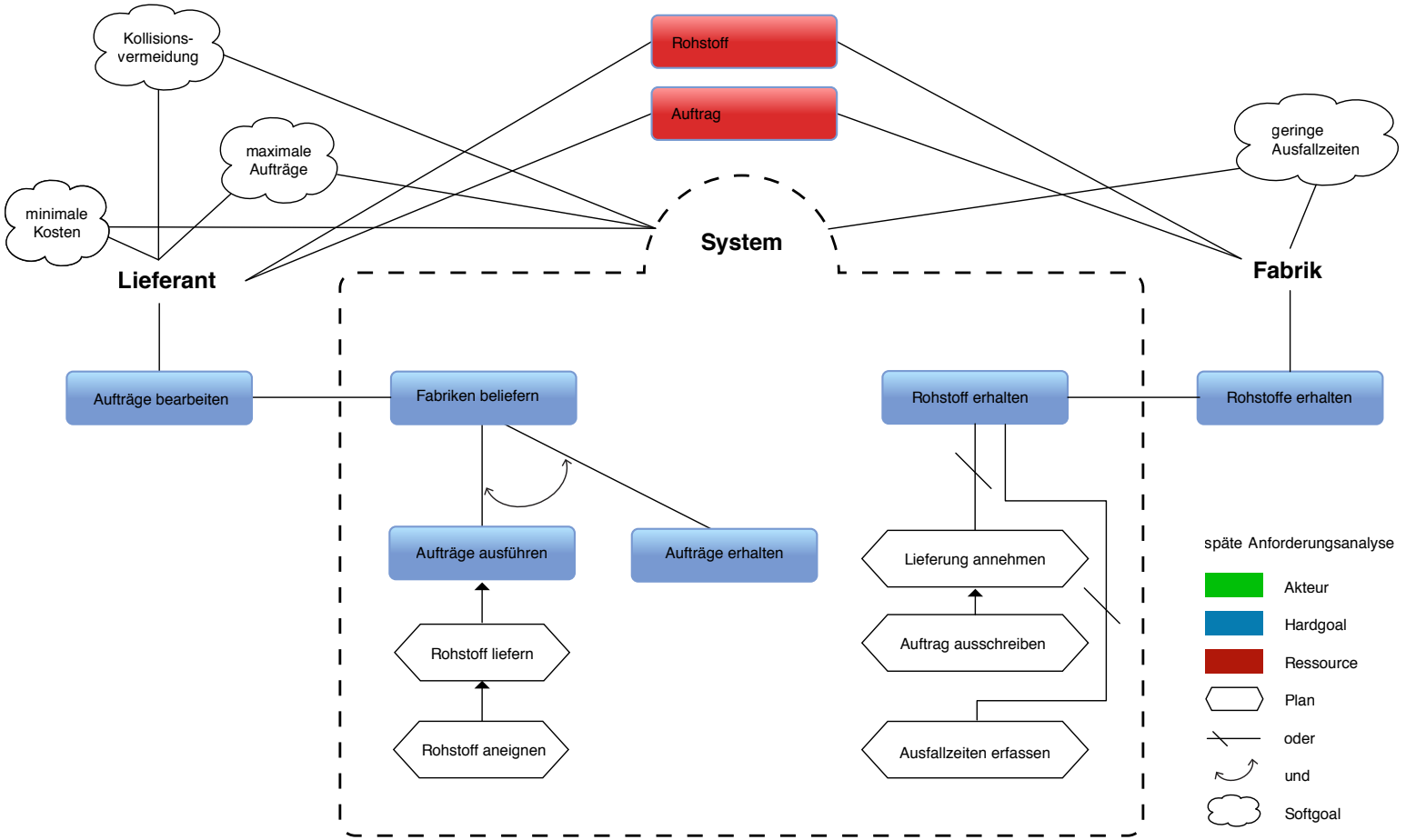

Abbildung 4.5: Späte Anforderungsanalysephase: Es wurde das Zielsystem definiert und dessen Anforderungen bzgl. Hardgoals und Softgoals bestimmt. Das Zielsystem soll stellvertretend für die Akteure dessen Ziele mittels konkreter Handlungsanweisungen erfüllen und dabei die Einhaltung der Softgoals berücksichtigen.

4.5.3 Architekturdesignphase

In der Architekturdesignphase wird das in den vorhergehenden Schritten ausgearbeitete Zielsystem genauer spezifiziert. Dies geschieht in drei konsekutiven Schritten.

Schritt 1:
Im ersten Schritt wird das Zielsystem wenn nötig in Subsysteme unterteilt. Dies ist insbesondere dann notwendig, wenn das Zielsystem mehrere Unterziele und Abhängigkeiten beinhaltet. Diese entsprechen hier überwiegend den bisherigen Akteuren, dem Rohstoffgenerator, der Fabrik und den Lieferanten.

Schritt 2:
In diesem Schritt soll erarbeitet werden, welche Eigenschaften, Pläne und Unterziele die einzelnen Agenten benötigen, um die vorab definierten Ziele zu erfüllen. Mittels Means−End−Analyse wurden Ziel−Plan−Hierarchien entwickelt. Unter anderem wurde hier die Fahrt zu einem Ziel explizit als Subgoal definiert, da die Ausführung dieser Handlung sehr komplex ist und eigene Pläne, zur Vermeidung von Kollisionen oder das Befahren der kürzesten Wege beinhaltet.

Schritt 3:
Das Verhalten wird für ein homogenes MRS entworfen. Die konkreten Entitäten des Agententyps „Lieferant“ stehen nicht in Konkurrenz zueinander sondern müssen ihr Handeln zum Wohle der gemeinschaftlichen Ziele koordinieren. Die Aufgabe des Lieferanten ist es, zu prüfen ob die Bedingungen für eine Auftragsannahme vorliegen: Sind offene Aufträge und die korrespondierenden Rohstoffe vorhanden? Stehen Roboter zur Ausführung des Auftrags zur Verfügung? Wenn dem so ist, soll der Auftrag an den Roboter mit dem geringsten Weg zwischen aktueller Position des Lieferanten, des Rohstoffs und der Fabrik, vergeben werden. Diese Eigenschaft der Lieferanten soll in Form eines Auftragsvergabeservice als Capability des Agententyps „Lieferant“ modelliert werden.

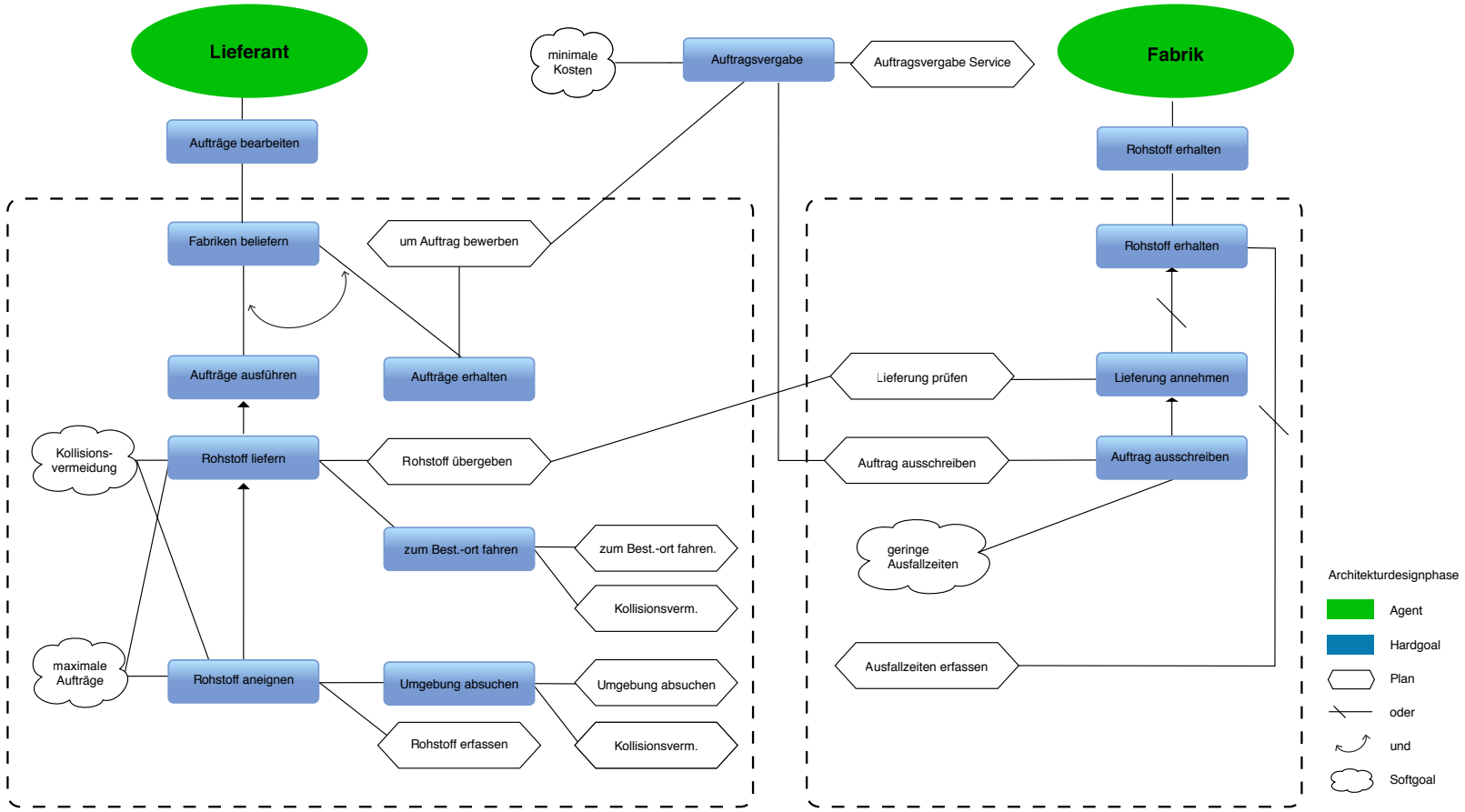

Abbildung 4.6: Architekturdesignphase: In dieser Phase wird die globale Architektur des Systems entworfen, in dem Agententypen definiert werden, die untereinander durch Datenaustausch und Kontrollfluß verbunden sind. Als Agententypen wurden die Fabriken, die Lieferanten und der Rohstoffgenerator festgelegt. Der Auftragsvergabeservice wurde als notwendiges Subsystem der Lieferanten identifiziert, und soll als eigenständige Capability aller Lieferanten umgesetzt werden.

4.5.4 Detaillierte Designphase

In dieser Phase werden die einzelnen Plankörper konkretisiert dargestellt. Hierbei dient die Verwendung von UML der Beschreibung einzelner Plankörper und AUML der Veranschaulichung der Interaktion mehrerer Agenten. Es findet eine Ausgestaltung konkreter Plankörper und der Agentendefinitionen statt. In den vorhergehenden Phasen wurde schrittweise das Gesamtsystem ausgehend von der Metaebene bis hin zu ausführbaren Aktionen erschlossen. Nachfolgend sind alle resultierenden Pläne aufgeführt, deren Ausführung sichtbare Handlungen der Roboter zur Folge haben und im nächsten Kapitel als Java Klassen implementiert werden sollen:

Rohstoffgenerator:
Rohstoff erzeugen−Plan
Fabrik:
Auftrag ausschreiben−Plan
Prüfe Lieferung−Plan
Erfasse Ausfallzeit−Plan
Spediteur:
Um Auftrag bewerben−Plan
Rohstoff liefern−Plan
Zum Bestimmungsort fahren−Plan
Umgebung erkunden−Plan
Rohstoff erfassen−Plan
Auftragsvergabeservice:
Auftragsvergabe−Plan

Was in dieser Phase nicht explizit erwähnt wird, ist das auslösende Ereignis oder die verwendete Form der Kommunikation: Es wurde bereits die Beziehung zwischen Warenanahme der Fabrik und Erzeugung neuer Rohstoffe angesprochen. Es gibt mehrere Möglichkeiten, das Erzeugen neuer Rohstoffe anzuregen. Es gibt unter anderem die Variante, dass die Fabriken explizit der Umgebung mitteilen, dass ein Rohstoff verbraucht wurde. In diesem Fall wurde jedoch für die Verwendung einer indirekten Kommunikation

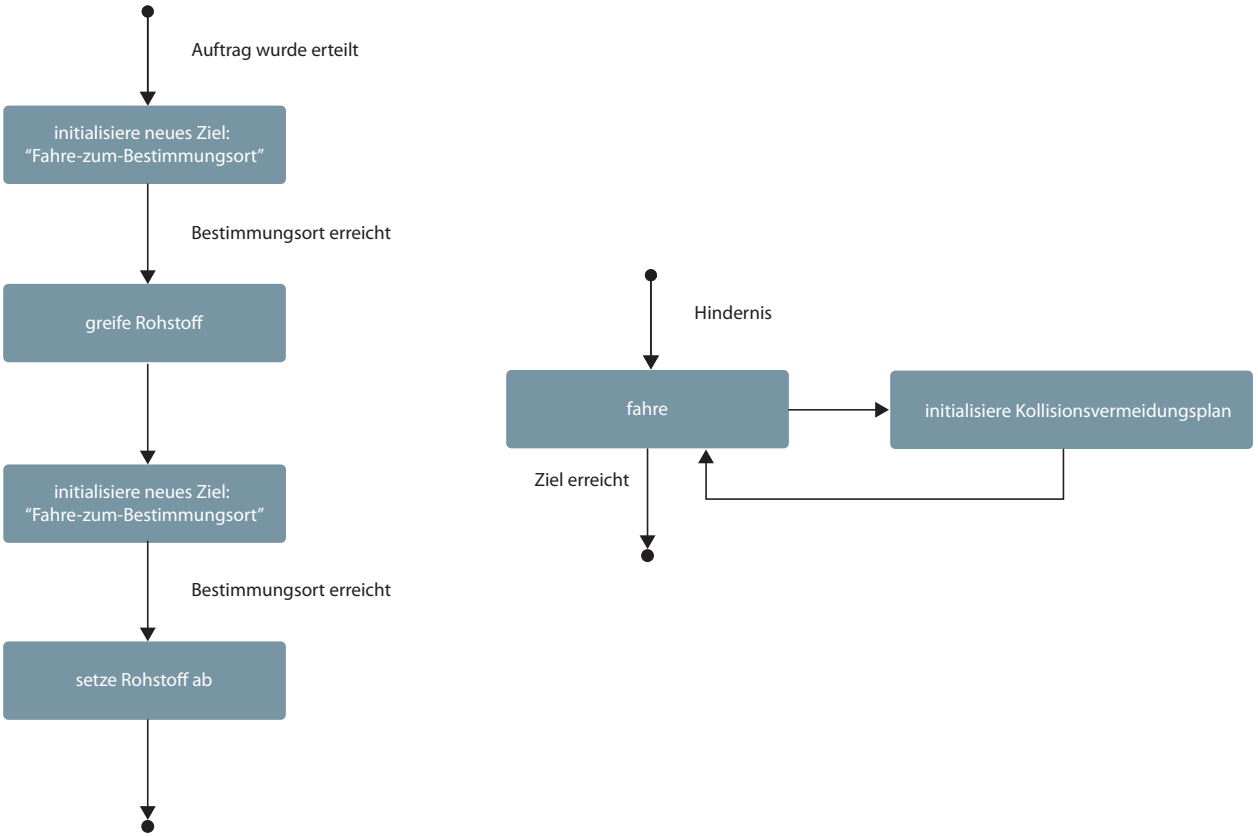

Abbildung 4.7: Veranschaulichung der Pläne „Liefere Rohstoff “ und „Fahre zum Bestimmungsort “ mittels UML.

entschieden. Der Rohstoffgenerator überwacht die Anzahl der Rohstoffe. Bei einer Warennannahme registriert der Rohstoffgenerator die Veränderung an der Anzahl verfügbarer Rohstoffe, wodurch eine entsprechende Planausführung veranlasst wird.

4.5.5 Implementationsphase

An dieser Stelle wird das fertige System schrittweise anhand der Entwürfe implementiert. Diese Phase wird im folgenden Kapitel konkret anhand von Jadex behandelt.

4.6 Implementation

4.6.1 Verwendung von JADEX

Die Implementation erfolgt in Jadex, einem Framework zur Entwicklung von BDI-Agenten unter der Verwendung von Java und XML. Jadex bringt von sich aus keine gesonderte IDE für die Implementation mit, allerdings werden hilfreiche Entwicklungstools als Plug-In für Eclipse zur Verfügung gestellt.
Jadex verfügt über eine grafische Oberfläche, dem JCC, über die nicht nur Agenten oder ganze Szenarien gestartet werden können, sondern enthält darüber hinaus ein grafisches Debugging-Tool.

Aktuelle Jadex Versionen basieren auf einer Service Component Architecture (SCA), bei der ein MAS aus einer Vielzahl einzelner Komponenten besteht, deren Kommunikation über Serviceaufrufe realisiert wird. Um das reaktive Verhalten von BDI-Agenten beizubehalten, wurden diese Komponenten in der Form angepasst, dass alle vorhandenen Komponenten eines MAS zur Laufzeit aktiv sind. Daher werden derzeit Agenten (Micro-Agenten, BDI-Agenten aber auch Services und Capabilities) als so genannte *Active Components* (AC) modelliert, welche selbst Services anbieten und verwenden können. Dadurch lassen sich unter anderem auch Konzepte Service orientierter Anwendungen (SOA) in JADEX wieder finden. Die Kommunikation zwischen Agenten wurde in JADEX ursprünglich ausschließlich über Nachrichten gemäß dem FIPA Standard [8] realisiert. Dies wird seit der Umstellung auf die SCA nur noch für Verhandlungen explizit empfohlen. In den meisten anderen Fällen ist die Kommunikation über Services deutlich leichtgewichtiger und erhöht somit die Reaktionsgeschwindigkeit der Agenten.

Die konkrete Umsetzung des BDI-Konzepts kommt in Jadex insbesondere bei der Definition eines Agententyps zum Tragen.

Ein Agententyp wird in Jadex durch Verwendung eines bestimmten XML-Schemas spezifiziert. Anhand des so genannten ADF (Agent Definition File) ist sehr gut erkennbar, wie die drei Säulen des BDI-Konzepts in Jadex umgesetzt werden.

Dies ist ein Auszug aus dem ADF des Fabrik-Agenten:

```
1  ...
2          <beliefs>
3                  <beliefset name="board" class="Board_Orders1">
4                          <fact>Board_Orders1.getInstance().getOpenOrders()</fact>
5                  </beliefset>
```

```
6                   <belief name="due" class="Due">
7                           <fact>Due.getInstance().getDue()</fact>
8                   </belief>
9                   <beliefset name="existingCommodities" class="Board_Found_Commodities">
10                          <fact>Board_Found_Commodities.getInstance().getExistingCommodity()
11                          </fact>
12                  </beliefset>
13                  <belief name="checkTime" class="Due" updaterate="1000">
14                          <fact>Due.getInstance().getDue().getCheckTime()</fact>
15                  </belief>
16                  <belief name="time" class="long" updaterate="100">
17                          <fact>$scope.getTime()</fact>
18                  </belief>
19                  <belief name="order1" class="Order1[]">
20                          <fact>$args.order1</fact>
21                  </belief>
22          </beliefs>
23          <goals>
24                  <achievegoalref name="cms_create_component">
25                          <concrete ref="cms.cms_create_component" />
26                  </achievegoalref>
27                  <performgoal name="AcceptDeliveryGoal" />
28                  <performgoal name="EvaluationGoal" />
29                  <performgoal name="NewOrderGoal" recur="true" recurdelay="1000">
30                          <parameter name="order1" class="Order1">
31                                  <value>$order1</value>
32                          </parameter>
33                  </performgoal>
34                  <performgoal name="OrderDueGoal" recur="true" recurdelay="100">
35                          <creationcondition language="jcl">$beliefbase.checkTime
36                                  &lt; $beliefbase.time</creationcondition>
37                  </performgoal>
38          </goals>
39          <plans>
40                  <plan name="Accept_delivery_plan">
41                          <body class="Accept_delivery_plan" />
42                          <trigger>
43                                  <goal ref="AcceptDeliveryGoal" />
44                          </trigger>
45                  </plan>
46                  <plan name="Evaluation_plan">
47                          <body class="Evaluation_plan" />
48                          <trigger>
49                                  <goal ref="EvaluationGoal" />
50                          </trigger>
51                  </plan>
52                  <plan name="New_order_plan">
53                          <parameter name="order1" class="Order1"></parameter>
54
55                          <body class="New_order_plan" />
56                          <trigger>
57                                  <goal ref="NewOrderGoal" />
58                          </trigger>
59                  </plan>
60                  <plan name="Order_due_plan">
61                          <parameter name="order1" class="Order1[]"></parameter>
62                          <parameter name="time" class="long"></parameter>
63                          <body class="Order_due_plan" />
64                          <trigger>
65                                  <goal ref="OrderDueGoal" />
66                          </trigger>
67                  </plan>
68          </plans>
69  ...
```

In den Zeilen 2-22 werden die relevanten Fakten der Welt eingebunden. In den Zeilen

23-38 werden die Ziele des Agenten formuliert und in den Zeilen 39-68 sind die vorhandenen Pläne aufgeführt. Die Reihenfolge dieser Tags kann nicht beliebig gewählt werden und entspricht im übrigen auch der Funktion
„$plan : 2^{Bel} \times 2^{Int} \times 2^{Ac} \rightarrow$ Plan [27]“aus Kapitel 3.2.3,
welche die Grundstruktur des Entscheidungsfindungsprozesses bzw. die Planauswahl eines BDI-Agenten beschreibt.

Die Grundlage bildet die Wissensbasis (Beliefbase) welche maßgeblich für die Auswahl und Verfolgung bestimmter Ziele verantwortlich ist. Die Planauswahl steht am Ende dieses Auswahlprozesses, da auch mehrere Pläne zur Erreichung eines Ziels vorhanden sein können.

Nachfolgend wird die Umsetzung des BDI-Konzepts in Jadex beschrieben:

Beliefbase

Einzelne Fakten des Umgebungswissens können als Beliefs oder auch mehrere zusammengefasst als Beliefsets erfasst werden. Dabei kann es sich auch um ganze Java-Objekte als Fakten handeln. Diese können mit einer OQL ähnlichen Anfragesprache der „Wissensdatenbank “ entnommen werden. Dadurch, dass alle Fakten an zentraler Stelle erfasst und verwaltet werden, verfügen alle Agenten über das gleiche Wissen und die Ausführung / Aktivierung der Pläne kann exakt gesteuert werden. Die Beliefbase von Jadex ist keine statische Wissensdatenbank sondern eine aktive Komponente mit der einzelne oder auch komplexe zusammenhängende Fakten der Welt auf Veränderungen untersucht werden können. Bei den hierarchischen Steuerungsarchitekturen wurde bemängelt, dass Fakten nicht zu beliebigen Zeitpunkten auf Veränderungen untersucht werden können bzw. dass das System sehr langsam wird, da alle Fakten gleichwertig behandelt werden und keine Unterscheidung möglich ist. In Jadex können Fakten, bei denen eine hohe Wahrscheinlichkeit einer relevanten Änderung angenommen wird, in beliebigen Abständen aktualisiert werden. Dies kann durch Angabe einer „updaterate“ im entsprechenden Belief-Tag ausgelöst werden.

```
1  ...
2  <belief name="checkTime" class="Due" updaterate="1000">
3          <fact>Due.getInstance().getDue().getCheckTime()</fact>
4  </belief>
5   ...
```

In diesem Fall enthält der Belief-Fact einen Zeitpunkt, zu dem ein Plan überprüfen soll, ob einer der Fabriken einen neuen Auftrag ausschreiben will. Dieser Fakt wird hier alle 1000 Millisekunden überprüft.

Goals

Goals sind ein zentrales Konzept in Jadex. Sie werden hier als konkrete vorübergehende Pläne wahrgenommen. Diese Ziele befinden sich zur Laufzeit immer in einem der Zustände *Ziel erreicht*, *Zielerreichung nicht möglich* oder *Zielerreichung nicht mehr gewünscht*. Dies ist der gewünschten Handlungsfähigkeit in einer dynamischen Umgebung geschuldet. Damit kann ein Ziel jederzeit unterbrochen werden, wenn es die Fakten der Beliebase erfordern. Es gibt einen anwendungsspezifischen Goal-Deliberation-Mechanismus, der die Zustandsübergänge von allen angenommenen/ akzeptierten Goals überwacht. Es ist eine Art Übersicht, welche Goals aktiv sind und welche lediglich akzeptiert wurden. Einige Goals sind auch nur gültig, wenn bestimmte Vorbedingungen erfüllt sind. Das heißt sie sind abhängig von der aktuellen Beliefbase des Agenten. Sobald ein Goal ungültig wird, wird es ruhend gestellt, bis die Bedingungen (wieder) erfüllt sind. Grundsätzlich kann ein Agent auch widersprüchliche Ziele verfolgen.

```
1  ...
2  <performgoal name="OrderDueGoal" recur="true" recurdelay="100">
3          <creationcondition language="jcl">$beliefbase.checkTime &lt; $beliefbase.time</
               creationcondition>
4  </performgoal>
5   ...
```

Bei diesem Auszug des Fabrikagenten ist eine weitere Eigenschaft des Goal-Konzepts in Jadex erkennbar. Es handelt sich um ein *Perform-Goal*, einem von drei Zieltypen die Jadex unterstützt. Perform-Goals sind nicht zwangsläufig an einen Weltzustand gebunden. Anders ist es bei einem Maintain-Goal, dass einen Weltzustand erhalten oder einem Achieve-Goal, welches einen bestimmten Zustand herbeiführen soll.

Plans

Pläne beinhalten das „sichtbare“ Verhalten des Agenten. Ein Plan besteht immer aus einem Plankopf und einem Plankörper. Der Plankörper enthält eine sequentielle Folge von atomaren Aktionen. Wie zum Beispiel „sendeNachricht(Empfänger) “ oder „turnright(Gradzahl) “. Der Plankörper wird als gewöhnliche Java-Klasse umgesetzt. Da Jadex in Java [12] implementiert ist, können natürlich alle bekannten Packages von Java für die Implementation des Plankörpers verwendet werden. Sobald ein Plan ausgewählt wurde, wird der Plankörper sequentiell durchlaufen und die dort definierten Aktionen ausgeführt. Jadex stellt jedoch auch Mittel zur Verfügung, um die weitere Ausführung des Plankörpers zwischen einzelnen Aktionen zu unterbrechen.

Zur Verknüpfung dieses Plans mit dem Agenten ist der Plankopf notwendig. Er ist ist im Gegensatz zum Plankörper keine eigene Java-Klasse, sondern Teil des ADF.

```
1  ...
2  <plan name="Order_due_plan">
3          <parameter name="order1" class="Order1[]"></parameter>
4          <parameter name="time" class="long"></parameter>
5          <body class="Order_due_plan" />
6          <trigger>
7                  <goal ref="OrderDueGoal" />
8          </trigger>
9  </plan>
10  ...
```

Hier ist ersichtlich, das der Plan „OrderDuePlan“durch das entsprechende Ziel ausgelöst wird. Der Plankopf kann also Vorbedingungen enthalten kann, die zur Aktivierung des hinterlegten Plans erfüllt sein müssen und auch eine Priorität, die bei mehreren ausführbaren Plänen darüber entscheidet, welcher der Pläne vorrangig auszuwählen ist. Per default haben alle Pläne die Priorität 0.

Jadex unterscheidet zwischen 2 Arten von Plänen, einem Service Plan und dem Passive Plan. Letzterer entspricht dem Plantyp, der standardmäßig in PRS-Systemen verwendet wird. Er wird erst aktiviert und instanziert, wenn die Vorbedingungen des Plankopfes erfüllt sind. Im Gegensatz dazu wird der Service Plan beim Erzeugen des Agenten instanziert und existiert dann quasi als Dämon im Hintergrund.

Alle Pläne eines Agenten werden in einer Planbibliothek bereitgestellt. Ein Manager überprüft, welche Pläne aufgrund der Definition des Plankopfes im ADF ausführbar sind und stellt diese zur Auswahl zur Verfügung. Alle ausgeführten Pläne sind direkt mit Aktionen verknüpft. Ein Plan gilt dann als erreicht, wenn alle Aktionen ausgeführt wurden, unabhängig davon, ob die Aktionen die gewünschte Zustandveränderung der Welt erreicht haben.

Plankörper und die Verwendung von RSAL

Während also eine Funktion anhand der im ADF festgelegten Bedingungen darüber entscheidet, welcher Plan zur Ausführung kommt, wird im Plankörper die eigentliche Handlung beschrieben. An dieser Stelle kommt die Verwendung von RSAL zum Tragen. Anhand des bisher verwendeten Fabrikagenten lässt sich auch dies sehr gut exemplarisch darstellen. Die vorhandenen Rohstoffe des Szenarios werden in der Simulationsumgebung Stage durch farbige Markierungen dargestellt. Das Framework RSAL ermöglicht den Zu-

griff auf Stage und erzeugt mittels eines „BlobAgents“ diese Markierungen in der 2,5D Darstellung der Umgebung. Wenn die Fabrik eine Rohstoff-Lieferung annimmt, wird ein Plan ausgeführt, der die zugehörige Markierung in der Karte entfernt:

```
...
try
                {
                        existing_commodities.remove_commodity(gelieferteWare);
                        existing_commodities.setSoll(false);
                        Map<String, Object> args = new HashMap<String, Object>();
                        args.put("X", delivery.getX);
                        args.put("Y", delivery.getY);
                        args.put("host", "\"localhost\"");
                        args.put("port", 6665);
                        args.put("blob", delivery.getName);
        // begin: Verbindung zu RSAL
                        IGoal goal = createGoal("cms_create_component");
                        goal.getParameter("type")
                                        .setValue("jadex/agent/CollectAgent.class");
                        goal.getParameter("arguments").setValue(args);

                        try
                        {
                                System.out.println("dispatchSubgoal...");

                                dispatchSubgoalAndWait(goal);
        //end: Verbindung zu RSAL
                                System.out.println("Create CollectAgent ok");
                        } catch (GoalFailureException failure)
                        {
                                failure.printStackTrace();
...
```

Wie in dem Quellcode erkennbar ist, erfolgt die Einbindung von RSAL über die Erzeugung eines neuen untergeordneten Ziels. Als Argumente werden die Werte der, den Rohstoff repräsentierenden Markierung, mitgegeben. Analog wird auch die Strategie zur Kollisionsvermeidung innerhalb eines Plans angestoßen.

Zusammenführung einzelner Agenten zu einem MAS

Nachdem einzelne Agententypen definiert wurden, müssen diese nun zu einem MAS zusammengeführt werden. Dies geschieht ebenfalls mittels XML. Das Anwendungsszenario entspricht, wie auch die einzelnen Agenten, einer Aktiven Komponente (AC). AC‘s können in Hierarchien angeordnet werden. Für das MAS werden die benötigten Agenten als Subkomponenten der Anwendung eingebunden:

```
...
        </imports>
        <arguments>
                <argument name="port"> 6665 </argument>
                <argument name="host"> "localhost" </argument>
        </arguments>
        <componenttypes>
                <componenttype name="Environment0"
                        filename="rohstoffgenerator/Environment.agent.xml" />
                <componenttype name="Explore0" filename="jadex/agent/ExploreAgent.class" master
                    ="true"/>
```

```
11                  <componenttype name="Factory0" filename="factory/Factory.agent.xml" />
12                  <componenttype name="Factory1" filename="factory/Factory.agent.xml" />
13                  <componenttype name="Robot0" filename="robots/Forwarder.agent.xml"/>
14          </componenttypes>
15          <properties>
16                  <property name="clock" class="IFuture">
17                          SServiceProvider.getService($component.getServiceProvider(),
18                          IClockService.class, RequiredServiceInfo.SCOPE_PLATFORM)
19                  </property>
20          </properties>
21          <configurations>
22                  <configuration name="1Environment␣+␣2␣Factory␣virtual,␣1Forwarder">
23                          <components>
24                                  <component type="Robot0" master ="true"/>
25                                  <component type="Factory0" master="true">
26                                          <arguments>
27                                                  <argument name="order1">
28                                                          new Order1[]
29                                                          {
30
31  new Order1($properties.clock.getTime()+3,"Agent␣1␣erster␣Auftrag",
32  new Date($properties.clock.getTime()+60000), 100, 120, true, $properties.clock),
33  new Order1($properties.clock.getTime()+1000,"Agent␣1␣zweiter␣Auftrag",
34  new Date($properties.clock.getTime()+60000), 40, 60, true, $properties.clock),
35  new Order1($properties.clock.getTime()+2000,"Agent␣1␣dritter␣Auftrag",
36  new Date($properties.clock.getTime()+60000), 5, 10, true, $properties.clock),
37  new Order1($properties.clock.getTime()+800,"Agent␣1␣vierter␣Auftrag",
38  new Date($properties.clock.getTime()+60000), 30, 65, true, $properties.clock)
39                                                          }
40                                                  </argument>
41                                          </arguments>
42                                  </component>
43                                  <component type="Factory1" master="true">
44                                          <arguments>
45                                                  <argument name="order1" >new Order1[]
46                                                          {
47
48  new Order1($properties.clock.getTime()-3,"Agent␣2␣erster␣Auftrag",
49  new Date($properties.clock.getTime()+90000), 100, 120, true, $properties.clock),
50  new Order1($properties.clock.getTime()+2900,"Agent␣2␣zweiter␣Auftrag",
51  new Date($properties.clock.getTime()+70000), 40, 60, true, $properties.clock),
52  new Order1($properties.clock.getTime()+1000,"Agent␣2␣dritter␣Auftrag",
53  new Date($properties.clock.getTime()+80000), 5, 10, true, $properties.clock),
54  new Order1($properties.clock.getTime()+40,"Agent␣2␣vierter␣Auftrag",
55  new Date($properties.clock.getTime()+100000), 30, 65, true, $properties.clock)
56                                                          }</argument>
57                                          </arguments>
58                                  </component>
59                          </components>
60                  </configuration>
61          </configurations>
62  </applicationtype>
63  ...
```

In diesem Beispiel wird noch einmal klar, das eine Agentendefinition eine Schablone für jede Entität dieses Typs ist. Da in dieser Konfiguration mehrere Fabriken mit unterschiedlichen Anforderungen existieren sollen, werden auch mehrere Agenten diesen Typs erzeugt (Zeile 25-42 und 43-58) und deren individuelle Bedarfsanforderungen als Argumente übergeben.
Mit dem Start dieser Anwendung im JCC, werden alle Agenten und sonstigen Komponenten (Services oder Capabilities) initialisiert.

Zusammenfassung

In diesem Kapitel wurde die praktische Umsetzung der Modellierung eines MAS vorgestellt. Der Entwurf sollte einem methodischen Vorgehen entsprechen. Hiefür gibt es verschiedene Möglichkeiten, die in Abhängigkeit des verwendeten Agentenframeworks gewählt werden sollte. In einer Untersuchung von Braubach wurde eine Auswahl dieser Methoden auf ihre Tauglichkeit in Bezug auf Jadex geprüft. Dabei schnitt Tropos am besten ab und wurde daher auch in dieser Arbeit verwendet. In der Implementationsphase wurden die herausgearbeiteten Ziele und Pläne einzelner Agenten mittels Jadex umgesetzt und in einem XML-File zu einer Multi-Agenten Anwendung zusammengeführt.

5 Evaluation

In diesem Kapitel wird die Handhabung und Eignung der, in dieser Arbeit verwendeten Tools, zur Modellierung und Entwicklung von BDI-Agenten für ein MRS bewertet und eine Einschätzung zur praktischen Umsetzung eines solchen Vorhabens mit aktuell verfügbaren Mitteln vorgenommen.

Tropos

Tropos eignete sich sehr gut, für das erste Erfassen der Aufgabenstellung. Es bietet zunächst eine relativ natürliche Herangehensweise und hilft bei der Zuordnung der geforderten Eigenschaften zu Agententypen und Capabilities und unterstützt den Entwickler beim Erstellen von Planhierarchien. Allerdings wird die Darstellung relativ unübersichtlich, wenn sehr viele Abhängigkeiten zwischen unterschiedlichen Akteuren bzw. Agententypen bestehen. Die Abgrenzung der einzelnen Phasen ist nicht ganz so natürlich, wie zunächst angenommen und die Grenzen dieser Methode zeigen sich in der detaillierten Designphase, in der dann auf die bewährten Modellierungsmöglichkeiten der objektorientierten Programmierung zurückgegriffen wird.

Jadex

Im XML-Schema von Jadex werden die drei Säulen des BDI-Konzepts sehr direkt abgebildet, sodass die Übertragung des Systementwurfs in Quellcode weitestgehend natürlich umgesetzt werden kann. Da mit der Verwendung von Java nicht nur eine plattformunabhängige, sondern auch bekannte Programmiersprache verwendet wird, ist die Programmierung der Plankörper unproblematisch umzusetzen. Allerdings erfordert die API von Jadex und insbesondere das XML-Schema eine gewisse Einarbeitungszeit. Jadex hat sich als umfassendes Agentenframework in der Forschung etabliert und wird daher stetig weiterentwickelt. Diese Weiterentwicklung beinhaltet neben Erweiterungen und Anpassungen der API auch sinnvolle Tools zur Unterstützung des Programmierers. Da Agenten üblicherweise nur in Form reiner Softwareagenten anzutreffen sind, gibt es jedoch so gut wie keine Middleware zur Übertragung der Agentensoftware auf ein MRS und eine Abwärtskompatibilität zu den wenigen bestehenden Frameworks, wie zum Beispiel RSAL, kann nicht gewährleistet werden, da hierfür noch keine ausreichend große Community existiert.

RSAL

Die Basisverhalten von RSAL können in Form von untergeordneten Zielen sehr einfach in den Plankörper von Agenten eingebunden werden. Somit stellt es eine sinnvolle Basis zur Implementation von Top-Level Verhalten dar. RSAL basiert allerdings auf JADEX 2.0rc6, das noch keinen Component-Kernel für BDI-Agenten enthält, um die Kommunikation im MAS über Services zu ermöglichen. Daher musste zunächst RSAL an die aktuelle Version von JADEX (2.1) angepasst werden, um das gewünschte MRS realisieren zu können.

Fazit

Der Einsatz von Roboter birgt ein großes Potential, das bislang nicht ausgeschöpft wird. Die Robotikforschung stellt, bezugnehmend auf die Konstruktion, Roboter zur Verfügung, die mit Ausnahme spezieller Bauarten, wie humanoide Roboter, eine solide Basis für die Erarbeitung neuer Einsatzmöglichkeiten und Anwendungsfelder darstellt. Für Basisverhalten wie der kollisionsfreien Pfadplanung, gibt es bereits bewährte Lösungsansätze, deren Funktionalität derzeit in vielen neuen Anwendungskontexten erprobt wird. Leider müssen diese, aufgrund fehlender Standards für den Zugriff auf die Hardware der Roboter, meist für jeden Robotertyp neu implementiert werden. Dies erschwert die Entwicklung und Umsetzung komplexer Verhalten und ist ein möglicher Grund dafür, dass der Einsatz von Roboter häufig nur im Kontext spezieller, für den Einsatz von Robotern vorgesehener Umgebungen, gesehen wird. Dennoch liegt der nächste notwendige Schritt der anwendungsorientierten Robotik darin, die Funktionalität der Roboter um Konzepte zu erweitern, die eine gefahrlose und sinnvolle Integration von Robotersystemen in den Alltag von Menschen ermöglichen. Eine neue Positionierung von Robotersystemen als Assistenten im menschlichen Umfeld, wie beispielsweise bei rettenden Robotersystemen, beinhaltet den Einsatz in einer komplexen, dynamischen Umgebung. Daher müssen Roboter nicht nur intelligenter sondern auch sicherer werden um keine Menschen direkt oder indirekt zu verletzen. Um diesen Anforderungen gerecht zu werden, müssen Roboter sowohl über reaktive als auch proaktive Verhaltensweisen verfügen, indem sie eine stetige Aktualisierung des Weltmodells vornehmen um die Folgen ihres Handelns in den Entscheidungsfindungsprozess einbeziehen zu können und angemessen zu reagieren. Diese Arbeit zeigt, dass ein solches Verhalten mit dem hier vorgestellten Agentenparadigma realisiert werden kann. Das BDI-Konzept erleichtert zusätzlich die Umsetzung, da es an

die menschliche Denkweise angelehnt ist. Frameworks wie Jadex ermöglichen die Implementation solcher Verhalten und bieten durch die Verwendung von Java eine gewisse Kompatibilität zu verschiedenen Roboterplattformen. Da Java weit verbreitet ist, entfällt für die Programmierer eine längere Einarbeitungszeit und ein bestehendes Verhalten kann aufgrund der Struktur des Quellcodes und der Verwendung von XML sehr einfach um neue Pläne ergänzt werden.

Diese Arbeit zeigt, dass die Umsetzung eines BDI-Verhaltens für ein MRS grundsätzlich möglich ist. Dennoch führen fehlende Standards und die Tatsache, dass bisher kaum Versuche unternommen werden, komplexe Verhalten, insbesondere mittels Agenten, auf Robotersystemen umzusetzen dazu, dass es wenige Tools gibt, die dieses Vorhaben unterstützen. Dadurch ist der Programmierer bei der Auswahl eines geeigneten MRS und kompatibler Frameworks sehr eingeschränkt. In diesem konkreten Fall war im Vorfeld die Anpassung der Middleware RSAL an das Agentenframework notwendig, da aufgrund der bisher kleinen Community, die diese Middleware nutzt, eine fortlaufende Anpassung in keinem angemessenem Verhältnis steht. Es ist jedoch davon auszugehen, dass in den kommenden Jahren ein vermehrtes Interesse an der Entwicklung komplexer Verhalten entstehen wird und damit auch vermehrt geeignete Tools entwickelt werden.

6 Ausblick

In diesem Abschnitt sollen mögliche Erweiterungen und praktische Anwendungsmöglichkeiten der vorliegenden Arbeit aufgegriffen werden. Es wurde anhand der Umsetzung eines logistischen Szenarios gezeigt, dass die Umsetzung eines BDI-Verhaltens für ein MRS möglich ist. Dafür wurde das gewählte Szenario bewusst abstrakt gewählt. Lediglich die Wahl der Bezeichner für die Akteure impliziert einen konkreten Einsatzkontext. Dabei ist diese Struktur auch in anderen Bereichen anzufinden und bietet daher Potential für reale Anwendungssituationen und zahlreiche Forschungsarbeiten. In den einleitenden Kapiteln wurde bereits die Notwendigkeit der Anpassung von Robotersystemen an komplexe Umgebungen beschrieben. Nachfolgend werden konkrete Anwendungsbeispiele aufgezeigt:

Auffüllen von Regalen, zum Beispiel im Supermarkt:
Sensoren können erfassen, ob ein Produkt in absehbarer Zeit im Regal vergriffen ist und fordern Nachschub für dieses bestimmte Produkt an. Mobile Roboter erfüllen diese Bedarfsanforderung, indem sie die entsprechenden Produkte aus dem Lager oder anderen Auslageflächen holen und das Regal auffüllen.
Diese reale Anwendung kann mittels mobiler Roboter, die mit einem Greifer und einer Ladefläche ausgestattet sind, umgesetzt werden. Die Anforderungen kommen spontan. Es ist denkbar, dass mehrere Standorte das gleiche Produkt anfordern und dieses Produkt nicht mehr in ausreichender Menge im Lager vorrätig ist. In diesem Fall müsste das MRS aufgrund seiner Wissensbasis entscheiden, welche Verkaufsfläche bevorzugt aufgefüllt werden sollte. Die Umgebung wäre sehr dynamisch, da die Platzierung der Waren monatlich variieren kann und der Einsatz des MRS in unmittelbarer Nähe von Menschen erfolgt.

Be- und Entladung von Flugzeugen:
Auch in diesem Beispiel haben wir es mit einer dynamischen Umgebung zu tun, da sich andere Fahrzeuge oder Menschen auf dem Rollfeld befinden. Ein MRS kann eingesetzt werden, um Flugzeuge zu be- oder entladen. Die Anforderungen kommen ebenfalls spontan, da die Ankunftszeit der Flugzeuge Verspätungen unterliegen kann und die Stellplätze variabel zugeordnet werden. Zur Einhaltung strenger Entladezeiten könnte ein intelligentes MRS die Anzahl der verwendeten mobilen Roboter anpassen.

Diese beiden Beispiele zeigen jeweils einen Anwendungsfall, der sehr gut durch Robotersysteme erledigt werden könnte. Die dynamisch ändernden Randbedingungen und die Einbettung in eine sicherheitskritische Umgebung, in der Menschen neben Robotern agieren, sind ein wichtiger Grund, der den Einsatz von Roboter für solche Zwecke bisher unmöglich macht.
Diese sehr konkreten Anwendungsfälle stellen eine Erweiterung des implementierten logistischen Szenarios aus dieser Arbeit dar. Die Erforschung der Praktikabilität sowohl aus Sicht der Robotik, als auch aus einer betriebswirtschaftlichen Perspektive, bietet viele interessante Forschungsmöglichkeiten. Desweiteren können die Grenzen der Funktionalität des implementierten Verhaltens hinsichtlich der Skalierbarkeit getestet werden und ein Vergleich der Umsetzung dieses Szenarios mittels anderer Ansätze aus dem Bereich der Künstlichen Intelligenz angestrebt werden.

7 Anhang

ADF des Fabrik-Agenten

```
1  <?xml version="1.0" encoding="UTF-8"?>
2          <!-- Fordert Rohstoffe an und nimmt diese entgegen -->
3
4  <agent xmlns="http://jadex.sourceforge.net/jadex" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance"
5          xsi:schemaLocation="http://jadex.sourceforge.net/jadex
6  ~~~~~~~~~~~~~~~~~~~~~~~~~http://jadex.sourceforge.net/jadex-component-2.0.xsd"
7          name="Factory" package="factory">
8          <imports>
9                  <import>java.util.*</import>
10                 <import>jadex.commons.*</import>
11                 <import>general_data.*</import>
12                 <import>jadex.rules.rulesystem.rules.functions.*</import>
13                 <import>jadex.application.runtime.*</import>
14                 <import>order.application.xml</import>
15                 <import>jadex.service.ReceiveNewGoalService</import>
16                 <import>jadex.service.IReceiveNewGoalService</import>
17                 <import>beliefbase.*</import>
18                 <import>factory.*</import>
19                 <import>jadex.bdi.runtime.*</import>
20                 <import>jadex.bridge.service.types.clock.*</import>
21                 <import>jadex.bridge.service.*</import>
22                 <import>jadex.base.fipa.*</import>
23                 <import>jadex.bdi.planlib.protocols.*</import>
24                 <import>jadex.bridge.service.clock.IClockService</import>
25         </imports>
26         <capabilities>
27                 <capability name="cms" file="jadex/bdi/planlib/cms/CMS.capability.xml" />
28         </capabilities>
29         <beliefs>
30                 <beliefset name="board" class="Board_Orders1">
31                         <fact>Board_Orders1.getInstance().getOpenOrders()</fact>
32                 </beliefset>
33                 <belief name="due" class="Due">
34                         <fact>Due.getInstance().getDue()</fact>
35                 </belief>
36                 <beliefset name="existingCommodities" class="Board_Found_Commodities">
37                         <fact>Board_Found_Commodities.getInstance().getExistingCommodity()
38                         </fact>
39                 </beliefset>
40                 <belief name="checkTime" class="Due" updaterate="1000">
41                         <fact>Due.getInstance().getDue().getCheckTime()</fact>
42                 </belief>
43                 <belief name="time" class="long" updaterate="100">
44                         <fact>$scope.getTime()</fact>
45                 </belief>
46                 <belief name="order1" class="Order1[]">
47                         <fact>$args.order1</fact>
48                 </belief>
49         </beliefs>
50         <goals>
51                 <achievegoalref name="cms_create_component">
52                         <concrete ref="cms.cms_create_component" />
53                 </achievegoalref>
54                 <performgoal name="AcceptDeliveryGoal" />
55                 <performgoal name="EvaluationGoal" />
```

```
56                  <performgoal name="NewOrderGoal" recur="true" recurdelay="1000">
57                          <parameter name="order1" class="Order1">
58                                  <value>$order1</value>
59                          </parameter>
60                  </performgoal>
61                  <performgoal name="OrderDueGoal" recur="true" recurdelay="100">
62                          <creationcondition language="jcl">$beliefbase.checkTime
63                                  &lt; $beliefbase.time</creationcondition>
64                  </performgoal>
65          </goals>
66          <plans>
67                  <plan name="Accept_delivery_plan">
68                          <body class="Accept_delivery_plan" />
69                          <trigger>
70                                  <goal ref="AcceptDeliveryGoal" />
71                          </trigger>
72                  </plan>
73                  <plan name="Evaluation_plan">
74                          <body class="Evaluation_plan" />
75                          <trigger>
76                                  <goal ref="EvaluationGoal" />
77                          </trigger>
78                  </plan>
79                  <plan name="New_order_plan">
80                          <parameter name="order1" class="Order1"></parameter>
81
82                          <body class="New_order_plan" />
83                          <trigger>
84                                  <goal ref="NewOrderGoal" />
85                          </trigger>
86                  </plan>
87                  <plan name="Order_due_plan">
88                          <parameter name="order1" class="Order1[]"></parameter>
89                          <parameter name="time" class="long"></parameter>
90                          <body class="Order_due_plan" />
91                          <trigger>
92                                  <goal ref="OrderDueGoal" />
93                          </trigger>
94                  </plan>
95          </plans>
96          <services>
97                  <requiredservice name="clockservice" class="IClockService">
98                          <binding scope="platform" />
99                  </requiredservice>
100         </services>
101         <configurations>
102                 <configuration name="default">
103                         <goals>
104                                 <initialgoal ref="NewOrderGoal" />
105                                 <initialgoal ref="OrderDueGoal" />
106                         </goals>
107                 </configuration>
108         </configurations>
109 </agent>
```

Konfiguration eines MAS

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <applicationtype xmlns="http://jadex.sourceforge.net/jadex"
4          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5          xsi:schemaLocation="http://jadex.sourceforge.net/jadex␣http://jadex.sourceforge.net/
               jadex-application-2.1.xsd"
6          name="bac_behaviour" package="scenario">
7          <imports>
```

```
8                 <import>jadex.agent.*</import>
9                 <import>rohstoffgenerator.Environment</import>
10                <import>factory.Factory</import>
11                <import>robots.Forwarder</import>
12                <import>robots.Forwarder.agent.xml</import>
13                <import>general_data.*</import>
14                <import>java.util.*</import>
15                <import>jadex.commons.future.*</import>
16                <import>jadex.bridge.service.*</import>
17                <import>jadex.bridge.service.types.clock.IClockService</import>
18                <import>jadex.bridge.service.search.*</import>
19                <import>general_data.Order1</import>
20                <import>factory.*</import>
21                <import>factory.Due</import>
22        </imports>
23        <arguments>
24                <argument name="port"> 6665 </argument>
25                <argument name="host"> "localhost" </argument>
26        </arguments>
27        <componenttypes>
28                <componenttype name="Environment0"
29                        filename="rohstoffgenerator/Environment.agent.xml" />
30                <componenttype name="Factory0" filename="factory/Factory.agent.xml" />
31                <componenttype name="Factory1" filename="factory/Factory.agent.xml" />
32                <componenttype name="Forwarder0" filename="robots/Forwarder.agent.xml"></
                      componenttype>
33        </componenttypes>
34        <properties>
35                <property name="clock" class="IFuture">
36                        SServiceProvider.getService($component.getServiceProvider(),
37                        IClockService.class, RequiredServiceInfo.SCOPE_PLATFORM)
38                </property>
39        </properties>
40
41        <configurations>
42                <configuration name="Environment␣+␣2␣Factory␣virtual,␣1Robot">
43                        <components>
44                                <component type="Environment0" master="true" />
45                                <component type="Forwarder0" master ="true"/>
46
47                                <component type="Factory0" master="true">
48                                        <arguments>
49                                                <argument name="order1">
50                                                        new Order1[]
51                                                        {
52
53                        new Order1($properties.clock.getTime()+3,"Agent␣1␣erster␣Auftrag",
                                                                                        new date($
                              properties.clock.getTime()+60000), 100, 120, true, $properties.
                              clock),
                                                                        new Order1($properties.clock.
                              getTime()+5000,"Agent␣1␣zweiter␣Auftrag",
                                                                                new Date($properties.clock.getTime()
                              +60000), 40, 60, true, $properties.clock),
54                        new Order1($properties.clock.getTime()+15700,"Agent␣1␣dritter␣Auftrag",
                                                                                        new Date($properties.clock
                              .getTime()+60000), 5, 10, true, $properties.clock),
55                        new Order1($properties.clock.getTime()+30000,"Agent␣1␣vierter␣Auftrag",
56                        new Date($properties.clock.getTime()+60000), 30, 65, true, $properties.
                              clock)
57
58                                                        }
59                                                </argument>
60                                        </arguments>
```

```
61                                  </component>
62                                  <component type="Factory1" master="true">
63                                          <arguments>
64                                                  <argument name="order1" >new Order1[]
65                                                          {
66
67                                                                  new Order1($properties.clock.
                                                                        getTime()+2900,"Agent 2
                                                                        erster Auftrag",
68                                                                  new     Date($properties.clock.
                                                                        getTime()+90000), 100,
                                                                        120, true, $properties.
                                                                        clock),
69                                                                  new Order1($properties.clock.
                                                                        getTime()+19000,"Agent 2
                                                                        zweiter Auftrag",
70                                                                  new Date($properties.clock.
                                                                        getTime()+70000), 40, 60,
                                                                        true, $properties.clock),
71                                                                  new Order1($properties.clock.
                                                                        getTime()+23000,"Agent 2
                                                                        dritter Auftrag",
72                                                                  new Date($properties.clock.
                                                                        getTime()+80000), 5, 10,
                                                                        true, $properties.clock),
73                                                                  new Order1($properties.clock.
                                                                        getTime()+40000,"Agent 2
                                                                        vierter Auftrag",
74                                                                  new Date($properties.clock.
                                                                        getTime()+100000), 30, 65,
                                                                         true, $properties.clock)
75
76                                                          }</argument>
77                                          </arguments>
78                                  </component>
79                          </components>
80                  </configuration>
81          </configurations>
82  </applicationtype>
```

Danksagung

Ich möchte mich bei meinem Erstbetreuer Prof. Dr. Jianwei Zhang und dem Arbeitsbereich TAMS (Technische Aspekte Multimodaler Systeme) für die Betreuung meiner Bachelorarbeit und das zur Verfügungstellen der Hardware bedanken.
Ich danke meinem Zweitbetreuer Dr. Alexander Pokahr und Dr. Lars Braubach vom Arbeitsbereich VSIS (Verteilte Systeme und Informationssysteme) für die Unterstützung hinsichtlich der Modellierung der Agenten insbesondere zur Verwendung von Jadex.
Besonderen Dank an Dipl.-Inf. Denis Klimentjew und M.Sc.Informatik Sebastian Rockel für die Hilfe bei der Themenfindung und Vorbereitung. Euer Korrekturlesen und Mut machen, während des gesamten Schreibprozesses haben mir sehr geholfen.

Literaturverzeichnis

[1] Website. http://www.iros2012.org/site/, abgerufen am 06.03.2012.

[2] Website. http://www.icra2012.org/, abgerufen am 06.03.2012.

[3] Website. http://www.robocup.org/, abgerufen 08.03.2012.

[4] Website. http://asimo.honda.com/, abgerufen 08.03.2012.

[5] Website. http://www.ros.org/wiki/, abgerufen am 01.03.2012.

[6] Website. http://aosgrp.com/products/jack/index.html, abgerufen 01.03.2012.

[7] Website. http://jason.sourceforge.net/Jason/Jason.html, abgerufen 01.03.2012.

[8] Website. http://www.fipa.org/, abgerufen am 01.03.2012.

[9] Website. http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview, abgerufen am 15.02.2012.

[10] Website. http://googleblog.blogspot.de/2010/10/what-were-driving-at.html, abgerufen am 24.03.2012.

[11] Website. http://www.mobilerobots.com/ResearchRobots/P3AT.aspx, abgerufen am 01.03.2012.

[12] Website. http://openjdk.java.net/, abgerufen am 10.03.2012.

[13] Bernhard Bauer. Uml class diagrams revisited in the context of agent-based systems. In *Agent-Oriented Software Engineering II*, Second International Workshop, AOSE 2001, Michael J.Wooldridge, Gerhard Weiß, Paolo Ciancarini, pages 101–118. Mai,2001.

[14] Michael E. Bratman. *Intention, Plans and Practical Reason.* 1987.

[15] Lars Braubach. *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme.* 1st edition, Dezember 2007.

[16] Richard T. Vaughan Brian Gerkey and Andrew Howard. "the player/stage project: Tools for multi-robot and distributed sensor systems". In *Proceedings of the 11th International Conference on Advanced Robotics, Coimbra, Portugal (ICAR'03)*, pages 317–323. http://www.isr.uc.pt/icar03/, June 2003.

[17] Rodney A. Brooks. Artificial intelligence. In *Artificial Intelligence Vol. 47*, Vol. 47, Issues 1-3, pages 139–159. Januar 1991.

[18] George F. Coulouris. *Distributed Systems.* Addison-Wesley, 5 edition, 2012.

[19] Phillip John McKerrow. *Introduction to Robotics.* Addison-Wesley Publishers Ltd., 1991.

[20] Robin R. Murphy. *Introduction to AI Robotics.* MIT Press, Cambridge, MA, USA, 1st edition, 2000.

[21] M. Wooldridge N. Jennings, Katia P. Sycara. A roadmap of agent research and development. In *Journal of Autonomous Agents and Multi-Agent Systems 1(1998).* 1998.

[22] Paolo Giorgini Fausto Giunchiglia John Mylopoulos Paolo Bresciani, Anna Perini. Autonomous Agents and Multi-Agent Sytems, 8, 203–236, 2004. Kluwer Academic Publishers.

[23] Anand Rao. *AgentSpeak(L): BDI agents speak out in a logical computable language*, volume 1038/1996. Springer Berlin / Heidelberg, http://dx.doi.org/10.1007/BFb0031845, 1996.

[24] Tony Johnson Richard Murch. *Agententechnologie: Die Einführung.* Addison-Wesley, 2000.

[25] Sebastian Rockel. A multi-robot platform for mobile robots with multi-agent technology. Master's thesis, Technical Aspects of Multimodal Systems, Distributed Systems and Information Systems, University of Hamburg, Juli 2011.

[26] P. Norvig S. Russel. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 2003.

[27] Michael Wooldridge. *An Introduction to MultiAgent Systems.* John Wiley and Sons Ltd, second edition, 2009.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.
Ich bin mit der Einstellung in den Bestand der Bibliothek des Fachbereichs einverstanden.

Anja Richter

Hamburg, den 13. April 2012