Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter in der Arbeitsgruppe Wirtschaftsinformatik, insbesondere CIM, am Heinz Nixdorf Institut der Universität Paderborn und des Fraunhofer Anwendungszentrums für logistikorientierte Betriebswirtschaft.

Für die während dieser Zeit erfahrene persönliche und fachliche Unterstützung möchte ich mich in erster Linie bei Herrn Prof. Dr.-Ing. habil. Wilhelm Dangelmaier, Leiter der Arbeitsgruppe Wirtschaftsinformatik, insbesondere CIM, am Heinz Nixdorf Institut der Universität Paderborn und des Fraunhofer Anwendungszentrums für logistikorientierte Betriebswirtschaft, bedanken. Insbesondere sein Vertrauen, seine stete Gesprächsbereitschaft und der gewährte Freiraum bildeten die Grundpfeiler bei der Erstellung dieser Arbeit. Herrn Prof. Dr.-Ing. Jürgen Gausemeier danke ich sehr herzlich für die eingehende Durchsicht der Dissertation und die Übernahme des Koreferats.

Weiterhin gilt mein Dank allen wissenschaftlichen Mitarbeitern der Arbeitsgruppe, die durch viele gemeinsame Diskussionen und wertvolle Anregungen zur Entstehung dieser Arbeit beigetragen haben. Des Weiteren bedanke ich mich bei Herrn Dipl.-Wirt. Ing. Andreas Schmidt für die gute Zusammenarbeit, die zahlreichen, gemeinsamen Veröffentlichungen und die interessanten Diskussionen im Rahmen des Sonderforschungsbereichs 614, der die Grundlage dieser Dissertation bildete. Herrn Hendrik Renken, Herrn Daniel Brüggemann und Herrn Heiner Giefers danke ich für ihre Unterstützung bei der Umsetzung der in dieser Arbeit entwickelten Konzepte.

Mein besonderer Dank gilt meiner lieben Ehefrau Regina, die immer verständnisvoll war und durch ihre liebevolle Art mir kontinuierlich neue Kraft zur Durchführung meiner Dissertation gegeben hat. Sie hat mir immer den erforderlichen Rückhalt gegeben, obwohl sie auf manche gemeinsame Stunde verzichten musste.

Mein weiterer besonderer Dank gilt meinen Eltern, denen ich meine gesamte Ausbildung bis zur Promotion verdanke. Ohne ihre Unterstützung hätte ich das Erreichte nie geschafft.

Paderborn, 26.10.2006

Geleitwort des Herausgebers

Die Industrie- und Dienstleistungsgesellschaft ist einem stetigen Wandel unterworfen. Die Herausforderungen an sie werden immer komplexer. Das äußert sich im verstärkten internationalen Wettbewerb, aber auch in dem Bestreben der Gesellschaft, das Erreichte zu sichern. Adäquate Problemlösungen sind daher in zunehmender Weise nur fachübergreifend realisierbar. Im Heinz Nixdorf Institut leisten wir mit der interdisziplinären Zusammenarbeit vor allem zwischen der Informatik und den Ingenieur- und Wirtschaftswissenschaften unseren Beitrag zur Bewältigung dieser Aufgaben.

Der Forschungsschwerpunkt der von mir geleiteten Fachgruppe „Wirtschaftsinformatik, insbesondere CIM“ liegt dabei auf allen technisch-betriebswirtschaftlichen Fragen, die bei der Gestaltung und Durchführung von inner- und überbetrieblichen Unternehmensprozessen auftreten und mittels innovativer Informationstechnik einer Lösung zugeführt werden können.

Einen wichtigen Teilbereich der inner- und überbetrieblichen Unternehmensprozesse stellt dabei der Transport, sowohl von Gütern als auch von Personen, dar. Das Ziel von Herrn Scheideler bestand dabei darin, ein Transportsystem zu entwickeln, das in der Lage ist sich eigenständig, sowohl technisch als auch logistisch, den Anforderungen der Umwelt dynamisch anzupassen. Dies geschieht durch eine ständige Verarbeitung der gesammelten Erfahrungen im Online-Betrieb. Die Erforschung dieses Bereiches ist umso wichtiger, da Prozesse komplexer werden und sich zentral fast nicht mehr steuern lassen. Die Dezentralisierung von Prozessen ist demnach stetig auf dem Vormarsch. Um hierbei nicht höhere Komplexitätskosten (z.B. mehr Personal) in Kauf nehmen zu müssen, wird man in der Zukunft nicht um eine Steigerung der Selbstständigkeit von technischen Systemen umher kommen.

Herr Scheideler widmete seine Arbeit dieser Problematik, indem er einen Beitrag zur Selbstoptimierung von Fahrzeugen aufgrund von Erfahrungen leistete. Hierzu wurden die gängigen Methoden der Informatik (Methoden der Wissensbasierten Systeme, Such- und Explorationsalgorithmen) mit den mechatronischen Anforderungen eines Fahrzeugs verknüpft. Herr Scheideler validierte seine Untersuchungen sowohl theoretisch als auch praktisch am Prüfstand.

Paderborn, 26.10.2006 Prof. Dr.-Ing. habil. Wilhelm Dangelmaier

Ein Beitrag zur erfahrungsbasierten Selbstoptimierung einer Menge technisch homogener fahrerloser Fahrzeuge

Dissertation

zur Erlangung der Würde eines

DOKTORS DER WIRTSCHAFTSWISSENSCHAFTEN

(Dr. rer. pol.)

der Universität Paderborn

vorgelegt von

Dipl.-Wirt.-Ing. Peter Scheideler

33161 Hövelhof

Paderborn, 26.10.2006

Dekan:	Prof. Dr. Peter F. E. Sloane
Referent:	Prof. Dr.-Ing. habil. W. Dangelmaier
Koreferent:	Prof. Dr.-Ing. J. Gausemeier

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AMS	Autonomes Mechatronisches System
CBR	Case-Based Reasoning
CSP	Constraint Satisfaction Problem
DAML	DRAPA Agent Markup Language
DRAPA	Defense Advanced Research Projects Agency
ERM	Entity Relationship Model
FIPA	Foundation for Intelligent Physical Agents
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
MFM	Mechatronisches Funktionsmodul
OCM	Operator-Controller-Modul
Oil	Ontology Inference Layer
RDF	Resource Description Framework
URI	Unified Resource Identifier
VMS	Vernetztes Mechatronisches System

1 Einleitung

Im beginnenden 21. Jahrhundert sind zwei Tendenzen im Bereich der Verkehrs- und Transportnetze zu erkennen: das immer größere Verkehrsaufkommen und die fortschreitende Automatisierung [Ste04-ol]. Bei der Automatisierung geht es um eine intelligente, computergestützte Steuerung der in den Verkehrs- und Transportnetzen operierenden Fahrzeuge. Diese Fahrzeuge werden oft als Automated Guided Vehicles (AGVs) bezeichnet [GKM+04], [Hol86]. Für diese fahrerlosen Fahrzeuge ist es notwendig, dass sie ihre Entscheidungen selbständig treffen, bewerten und verbessern können. Dies gilt insbesondere dann, wenn es darum geht, zur Laufzeit auf die dynamischen Veränderungen der Umwelt zu reagieren [ANS00].

Aufgrund der technischen Komplexität werden diese fahrerlosen Fahrzeuge in dieser Arbeit hierarchisch in einzelne Module (z.B. Antriebs- und Bremsmodul, Spurführungsmodul, Feder- und Neigemodul) untergliedert. Dies hat den Vorteil, dass Probleme speziellen Modulen zugeordnet und damit gesondert untersucht werden können. So ist das Antriebs- und Bremsmodul verantwortlich für das Bremsen und Beschleunigen und das Feder- und Neigemodul verantwortlich für den Fahrkomfort. Jedes dieser Module muss für sich in der Lage sein, selbständig einen möglichst optimalen Arbeitspunkt zu finden. Eine Möglichkeit dieser Selbstoptimierung besteht in der modellbasierten Optimierung. „Sie erfolgt auf Grundlage ausformulierter mathematischer Modelle. Es existieren detaillierte System- und Umfeldmodelle und symbolische mathematische Formalisierungen physikalischer Effekte“ [SFB04, S.24]. Dies setzt jedoch voraus, dass das mathematische Modell das System und das Umfeld zu 100% darstellen kann. Sind diese Modelle unvollständig oder fehlen ihnen Parameter, können die Systeme (hier die Module) ihren optimalen Zustand nicht einnehmen. In dieser Arbeit wird daher ein Ansatz verfolgt, in dem die Module ausschließlich durch eine Rückkopplung der Erfahrungen lernen und sich dadurch selbständig verbessern. Dieser Prozess führt zu einer kontinuierlichen erfahrungsbasierten Selbstoptimierung des Verhaltens der einzelnen Module und damit zu einer insgesamten erfahrungsbasierten Selbstoptimierung des Verhaltens eines fahrerlosen, schienengeführten Fahrzeugs.

Die Grundlagen zur Erreichung einer erfahrungsbasierten Selbstoptimierung bilden die Bereiche der *Erfahrungsakquisition*, *Wissensrepräsentation*, *Erfahrungsaustausche* und *Erfahrungsverarbeitung* [Hel96]. Bei der Erfahrungsakquisition ist es das vorrangige Ziel, Erfahrungen zu generieren. Dies geschieht mittels geeigneter Explorationsstrategien oder der Kommunikation mit anderen Fahrzeugen. Dieser Kommunikationsprozess stellt sich jedoch als schwierig heraus, wenn es sich um mehrere, heterogene Wissensbasen handelt. Die Erfahrungsakquisition über mehrere Fahrzeuge kann dann vor allen durch die Heterogenität der Wissensrepräsentation, die Heterogenität der Implementierung, lexikalische Probleme, Synonyme, implizites Wissens und das nicht Vorhandensein von Allgemeinwissen [Gom98] gestört werden. In diesem Fall ist es unerlässlich, einen Weg zu finden, *heterogene Wissensbasen* so miteinander zu verknüpfen, dass dennoch ein eindeutiger Erfahrungsaustausch für die weitere *Verarbeitung der Erfahrungen* stattfinden kann. Bei der

Erfahrungsverarbeitung handelt es sich um jene Wissenschaft, die sich mit der Automatisierung der Transformation von Erfahrungen in Wissen befasst, um sinnvolles Handeln und Entscheiden und damit die Akquisition neuer Erfahrungen zu ermöglichen [GRS03], [Hel96].

Klassische Verfahren im Bereich der Erfahrungsverarbeitung sind in dieser Arbeit nur bedingt einsetzbar, da sie oft von homogenen Wissensbasen und einer zentralen Datenhaltung ausgehen. Diese Vorstellungen sind idealisiert und führen im besten Falle zur Optimierung eines Fahrzeugs bzw. einer Menge von exakt gleichen Fahrzeugen. In der vorliegenden Arbeit wird jedoch ein Transportsystem betrachtet, das zwar aus technisch homogenen Fahrzeugen besteht, diese aber unterschiedliche Vorstellungen über die Umwelt besitzen. Dieses äußert sich darin, dass die Fahrzeuge mit unterschiedlichen Zielen, unterschiedlichen Verhaltensweisen und unterschiedlichen Darstellungen der Umwelt (Terminologie) operieren. Kommunizieren diese Fahrzeuge miteinander, ist es wichtig, dass der Empfänger dieselben Vorstellungen von dem hat, was der Sender mitteilen wollte. Ist dem nicht so, kann es zu Missverständnissen und gefährlichen Situationen kommen. Deshalb muss von fahrerlosen Fahrzeugen, die zwar nicht über die menschliche Intuition verfügen, aber dennoch zu einem bestimmten Grad intelligent agieren sollen, verlangt werden, dass sie zum einen eigenständige Entscheidungen treffen und umsetzen und sich zum anderen mit anderen fahrerlosen Fahrzeugen austauschen und dabei die Semantik und Syntax des anderen verstehen können. Dieses Problem ist Grundlage der vorliegenden Arbeit, die sich mit dem Erwerb, dem Austausch, der Verarbeitung und der Integration der Erfahrungen mehrerer technisch homogener, in ihren Wissensbasen jedoch heterogener fahrerloser Fahrzeuge beschäftigt.

2 Probleme der erfahrungsbasierten Selbstoptimierung einer Menge technisch homogener fahrerloser Fahrzeuge

In diesem Kapitel werden die verwendeten *technisch homogenen fahrerlosen Fahrzeuge* mit deren Strukturen, Zielen, Parametern, deren Umwelt, Verhaltensweisen[1] und möglichen Kooperationsarten näher spezifiziert. Die Spezifizierung der Fahrzeuge basiert auf den allgemeinen Aspekten eines selbstoptimierenden *technischen Systems* (Näheres siehe [FGK+04]).

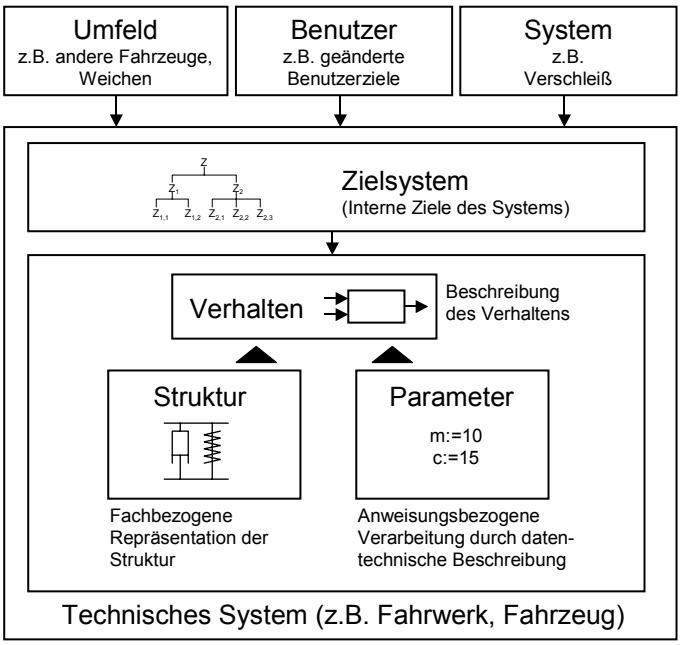


Abbildung 1: Die Aspekte eines selbstoptimierenden technischen Systems [SFB04]

Anschließend werden die jeweiligen Problemansätze zur notwendigen *erfahrungsbasierten Selbstoptimierung* einer Menge technisch homogener fahrerloser Fahrzeuge erläutert. Hierzu gehört die Kommunikation der Fahrzeuge untereinander und die hierfür notwendige Spezifizierung einer gemeinsamen Sprache bzw. die Auflösung von Sprachdifferenzen. Basierend auf einer generischen Sprachebene tauschen sie *Erfahrungen*, abgespeichert in einer *Wissensbasis*, aus. Zur Erlangung von Erfahrungen werden geeignete Explorationsstrategien auf unterschiedlichen Ebenen (AMS, MFM; siehe Abbildung 3) untersucht, die den *Zustand* des jeweiligen Fahrzeugs durch die Auswahl geeigneter *Verhaltensweisen* verbessern. Dabei basieren die Verhaltensweisen nicht nur auf dem *Ziel* der Erweiterung der Erfahrungen durch Erprobung neuer, unbekannter Zustände (*exploratives Verhalten*), sondern auch auf dem Ziel der Nutzung der existierenden Erfahrungen für bereits bekannte Zustände (*exploitatives Verhalten*).

[1] „The ways it [a technical system] acts [...] are called its behaviours" [Sto00, S. 1]. Typische Verhaltensweisen in roboter-basierten Fußballspielen sind „face ball, pass ball, chase ball, dribble ball, avoid obstacle" [GHR+03].

Definition 2-1: Technisches System *Technische Systeme* „repräsentieren [...] konkrete künstliche Gegenstände, die aus natürlichen Beständen gemacht werden und greifbare Wirklichkeit in Zeit und Raum sind“ [Rop99, S.119]. Diese Systeme sind eingebettet in eine natürliche, technische und gesellschaftliche Umgebung und besitzen einen inneren Zustand. Zwischen der Umgebung und dem technischen System besteht eine Input/Output Beziehung. Jeder In- und Output kann dabei der Masse, der Energie oder der Information zugeordnet werden. Der innere Zustand ist durch die Größen Masse, Energie, Information, Raum und Zeit definiert. Aufgrund des inneren Zustandes verhält sich ein technisches System deterministisch (siehe hierzu Bild 16 in [Rop99, S.120]).

Definition 2-2: Technisch homogene fahrerlose Fahrzeuge *Fahrerlose Fahrzeuge* werden dann als *technisch homogen* bezeichnet, wenn sie sich hinsichtlich ihrer technischen Struktur nicht unterscheiden.

Definition 2-3: Erfahrungen „*Erfahrung* ist eine allgemeine Bezeichnung für Kenntnisse und Verhaltensweisen, die man durch Wahrnehmung und Lernen erwirbt oder erworben hat“ [Wik04-ol]. Erfahrung ist somit erlebtes, praktisch überprüftes Wissen.

Definition 2-4: Erfahrungsbasierte Selbstoptimierung „Für die *Selbstoptimierung* ist wesentlich, dass Ziele durch das [technische] System situationsbedingt festgelegt oder angepasst werden“ [FGK+04, S.22]. Dabei passt das technische System sein „Systemverhalten [aufgrund von Erfahrungen] an, ohne dabei auf explizite Modelle zurückzugreifen“ [FGK+04, S. 31].

Definition 2-5: Wissensbasis Die *Wissensbasis* ist der Bereich eines Systems, „der das Fachwissen in einer beliebigen Repräsentationsform enthält. Ergänzt wird die Wissensbasis durch eine Inferenzmaschine, also eine Hard- oder Software, mit der auf der Wissensbasis operiert werden kann“ [Wik04-ol].

Definition 2-6: Systemverhalten „Vom *Verhalten* eines *Systems* spricht man dann, wenn eine Veränderung des Zustandes bzw. der Zustandsgrößen des Systems auf der Makroebene beobachtet werden kann. Als Ereignis wird der Übergang von einem Zustand in einen anderen bezeichnet. Diese Veränderungen können selbständig, ohne Einflüsse von außen erfolgen oder mit einem Einfluss von außen zusammenhängen. Die Zahl der Möglichkeiten, welche Einflüsse (Eingaben) auf ein System einwirken können, ebenso die Zahl der Wirkungsmöglichkeiten (Ausgaben), hängt von der Struktur des Systems ab. Von dem einfachsten System mit nur einer Eingabemöglichkeit und einer Ausgabemöglichkeit (Beispiel: Kniesehnenreflex) bis zu sehr vielen Möglichkeiten bei adaptiven und lernenden Systemen sind alle Übergänge denkbar“ [Wik04-ol].

Definition 2-7: Exploratives Verhalten Unter *explorativem Verhalten* versteht man die Bereitschaft, sich in Zustände zu begeben, die ein hohes Maß an Unbestimmtheiten enthalten [DM95].

Definition 2-8: Exploitatives Verhalten Unter *exploitativem Verhalten* versteht man die Bereitschaft, sich in Zustände zu begeben, die gemäß der gesammelten Erfahrungen optimal sind [DM95].

2.1 Die Domäne der fahrerlosen Fahrzeuge

2.1.1 Einordnung in den Bereich der technischen Systeme

Die fahrerlosen Fahrzeuge gehören zu dem Bereich der technischen Systeme. Die technischen Systeme untergliedern sich laut [Wen05-ol] und [Rop99] in Produktionssysteme, Transportsysteme und Speicherungssysteme.

Funktion / Output	Produktionssystem	Transportsystem	Speicherungssystem
Masse	Verfahrenstechnik Fertigungstechnik	Fördertechnik Verkehrstechnik Tiefbautechnik	Behältertechnik Lagertechnik Hochbautechnik
Energie	Energiewandlungstechnik	Energieübertragungs-technik	Energiespeicherungs-technik
Information	Informationsverarbeitung, Mess-, Steuer,- und Regel-technik	Informations-übertragungstechnik	Informations-speicherungstechnik

Tabelle 1: Klassifikation der technischen (Sach)-Systeme [Rop99, S.131]

Die „Masse“-Transportsysteme können sowohl mobiler oder immobiler Art als auch frei oder geführt sein. Zu den freien, mobilen „Masse“-Transportsystemen gehören z.B. die PKWs und Flugzeuge. Zu den geführten, mobilen „Masse“-Transportsystemen gehört die Bahn. Sowohl die mobilen als auch die immobilen „Masse“-Transportsysteme können mit oder ohne Bediener ausgestattet sein. Ein Verkehrs-Flugzeug wäre z.B. ein freies, mobiles „Masse“-Transportsystem mit Bediener, da es durch den Flugzeugkapitän eigenständig die Richtung, die Geschwindigkeit und die Flughöhe bestimmen kann. Das in dieser Arbeit untersuchte technische System ist ein schienengeführtes, mobiles „Masse“-Transportsystem ohne Bediener.

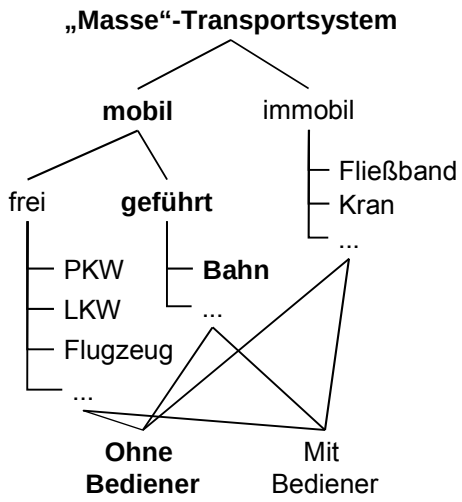


Abbildung 2: Arten eines „Masse“-Transportsystems

2.1.2 Die mechatronische Struktur

Module und Hierarchien sind wichtige Voraussetzungen, um komplexe Probleme in technischen Systemen zu lösen. Die Aufteilung in Module erlaubt es dem Ingenieur, einzelne Module mit ihren Haupt- und Nebenfunktionen[2] unabhängig vom Gesamtsystem zu entwickeln, zu testen und zu produzieren. Diese Module können mechatronische Systeme sein, die mit Aktoren, Sensoren und informationsverarbeitenden Einheiten ausgestattet sind [OHK+02]. Die Hierarchie dieser Module erlaubt es dem Gesamtsystem (hier dem Fahrzeug), die Funktionen der einzelnen Module aufeinander abzustimmen.

Die unterste Hierarchiestufe in einem Fahrzeug (Abbildung 3) wird als *mechatronisches Funktionsmodul (MFM)* bezeichnet. Es besteht aus den oben erwähnten Basiseinheiten (Aktoren, Sensoren und informationsverarbeitenden Einheiten) zuzüglich der jeweils benötigten Hardware bzw. Software zur Kontrolle des MFM. Mechatronische Funktionsmodule können mit weiteren MFM zusammengeschlossen werden, in der ein MFM als Aktor eines anderen MFM fungieren kann [Gau02], [OHK+02].

Auf der nächst höheren Stufe befinden sich die *autonomen mechatronischen Systeme (AMS* – hier das Fahrzeug selbst), die aus informationstechnisch und/oder mechanisch gekoppelten MFM aufgebaut sind. Ein AMS muss in der Lage sein, unterschiedliche Umwelteinflüsse bewerten und geeignete Verhaltensweisen auszuwählen. So sollte ein Fahrzeug erkennen können, ob und was für ein Hindernis im Weg steht. Hat das Fahrzeug ein Hindernis erkannt, muss es ein passendes Verhalten auswählen können, um das Hindernis zu umfahren oder gegebenenfalls zu bremsen. AMS weisen eine Informationsverarbeitung auf, die sowohl das eigene Verhalten (z.B. Hindernis umfahren) als auch das Verhalten der untergeordneten MFM

[2] „Neben den Hauptfunktionen des verschleißarmen Antriebs und Bremsens besitzt das Antriebs- und Bremsmodul die Nebenfunktion Energie zu übertragen (von der Strecke zum Fahrzeug) sowie die Regelung der Nickbewegung der Einzelachse eines jeden Fahrmoduls“. [FGK+04, S.91]

steuert (z.B. Spurführungsmodul nach links steuern, Feder- und Neigemodul leicht neigen) [SS03a].

Interagieren AMS mit anderen AMS, so schließen sie sich zu *vernetzten mechatronischen Systemen (VMS)* zusammen. VMS entstehen also durch eine Kopplung der beteiligten AMS. Hierzu muss jedes AMS wissen, mit welchem AMS es zu kooperieren hat und welches Verhalten in Verbindung mit anderen AMS temporär abhängig voneinander durchgeführt werden soll und kann. Dies kann z.B. das Zusammenschließen zu einer Kolonne (siehe Abbildung 3) oder die Lösung eines Kreuzungsproblems sein. Aufgrund der komplexen und dynamischen Umwelt sowie des autonomen Verhaltens jedes AMS wird es auf der Ebene der vernetzten mechatronischen Systeme oft nicht möglich sein, mit vertretbarem Aufwand (also in vertretbarer Zeit) mathematische Optima zu berechnen. Eine suboptimale Lösung wird hier durch generelle Absprachen bzw. Regeln (rechts vor links), einer Kommunikation (freiwillig Vorfahrt gewähren) oder Kompensationszahlungen (für das „Recht der Vorfahrt“ zahlt Fahrzeug A an Fahrzeug B x Einheiten) erreicht.

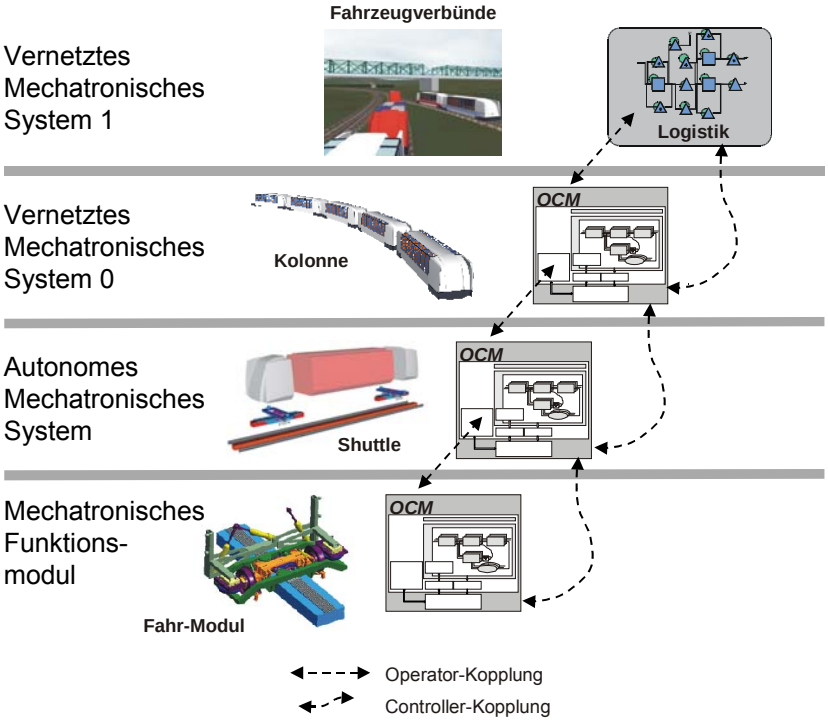


Abbildung 3: Aufbau eines hierarchischen, mechatronischen Transportsystems [OHK+02]

2.1.3 Kooperationsarten

Die möglichen Kooperationsarten zwischen technisch homogenen fahrerlosen Fahrzeugen können in zwei Dimensionen beschrieben werden: der Heterogenität (bezogen auf deren Ziele, Verhaltensweisen und Domänenwissen) der Wissensbasis der Fahrzeuge und deren Kommunikationsbereitschaft [Sto00].

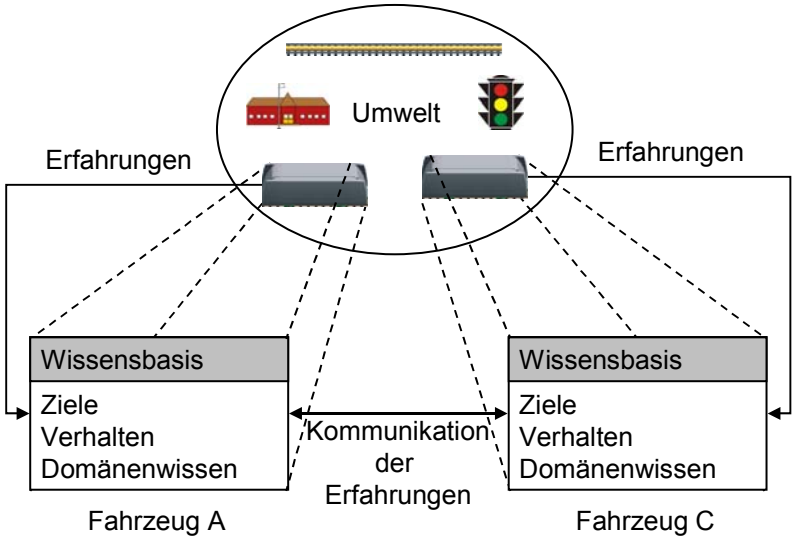


Abbildung 4: Allgemeiner Multi-Fahrzeug-Ansatz [Sto00]

Homogene, nicht-kommunizierende Fahrzeuge haben identische Ziele, Verhaltensweisen und ein identisches Domänenwissen. Sie unterscheiden sich ausschließlich hinsichtlich ihres Sensorinputs und den daraus abgeleiteten Verhaltensweisen. Eine Interaktion findet dadurch statt, dass andere Fahrzeuge per Sensor wahrgenommen werden können, um z.B. zu überholen oder einem Stau auszuweichen. *Heterogene, nicht-kommunizierende Fahrzeuge* besitzen unterschiedliche Ziele, Verhaltensweisen und ein unterschiedliches Domänenwissen. So kann das Ziel eines Fahrzeugs z.B. darin bestehen, die anderen Fahrzeuge absichtlich zu behindern. *Homogene, kommunizierende Fahrzeuge* besitzen dieselben Eigenschaften wie homogene, nicht-kommunizierende Fahrzeuge, können darüber hinaus aber Erfahrungen direkt untereinander austauschen. So ist es möglich Erfahrungen, die von Fahrzeug A in einem Teil der Umwelt erlangt wurden, an ein anderes Fahrzeug C zu übermitteln. *Heterogene, kommunizierende Fahrzeuge* besitzen dieselben Eigenschaften wie heterogene, nicht-kommunizierende Fahrzeuge, können aber im Gegensatz zu jenen Erfahrungen nicht direkt untereinander austauschen, da sie z.B. andere Sprachen zur Kommunikation von Erfahrungen verwenden.

In Tabelle 2 sind sämtliche Kooperationsmöglichkeiten mit deren zugehörigen Problembereichen aufgeführt. In dieser Arbeit wird jedoch nur der Bereich der heterogenen, kommunizierenden Fahrzeuge näher untersucht.

Homogene, nicht-kommunizierende Fahrzeuge	**Heterogene, nicht-kommunizierende Fahrzeuge**
• Reaktive vs. deliberative Fahrzeuge • Lokale vs. globale Perspektive • Modellierung der Zustände anderer Fahrzeuge • Wie können andere Fahrzeuge beeinflusst werden?	• Wohlwollende vs. konkurrierende Fahrzeuge • Stabile oder sich entwickelnde Population • Modellierung der Ziele, der Verhaltensweisen und des Wissens der anderen Fahrzeuge • Ressourcenmanagement (abgestimmtes Verhalten) • Soziale Konventionen • Rollen
Homogene, kommunizierende Fahrzeuge	**Heterogene, kommunizierende Fahrzeuge**
• Verteiltes Messen	• Gegenseitige Verständigung • Planung von Kommunikation • Wohlwollende versus konkurrierende Fahrzeuge • Verhandlung • Ressourcenmanagement (abgestimmtes Verhalten) • Festlegungen

Tabelle 2: Kooperationsmöglichkeiten in einem Multi-Fahrzeug-Ansatz [Sto00]

Die Problembereiche der *heterogenen, kommunizierenden Fahrzeuge* sind wie folgt beschrieben:

Gegenseitige Verständigung
In einer Umwelt, in der mehrere Fahrzeuge, die von verschiedenen Konstrukteuren erstellt wurden, miteinander kommunizieren sollen, muss eine bestimmte gemeinsame Sprache oder ein gemeinsam genutztes Protokoll erstellt werden. Existierende Protokolle für die Ebenen des Informationsinhalts, der Formate einer Botschaft und der Koordinationskonventionen sind z.B. KIF [GF92] oder Ansätze im Bereich von Ontologien [Mae03], für das Format der Botschaft KQML [FMF+94] oder ACL [Fip02] und für die Koordinationskonventionen COOL [BF95]. Die vorliegende Arbeit untersucht in diesem Bereich ausschließlich die Problematik des Informationsinhalts. Hierzu wird der Ansatz der Ontologien verwendet.

Planung von Kommunikation
Tauschen Fahrzeuge Erfahrungen mit anderen Fahrzeugen aus, soll dadurch eine bestimmte Absicht bzw. ein bestimmter Effekt erzielt werden. Ein Effekt kann sein, die Einstellung eines Fahrzeugs zu seiner Umwelt oder über das Verhalten der anderen Fahrzeuge zu ändern. Es ist demnach wichtig, bei der Planung eines Fahrzeugs die Voraussetzungen und Effekte einer Kommunikation zu berücksichtigen. Die zugehörige Theorie dieses Ansatzes wird als „Sprech-Akt"-Theorie bezeichnet [Sto00], [CL95], [LS95]. Das bedeutet, dass die Fahrzeuge eines Unternehmens A absichtlich in ihrer Sprache miteinander kommunizieren, damit die

Fahrzeuge eines Unternehmens C die Erfahrungen von A nicht nutzen können. Eine gegenseitige Verständigung wird dadurch absichtlich vermieden. Dieser Wechsel der Kommunikationsstrategie soll in der vorliegenden Arbeit nicht weiter untersucht werden.

Wohlwollende versus konkurrierende Fahrzeuge

Eines der herausragenden Probleme bei der Konstruktion eines Fahrzeugs besteht darin, ob das Fahrzeug ein wohlwollendes oder ein konkurrierendes Verhalten zeigen soll. Ein wohlwollendes Verhalten ist auch bei unterschiedlichen Zielen möglich, wenn die Fahrzeuge sich in der Zielerreichung jedes einzelnen Fahrzeugs unterstützen [GR94]. Auf der anderen Seite können die Fahrzeuge „egoistisch“ sein und nur ihre persönlichen Ziele in ihrem Verhalten berücksichtigen. Im Extrem richten sich die Aktionen des einen Fahrzeugs gezielt gegen die Ziele eines anderen, um die eigenen Ziele durchzusetzen [Sto00]. In der vorliegenden Arbeit agieren die Fahrzeuge sowohl wohlwollend als auch konkurrierend. So tauschen sie z.B. wahrheitsgemäß Erfahrungen über ihre Umwelt aus. Allerdings konkurrieren sie bei der Angebotsabgabe für einen Fahrauftrag.

Verhandlung

Grundlage einer Verhandlungsstrategie ist es, einen optimalen Punkt zwischen Angebot und Nachfrage zu bestimmen [Sto00]. Im „Kontrakt-Netz-Framework“ von Smith [Smi80] haben alle Fahrzeuge ihre individuellen Ziele, sind eigennützig und besitzen limitierte Schlussfolgerungsmechanismen. Sie bieten auf vorgegebene Aufgaben anderer Fahrzeuge und können diese Aufgaben selbst durchführen oder sie an andere Fahrzeuge weiter vergeben. Dabei müssen die Fahrzeuge für die Ausschreibung ihrer Aufgaben zahlen und bekommen im Gegenzug das günstigste Angebot [SL95]. Eine genauere Untersuchung und Übertragung von Verhandlungsstrategien auf die hier verwendeten Fahrzeuge ist nicht Gegenstand dieser Arbeit.

Ressourcenmanagement

Fahrzeuge können ein abgestimmtes, abhängiges Verhalten aufgrund von knappen Ressourcen, die von mehreren Fahrzeugen gleichzeitig beansprucht werden, besitzen. Beispiele hierfür sind Netzwerkprobleme wie z.B. im Straßenverkehr, wo Straßen verstopfen können, oder beim Netzwerk Routing, wo mehrere Fahrzeuge ihre Nachricht durch ein limitiertes Netzwerk schicken müssen [Sto00]. Untersuchungen zu dem Bereich des Ressourcenmanagements werden in dieser Arbeit nicht vorgenommen.

Festlegungen

Kommunizieren Fahrzeuge untereinander, werden Absprachen getroffen, z.B. dass Fahrzeug A zu einem bestimmten Zeitpunkt am Ort X zu sein hat. In diesem Sinne gehen Fahrzeuge Verpflichtungen gegenüber anderen Fahrzeugen ein. Verpflichtungen sorgen dafür, dass ein System wesentlich „runder“ läuft, indem sich eine gewisse Vertrauensbasis unter den Fahrzeugen aufbaut [Cas95]. Verpflichtungen werden in dieser Arbeit nicht untersucht.

2.2 Generierung einer erfahrungsbasierten Selbstoptimierung

Zur Generierung einer Selbstoptimierung aufgrund von Erfahrungen einer Menge technisch homogener fahrerloser Fahrzeuge ist eine individuelle Wissensbasis jedes Fahrzeugs über seine Ziele, seine Verhaltensweisen und seine Umwelt (bzw. sein Domänenwissen) (vgl. Kapitel 2.2.1), eine gegenseitige Verständigung der Fahrzeuge untereinander (vgl. Kapitel 2.2.1), eine Erfahrungsakquisition (vgl. Kapitel 2.2.2) und die Auswahl eines spezifischen, verbesserten Verhaltens aufgrund von Erfahrungen (vgl. Kapitel 2.2.3) notwendig.

2.2.1 Wissensrepräsentation einer Domäne

Um Wissen beschreiben zu können, werden zunächst *Daten* benötigt. Daten enthalten lediglich eine syntaktische Dimension. Sie besitzen eine spezielle Form, der aber ohne zusätzliches Wissen kein Inhalt zuzuordnen ist. Daten können in *symbolischer* oder *sub-symbolischer* Repräsentation auftreten.

Definition 2-9: Daten Als *Daten* wird die symbolische oder sub-symbolische Repräsentation von Sachverhalten bezeichnet (z.B. den auf einem digitalen Thermometer ablesbaren Anzeigewert von „25“ für eine sub-symbolische Repräsentation.) [PRR03].

Definition 2-10: Symbolische Repräsentation Eine *symbolische Repräsentation* wid durch „sprachlich fassbare elementare Einheiten [erreicht], die durch Bezeichner gekennzeichnet sind“, z.B. Worte, Musiknoten [Bün05].

Definition 2-11: Sub-symbolische Repräsentation Eine *sub-symbolische Repräsentation* wird durch „Elementareinheiten [erreicht], die nicht durch Bezeichner charakterisiert“ sind z.B. Zahlen bzw. Zahlenfolgen [Bün05].

Erhalten Daten einen inhaltlichen Zusammenhang, spricht die Literatur von *Informationen* [AN95], [DP98].

Definition 2-12: Information *Informationen* entstehen aus einer Menge von interpretierbaren Daten, die im Zusammenhang gesehen einen Sinn ergeben [PRR03].

Informationen allein sind zur vollständigen Bewertung einer Situation nicht ausreichend. Dies ist jedoch unerlässlich für die Auswahl eines geeigneten Verhaltens. Ein Beispiel soll dies verdeutlichen: Die Information, dass am 31.12.2004 eine Außentemperatur von minus 10°C in München vorliegt, ist zunächst nur eine Feststellung. Um die Bedeutung dieser Information zu erhalten, wird Wissen benötigt. Wissen verknüpft systematisch Informationen, so dass prognostische oder explanatorische Aussagen abgegeben werden können [PRR03]. In diesem Fall wäre die prognostische Aussage: Wenn die Temperatur in München unter 0°C sinkt und

Niederschlag vom Himmel fällt, so ist dies mit einer hohen Wahrscheinlichkeit Schnee. Aus diesem Wissen schließt das Fahrzeug hinsichtlich seines Verhaltens, dass es seine Geschwindigkeit reduzieren muss.

Definition 2-13: Wissen „*Wissen* bezeichnet die Gesamtheit aller organisierten Informationen mitsamt ihrer wechselseitigen Zusammenhänge, auf deren Grundlage ein (vernunftbegabtes) System handeln kann“ [Wik04-ol].

Auch wenn die Literatur die Begriffe *Daten*, *Informationen* und *Wissen* unterschiedlich interpretiert, kommt sie immer zu einer sehr ähnlichen Definition. In Tabelle 3 sind die vier gängigsten Definitionen aufgeführt.

Autor	Daten	Information	Wissen
[Wii93]	-	Organisierte Fakten, um eine Situation oder einen Zustand zu beschreiben	Wahrheiten, Perspektiven, Überzeugungen, Beurteilungen
[SS97]	Noch nicht interpretierte Symbole	Daten mit Bedeutung	Die Fähigkeit, Bedeutungen zuzuweisen
[DP98]	Eine Menge diskreter Fakten	Eine Botschaft mit dem Zweck, die Wahrnehmung des Empfängers zu verändern.	Erfahrungen, Werte, Erkenntnisse und textabhängige Informationen
[QD99]	Ein Text, der keine Antwort auf bestimmte Fragen gibt	Ein Text, der die Fragen nach dem Wer, Wann, Was oder Wo beantwortet.	Ein Text, der die Frage nach dem Warum und Wie beantwortet

Tabelle 3: Definition des Begriffs Wissen [SS02]

Die vielleicht wichtigste Unterscheidung des Wortes Wissen liefern Nonaka und Takeuchi [NT97]. Sie unterscheiden Wissen in *explizites* und *implizites* oder *stilles* (tacit) *Wissen*.

Definition 2-14: Explizites Wissen *Explizites Wissen* ist formalisierbar. Es lässt sich in Form von Daten, Formeln, Verfahrensweisen, Texten o.ä. weitergeben.

Definition 2-15: Implizites Wissen *Implizites Wissen* ist nicht formalisiertes oder formalisierbares Wissen. Es ist verankert in den Erfahrungen und Tätigkeiten von Individuen und stellt nicht dargestellte Verbindungen zwischen explizitem Wissen her.

Implizites Wissen stellt somit das persönliche Wissen eines Individuums mit Idealen, Werten und subjektiven Einsichten dar. Explizites Wissen ist dagegen methodisch bzw. systematisch und liegt in artikulierbarer Form vor. Es kann mittels Informations- und Kommunikationstechnologie verarbeitet und verbreitet werden. Das Grundproblem des Wissensmanagements ist die Überführung von implizitem in explizites Wissen. Erst dann kann es im Rahmen einer Wissensrepräsentation verfügbar gemacht werden und ist somit über einzelne Fahrzeuge hinaus nutzbar [MCD99-ol].

Im Bereich des impliziten Wissens lassen sich zwei Formen von einander unterscheiden. Beim *technischen Wissen* handelt es sich um informelle und schwer dokumentierbare Fähigkeiten und Fertigkeiten. Hierzu zählt zum Beispiel auch das Know-how oder das Geschick eines Experten. Beim *kognitiven Wissen* handelt es sich um Schemata, mentale Modelle, Überzeugungen und Wahrnehmungen, die der einzelne Experte für selbstverständlich hält.

Im Bereich des expliziten Wissens wird zwischen *deklarativem* und *prozeduralem Wissen* unterschieden. Unter deklarativem Wissen wird statisches Wissen, also eine Art Faktenwissen verstanden. „Beispiele hierfür sind das kleine Einmaleins, das in der Grundschule gelehrt wird, oder der Sachverhalt, dass die Wirbelsäule des Menschen aus 29 Wirbeln besteht“ [Fau96]. Deklaratives Wissen kann durch zwei grundsätzlich verschiedene Formen repräsentiert werden: entweder als Inhalt einer sprachlichen Äußerung („Die menschliche Wirbelsäule besteht aus 29 Wirbeln.“) oder mittels einer bildlichen Darstellung [Alb-03].

Bei der prozeduralen Repräsentation von Wissen steht der Ablauf der Berechnungen zur Problemlösung im Vordergrund. Prozessabläufe lassen sich hierdurch leicht beschreiben. Das prozedurale Wissen besteht entweder aus Rechenoperationen wie z.B. der Multiplikation von zwei Zahlen oder aus Wenn-Dann-Regeln, die das Handlungswissen (Verhaltensweisen) eines Fahrzeugs konstituieren, das in einer Situation anwendbar ist [BP94].

Bei der künstlichen Intelligenz werden Verhaltensweisen oft auch als Aktionen bezeichnet, die der obersten Ebene der Wissenspyramide zugeordnet sind (Abbildung 5).

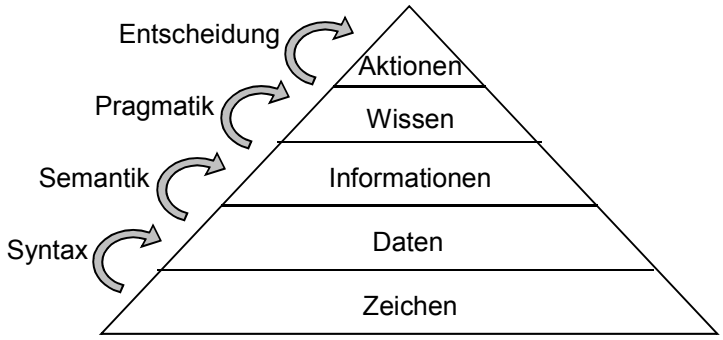


Abbildung 5: Wissenspyramide nach [WDA99]

Verhaltensweisen (Wenn Temperatur = -5°C, dann Pullover anziehen), Domänenwissen (Wenn Temperatur = -5°C, dann Wasser = gefroren), Informationen (Temperatur = 10°C) und Ziele (hoher Fahrkomfort) werden in dieser Arbeit als Erfahrungen definiert, die verbreitet werden können.

2.2.1.1 Wissensrepräsentation einer Domäne durch Ontologien

Grundvoraussetzung für eine erfahrungsbasierte Selbstoptimierung von Fahrzeugen ist, dass das Wissen über eine Domäne explizit repräsentiert ist. Dies ist Aufgabe der Wissensrepräsentation, die Wissen über die Umwelt oder über eine spezifische Domäne so verfügbar macht, dass es mit algorithmischen Methoden geschickt verarbeitet werden kann. Die Wissensrepräsentation definiert also im Wesentlichen die formellen Datenstrukturen, mit denen Wissen im Rechner abgebildet, also formalisiert wird [BK03]. Praktisch kann man eine Wissensrepräsentation als eine strukturierte Abbildung der Umwelt bezeichnen.

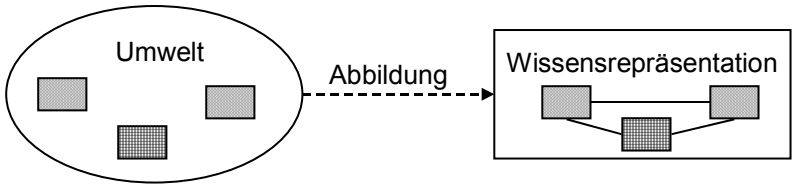


Abbildung 6: Abbildung der Umwelt in einer Wissensrepräsentation

Um Wissen explizit repräsentieren zu können, bedarf es einer logischen Verknüpfung aller Begriffe einer Domäne. In WordNet[3], einer lexikalischen online Datenbank von englischen Begriffen mit dem Ziel, diese sowohl eindeutig syntaktisch (Subjekt, Prädikat, Objekt) als auch semantisch zuordnen zu können, ist jeder Begriff einem bestimmten Zweck zugeordnet. So kann ein Begriff sowohl als eine Begriffsform als auch als eine Begriffsbedeutung eingesetzt werden.

Definition 2-16: Begriffsform Die *Begriffsform* bezeichnet die symbolische Darstellung eines Begriffs, der verbreitet werden soll, als Zeichenkette aus einem Alphabet ohne konkrete Bedeutung. Eine Begriffsform kann ein Substantiv, Verb, Adjektiv oder Adverb sein.

Definition 2-17: Begriffsbedeutung Die *Begriffsbedeutung* bezeichnet das lexikalische Konzept und dementsprechend die Bedeutung (Semantik) der Begriffsform. Begriffsbedeutungen werden erst auf Anfrage verbreitet.

Begriffsform und Begriffsbedeutung spannen eine n:m-Matrix auf, in der Begriffsformen verschiedene Begriffsbedeutungen und Begriffsbedeutungen verschiedene Begriffsformen haben können [MBF+93]. Die Sprachwissenschaft spricht hier von *Polysemie* bzw. *Synonymie* [Pfl01]. Beide Arten können große Probleme zwischen einer Maschine-zu-Maschine-Kommunikation bewirken, da Maschinen grundsätzlich nicht wissen können, welche Begriffsbedeutung bzw. welche Begriffsform im speziellen Fall gemeint ist [Mae03].

[3] http://www.cogsci.princeton.edu/~wn/, WordNet ([Fel98], [HF97]) umfasst mehr als 118.000 Begriffsformen und mehr als 90.000 unterschiedliche Begriffsbedeutungen [Mil95].

Begriffsbedeutung	Begriffsform				
	F_1	F_2	F_3	…	F_n
B_1	=	=			
B_2		=			
B_3			=		
…				…	
B_m					=

Tabelle 4: Die lexikalische Matrix [MBF+93]

Definition 2-18: Polysemie *Polysemie* bedeutet, dass eine Begriffsform mehrere Begriffsbedeutungen haben kann (z.B. Bach = kleiner Fluss oder Komponist; $f = b_1 \vee b_2$).

Definition 2-19: Synonymie *Synonymie* bedeutet, dass verschiedene Begriffsformen eine ähnliche oder gar gleiche Begriffsbedeutung haben können (z.B. Orange = Apfelsine oder Zündholz = Streichholz; $f_1 \vee f_2 = b$). *Synonymie* ist hierbei eine symmetrische Relation zwischen Begriffsformen.

Alle Begriffsformen und –bedeutungen zusammen beschreiben die Terminologie einer Domäne.

Definition 2-20: Terminologie Als *Terminologie* wird die Menge aller Begriffsformen $F=\{f_1,f_2,...,f_n\}$ und Begriffsbedeutungen $B=\{b_1,b_2,...,b_m\}$ bezeichnet, die einer spezifischen Domäne zugeordnet sind.

Werden *Begriffsformen* (z.B. Jaguar) und *Begriffsbedeutungen* (z.B. Auto) einer Terminologie zueinander in Beziehung gesetzt, entsteht ein geschlossenes Weltbild einer Domäne. Dieser Prozess wird als Entwicklung einer *Ontologie* bezeichnet [Zel02].

Definition 2-21: Ontologie Eine *Ontologie* ist eine strukturierte Darstellung einer Terminologie und somit eine formale, explizite Spezifikation einer gemeinsamen Konzeptionalisierung [Gru93].

Diese Definition beinhaltet vier Schlüsselwörter von zentraler Bedeutung [Unt01-ol]:

- *Konzeptionalisierung* (*conceptualization*) bezeichnet ein abstraktes Modell bestimmter Begriffsformen mit deren identifizierten relevanten Begriffsbedeutungen.
- *Explizit* (*explicit*) bedeutet, dass die Art und Bedingung einer jeden Begriffsform ausdrücklich angegeben und definiert wird.

- *Formal* (*formal*) müssen Ontologien sein, um maschinenverständlich zu sein, was eine natürlichsprachliche Darstellung ausschließt.
- *Gemeinsam* (*shared*) signalisiert, dass es sich bei einer Ontologie nicht um eine einzelne, individuelle Ansicht von Wissen handelt, sondern um eine Darstellung, auf die sich eine bestimmte Benutzergruppe geeinigt hat und über die folglich ein Konsens besteht.

Eine Ontologie beschreibt also explizit eine formale, verteilte Konzeptionalisierung eines bestimmten, eine Gruppe interessierenden Bereichs [Unt01-ol], [Alp03].

Formal besteht eine Ontologie aus einem 5er-Tupel $O:= \{C,R,H^C,S,A\}$ [Mae03] mit einer Menge von Begriffsformen (z.B. Shuttle) und Begriffsbedeutungen (z.B. Fahrzeug), die durch eine Menge von hierarchischen is_a-Relationen *R* verbunden sind (Shuttle is_a Fahrzeug; siehe Abbildung 7). Begriffsformen sowie Begriffsbedeutungen werden mittels Klassen C dargestellt, wodurch eine Klassenhierarchie H^C ($H^C \subseteq C \times C$) entsteht. Zusätzlich zu den Klassen existieren Slots[4] *S* (relationale S^R, deskriptive S^D und prozedurale S^P) und Axiome *A*. Hierbei gilt, dass S^R für alle nicht taxonomischen (is_a) Beziehungen steht [Mae03]. Mit S^R können beliebige Arten von Relationen zwischen Klassen dargestellt werden, wie z.B. ein Fahrzeug fährt_auf Schiene oder Schiene A länger_als Schiene B (Näheres siehe Kapitel 3.1.2.2)

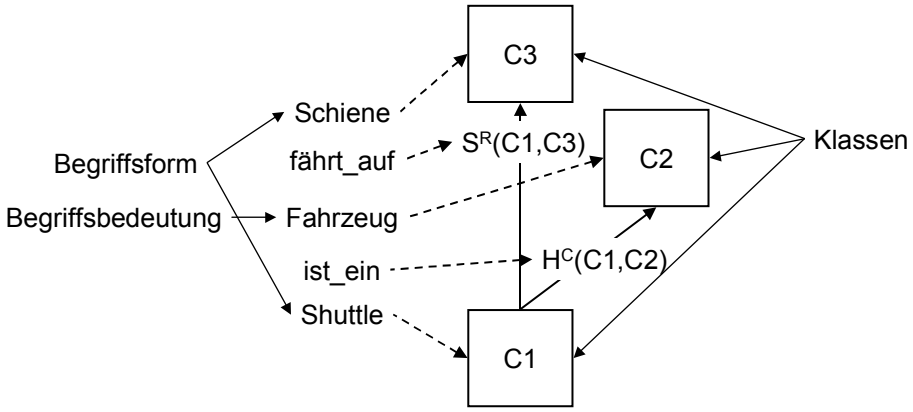


Abbildung 7: Umwandlung von Begriffsformen und -bedeutungen in Klassen

Auf Grund der sprachlichen Vielfältigkeit der relationalen Slots werden Ontologien meist im Zusammenhang mit den semantischen Netzen oder Frames (siehe Kapitel 3.1) genannt. Der Unterschied dieser Darstellungsformen besteht in ihrer Ausdrucksstärke (Abbildung 8). Der Nachteil der semantischen Netze besteht darin, dass sie nicht genügend auf die kontextuellen Begriffsbedeutungen eingehen, sondern nur generelle Verhältnisse von Begriffsformen ausdrücken und in deren Belegung mit spezifischen Merkmalen unflexibel sind. Zusätzlich

[4] Slots besitzen relationale, deskriptive bzw. prozedurale Eigenschaften, die einer Klasse innerhalb einer Ontologie zugewiesen werden.

sind sie nicht dynamisch, d.h. sie können nicht ohne weiteres neue Erfahrungen aufnehmen und besitzen Defizite in der Regeldarstellung [Vis03-ol].

Ontologien erheben stets den Anspruch einer gewissen Objektivität. Dieser Anspruch wird jedoch dadurch gefährdet, dass Ontologien stets durch Menschen erstellt werden, die eine subjektive Wahrnehmung ihrer Umwelt besitzen. Da jede Benutzergruppe eine Domäne auf unterschiedliche Weise wahrnimmt, werden daher auch unterschiedliche Begriffe (Begriffsformen und –bedeutungen) für ihre Modellierung verwendet. Ontologien orientieren sich demnach nicht an objektiven Gegebenheiten der realen Welt, sondern an den Erfordernissen, die sich aus unserer spezifischen Sichtweise auf die Welt ergeben. Dementsprechend gibt es theoretisch unendlich viele Ontologien. Aus diesem Grunde wird der Ansatz der *Basisontologie* eingeführt [Stu03]. Über eine Basisontologie lassen sich verschiedene spezifische Ontologien eindeutig definieren.

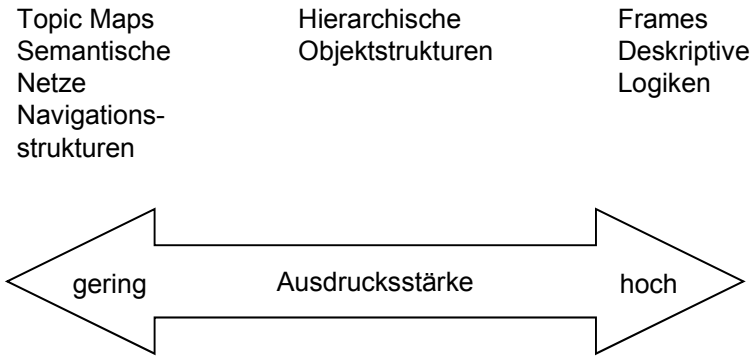


Abbildung 8: Ausdrucksstärke verschiedener ontologischer Darstellungsformen [KTT01]

Definition 2-22: Basisontologie Eine *Basisontologie* ist eine standardisierte, allgemeingültige Beschreibung einer Domäne, über die individuelle Ontologien spezifiziert werden können.

Entwicklungen im Bereich der Ontologien werden vor allem vom Konsortium W3C (www.w3c.org) vorangetrieben. Dieses entwickelte eine Sprache namens RDF[5] (Resource Description Framework), mit Hilfe derer Wissen explizit in Form einer Ontologie dargestellt werden kann, um es für Softwaresysteme verständlich und lesbar zu machen. Dadurch wird der Austausch von Erfahrungen erst möglich. Zusammen mit der DRAPA (Defense Advanced Research Projects Agency) entwickelte das W3C eine weitere Sprache namens DAML (DRAPA Agent Markup Language), die auf RDF aufbaut und diese um weitere ausdruckstarke Konstrukte erweitert, die die Interaktion im Web vereinfacht [HM00].

[5] Siehe Kapitel D

2.2.1.2 Verständigungsprobleme bei heterogenen Ontologien

Um ein Problem lösen zu können, bedarf es oft des Zugriffs auf Erfahrungen, die in den Wissensbasen verschiedener Fahrzeuge gespeichert sind. Um diese Erfahrungen nutzen zu können, müssen sie durch Kommunikation ausgetauscht werden können. Kommunikation ist ein Prozess, bei dem mindestens folgende Elemente vorhanden sind: eine Mitteilung (Nachricht), ein Sender und mindestens ein Empfänger. Hierbei wird eine Nachricht von einer sendenden Instanz an mindestens eine empfangende gerichtet, wobei die Nachricht durch Begriffe in einem Medium (z.B. FIPA-ACL[6]) ausgedrückt und durch einen Kanal (Leitung, Wellen) übermittelt wird [Woo02], [Fer99]. Kommunikation kann nur dann erfolgreich sein, wenn die Begriffe für alle Kommunikationspartner verständlich und eindeutig sind.

Es gibt zahlreiche Beispiele für Interpretationsmöglichkeiten und die daraus entstehenden Mehrdeutigkeiten von Begriffen. Für ein isoliertes System, das nur eine lokale Wissensbasis verwendet, sind diese Probleme kaum relevant. Die Schwierigkeiten treten erst dann auf, wenn ein derartiges System in andere Systeme integriert wird bzw. mit ihnen Erfahrungen austauschen möchte. Die Mehrdeutigkeit der Begriffe kann schnell zum Stillstand des Prozesses bzw. zur Destabilisierung eines Systems führen. Die Notwendigkeit zur Datenhaltung in vernetzten Softwaresystemen setzt demnach eine terminologische Basis voraus, um eine entsprechende Semantik sicherzustellen, die eine sinnvolle Kommunikation überhaupt erst ermöglicht. Die nachfolgenden Ausführungen sollen dabei die Probleme andeuten, die bei der Kommunikation mittels heterogener Ontologien beachtet werden müssen.

Abbildung 9 stellt den Zusammenhang zwischen Begriffsform und Begriffsbedeutung innerhalb einer Kommunikation zwischen heterogenen Ontologien dar.

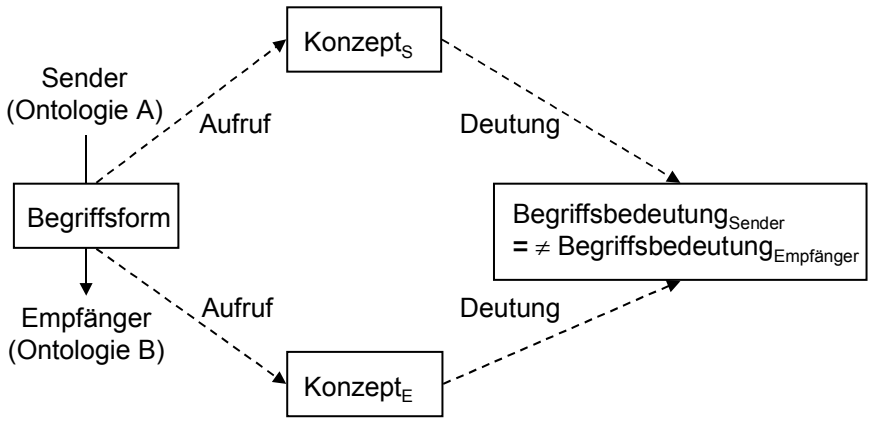


Abbildung 9: Semiotisches Dreieck (in Anlehnung an: [Unt01-ol])

Der Sender und der Empfänger tauschen eine Begriffsform aus wobei jeder über seine individuelle Konzeptionalisierung der Welt seine eigene Begriffsbedeutung besitzt. Dies kann

[6] http://www.fipa.org/specs/fipa00061/

dazu führen, dass die entsprechende Begriffsbedeutung, die der Sender vermitteln will, nicht oder falsch vom Empfänger identifiziert wird.

Aufgrund dieser Problematik sind im Bereich der Kommunikation über heterogene Ontologien grundsätzlich die Beziehungen *Konsens*, *Konflikt*, *Korrespondenz* und *Kontrast* möglich.

Definition 2-23: Konsens In zwei heterogenen Ontologien wird dieselbe Begriffsform für dieselbe Begriffsbedeutung verwendet $(f^S = f^E, b^S = b^E)$.

Definition 2-24: Konflikt In zwei heterogenen Ontologien wird dieselbe Begriffsform für unterschiedliche Begriffsbedeutungen verwendet $(f^S = f^E, b^S \neq b^E)$.

Definition 2-25: Korrespondenz In zwei heterogenen Ontologien werden verschiedene Begriffsformen für dieselbe Begriffsbedeutung verwendet $(f^S \neq f^E, b^S = b^E)$.

Definition 2-26: Kontrast In zwei heterogenen Ontologien werden verschiedene Begriffsformen für verschiedene Begriffsbedeutungen verwendet $(f^S \neq f^E, b^S \neq b^E)$.

Der Sachverhalt der Definition 2-23 stellt hierbei keine weiteren Schwierigkeiten für die Kommunikation dar, da hier eine Eindeutigkeit der Begriffsformen (gegeben durch die selben Begriffsbedeutungen) gegeben ist. Problematischer ist dies für die Definition 2-24, Definition 2-25 und Definition 2-26, weil hier keine Gleichheit von Begriffsformen und Begriffsbedeutungen vorliegt.

Neben der reinen Begriffsproblematik besteht eine weitere Schwierigkeit in den unterschiedlichen, gelernten Verhaltensweisen und Zielen (ausgedrückt durch Regeln) der technisch homogenen fahrerlosen Fahrzeuge. So kann ein Fahrzeug in einer bestimmten Umweltsituation bereits einen Bremsvorgang einleiten, in der ein anderes Fahrzeug seine Geschwindigkeit beibehält.

Dieses unterschiedliche Verhalten ist darauf zurückzuführen, dass die Fahrzeuge unterschiedliche *Regeln* (z.B. hervorgerufen durch den unterschiedlichen Verschleiß der Module) gelernt und unterschiedliche Begriffsformen, Variablen und Operatoren in ihren Regeln mit einbezogen haben. Solche unterschiedlichen Regelmengen sind insbesondere dann kritisch, wenn die Fahrzeuge voneinander lernen oder miteinander kooperieren möchten (z.B. wenn sie sich in einem Konvoi befinden oder beabsichtigen, sich zu einem Konvoi zusammen zu schließen). Hier muss das eine Fahrzeug wissen, wie sich das andere Fahrzeug verhält.

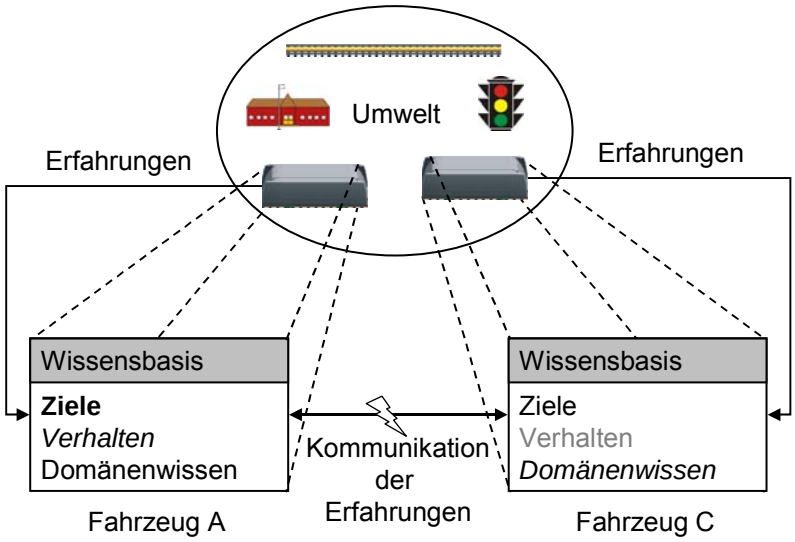


Abbildung 10: Fahrzeuge mit unterschiedlichen Zielen, Verhaltensweisen und Domänenwissen

Definition 2-27: Kriterien zum Vergleich von Regelmengen *Kriterien zum Vergleich von Regeln* zwischen heterogenen Ontologien lassen sich durch die Regellänge, Regelbegriffe, Regelvariablen und Regeloperatoren bestimmen.

2.2.1.3 Auflösung der Verständigungsprobleme

Um die Erfahrungen eines Fahrzeugs den anderen Fahrzeugen mitteilen zu können, müssen sie mit den Erfahrungen anderer Fahrzeuge in eine logische Beziehung gebracht werden [PGM99]. Hierfür wird eine Basisontologie verwendet (siehe Abbildung 11), die eine Lösung der Konflikt-, Korrespondenz-, Kontrast- und Regelmengenproblematik erreicht, indem sich die Ontologien der Fahrzeuge A bis D auf eine gemeinsame Basisontologie beziehen. Die Basisontologie dient hierbei als allgemeingültige Beschreibung der Umwelt.

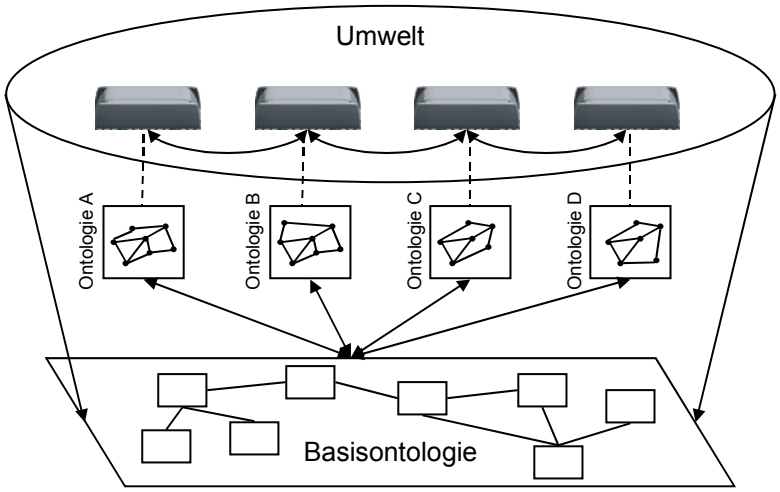


Abbildung 11: Basisontologie für den Austausch von Erfahrungen (in Anlehnung an [Mae03])

Grundsätzlich leistet eine Basisontologie folgende Aufgaben [Tam01-ol]:

- Unterstützung der Kommunikation mehrerer heterogener Ontologien durch die Definition einer standardisierten Terminologie
- Speicherung von allgemein gültigem Wissen einer Domäne (z.B. rechts-vor-links)
- Förderung des Erfahrungsaustausches („People can't share knowledge if they don't speak a common language" [Dav97]).

Bei dem *Konsens* (siehe Definition 2-23) wird davon ausgegangen, dass sowohl die Ontologie von A als auch die Ontologie von C mit denselben Begriffsformen arbeitet. So wird die Begriffsform „Bach" in beiden Ontologien der Begriffsbedeutung „Komponist" zugeordnet. Eine Übersetzung ist demnach nicht notwendig. Die FIPA spricht in diesem Fall von einer Form der „Identical Ontologies" - d.h. Begriffsformen und die Repräsentationssprache sind identisch - oder „Equivalent Ontologies" wobei Begriffsformen identisch sind, die Repräsentationssprache jedoch unterschiedlich ist. Dieser Sachverhalt wird allerdings in dieser Arbeit nicht weiter behandelt [FIP01].

Definition 2-28: Darstellung des Konsens Sei X eine Begriffsform der Ontologie A, Y eine Begriffsform der Ontologie C und B eine Begriffsbedeutung aus der Basisontologie, dann gilt: Konsens (X,Y) :- Begriffsbedeutung (B_1,X), Begriffsbedeutung (B_2,Y), $X = Y, B_1=B_2$

Beim *Konflikt* (siehe Definition 2-24) wird davon ausgegangen, dass dieselbe Begriffsform für unterschiedliche Begriffsbedeutungen verwendet wird. Dieselbe Begriffsform wird demnach einer anderen Begriffsbedeutung in der Basisontologie zugeordnet. Dies ist z.B. für die Begriffsform „Bach" der Fall: Hierbei kann es sich um einen Komponisten oder um einen Wasserweg handeln. Normalerweise dürfte dieser Fall der Problematik nicht auftauchen, da durch die Wahl der Domäne bereits festgelegt wurde, ob sich die Fahrzeuge über den Bereich der Natur oder über den Bereich der Musik unterhalten. Diese eindeutige Domänenfestlegung kann in dem vorliegenden Szenario jedoch nicht gewährleistet werden, da dem Empfänger nicht gesagt wird, in welcher Domäne (z.B. Logistik, Mechatronik, Sicherheitstechnik) sich der Sender zurzeit befindet.

Definition 2-29: Darstellung des Konflikts Sei X eine Begriffsform der Ontologie A, Y eine Begriffsform der Ontologie C und B eine Begriffsbedeutung aus der Basisontologie, dann gilt: Konflikt (X,Y) :- Begriffsbedeutung (B_1,X), Begriffsbedeutung (B_2,Y), X = Y, B_1\=B_2

Bei der *Korrespondenz* (siehe Definition 2-25) wird davon ausgegangen, dass verschiedene Begriffsformen für ein und dieselbe Begriffsbedeutung verwendet werden. Die FIPA bezeichnet eine solche Art der Problematik als „Translatable Ontologies" [FIP01]. Dies bedeutet, dass es möglich ist, die Begriffsform der einen Ontologie in eine Begriffsform der anderen Ontologie zu übersetzen, ohne dass dadurch ein Informationsverlust entsteht.

Definition 2-30: Darstellung der Korrespondenz: Sei X eine Begriffsform der Ontologie A, Y eine Begriffsform der Ontologie C und B eine Begriffsbedeutung aus der Basisontologie, dann gilt: Korrespondenz (X,Y) :- Begriffsbedeutung (B_1,X), Begriffsbedeutung (B_2,Y), X\=Y, B_1=B_2

Der *Kontrast* der Begriffsformen (siehe Definition 2-26) drückt sich darin aus, dass sie nichts miteinander gemein haben. Die Begriffsformen werden in den Ontologien unterschiedlich bezeichnet und auch unterschiedlich in die jeweilige Taxonomie eingeordnet. Eine Kommunikation, auch über eine Basisontologie, ist somit nicht möglich, da die Begriffsformen nichts miteinander zu tun haben. Aus diesem Grunde wird diese Problematik in der vorliegenden Arbeit nicht weiter untersucht.

Definition 2-31: Darstellung des Kontrasts Sei X eine Begriffsform der Ontologie A, Y eine Begriffsform der Ontologie C und B eine Begriffsbedeutung aus der Basisontologie, dann gilt: Kontrast (X,Y) :- Begriffsbedeutung (B_1,X), Begriffsbedeutung (B_2,Y), X\= Y, B_1\= B_2

Die schwierigste Form der Kommunikationsproblematik ist der Vergleich der Regelmengen. Hierbei wird angenommen, dass die Fahrzeuge A und C unterschiedlich gelernte Regelmengen besitzen.

Definition 2-32: Vergleich der Regelmengen Seien R_A und R_C beliebige Regeln aus der Verhaltens- bzw. Zielmenge der Fahrzeuge A und C, P_A und P_C die Prämissen und C_A und C_C die Konklusionen dieser Regeln, dann gilt: Gleiche_Regel (A,C) :- Prämisse (P_A, R_A), Prämisse (P_C, R_C), $P_A = P_C$, (Konsens (C_A,C_C) $\vee$ Korrespondenz (C_A,C_C) $\vee$ Konflikt (C_A,C_C))

Eine Prämisse ist hierbei eine Menge von 6er-Tupeln $T:=\{S, F, O^{min}, W^{min}, O^{max}, W^{max}\}$, die sich aus den Begriffsformen *F* und deren Slots *S* sowie den Wertebereichen der Slots W^{min}, W^{max} und den Operatoren der Wertebereiche O^{min}, O^{max} zusammensetzt (Näheres siehe Kapitel 5.2).

Mögliche Transformationsalgorithmen zur Auflösung von Konflikten, Korrespondenzen und unterschiedlicher Regelmengen sind durch die Methoden des *Mergens*, *Mappens* bzw. *Übersetzens* gegeben. Diese Methoden werden in Kapitel 3.2 näher beschrieben.

2.2.2 Erfahrungsakquisition

2.2.2.1 Möglichkeiten der Erfahrungsakquisition

Es sind prinzipiell drei Formen der Erfahrungsakquisition zu unterscheiden:

- Ein Knowledge Engineer [BK03] erhebt die Erfahrungen von Experten und baut eine Wissensbasis in Form einer Ontologie auf bzw. füllt diese Wissensbasis manuell mit neuen Erfahrungen (*indirekte Erfahrungsakquisition*) [GRS03].

- Ein Experte baut selbständig die Wissensbasis in Form einer Ontologie unter Zuhilfenahme eines Wissenserwerbsprogramms auf und füllt diese Wissensbasis manuell mit neuen Erfahrungen (*direkte Erfahrungsakquisition*) [GRS03].
- Eine Wissensbasis ist bereits indirekt oder direkt entwickelt worden. Neue Erfahrungen werden mit Hilfe einer Simulation, eines Tests unter Laborbedingungen oder im realen Echtzeitbetrieb akquiriert (*automatische Erfahrungsakquisition*).

Untersucht man die Methoden der *indirekten Erfahrungsakquisition*, so ist die vom Knowledge Engineer am häufigsten verwendete Methode das unstrukturierte Interview, bei dem der Experte erzählt, wie er ein Problem löst. Der Knowledge Engineer versucht durch mehr oder weniger spontane Zwischenfragen ein vollständiges Bild zur umfassenden Repräsentation einer Domäne durch eine Ontologie zu erhalten. „Im Idealfall erfolgt diese Konstruktion einer Ontologie durch Komposition und Anpassung bereits existierender Ontologien“ [PSS03, S. 623].

Bei der *direkten Erfahrungsakquisition* versuchen Experten, meist interaktiv, eine Wissensbasis aufzubauen, wobei als Ergebnis eine Menge von Produktionsregeln entsteht. Die dazu verwendete Methode ist häufig das so genannte Repertory-Grid[7]. Es ist vorwiegend für Aufgabenstellungen geeignet, die mit Klassifikationen zu tun haben. Bei dieser Methode werden im ersten Schritt die verschiedenen zu klassifizierenden Objekte festgelegt. Bei einem Beratungssystem für Städtereisen beispielsweise wären dies die Orte (Paris, London, Nizza etc.). Danach wird nach charakteristischen Eigenschaften für Dreiergruppen von Objekten gesucht, wobei eine Eigenschaft für jeweils zwei und die gegensätzliche Eigenschaft für das dritte Objekt stehen soll. In unserem Beispiel hieße dies „viele Museen“ – „wenige Museen“. Anschließend werden alle Objekte nach dieser Eigenschaft auf einer Skala von eins bis fünf bewertet. Inkrementell werden neue Eigenschaften und auch neue Objekte zum System hinzugefügt. Dieses Erfahrungsakquisitionssystem erlaubt es, relativ schnell eine Wissensbasis aufzubauen, wobei dies jedoch nur für sehr einfache Problemstellungen möglich ist.

Um einen langwierigen Erfahrungsakquisitionsprozess zu umgehen, wird oft auf die *automatische Erfahrungsakquisition* zurückgegriffen [BK03]. Hierbei können grundsätzlich drei Wege bestritten werden: Der direkte Einsatz im realen Echtzeitbetrieb, das Testen des Fahrzeugs unter Laborbedingungen (Teststrecke) oder das vorherige Testen des Fahrzeugs im Simulationslauf [Pro03-ol].

Der Nachteil des direkten Einsatzes im realen Echtzeitbetrieb besteht darin, dass Erfahrungen teilweise unter hohen Kosten erworben werden. Zudem kann es zu sicherheitskritischen Umweltsituationen kommen, die ein Testobjekt noch nicht beherrschen kann. Solange es sich bei dem Testobjekt um unkritisches Material handelt (z.B. Holz oder Kohle), wäre dies kein Problem. Werden jedoch Menschen in einem Testfahrzeug befördert, fällt diese Variante sofort durch. Das Testen unter Laborbedingungen (Teststrecke) stellt die interessanteste

[7] Eine Demosoftware kann unter http://www.csd.abdn.ac.uk/~swhite/repgrid/repgrid.html gestartet werden.

Variante dar, da hier quasi reale Bedingungen vorherrschen, dessen Erfahrungen wichtig für den späteren realen Echtzeitbetrieb sind. Da auch dieser Einsatz sehr teuer werden kann, sollten als Vorstufe zu diesem Schritt zunächst die Erfahrungen anhand einer Simulation erworben werden.

Neben anderen wichtigen Eigenschaften (siehe Abbildung 12) bietet eine Simulation vor allem die Sicherheit, zukünftige Entscheidungen auf fundierte Erkenntnisse bzw. Erfahrungen zu stellen und etwaige Schwachstellen bereits vor dem Betrieb eines fahrerlosen Fahrzeugs zu eliminieren [Pro03-ol].

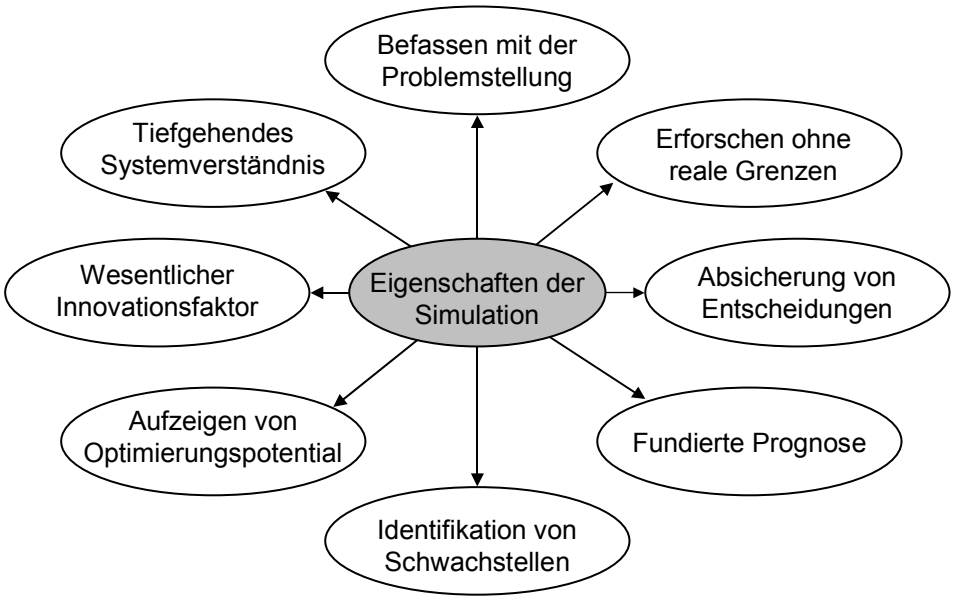


Abbildung 12: Eigenschaften einer Simulation [Pro03-ol]

2.2.2.2 Simulationsgestützte Erfahrungsakquisition

Im Gegensatz zu analytischen Verfahren bietet die Simulation keine automatischen Algorithmen für eine bereits bekannte optimale Lösung, sondern ist in erster Linie ein Datenlieferant für eine nachgelagerte Analyse [Pro03-ol]. Somit ist sie sehr gut für den Einsatz von Fahrzeugen geeignet, die sich aufgrund historischer Erfahrungen iterativ einem optimalen Punkt nähern. Dabei liefert die Simulation in der Regel keine optimale, sondern lediglich eine möglichst beste Lösung [Pag91]. Ziel von Simulationen ist es, das reale Fahrzeug möglichst gut abzubilden, da Experimente am realen Fahrzeug unmöglich, gefährlich oder zu teuer wären.

Die Gewinne, die durch eine Simulation erzielt werden können, werden von Szczerbicka und Uthmann [SU00] wie folgt beschrieben: Simulationsmodelle können als Werkzeuge zur Gewinnung und Strukturierung von Domänenwissen dienen und dieses validieren. Sie können zudem als Quelle für Erfahrungen eingesetzt werden, z.B. indem Ergebnisse von Simulationsläufen in die Fallbasis von Case-Based-Reasoning-Systemen [Ber02] integriert

werden. Szczerbicka und Uthmann [SU00] weisen jedoch darauf hin, dass es nicht immer möglich und sinnvoll ist, in einer Wissensbasis sämtliche verfügbaren Erfahrungen in Fällen vorzuhalten. Dies würde die Schlussfolgerungseffizienz eines Fahrzeugs mit zunehmender Zahl an Fällen stark belasten. Das gilt vor allem dann, wenn das Fahrzeug von einem sehr dynamischen Verhalten, also einem schnellen Anwachsen der Fallbasis geprägt ist.

Neben der reinen Datenlieferung können Simulationen auch für Schlussfolgerungen zur Validierung bzw. Entwicklung von Modellen eingesetzt werden. Als Beispiel sei hier die Methode des *deduktiven* und des *induktiven Schließens* genannt. Im deduktiven Fall untersucht die Simulation, ob die theoretische Annahme (das mathematische Modell) korrekt ist. Im induktiven Fall wird erst aus der Menge aller Erfahrungen ein experimenteller Zusammenhang der Variablen erzeugt [BK03]. Die in dieser Arbeit vorliegenden Untersuchungen beschränken sich auf eine induktive Analyse.

Definition 2-33: Deduktives Schließen Das *deduktive Schließen* ist das Schließen von allgemeinen Sachverhalten auf spezielle Fälle.

Definition 2-34: Induktives Schließen Beim *induktiven Schließen* wird ein allgemeiner Sachverhalt aus beobachteten, speziellen Sachverhalten abgeleitet.

2.2.3 Erfahrungsbasierte Selbstoptimierung aufgrund veränderter Verhaltensweisen

Für eine erfahrungsbasierte Selbstoptimierung benötigt ein Fahrzeug grundsätzlich vier verschiedene Komponenten [RN03]: Ein *Lernelement* (siehe Kapitel 2.2.3.3), das verantwortlich dafür ist, dass die Wissensbasis durch Erfahrungen ständig erweitert wird (z.B. um neue Regeln); eine *Wissensbasis* (siehe Kapitel 2.2.3.1 und 2.2.3.2), die aus einer Menge von Zielen, Verhaltensweisen und Domänenwissen besteht, auf die sich das Fahrzeug stützen kann; eine *Kritikkomponente* (siehe Kapitel 2.2.3.3), die die tatsächlichen Erfahrungen mit den erwarteten Erfahrungen vergleicht, kritisch hinterfragt und darauf basierend dem Lernelement ein Feedback übermittelt, an welcher Stelle gelernt werden muss und ein *Problemgenerator* (siehe Kapitel 2.2.3.4), der ein Verhalten vorschlägt, das zu neuen und informativen Erfahrungen führt. Sollte dieser Generator fehlen, würde ein Fahrzeug immer nur das nach seinem aktuellen Wissen beste Verhalten ausführen, aber nie etwas Neues ausprobieren, das zwar kurzfristig ungünstiger sein kann, aber langfristig eventuell eine Verbesserung bewirkt.

Definition 2-35: Lernen Ein Fahrzeug gilt als lernend, wenn seine Leistungen zur Erfüllung einer Aufgabe durch seine Erfahrungen und den optimalen Einsatz seiner Ressourcen ständig verbessert werden.

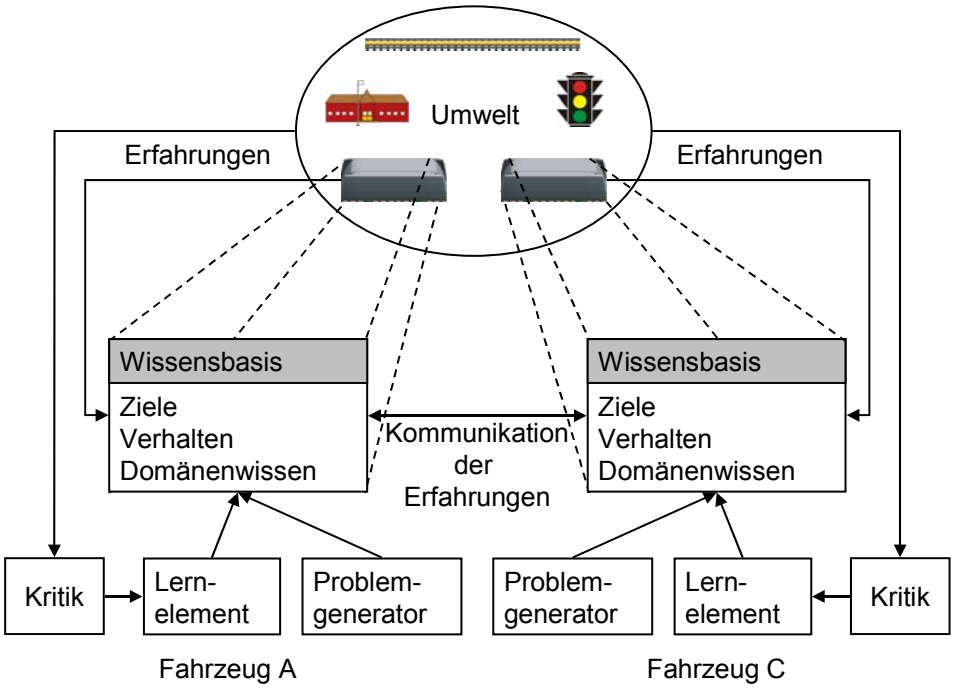


Abbildung 13: Allgemeines Modell eines selbstoptimierenden Fahrzeugs (siehe [RN03])

2.2.3.1 Verhaltensweisen eines Fahrzeugs

2.2.3.1.1 Verhaltensweisen der MFM-Ebene

Die AMS-Ebene „kontrolliert“ die Verhaltensweisen der MFM. Sie ist in der Lage, die Umweltsituation zu bewerten und daraus Handlungsempfehlungen an die MFM weiterzuleiten. Grundsätzlich wird dabei Folgendes angenommen: Je niedriger die Ebene eines Moduls ist, desto zeitkritischer verläuft der reibungslose Informationsprozess [OHK+02]. Umgekehrt bedeutet dies jedoch auch, dass die Komplexität und die Freiheitsgrade auf höheren Levels größer sind als auf den unteren Levels [LKS00], [LHL01]. Sind die Entscheidungen für die Auswahl der geeigneten Verhaltensweisen auf der VMS- bzw. AMS-Ebene einmal fixiert, besteht die Aufgabe der MFM darin, diese möglichst optimal in kurzer Zeit auszuführen. Die Freiheitsgrade der MFM bewegen sich dabei immer innerhalb des von dem AMS vorgegebenen Rahmens.

Die physikalischen bzw. mechatronischen Verhaltensweisen der MFM-Ebene beschränken sich auf die in Abbildung 14 dargestellte Struktur. Diese wird (inklusive ihrer möglichen Verhaltensweisen) im Folgenden genauer beschrieben.

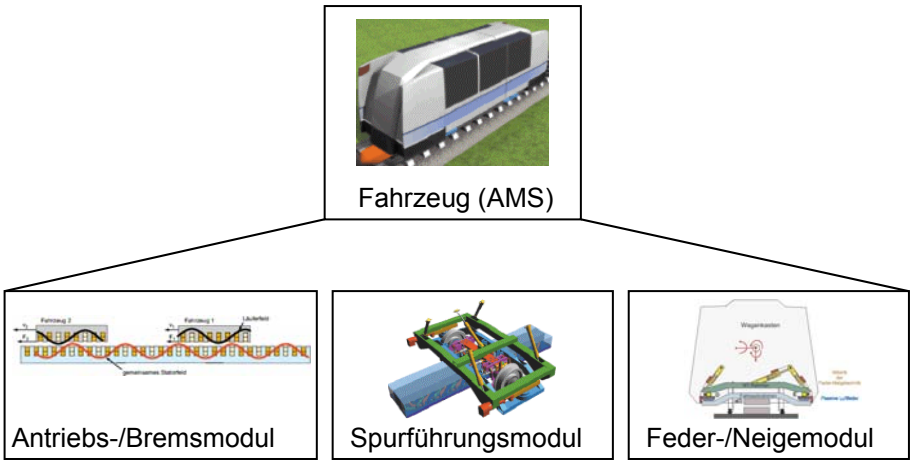


Abbildung 14: Aufbau eines AMS aus mehreren MFM

Das Antriebs- und Bremsmodul:

„Das Antriebsmodul des [Fahrzeugs] besteht aus einem doppelt gespeisten Langstatorlinearmotor. Der eigentliche Antrieb wird bei diesem Konzept nicht mehr im Fahrzeug mitgeführt. Die im Stator liegenden Drehstromwicklungen bilden ein magnetisches Feld, welches sich entlang der Schiene fortbewegt und so das Fahrzeug mit sich führt. Die magnetischen Kraftwirkungen zwischen Stator- und Läufermagnetfeld sind für das *Beschleunigen* und *Bremsen* des Fahrzeugs, ohne Nutzung des Rad-/Schiene-Kontaktes, verantwortlich. Die Doppelspeisung ermöglicht die beliebige Ausrichtung des Fahrzeugmagnetfeldes. Dadurch können mehrere [Fahrzeuge] auf dem gleichem Statorabschnitt betrieben werden. Die Räder des Fahrzeugs werden nur noch zum Führen und Lenken genutzt, wodurch der Radverschleiß zusätzlich reduziert wird. Der Sekundärteil des Motors (Läufer) befindet sich am Fahrzeug. Hierdurch werden die ungefederten Massen im Fahrzeug verringert, da auf ein Getriebe gänzlich verzichtet wird. Zudem kann der Motor an die Landschaftstopologie angepasst werden, also z.B. an Streckensteigungen stärker dimensioniert werden“ [NBP02], [NBP03-ol].

Das Spurführungsmodul

„Das Spurführungsmodul ist für die *sichere* und *verschleißarme Führung* des Fahrzeugs im Gleis verantwortlich und bildet die Basis des mechatronischen [Fahrzeug-] Fahrwerks. Um einen hohen Fahrkomfort zu gewährleisten, wurde konsequent darauf geachtet, eine optimale Entkopplung zwischen dem Aufbau und dem Fahrwerk zu erreichen. So kann beispielsweise durch den Einsatz von Losradsätzen auf Schlingerdämpfer verzichtet werden. Diese sind bei Radsatzfahrwerken als sicherheitsrelevantes Bauteil unverzichtbar, stellen aber eine unerwünschte Schallbrücke dar“ [NBP02].

„Da die Fahrzeuge nicht über die Räder angetrieben werden, kann mit dem Losradprinzip auch der Längsschlupf zwischen Rad und Schiene ausgeschaltet werden. Der Verschleiß an beiden Elementen wird somit deutlich reduziert. Zusätzlich wird der verschleißintensive Spurkranzanlauf der Räder durch die aktive Lenkung der Losradachsen verhindert und mit

einem künstlichen Sinuslauf für eine gleichmäßige Beanspruchung des Radprofils gesorgt. Da der Schienenverlauf bekannt ist, ergibt sich für die Fahrzeuge außerdem die Möglichkeit, die Lenkung vorzusteuern und damit eine *optimale Spurführung* zu gewährleisten“ [NBP02].

Das Feder- und Neigemodul

„Ein *guter Fahrkomfort* im Wagenkasten erfordert möglichst schwache Dämpfer in der Sekundärfederung. In dem vorgestellten Modulkonzept entfallen daher alle Dämpfer der Sekundärfederung. Das Feder- und Neigemodul kombiniert eine niederfrequent abgestimmte passive Aufhängung des Wagenkastens mittels einer Luftfeder (Sekundärfederung) mit einer aktiven Fußpunktverstellung, so dass der Wagenkasten *aktiv gedämpft* und *geneigt* werden kann. Die für die Regelung der Fußpunktverstellung benötigten Informationen werden von geeigneten Sensoren bereitgestellt und mittels einer Mehrgrößenregelung verarbeitet. Durch diese aktive Federungstechnik wird eine weitgehend dynamische Entkopplung des Wagenkastens vor allem im oberen Frequenzbereich erreicht, d.h. die von den Unebenheiten in der Schiene (quer und vertikal) eingeleiteten Störungen werden so gut wie nicht mehr auf den Wagenkasten übertragen. Das Ergebnis ist ein bisher unerreichter *Fahrkomfort in vertikaler und horizontaler Richtung*“ [NBP02].

Aus den Beschreibungen lassen sich sowohl die möglichen Verhaltensweisen als auch die Zielgrößen für ein MFM herausfiltern. Die Zielgrößen lauten z.B. *Fahrsicherheit* und *Fahrkomfort* und können von allen Modulen gleichermaßen beeinflusst werden. Die Verhaltensweisen sind jedoch modulspezifisch und lauten:

Antriebs- und Bremsmodul	→	*Fahrzeug bremsen, Fahrzeug beschleunigen*
Spurführungsmodul	→	*Fahrwerk lenken, Fahrwerk spurführen*
Feder- und Neigemodul	→	*Aufbau federn, Aufbau dämpfen und Aufbau neigen*

Diese Arbeit beschränkt sich auf die Darstellung der Verhaltensweisen und Ziele des Feder- und Neigemoduls und untersucht deren mögliche erfahrungsbasierte Selbstoptimierung.

2.2.3.1.2 Verhaltensweisen der AMS-Ebene

Der Fahrgast knüpft an seine Fahrt bestimmte Bedingungen (Constraints des Kunden C^c). Diese Bedingungen erlegen dem Fahrzeug Beschränkungen in seinen Verhaltensmöglichkeiten auf, wie z.B. nur eine bestimmte Strecke zu verwenden. Sind die Bedingungen des Fahrgastes für das Fahrzeug nicht zu erfüllen, kann ein Auftrag auch abgelehnt werden oder es wird erst gar kein Angebot abgegeben.

$$C^c \equiv \left\{ v^s, v^z, t^c, \omega^c, d^c, m^c, typ^c, L^c \right\}$$

v^s ist der Startbahnhof, v^z der Zielbahnhof, t^c der gewünschte Startzeitpunkt, ω^c ein Faktor zur Festlegung für eine entweder möglichst billige oder möglichst schnelle Verbindung, d^c ist die

maximale Reisezeit, m^c sind die maximalen Kosten, typ^c ist der gewünschte Schienenfahrzeugtyp und L^c sind die lokalen Bedingungen der Streckenabschnitte (z.B. keine Tunnels, nur Panoramastrecken).

Die durch den Fahrgast definierten Constraints lassen sich in zwei Gruppen einteilen. Lokale Constraints (hier L^c) interagieren disjunkt mit dem Streckennetz, d.h. ein lokaler Constraint stellt eine bestimmte Anforderung an jeden Streckenabschnitt. Ein Beispiel für solch ein Constraint ist der Wunsch des Fahrgastes, auf Hochgeschwindigkeitsstrecken von Bahnhof A zu Bahnhof B gebracht zu werden. Durch solch ein Constraint verlieren alle „normalen" Trassen ihre Gültigkeit und dürfen vom Fahrzeug nicht benutzt werden. Globale Constraints (hier d^c und m^c) interagieren dagegen mit dem bisher gefahrenen Weg. Ein Beispiel ist die Zeit. Möchte der Fahrgast in einer Stunde von Bahnhof A nach Bahnhof B gebracht werden, sind nur noch Wege zwischen A und B erlaubt, die in der Summe der Streckenabschnitte nicht länger als eine Stunde dauern.

Das Fahrzeug muss die Constraints des Fahrgastes so verarbeiten, dass es im Sinne seiner Wünsche die optimale Route findet, d.h. optimal im Hinblick auf die globalen Constraints. Sind die Möglichkeiten des Fahrzeugs und die Wünsche des Fahrgastes nicht in Einklang zu bringen, kommt es zu keiner Angebotsabgabe. Stellt das Fahrzeug während des Transportauftrages fest, dass die Constraints mit dem derzeitigen Streckenplan nicht mehr einzuhalten sind, kann es z.B. eine neue Route wählen oder auch die Geschwindigkeit erhöhen. Werden die Constraints des Fahrgastes verletzt, kommt es zu Kompensationszahlungen.

Aus den Beschreibungen lassen sich sowohl die möglichen Verhaltensweisen als auch die Zielgrößen für ein Fahrzeug herausfiltern. Zielgrößen sind z.B. ein *gewinnorientiertes, auslastungsorientiertes, schnelles oder komfortables Fahren*. Als entsprechende Verhaltensweisen hierzu gelten z.B. eine *Verbindung planen,* den *Streckenabschnitt wechseln,* ein *Fahrzeug überholen,* das *Fahrzeug beschleunigen,* einen *Auftrag annehmen* und einen *Auftrag ablehnen*.

Diese Arbeit untersucht das Ziel des gewinnorientierten Fahrens in Verbindung mit den Verhaltensweisen *Auftrag annehmen*, *Auftrag ablehnen* und *Verbindung planen*. Sie geht demnach näher auf die planerischen Verhaltensweisen und die Selbstoptimierung dieser durch die Verarbeitung der Erfahrungen ein.

2.2.3.2 Methodische Ansätze zur Auswahl geeigneter Verhaltensweisen

Der Bereich der *wissensbasierten Systeme* bietet vier verschiedene Lösungstechniken zur Auswahl geeigneter Verhaltensweisen im symbolischen Bereich an [Kös00], [BK03]:

- **Regelbasiertes Schließen:** Sie ist eine der am häufigsten eingesetzten Technik im Bereich der wissensbasierten Systeme, die Schlussfolgerungsmechanismen in Form von Wenn X Dann Y Regeln verwendet.
- **Fallbasiertes Schließen:** Das aktuell zu lösende Problem weist Ähnlichkeiten zu früher bereits gelösten Problemen auf, die in einer Fallbasis abgespeichert sind. Ist eine geeignete Lösung gefunden, kann sie an die aktuelle Situation adaptiert werden.
- **Constraint Techniken:** Dieser Ansatz arbeitet mit der Aufstellung von Einschränkungen (Constraints) des Wertebereichs für Objekte. Constraints können explizit oder in Form von Prädikaten, Funktionen oder Formeln angegeben werden.
- **Klassische und nicht-klassische Logik**: Diese Ansätze umfassen einerseits alle Aussagen- oder Prädikatenlogiken, andererseits aber auch darüber hinausgehende oder von ihr abweichende Folgerungsmechanismen. Diese dienen häufig zur Berücksichtigung von unvollständigen oder unscharfen Informationen (z.B. Fuzzy Logic [JM96]).

Bei Regelbasierten Systemen wird der „Wenn-Teil“ als *Prämisse* oder *Antezedenz* der Regel bezeichnet, während der „Dann-Teil“ *Konklusion* oder *Konsequenz* genannt wird [LC01]. Ist die Prämisse erfüllt, so wird davon gesprochen, dass die Regel feuert. Feuert eine Regel, dann wird die Konklusion ausgeführt. Die Konklusion beinhaltet hierbei das zugeordnete Verhalten, das durch eine Umweltsituation (Prämisse) ausgelöst wurde.

Regeln werden gerne in Produktionssystemen zur Steuerung eingesetzt weshalb Regeln oft als Produktionsregeln bezeichnet werden, wobei die Konsequenz einer Regel oft mit einem Verhalten verbunden ist [BK03]:

Wenn Druck > 50 bar Dann Ventil öffnen

Werden mehrere Prämissen (Umweltsituationen) zum Feuern einer Regel benötigt, so wird von einer Vorwärtsverkettung oder einer datengetriebenen Inferenz gesprochen [LC01]. Die Umweltsituationen werden hierbei in konjunktiver Form ($\wedge$) dargestellt.

Wenn Druck > 50 bar $\wedge$ Temperatur > 100°C Dann Ventil öffnen

Werden weitere Regeln hinzugefügt, baut sich eine Regelbasis auf. Dieser Prozess kann jedoch nur bedingt durch Techniken des maschinellen Lernens automatisiert werden. Je anspruchsvoller und komplexer die Konzeption eines Fahrzeugs ist, umso mehr ist seine Leistungsfähigkeit auf eine Regelgenerierung „von Hand“ angewiesen. Dies ist arbeits- und zeitintensiv und setzt einen wohlverstandenen und formalisierbaren Problembereich voraus. Aus diesem Grunde greifen Experten mehr und mehr auf Erfahrungswissen, abgespeichert in Fällen, zurück [BK03]. Genau diesen Ansatz verfolgt das fallbasierte Schließen [Ber02], eine Methode, die auf der grundsätzlichen Annahme über die Umwelt beruht, dass ähnliche Probleme ähnliche Lösungen (Verhaltensweisen) besitzen. Eine Umweltsituation, die bereits in der Vergangenheit durch ein Verhalten erfolgreich gelöst wurde, kann im aktuellen Fall

wiederum durch dasselbe Verhalten (eventuell adaptiert) gelöst werden. Diese Methode hat den Vorteil, dass, auch wenn die Regeln, nach denen bestimmte Dinge funktionieren, nicht genau bekannt sind, doch auf gewisse Regelmäßigkeiten der Welt zurückgegriffen werden kann.

Um den Bereich möglicher Lösungen einzuschränken, wird auf Constraint-Techniken zurückgegriffen. Constraints definieren eine ungerichtete Relation zwischen Variablen. Constraint-Probleme lassen sich als Graph darstellen, indem Variablen durch Knoten und Constraints als Kanten zwischen den entsprechenden Knoten repräsentiert werden (siehe Abbildung 15). Constraints können in Form von Tabellen (nur die Werte einer Tabelle sind möglich), Funktionen bzw. Formeln (z.B. Normalgewicht = Körpergröße – 100), Prädikaten (Untergewicht :- Gewicht/Normalgewicht < 0,8), expliziten Darstellungen von Relationen (z.B. B = {(schwarz, weiß), (blau, schwarz)}) und aussagelogischen Formeln ($\alpha \wedge \beta \vee \neg\gamma \rightarrow \tau$) abgebildet werden [Ste03-ol], [RN03].

Definition 2-36: Constraint Satisfaction Problem Ein *Constraint-Satisfaction-Problem* wird durch ein Tripel CSP (V, D, C) beschrieben, wobei $V = \{v_1, v_2,...,v_n\}$ eine endliche Menge von Umweltvariablen mit assoziiertem (diskretem oder kontinuierlichem) Wertebereich einer Domäne $D = \{D_1, D_2,...,D_n\}$ ist, für die gilt $\{v_1:D_1, v_2:D_2,..., v_n:D_n\}$. C ist eine endliche Menge von Constraints, wobei jedes Constraint $c^i(V^i)$ eine Teilmenge $V_i = \{v_{i1}, v_{i2},..., v_{in}\} \subseteq V$ der Variablen zueinander in Relation setzt und deren gültige Wertekombination auf eine Teilmenge von $\{D_{i1}xD_{i2}x...xD_{in}\}$ beschränkt.

Das Ziel eines *CSP* besteht in einer eindeutigen Belegung der Variablen mit genau einem Wert aus dem jeweiligen Wertebereich, so dass alle Constraints erfüllt sind. Der *Lösungsraum eines CSP* wird durch alle möglichen unterschiedlichen Lösungen aufgespannt. Ein Beispiel soll dies verdeutlichen. Hierzu wird die häufig verwendete Form des Constraint-Graphen [RN03] verwendet:

Die Umweltsituation besteht aus der Menge aller Arbeitsplätze A_i einer Firma X. Die Firma beschäftigt drei verschiedene Nationalitäten N = {Deutsch, Spanisch, Amerikanisch}. Das Constraint besteht darin, dass keine Nationalität neben einer gleichen sitzen darf ($A1 \neq A2$, $A1 \neq A3$ etc.), um einen möglichst hohen Integrationsprozess zu erreichen. Das Verhalten bestehen darin, die Arbeitsplätze mit den entsprechenden Nationalitäten zu besetzen.

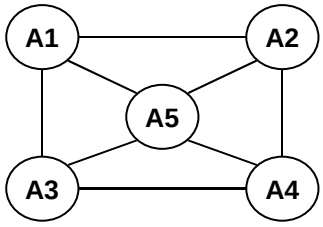


Abbildung 15: Constraint-Graph

Eine mögliche Lösung wäre: A1 = {Deutsch}, A2 = {Spanisch}, A3 = {Spanisch}, A4 = {Deutsch}, A5 = {Amerikanisch}.

CSP-Verfahren werden vor allem beim Planen (Fabrikplanung, Auftragsplanung etc.), Scheduling (Fahrpläne, Arbeitspläne etc.) und Optimieren (Optimale Konfiguration eines Computers etc.) eingesetzt [RN03], [Ste03-ol].

Die *Logik* beschäftigt sich mit der Formalisierung von Situationsbeschreibungen und folgerichtigen Schlüssen. Formal ist eine Logik eine Menge von Regeln, die definiert ist durch [RN03]:

- Eine Menge von Symbolen
- Eine Menge von Termen bzw. Formeln, entwickelt aus den Symbolen
- Eine Menge von Axiomen (Grundregeln) und
- Eine Menge von Inferenzregeln, um weitere Formeln aus bestehenden Formeln ableiten zu können.

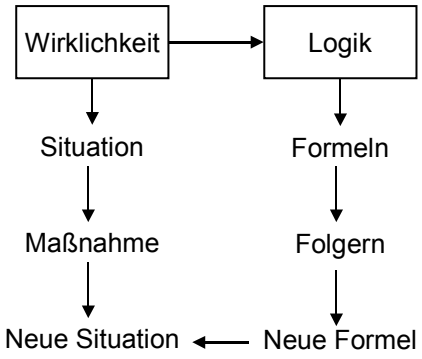


Abbildung 16: Logik [Sie92]

In der *klassischen Logik* werden Annahmen zugrunde gelegt, die plausibel und unproblematisch sind für Schlussfolgerungen. Klassische Logiken machen hierbei zwei grundlegende Annahmen [Bus99]:

- **Zweiwertigkeit**: Es wird davon ausgegangen, dass jede Behauptung entweder wahr oder falsch ist.
- **Extensionalität**: Sie besagt, dass eine mit „und“ verknüpfte Aussage genau dann und nur dann wahr ist, wenn beide Teilaussagen wahr sind [Sch03].

Nicht-klassische Logiken lockern mindestens eine dieser beiden Grundannahmen auf. Typische Vertreter sind alle Formen von *nicht-monotonen Logiken*. Nicht-monotone Logiken entstanden aufgrund von drei wichtigen Annahmen [BK03]:

- Ereignisse sind nicht immer vollständig vorhersehbar (also nicht immer deduktiv).

- Aussagen sind nicht immer eindeutig (z.B. die Aussage: Ein Hund ist groß. Für eine Maus trifft dies zu, für einen Elefanten jedoch nicht. Siehe hierzu auch Fuzzy Logic [BE05])
- Aussagen unterliegen temporären Veränderungen. So kann die Aussage, dass Dirk mit Sabine verheiratet ist, morgen schon nicht mehr stimmen.

Die Zielsetzung nicht-monotoner Logiken besteht darin, revidierbares Schließen zu ermöglichen. Sie erlauben es, Schlussfolgerungen, die sich als falsch oder unpassend erwiesen haben, zurückzunehmen und stattdessen alternative Schlussfolgerungen abzuleiten. Entsprechendes gilt für die Verhaltensweisen. Diejenigen, die sich als nicht optimal herausgestellt haben, müssen verändert werden können, auch wenn sie sich in den letzten Jahren als richtig und optimal erwiesen haben.

2.2.3.3 Auswahl geeigneter Lernverfahren zur Verbesserung der Verhaltensweisen

Um die *erfahrungsbasierte Selbstoptimierung* eines Fahrzeugs zu erreichen, muss es die Fähigkeit des Lernens besitzen. Dabei ist zunächst zu klären, welche Art von Erfahrungen in welcher Form zur Verfügung steht und welche Erfahrungen überhaupt für einen Lernprozess benötigt werden [Mit97]. Die Literatur spricht in diesem Fall von *Trainingsdaten* [WE00]. Trainingsdaten (Erfahrungsdaten) sind entscheidend für die spätere Struktur eines Lernsystems. Mit ihnen können erste Entscheidungsbäume, Regeln, Algorithmen, Neuronale Netze oder Instanzbeziehungen aufgebaut werden (siehe hierzu auch [WE00]). Liegen die entsprechenden Erfahrungen vor, ist die Zielfunktion die nächste entscheidende Einflussgröße. Da sich nicht jedes Verhalten zum Lernen eignet, muss die Zielfunktion bestimmen, was gelernt werden bzw. welches Verhalten verbessert werden soll und wie das gelernte Verhalten bewertet werden kann. Das Ziel einer erfahrungsbasierten Selbstoptimierung sollte es stets sein, aus der Vielzahl von Möglichkeiten die beste Lösung zu finden. In unserer Domäne könnte dies der kürzeste oder der schnellste Weg oder die komfortabelste oder vielleicht sogar die landschaftlich reizvollste Strecke sein, soweit der Benutzer dieses Ziel so formuliert.

Im Bereich des *Maschinellen Lernens* existieren eine Vielzahl von Lernverfahren, um einem Fahrzeug selbstoptimierende Eigenschaften zu geben (Entscheidungsbäume, künstliche Neuronale Netze, Bayessche Netze, genetische Algorithmen, regelbasiertes Lernen, instanzbasiertes Lernen, Reinforcement Learning [RN03] – um hier nur die bekanntesten zu nennen). Obwohl sie alle prinzipiell das gleiche Ziel, die Optimierung des Verhaltens eines Systems, verfolgen, unterscheiden sie sich doch wesentlich in ihren Vorgehensweisen und der Verarbeitung von Daten. Sehr gut ist dies in den Beispielen von Mitchell [Mit97] zu sehen, in denen mehrere unterschiedliche Einsatzgebiete des Lernens beschrieben werden, die eine völlig andere Herangehensweise verlangen. Im ersten Beispiel geht Mitchell auf die Lernverfahren zur Spracherkennung ein. Mit Hilfe von Neuronalen Netzen und versteckten Markov-Modellen können Reden analysiert und für den individuellen Vortragenden optimiert

werden. Ziel dabei ist es, die Reden von Einzelpersonen automatisch durch die richtige Wahl der Syntax sowie der Semantik zu verbessern [Lee89]. Im zweiten Beispiel wird eine Lernmethode für eine autonome Fahrzeugsteuerung beschrieben. Im ALVINN (Autonomous Land Vehicle In a Neuronal Network) -System [Pom89] ist es möglich, Fahrzeuge ohne Fahrzeugführer auf öffentlichen Straßen fahren zu lassen, ohne dass es zu Unfällen kommt. Die Aktionen sind Lenken, Beschleunigen und Bremsen. Als Sensoren wurden eine Farb-Stereo-Videokamera, Laser und Radar verwendet. Das Bild der Kamera wurde in einen Pixelarray transferiert und an einem 30x32-Gitter von Eingabeeinheiten umgesetzt. Die Ausgabeschicht hat 30 Einheiten, von denen jede einer Steueraktion entspricht. Es wird diejenige Aktion ausgeführt, deren zugehörige Einheit den höchsten Aktivierungswert hat. ALVINN fuhr mit 70 mph über eine Entfernung von 90 Meilen auf öffentlichen Landstraßen in der Nähe von Pittsburgh. Leider konnte das Fahrzeug jedoch nur auf den Straßen fahren, für die es trainiert worden war. Zudem zeigte es sich nicht robust gegenüber wechselnden Lichtverhältnissen und unbekannten Fahrzeugen. Mit Hilfe von Entscheidungsbäumen konnte die NASA im dritten Beispiel Himmelskörper klassifizieren [FSW+95]. Aus einer großen Datenbank wurden die Bilder von neuen Himmelskörpern mit denen bereits bekannter verglichen. Dadurch konnten die neuen Himmelskörper schneller und eindeutiger bestimmt werden.

Die Beispiele zeigen, dass nicht jedes Verfahren für alle Anwendungen gleich gut geeignet ist. So sind für kontinuierliche und diskrete Ein- und Ausgabewerte im sub-symbolischen Bereich eher Neuronale und Bayessche Netze, für rein diskrete Werte im symbolischen Bereich eher Entscheidungsbäume, regelbasiertes Lernen oder das instanzbasierte Lernen zu verwenden [Dil03-ol]. Teilweise greifen Forscher auch auf hybride Verfahren (z.B. Neuro-Fuzzy-Systeme [SSK+04]) zurück, um je nach Lernstufe mehrere lernende Verfahren in einer Anwendung einzusetzen.

2.2.3.4 Exploration versus Exploitation

Zur erfahrungsbasierten Selbstoptimierung gehört das selbst-initiierte Ausprobieren neuer Verhaltensweisen. Hierbei kann das Ausprobieren auf explorativen oder exploitativen Ansätzen beruhen. *Explorative Ansätze* sind zu wählen, wenn das Fahrzeug kein Wissen oder nur Schätzungen über den realen Zustand und das notwendige Verhalten in einer Umgebung hat. Oft ist hierfür nur ein zufälliges Ausprobieren eines neuen Verhaltens geeignet, um mögliche, bessere Zustände zu finden. *Exploitative Ansätze* sind zu wählen, wenn das Fahrzeug bereits Erfahrungen sowohl über den realen Zustand als auch über die notwendigen Verhaltensweisen besitzt. Hierbei geht es lediglich darum, eine richtige Auswahl des nach bisherigen Erfahrungen optimalen Verhaltens zu treffen. In der Literatur wird in diesem Zusammenhang von dem Exploration-Exploitation-Dilemma gesprochen [Sch03a-ol] [Ban01-ol] [HSH03]. Es ergibt sich aus dem Konflikt, vor jedem Verhalten entscheiden zu müssen, ob ein gemäß der gesammelten Erfahrungen optimaler Schritt *(exploitation)* oder ein Schritt, der das Fahrzeug in unbekannte Systemzustände bringt *(exploration)*, um somit neue Erfahrungen zu sammeln, durchzuführen ist. Allgemein wird dieses Problem im Bereich des

Reinforcement Learnings [SB98] [Mit97] [GRS03] betrachtet. Reinforcement Learning ist eine Methode aus dem Bereich der Künstlichen Intelligenz, in dem ein Fahrzeug nicht gesagt bekommt, welches Verhalten es auszuführen hat. Somit muss es selbständig und ohne Hilfe anderer durch einen Trial-and-Error-Ansatz seine Umwelt erforschen. Verhaltensweisen, die sich als erfolgreich erwiesen haben, werden verstärkt. Dadurch werden die in einer Situation möglichen Verhaltensweisen in eine Rangfolge des höchsten Nutzens gebracht und diese in Zukunft bevorzugt ausgewählt. Der Ansatz des Reinforcement Learning bezieht sich jedoch auf Problembereiche mit ausschließlich sub-symbolischen Daten und kann daher in dieser Arbeit nicht verwendet werden.

In unserem Szenario findet eine Exploration bzw. Exploitation dadurch statt, dass in einer unbekannten Situation neue Zustände über *Heuristiken* abgeschätzt werden oder auf die Erfahrungen anderer Fahrzeuge zugegriffen wird. Kennt ein Fahrzeug z.B. nicht den besten Weg von A nach B oder ist es sich mit seiner Systemeinstellung vor einer Kurve nicht sicher, befragt es andere Fahrzeuge, die dieses Problem bereits gelöst haben. So können neue Erfahrungen viel schneller und sicherer akquiriert und verbreitet werden als durch einen „einfachen“ Reinforcement Ansatz. Problematisch wird die Situation jedoch dann, wenn kein Fahrzeug eine Antwort auf ein spezielles Problem kennt. Dann bleibt dem Fahrzeug nichts anderes übrig, als sich eigenständig an das Problem heranzutasten, indem es das möglichst beste Verhalten schätzt oder „blind“ exploriert.

Definition 2-37: Heuristik *Heuristische Verfahren* sind Lösungsverfahren, die nicht auf gesicherten Erkenntnissen beruhen, sondern auf Hypothesen, Analogien oder Erfahrungen aufbauen.

2.3 Anforderung

Die in Kapitel 2.1 und 2.2 herausgearbeiteten Problembereiche bei der erfahrungsbasierten Selbstoptimierung von technisch homogenen fahrerlosen Fahrzeugen mit unterschiedlichen Wissensbasen in einer sich dynamisch ändernden Umgebung zur Erzeugung best-möglicher Verhaltensweisen werden mit den unten aufgeführten Anforderungen konkretisiert.

2.3.1 Anforderungen an die Beschreibung des Domänenwissens

Die entwickelte Basisontologie muss in der Lage sein, das Wissen über die Domäne möglichst vollständig und real abzubilden, um einen eindeutigen Kommunikationsprozess gewährleisten zu können. Allen Repräsentationsformaten (siehe Tabelle 5) liegen hierbei gleiche Anforderungen zugrunde. Dies sind die *Verständlichkeit* (Inhalt der Wissensbasis muss für Knowledge Engineer und Benutzer leicht verständlich darstellbar sein), die *Ausdrucksstärke* (Umfang und Formalisierbarkeit der Umwelt müssen vollständig sein), die *Modularität* (Wissensbasis ist leicht veränderbar, modular aufgebaut, Teile sollen leicht hinzugefügt oder geändert werden können), die *Validierbarkeit* (Möglichkeit, aus bestehendem Wissen auf neues Wissen zu schließen) und die *Flexibilität* (dasselbe Modell

soll für Wissen unterschiedlicher Bereiche einsetzbar sein). Hars [Har94] hat alle fünf Anforderungen bewertet und über den Mittelwert die durchschnittlich geeignetste Repräsentationsform ermittelt, wobei als Einschätzungsgrade gering (1), mittel (2) und hoch (3) vergeben wurden (siehe Tabelle 5).

Kriterien	Logikbasierte Ansätze	Semantische Netze	Frames	Regeln
Verständlichkeit	gering	hoch	hoch	gering
Ausdrucksstärke	hoch	gering	mittel	hoch
Modularität	gering	hoch	hoch	gering
Validierbarkeit	mittel	mittel	mittel	gering
Flexibilität	gering	mittel	hoch	gering
Mittelwert	1,6	2,2	2,6	1,4

Tabelle 5: Anforderungen an klassische Ansätze zur Wissensrepräsentation [Har94]

Wichtigste Anforderung zur Nutzung von Wissensbasen ist die *Verständlichkeit*. Verständlichkeit bedeutet in diesem Zusammenhang vor allem, dass die wichtigen Informationen einfach und schnell herauszufiltern sind. Dies wird allgemein dadurch erreicht, dass ein Modell objektorientiert aufgebaut ist, d.h. dass alle ein Objekt betreffenden Aussagen an einer Stelle des Modells (bei Frames) oder zumindest in seiner unmittelbaren Nähe (bei semantischen Netzen) gefunden werden können. In der Regel werden deshalb auch graphische Darstellungen bei komplexen Problemstellungen höher bewertet als nicht-graphische, bei denen die Einordnung der einzelnen Objekte an unterschiedlichsten Stellen vorgenommen werden kann.

Fast umgekehrt verhält es sich mit der *Ausdrucksstärke*. Im Rahmen der Ausdrucksstärke kommt es darauf an, möglichst viele Umweltsituationen abbilden und formalisieren zu können (z.B. die Tasse steht auf dem Tisch; der Tisch steht im Esszimmer). Eine hohe Ausdrucksstärke besitzen regelbasierte und logikbasierte Formate, da sie Daten, Regeln und Operatoren explizit darstellen können und eine eindeutige Präzision der Aussage darstellen (z.B. wenn Farbe der Ampel Rot, dann bremse *oder* z.B. die Mutter meiner Frau ist meine Schwiegermutter). Semantische Netze bieten den geringsten Umfang, da sie in der „reinen" Form weder die Definition von Regeln noch von Operatoren erlauben (als hybride Systeme könnten jedoch Regeln in ein semantisches Netz integriert werden). Frames bieten über Constraints die Möglichkeit, Objekte eindeutig zu spezifizieren (z.B. ab 0° Celsius gefriert Wasser) und besitzen Möglichkeiten zum Einbinden von Prozeduren.

Die *Modularität* einer Darstellung folgt der Objektorientierung. Da Frames und semantische Netze in Objekten graphisch modelliert werden, ist eine Kapselung eines bestimmten Bereiches bzw. eine Abtrennung eines Bereiches sehr gut möglich. In logikbasierten und regelbasierten Ansätzen sind die Objekte über die gesamte Wissensbasis zerstreut.

Grundsätzlich muss bei dem Aufbau einer Wissensbasis auf die Konsistenz des Inhalts geachtet werden. Dies ist umso schwieriger, je komplexer eine Umweltsituation zu beschreiben ist. In regelbasierten Situationen kann es dazu kommen, dass spezifische Regeln

allgemeinen Regeln widersprechen (allgemeine Regeln: wenn die Ampel Grün zeigt, dann fahre ich los; spezifische Regel: wenn die Ampel Grün zeigt und von links nähert sich ein Polizeiauto im Einsatz, dann halte ich an). Deshalb ist die *Validierbarkeit* dieser Darstellungsform oft nur in recht einfach strukturierten Bereichen möglich. Gleiches gilt für die logikbasierten Ansätze (allgemeine Regel: Vögel können fliegen; spezifische Regel: Ein Pinguin ist ein Vogel – er kann aber nicht fliegen). Bei den Frames können solche Ausnahmen gut über die Constraints dargestellt werden. Somit ist die Konsistenz der Wissensbasis besser zu erhalten als in logik- oder regelbasierten Systemen.

Das letzte wichtige Anforderungskriterium ist die *Flexibilität*, mit der die Formate selbständig der dynamischen Umweltsituation angepasst werden können. Logik- und regelbasierte Systeme sind hierbei starr. Eine automatische Anpassung der Regeln ist nicht möglich. Hierzu bedarf es stets des Eingreifens eines Experten, der bei Regeländerungen jedoch darauf achten muss, nicht andere, noch bestehende Regeln zu verletzen. Aufgrund der Objektorientierung der framebasierten und semantischen Ansätze ist eine flexible Anpassung einfacher. Es müssen lediglich die Relationen eines Objektes sowie das Objekt selbst verändert werden, um die Wissensbasis an die sich verändernden Verhältnisse anzupassen.

Eine Anforderung, die alle Ansätze gleichermaßen betrifft und deshalb nicht in die Analyse mit aufgenommen wurde, ist die *Vollständigkeit*. Wird die Umwelt nicht umfassend erfasst, kann das Fahrzeug auch nicht wie gewünscht funktionieren.

2.3.2 Anforderungen an die Auflösung von Verständigungsproblemen

Der effektive Nutzen von Erfahrungen ist fundamental für die erfahrungsbasierte Selbstoptimierung eines Fahrzeugs. Dabei stellt sich jedoch zum einen das Problem der Nutzung von Erfahrungen, die von anderen Fahrzeugen generiert werden, und zum anderen das Problem der Heterogenität ihrer Wissensbasen. Die Integration der „fremden“ Erfahrungen in die eigene Wissensbasis wirft deshalb einige Probleme für die Fahrzeuge auf. Die Überwindung dieses Problems macht eine Übersetzung der Erfahrungen zwischen den verschiedenen Wissensbasen der beteiligten Fahrzeuge nötig. Diese Übersetzung ist aus zwei Gründen wichtig: Erstens existieren bereits Erfahrungen in unterschiedlichster Form und unterschiedlichsten Quellen, die nicht verloren gehen sollen, und zweitens wäre es wahrscheinlich zu teuer, die gesamten Erfahrungen eines Transportsystems in einer Wissensbasis vorzuhalten. Dementsprechend ist es notwendig, geeignete Lösungen zu entwickeln, die sich die bereits existierenden Erfahrungen zu Nutze machen können.

Die entwickelte Lösung sollte hierzu folgende Anforderungen erfüllen [Bac96]:

1. Der Ansatz sollte möglichst allgemein (*generisch*) gehalten werden. Zum einen sollten die Verfahren nicht bloß für zwei spezifische Wissensbasen anwendbar sein, und zum anderen sollten bereits entwickelte Sprachen und Techniken verwendet werden, auf denen die Ansätze aufbauen können.

2. Die entwickelte Lösung sollte problemlos *erweiterbar* sein und zwar in dem Sinne, dass die Integration von neuen Ansätzen in die bestehende Lösung möglich ist, ohne eine komplettes Redesign oder eine Rekonstruktion zu fordern.
3. Die entwickelte Lösung sollte den *gängigen Standards* genügen, da bereits seit einigen Jahren Vorarbeiten auf dem Gebiet der Übertragung von heterogenen Informationen geleistet wurden.

2.3.3 Anforderungen an die Methodik zur Auswahl geeigneter Verhaltensweisen

Die Methodik, Probleme zu lösen, indem aufgrund von bestimmten Umweltsituationen auf konkrete Verhaltensweisen geschlossen wird, ist dem Menschen schon seit Jahrtausenden vertraut [RS89]. Die Herausforderung besteht darin, diese Methodik mittels automatischer Problemlösungsprozesse einem Fahrzeug zu vermitteln. Hierzu müssen grundlegende Anforderungen gestellt werden [GRS03]:

- Um ein Problem (Umweltsituation) zu verstehen, muss auf Erfahrungen zurückgegriffen werden können, die bei Bedarf angepasst werden können.
- Die Erfahrungen müssen so abgelegt (indexiert) werden, dass sie schnell wieder hervorgerufen werden können.
- Das Wissen (die Wissensbasis) muss so dynamisch gestaltet sein, dass auch neue (bis dato unbekannte) Problemsituationen gelöst werden können.

Die Methodik versucht dabei nicht korrekte Lösungen zu garantieren, sondern es wird auf die Exaktheit der Lösung zugunsten einer approximativen Lösung verzichtet. Um hierbei nicht mit einer zufälligen Auswahl von approximativen Lösungen konfrontiert zu werden, wird der Begriff der *experimentellen Zustandsdarstellung* verwendet, der sich sowohl auf die Umweltsituation als auch auf die Lösung dieser Situation beziehen kann. Es wird davon ausgegangen, dass ähnliche Umweltsituationen ähnliche Verhaltensweisen hervorrufen, ohne den Anspruch zu erheben, eine optimale Lösung zu finden. Ein derartiger Verzicht auf Optimalität spiegelt sich darin wider, dass in dieser Methodik nicht, wie bei der klassischen Logik, der Wahrheitsbegriff, sondern der Nützlichkeitsbegriff im Vordergrund steht. Lösungen sind nicht wahr oder falsch, optimal oder suboptimal, sondern mehr oder weniger nützlich. Hierbei versucht das Fahrzeug sich durch die steigende Anzahl an Erfahrungen einem „Quasi“-Optimum anzunähern. Dieses Optimum könnte rein theoretisch auch mathematisch a-priori ohne große Erfahrungen bestimmt werden. In statischen, wenig komplexen und klar definierten Umgebungen ist dies sicherlich sinnvoll. In komplexen und dynamischen Umgebungen mit zweistelligen Dimensionen stoßen diese Methoden jedoch schnell an ihre Grenzen.

Durch folgende Anforderungen ist deshalb der spezielle Bereich der experimentellen Zustandsdarstellung abgesteckt [GRS03]:

- Es müssen ausreichend viele Erfahrungen vorliegen.
- Es muss einfacher und schneller gehen, den Fall mit einer Erfahrung zu lösen, als ihn direkt mit mathematischen Formeln und Funktionen zu berechnen.
- Besonders im Umfeld mit lebenden Objekten darf die Verwendung eines Verhaltens sicherheitskritischen Anforderungen nicht widersprechen.
- Die zur Verfügung gestellten Erfahrungen sind unvollständig, unsicher und dynamisch und können deshalb nicht einfach mit einem mathematischen Modell interpretiert werden.

Zur Validierung der Konzeption ist es erforderlich, eine Simulationsumgebung zu erstellen (für AMS) bzw. eine Testumgebung aufzubauen (für MFM), die die entwickelten Ansätze empirisch unterstützt.

3 Stand der Technik

3.1 Ansätze zur Wissensrepräsentation einer Domäne

Im Bereich der Wissensrepräsentation existiert eine Vielzahl von Modellen und Methoden, mit denen Wissen dargestellt werden kann. Besonders im Bereich der Ontologien sind eine Reihe von Methoden [UG96], [GW00], Repräsentationssprachen [KLW95], [DFH+00] und Modellierungswerkzeuge [FFR97], [GEF+99], [KMS+03] entwickelt worden. Nicht immer ist jedoch der Einsatz von Ontologien notwendig. Dieser hängt ab vom speziellen Anwendungsgebiet, von der Komplexität der zu modellierenden Umwelt und vom Experten, der das Wissen bereitstellt.

Die nachfolgenden Kapitel gehen kurz auf die verschiedenen Möglichkeiten der Wissensrepräsentation ein, bevor dann in Kapitel 3.1.3 die aktuellen Ontologien im technischen Bereich vorgestellt werden. Diese Ontologien basieren auf der Grundlage von semantischen Netzen, Regelnetzwerken, Frames oder der Prädikatenlogik.

3.1.1 Deklarative Wissensrepräsentation

3.1.1.1 Objekt-Attribut-Wert-Tripel

Diese „einfachste“ Form der Darstellung von Wissen steht in Analogie zu den relationalen Datenbanken und semantischen Netzen [Sch00-ol]. Dabei können *Objekte* entweder physische Entitäten oder begriffliche Einheiten sein. *Attribute* sind allgemeine Charakteristika oder Eigenschaften, die mit Objekten assoziiert werden. Größe, Form und Farbe sind typische Attribute von physischen Objekten. Das dritte Element des Tripels ist der Wert eines Attributs. Der *Wert* kennzeichnet die spezifische Beschaffenheit eines Attributs in einer bestimmten Situation.

Objekt	Attribut	Wert
Apfel	Farbe	Rot
Apfel	Herkunft	Israel
Apfel	Haltbarkeit	gut
Trauben	Farbe	Blau
Trauben	Herkunft	Italien
Trauben	Haltbarkeit	mittel

Tabelle 6: Objekt-Attribut-Werte Tabelle

3.1.1.2 Ansätze im Bereich der semantischen Netze

Der Ansatz des semantischen Netzes (engl. Semantic Web) besteht darin, Wissen durch Softwareprogramme verarbeiten zu können. Das semantische Netz besteht aus zwei Primitiven: dem Knoten, der durch einen Begriff beschrieben wird, sowie den Kanten, die einer zwei-stelligen Relation zwischen den Knoten entsprechen. Entscheidend hierbei ist, dass, im Gegensatz zum framebasierten Ansatz, die Knoten selbst keine Informationen

enthalten. Alles Wissen über die Knoten wird durch die Verbindungen repräsentiert, die von ihnen ausgehen. Ein Schwachpunkt hierbei besteht darin, dass n-stellige Verbindungen im semantischen Netz nicht erlaubt sind. Sollen n-stellige Verbindungen dennoch hergestellt werden, müssen diese über mehrere binäre Verbindungen dargestellt werden [Fer03].

Eine binäre Verbindung zwischen zwei Knoten kann unter anderem sein [Hel01]:

- Hierarchische Verbindung (auch „ist_ein“ Verbindung genannt): z.B. ist Shuttle ein Unterbegiff zu Schienenfahrzeug, Schienenfahrzeug ein Unterbegriff zu Fahrzeug. Manche semantischen Netze erlauben zudem multiple Vererbungen. So kann Fahrgast als Unterbegriff sowohl von Transportgut als auch von Mensch modelliert werden.
- Instanzverbindung: z.B. ist Siemens S-X23 eine Instanz_von Siemens-S, da Siemens S-X23 ein "Element der Menge der Objekte der Klasse Siemens-S" [Her01] ist. Instanzen besitzen Eigenschaften, die sie eindeutig als konkretes Objekt charakterisieren. So besitzt das Shuttle Siemens S-X23 eine Fabrikationsnummer (hier: X23), ein spezifisches Gewicht, eine spezifische Farbe etc. Die Eigenschaften werden durch Eigenschaftskanten (z.B. hat_Fabrikationsnummer) beschrieben [Abd00-ol].
- Partitive Verbindung: z.B. ist das Fahrwerk ein Teil_von Shuttle.
- Beziehungsverbindung: z.B. „Lampe liefert Licht“ oder „Fahrzeug fährt auf Schiene“.

Ein typisches semantisches Netz, das Wissen über einen elektrischen Raumerhitzer darstellt, findet sich in Abbildung 17. Basierend auf diesem Netz kann ein Softwareprogramm erkennen, wozu z.B. eine Lampe benötigt wird.

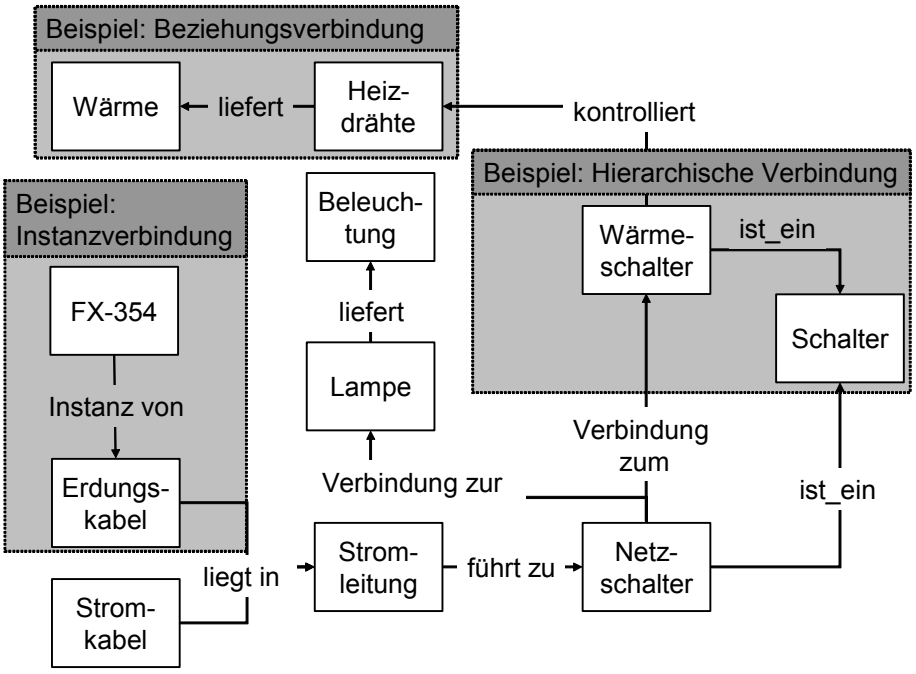


Abbildung 17: Semantisches Netz eines elektrischen Raumerhitzers [GJ02-ol]

Ein weiteres Semantisches Netz, erweitert durch eine Regel, stellt ein Automobil mit Relationen zum Motor und Fahrwerk dar (Abbildung 18).

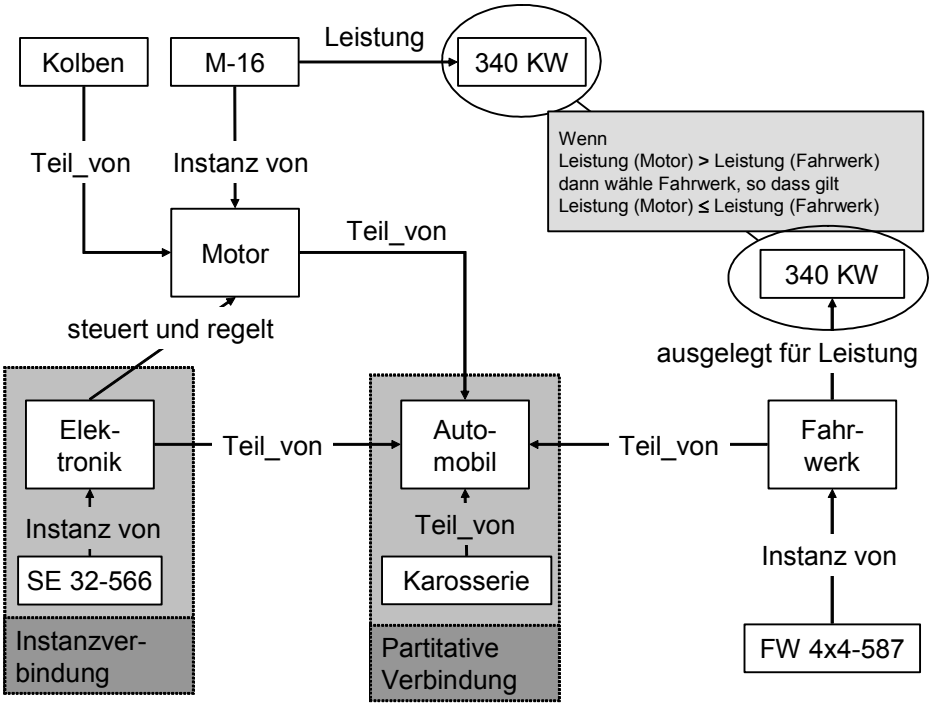


Abbildung 18: Semantische Darstellung eines Automobils [Ang03]

Obwohl als Bezeichner für Knoten sinntragende Begriffe verwendet werden, kann dieser Sinn nur vom Menschen identifiziert werden. Für ein Computersystem selbst wird dadurch keine Semantik definiert. Insofern ist der Ausdruck „Semantisches Netz“ eher irreführend. Die Semantik der semantischen Netze ergibt sich z.B. durch die Übersetzung der Kanten und Knoten in die Prädikatenlogik erster Stufe, die von einem Computer gelesen werden kann (siehe Abbildung 19) oder durch eine Einbettung des semantischen Netzes in eine komplexe ontologische Struktur.

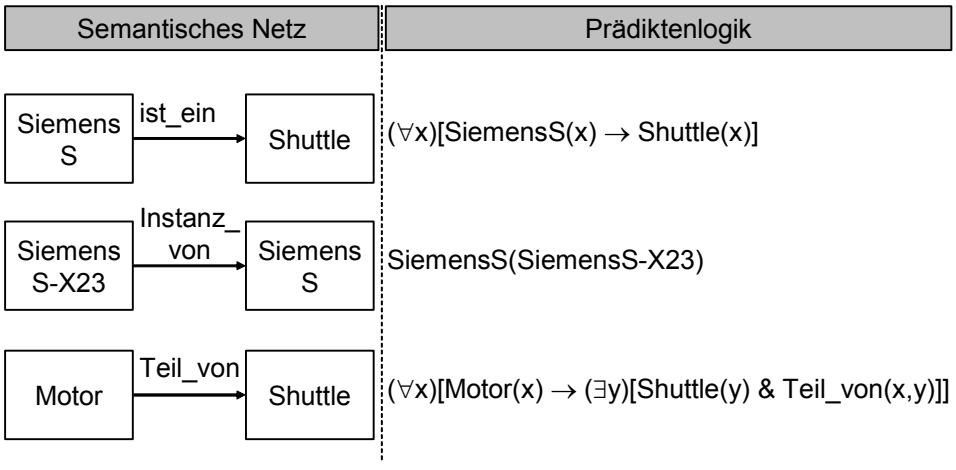


Abbildung 19: Übersetzung von Teilen eines semantischen Netzes in die Prädikatenlogik [Abd00-ol]

Eine weitere Anwendung im Bereich der semantischen Netze ist ein Wein-Agent, basierend auf dem Inference Web Browser[8] der Stanford University[9]. Dieser Agent generiert automatisch nach Angaben eines präferrierten Gerichts Vorschläge für die Weinauswahl. Weitere semantische Netz-Prototypen sind OntoWeb[10], Agentcities[11] oder FOAF (FriendOfAFriend)[12]. Das FOAF-Projekt stellt einen Ansatz für die Implementierung eines Vertrauensmodells dar. Über Homepages können Personen ihre persönlichen Informationen wie E-Mail-Adressen und Kontakte zu anderen Personen in maschinenverständlicher Sprache hinterlegen. Auf Basis dieser Informationen können E-Mails nach Vertrauenswürdigkeit gefiltert oder Personen mit ähnlichen Interessen gefunden werden.

3.1.1.3 Prädikatenlogik

Die *Prädikatenlogik* oder Logik erster Ordnung ist ein Teilgebiet der Logik. Man kann sie als Erweiterung der Aussagenlogik ansehen, die zusätzlich zur Verknüpfung von Aussagen („und“, „oder“) auch die Eigenschaften von Begriffen und deren Geltungsbereiche betrachtet, wobei erstere durch Prädikatssymbole und Funktionssymbole, letztere durch Quantoren beschrieben werden. Die Grundlagen für eine formale Sprache der Prädikatenlogik (erster Ordnung) wurden von Ludwig Gottlob Frege 1879 in seiner „Begriffsschrift“ [Wie04-ol] gelegt.

Meistens wird für die Rekonstruktion der Umwelt die Prädikatenlogik der ersten Stufe verwendet (englisch: first-order predicate calculus oder first order logic, FOL) [Kel03], da sie im Wesentlichen die geforderten Voraussetzungen für eine geeignete Notation erfüllt. Diese Eigenschaften sind Kommunikativität, Ausdrucksfähigkeit und Operationalisierbarkeit. Kommunikativität bedeutet, dass die Notation sowohl auf der syntaktischen als auch auf der semantischen Ebene eindeutig ist, damit zum Beispiel verschiedene Personen oder Gruppen über das Modell diskutieren können. Hinsichtlich der Ausdrucksfähigkeit muss die gewählte „Sprache“ in der Lage sein, die in der Domäne beobachteten Phänomene eindeutig darzustellen. Abschließend besteht mit der Forderung nach Operationalisierbarkeit die Aufgabe darin, das entstehende Modell in ein lauffähiges System umwandeln zu können, welches die vorgegebenen Aufgaben erfüllt [Mat97].

Die Prädikatenlogik der ersten Stufe ist ein sehr ausdrucksstarker Formalismus, was sowohl als Vorteil als auch als Nachteil gesehen werden kann. Einerseits lassen sich Sachverhalte eindeutig in dieser Sprache darstellen, andererseits ist ihre Ausdruckskraft zu mächtig, als dass man effiziente Verfahren für ihre Verarbeitung angeben könnte [RN03]. So ist die Frage, ob eine Aussage aus einer anderen logisch folgt, nicht unentscheidbar. Es lässt sich also ein Deduktionssystem implementieren, das korrekt und vollständig ist und immer terminiert. Der mögliche Einwand, man könnte das Wissen in geeigneter Form codieren und zusätzlich

[8] http://belo.stanford.edu:8080/iwbrowser/index.jsp
[9] http://ksl.stanford.edu/people/dlm/webont/wineAgent/
[10] http://ontoweb.aifb.uni-karlsruhe.de/
[11] http://www.agentcities.org/
[12] http://www.foaf-project.org/

Kontrollinformationen bereitstellen, so dass das System nach endlicher Zeit auf jeden Fall eine Antwort gibt, führt jedoch zu einem Übergang von der Wissensrepräsentation zur Programmierung. In diesem Zusammenhang kann das System Prolog (Programming in Logic) genannt werden, welches eben diese prozedurale Repräsentation des Wissens unterstützt [TW02].

Das zurzeit meist diskutierte Projekt in diesem Forschungsgebiet ist das KOWIEN-Projekt [Ala03]. KOWIEN (Kooperatives Wissensmanagement in Engineering-Netzwerken) befasst sich mit der Umsetzung von Wissensmanagement mit Kompetenzprofilen auf der Basis von Ontologien und verwendet die Repräsentationssprache F-Logic. F-Logic (Frame-Logic) ist eine formalsprachliche Spezifikation, die eine Kombination der Repräsentationsformalismen Frames und Prädikatenlogik darstellt [Dec98]. Die Anlehnung an prädikatenlogische Spezifikationen findet sich unter anderem in der Möglichkeit zur Definition allgemeingültiger Axiome (Inferenzregeln) wieder. Es kann in einer ontologiebasierten Faktenbasis Wissen über die Domäne abgelegt werden, anhand dessen mit Inferenzregeln „neues“ Wissen expliziert werden kann. Die bei KOWIEN eingesetzte Ontologie-Entwicklungsumgebung ist OntoEdit[13] der Firma Ontoprise.

Die KOWIEN Ontologie in der derzeitigen Fassung liefert lediglich rein taxonomische Beziehungen (Mercedes ist_ein Auto, oder Stunde ist_eine Zeitspanne) zwischen Begriffen. Die nicht taxonomischen Beziehungen (Auto fährt_auf Schiene oder Batterie liefert Strom) wurden bis jetzt nocht nicht definiert. Zudem stellen die Verfasser fest, dass "die derzeitige Struktur […] für Zwecke des Wissensmanagements […] weniger geeignet ist" [Ala03, S.45]. Eine Weiterentwicklung von KOWIEN wurde bis jetzt nicht vorangetrieben.

3.1.2 Prozedurale Wissensrepräsentation

3.1.2.1 Regelbasierte Systeme

Regeln stellen *Wenn-Dann Verbindungen* zwischen Begriffen her [BK03]. Häufig werden sie bei der Steuerung von Produktionsvorgängen eingesetzt, so dass die Konsequenz mit einer Aktion gleichzusetzen ist. Für Regeln dieser Art hat sich der Ausdruck „Produktionsregel“ durchgesetzt. Regeln, die immer gelten, werden deterministische Regeln genannt [BK03].

Alle Regeln zusammen bilden die Regelbasis. Um diese für Entwickler, Knowledge-Engineers und Experten transparent und beherrschbar zu halten, ist es notwendig, mehrere einzelne Regeln in Regelpaketen unter einem Namen zusammenzufassen. Ein einzelnes Regelpaket kann dabei getrennt von anderen ausgeführt werden. So lassen sich umfangreiche Wissensbasen in Teilprobleme zerlegen und sukzessive entwickeln. Gibt es Kriterien für die Aktivierung von Regeln, kann sogar deren Effizienz gesteigert werden. Außerdem wird dadurch ihre Verständlichkeit verbessert [BSW91].

[13] http://www.ontoprise.de/products/ontoedit

Werden mehrere Regelpakete miteinander verknüpft, spricht man von einem Regelnetzwerk. So könnte das in Abbildung 20 dargestellte Regelnetzwerk einen Produktionsprozess darstellen, in dem die Kreisbögen Konjunktionen ($\wedge$) und die fehlenden Kreisbögen Disjunktionen ($\vee$) symbolisieren. Die Begriffe A bis F wären dann die Rohstoffe, die Begriffe G bis L Zwischenprodukte und M das Endprodukt. Dabei stehen Disjunktionen für Fertigungsalternativen. Das Zwischenprodukt H könnte also aus dem Rohstoff C oder D hergestellt werden. Zu berücksichtigen ist allerdings, dass ein originäres Regelnetzwerk den Faktor Zeit außer Acht lässt und keine Reihenfolge für die Inferenz festlegt. In der KOWIEN Ontologie wird das Problem dadurch gelöst, dass der Zeitpunkt und die Zeitspanne explizit definiert werden. So können Aktionen oder Ereignisse in temporärer Relation zueinander gestellt werden. Die genaue Bestimmung der Zeit ist auch Gegenstand der in dieser Arbeit entwickelten Ontologie.

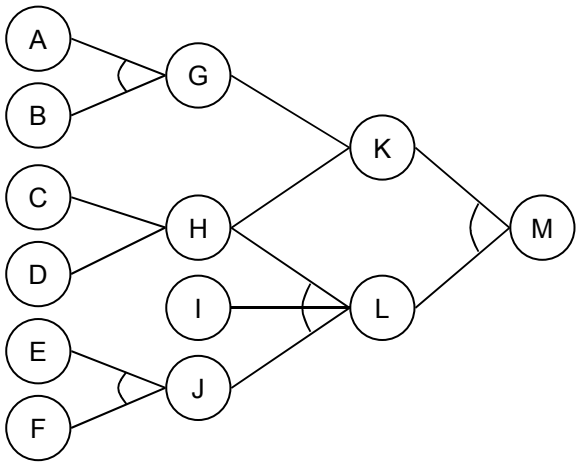


Abbildung 20: Regelnetzwerk

Das Durchlaufen und Auswerten der Regelbasis ist von links nach rechts oder von rechts nach links möglich [BK03]. Hierfür werden zwei Verfahren unterschieden:

Vorwärtsverkettung (datengetrieben)
Die *Vorwärtsverkettung* wird dann angewendet, wenn die Zustände der Zielobjekte noch nicht bekannt sind. Als Ausgangsbasis dient das bekannte fallspezifische Wissen (Erfahrungen). Vorteilhaft ist, dass auf Änderungen in der Wissensbasis schnell reagiert werden kann. Das Netzwerk wird, ausgehend von der neuen oder geänderten Erfahrung, vorwärtsverkettend durchlaufen [BK03].

Rückwärtsverkettung (zielgetrieben)
Um den Zustand oder die Situationsbeschreibung eines Begriffs zu überprüfen, wird im Allgemeinen die *Rückwärtsverkettung* angewendet. Bei dieser Inferenzmethode ist der Zielbegriff Ausgangspunkt der Analyse, so dass sie auch als „zielgetriebene“ oder „zielorientierte Inferenz“ bezeichnet wird.

Inferenzkomponenten lassen sich nicht immer nach Vorwärts- bzw. Rückwärtsverkettung einteilen. Meistens wird eine Wissensbasis mit einer kombinierten Strategie bearbeitet („means-end analysis“ [HHB00]). Diagnosesysteme arbeiten nach diesem Prinzip. Zunächst werden mit der datengetriebenen Inferenz aus Befunden mögliche Hypothesen über Ursachen abgeleitet, die dann durch die zielorientierte Inferenz falsifiziert bzw. verifiziert werden. Die meisten heute bekannten wissensbasierten Systemen bzw. Expertensysteme arbeiten auf Basis dieser erprobten und gut verstandenen Regeltechnik [BK03]. Für den Erfolg dieses Ansatzes gibt es u. a. folgende Gründe:

- Regeln stellen eine vertraute Ausdrucksweise dar, wie sie etwa in Gesetzen Verwendung findet [Kur89].
- Regeln bilden einen guten Kompromiss zwischen einer verständlichen Wissensdarstellung und formalen Ansprüchen.
- Expertenwissen lässt sich in großem Umfang in Form von Regeln darstellen.
- Eines der wichtigsten Ziele bei der Entwicklung von Expertensystemen wird erreicht, nämlich eine Nähe zum menschlichen Denken herzustellen [BK03].
- Regeln ermöglichen es, Wissen in viele kleine eigenständige Wissenseinheiten zu stückeln, so dass eine modulare und leicht veränderbare Wissensbasis geschaffen werden kann [Pup91].

Ein in der Literatur oft genanntes Regelsystem ist MYCIN [RN03], [Sho81]. MYCIN ist ein medizinisches Regelsystem (auch Expertensystem genannt). Es benutzt ein einfaches Regelsystem mit Sicherheitsfaktoren, um aus verschiedenen Symptomen die wahrscheinlichste Diagnose zu erstellen. Die Form der Regeln lautet:

$$CF[B] = CF[B, \{A\}] = 0{,}8*1{,}0 = 0{,}8$$

und bedeutet: Unter der Voraussetzung, dass die Wahrscheinlichkeit des Symptoms A =1 ist (also 100% sicher ist), ist die Wahrscheinlichkeit des Symptoms B gleich 0,8. CF (certainty factor) bezeichnet hierbei den Sicherheitsfaktor mit einem bestimmten Wahrscheinlichkeitsmaß zwischen 0 und 1, der die Korrelation dieser Regel ausdrückt (siehe Abbildung 21).

Um nun aus der evtl. großen Anzahl an anwendbaren Regeln einen Schluss ziehen zu können, werden die in den verschiedenen Regeln auftauchenden Sicherheitsfaktoren über geeignete Junktoren miteinander kombiniert und zu einem einzigen Wert verrechnet. Genauso verfährt man mit Regeln, die gegen ein Krankheitsbild sprechen. Diese Regeln ergeben durch geeignete Kombinationen wieder einen Wert. Aus diesen positiven und negativen Evidenzen errechnet sich dann die Gesamtwahrscheinlichkeit eines Krankheitsbildes.

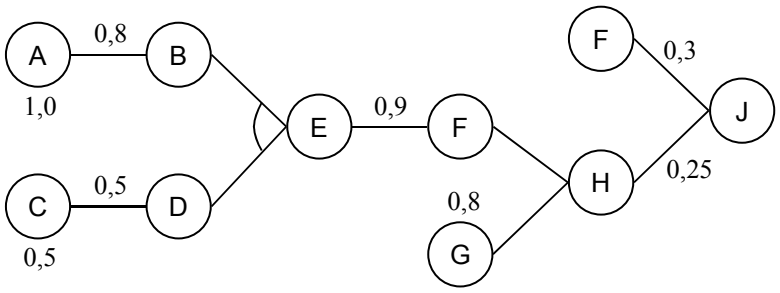


Abbildung 21: Regelnetzwerk mit Wahrscheinlichkeitsfaktoren [BK03]

In diesem Regelnetzwerk gilt z.B. eine Wahrscheinlichkeit:

- CF[B] = CF[B, {A}] = 0,8*1,0 = 0,8
- CF[B∧D] = min{0,8;0,25} = 0,25 = CF[E]
- CF[F∨G] = max{0,225;0,8} = 0,8 = CF[H]

3.1.2.2 Framebasierte Ansätze

Die Idee der *Frames* basiert auf der Annahme, dass unsere Informationsverarbeitung wesentlich von unseren Erfahrungen abhängt. Minsky [Min75] ging schon vor Jahrzehnten davon aus, dass immer dann, wenn eine neue Situation auftritt oder wenn sich die Sicht auf ein Problem gravierend ändert, man im Gedächtnis nach passenden erlebten Situationen sucht. Die gespeicherte Situation wird dann der neuen Situation angepasst.

Frames stellen eine Kombination aus semantischem Netz, Objekt-Werte-Tripel und Regeln dar [Ang03]. Ein Frame-System besteht hierbei aus einer Anzahl von Frames, die in semantischen Netzen als Knoten bezeichnet werden, und die über Beziehungen (Kanten) miteinander verknüpft sind. Im Gegensatz zu semantischen Netzen erlaubt die Repräsentation durch Frames jedoch eine komplexere strukturelle Darstellung von relevanten Eigenschaften und Ausprägungen eines Knotens. So können in einem Knoten Informationen wie im Objekt-Werte-Tripel abgespeichert werden. Zur Vervollständigung kann ein Knoten selbst, aber auch die Relation zwischen den Knoten Regeln besitzen [Ang03].

Der zentrale Modellierungsansatz von Frames besteht aus *Klassen* (die durch einen Begriff gekennzeichnet sind) mit bestimmten Eigenschaften und Verhaltensweisen, die *Slots* genannt werden (siehe Tabelle 7). Slots besitzen keine globale Reichweite, sondern sind nur auf die von ihnen beschriebenen Klassen und Unterklassen beschränkt. Slots können Werte (deskriptive Slots), Verweise auf andere Klassen (relationale Slots) und Prozeduren (prozedurale Slots) enthalten [Reu04], [Abd00-ol].

Klasse (wird durch einen Begriff gekennzeichnet)			
Name	**Typ**	**Kardinalität**	**Ausprägung**
Prozedurale Slots	Regel, Funktion	Single	Value={}
Relationale Slots	Klasse, Instanz	Single, Multiple	Klasse =()
Deskriptive Slots	Integer, Float, String, Symbol, Boolean,	Single, Multiple	

Tabelle 7: Struktur eines Frames

Die *deskriptiven Slots* können einzelne Werte (reelle Zahlen, Symbole, Boolsche Werte etc.) oder eine Menge von Werten beinhalten. Zusätzlich besteht mit Default-Werten die Möglichkeit, bei mehreren existierenden Werten in einem Slot eine Voreinstellung nach der wahrscheinlichsten Ausprägung eines Objektes vorzunehmen (z.B. die Default-Einstellung, dass ein Fahrrad zwei Räder hat). Die Einbettung von prozeduralen Anweisungen in einen *prozeduralen Slot* verdeutlicht den Vorteil von Frames z.B. gegenüber den semantischen Netzen. Prozedurale Elemente wie IF-NEEDED, IF-ADDED oder IF-REMOVED erlauben z.B. die kalkulatorische Berechnung von fehlenden Werten auf der Grundlage von statistischen Formeln bzw. die Ausführung von Verhaltensweisen [MAA+03]. Schließlich können *relationale Slots* oder so genannte Pointer in Frames gespeichert werden. In diesem Fall führt eine Verbindung von einem Slot eines Frames zu einer Instanz oder Klasse eines anderen Frames.

Bei allen Slots unterscheidet man grundsätzlich nach geerbten und proprietären Slots. *Geerbte Slots (SG)* spiegeln den taxonomischen Aufbau von Frame Systemen wider. Frames, die als Unterklasse von übergeordneten Frames gelten, erben die von der Oberklasse ausgewiesenen Slots, d.h. eine Mehrfachangabe von Slots in den einzelnen zusammenhängenden Klassen und Unterklassen ist nicht notwendig. Ebenso unterstützen Frames multiple Vererbungen. Multiple Vererbungen sind relevant, wenn eine Klasse eine Unterklasse von zwei oder mehreren Oberklassen darstellt. *Proprietäre Slots (SP)* beziehen sich ausschließlich auf einen Frame und sind nur innerhalb dieses Frames anwendbar.

Beispiele frame-basierter Systeme sind Ontolingua[14] bzw. das Nachfolgetool Protégé[15]. Anwendung finden Frames vor allem dann, wenn Erfahrungen akquiriert oder verwaltet werden sollen, wie im Projekt Gene Ontology [YKN+03], über das biologische Daten ausgetauscht werden können, im Projekt PC-DIY [Yan02], in dem PC-Anwender selbständig Fragen an eine PC-Hilfe-Datenbank richten und Antworten eingeben können oder im Projekt Cedex des Umweltbundesamtes in Österreich [SM02], das für die Abbildung, Verwaltung und Auswertung von Daten diverser ökologischer Projekte verwendet wird.

[14] http://www.ksl.stanford.edu/software/ontolingua/
[15] http://protege.stanford.edu/

3.1.3 Wissensrepräsentation im mathematisch-physikalischen Umfeld

Für die Wissensrepräsentation von Fahrzeugen müssen insbesondere mathematische bzw. physikalische Zusammenhänge modellierbar sein. Der Einsatz von Frames, der Prädikatenlogik und der semantischen Netzen hat gerade in diesem Umfeld neue Möglichkeiten geschaffen, Wissen formalisierbar zu machen. Dabei hat sich die Forschung in den letzten Jahren stark auf die Vermittlung von Faktenwissen konzentriert. Ein Problem hierbei bleibt die Darstellung von komplexeren Zusammenhängen wie z.B. die Darstellung von mathematischen Formeln. So können Formeln zwar per XML an jedes Softwareprogramm übermittelt werden; das Problem jedoch bleibt, dass die Programme die Formeln nicht interpretieren können. So kann der Buchstabe *s* sowohl das Zeichen für eine Strecke, aber auch für die Zeiteinheit Sekunde sein. Uns Menschen wird im Gesamtzusammenhang der Formel verständlich, um welche Größe es sich handeln muss. Ein Softwareprogramm benötigt jedoch eine genauere Darstellung der Zeichen in einem strukturierten Weltbild. Eine notwendige Voraussetzung hierfür bieten Ontologien, die komplexe mathematische Beziehungen bzw. Formeln darstellen können. Das Problem der Darstellung von nur binären Beziehungen innerhalb von Ontologien bleibt jedoch bestehen.

Die Ergebnisse dieser Arbeit im Bereich der Darstellung einer Basisontologie im mathematischen bzw. physikalischen Bereich wurden stark von EngMath [GO94] Physsys [Bor97], SUMO [NP02], DOLCE [GGM+03], CYC [Cyc04-ol] und den Arbeiten von Annamalai und Sterling [AS03] beeinflusst.

EngMath [Gen95] und Physsys [BAP+95] bauen auf der Sprache KIF[16] (Knowledge Interchange Format) auf und sind ein umfangreicher Versuch, die Semantik der mathematischen bzw. physikalischen Ausdrücke in Klassen zu spezifizieren. Es war die Vision der Autoren, durch eine beschreibende, maschinenlesbare Repräsentation der Mathematik bzw. Physik, eine eindeutige Kommunikation zwischen Softwareagenten zwecks Austauschs von mathematischem bzw. physikalischem Wissen zu erreichen [GO94]. Obwohl die Darstellung vollständig und richtig erscheint, ist sie schwer zu verstehen (lesen) und nur unzureichend auf derzeitig zur Verfügung stehende Systeme anzuwenden [AS03]. Die dort angewandten Regeln (Axiome) sind nicht intuitiv und für Physiker bzw. Mathematiker kaum zu verstehen, verfügen sie nicht über ein umfangreiches, fundiertes Wissen im Bereich von KIF. Ein weiteres Problem besteht in der Mächtigkeit der Sprache KIF. Es ist mit den derzeitig verfügbaren Sprachen (Daml, Oil, RDF) nicht möglich, die Eng-Math-Ontologie auf das Web zu übertragen und damit kommunizierbar zu machen, da diese Web-Sprachen keine mehrstelligen Beziehungen, Funktionen und Regeln (Axiome) darstellen können, die jedoch unerlässlich für die Darstellung von EngMath sind [AS03]. Die in der vorliegenden Arbeit entwickelte Basisontologie basiert zwar auch auf mathematischen Formeln, die jedoch nicht mit ihren einzelnen Größen kommuniziert werden. Die Formeln dienen als Prozeduren innerhalb einer Klasse zur Berechnung von Werten, die dann als Float (also als eine reelle Zahl) übergeben werden. Die Eindeutigkeit der Formeln resultiert daraus, dass sich die Berechnungen immer auf die gleiche Klasse der Basisontologie beziehen, in der eine

[16] http://logic.stanford.edu/kif/kif.html

Referenzformel für alle Systeme definiert ist. So beziehen sich z.B. alle Berechnungen zur Ermittlung der Vertikalbeschleunigung auf die entsprechende Klasse.

SUMO [NP02] wurde im Rahmen der IEEE Standard Upper Ontology Working Group entwickelt und basiert auf den Ansätzen von EngMath und Physsys. Das Ziel dieser Arbeitsgruppe ist die Entwicklung einer Standard Ober-Ontologie, die die Dateninteroperabilität, die Informationssuche und Abfrage, die automatische Schlussfolgerung und die Kommunikation unterstützt. SUMO ist bereits in viele verschiedene Repräsentationsformate übersetzt worden (DAML, LOOM, XML, Protégé 2000). Die Entwicklungsumgebung ist jedoch, wie auch bei EngMath, KIF bzw. seine weiterentwickelte Version SUO-KIF. Dies bedeutet, dass eine Portierung in das Web nur bedingt möglich ist. Mit SUMO wurden bereits eine Reihe von Domänen-Ontologien entwickelt (Service-Ontologien, Ontologie über biologische Viren, Finanzontologie, Ontologie über Geländemerkmale, Ontologie über Massenvernichtungswaffen und Terrorismus, Ontologie über Geographie, Staatswesen, Wirtschaft und Transport sowie eine Ontologie über chemische Elemente; siehe hierzu http://ontology.teknowledge.com). SUMO hat die höchste Ähnlichkeit mit der in dieser Arbeit entwickelten Basisontologie, die Teile von SUMO wiederverwendet. Allerdings war es nicht möglich, das selbstoptimierende, hierarchische mechatronische Transportsystem (siehe Abbildung 3) mit SUMO vollständig zu beschreiben. Zudem baut die Entwicklungsumgebung wie bei EngMath auf KIF bzw. einer weiterentwickelten Version SUO-KIF auf.

Eine weitere Möglichkeit der Darstellung mathematischer bzw. physikalischer Formeln bieten Annamalai und Sterling [AS03]. Sie entwickelten basierend auf der Logik von EngMath und SUMO eine weiterführende Methodik zur Darstellung von Funktionen im Web. Dieser Ansatz bietet interessante Möglichkeiten der Darstellung von Transportsystemen durch eine Ontologie. Der Ansatz der Übertragbarkeit arithmetisch-logischer Ausdrücke über OpenMath (http://www.openmath.org/cocoon/openmath/index.html) wurde in dieser Arbeit jedoch nicht verwendet, da nicht die einzelnen Elemente einer Formel, sondern nur deren Ergebnisse selbst übermittelt werden. Zudem ist die Ontologie nicht in der Lage, prozedurale Fakten (also z.B. Regeln) darzustellen, die für das vorliegende Szenario wichtig sind, um Entscheidungen treffen und ausführen zu können.

DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [GGM+03] wird innerhalb des WonderWeb-Projekts entwickelt. Das Projekt zielt darauf ab, eine Infrastruktur bereit zu stellen, die es erlaubt, Ontologien im semantischen Netz zu entwickeln. Im Gegensatz zu einfach strukturierten Ontologien (lediglich Taxonomien) liegt der Schwerpunkt von DOLCE in der Bereitstellung einer umfassenden Struktur für die Kommunikation verteilter künstlicher Agenten oder der Kommunikation zwischen Agenten und Menschen. Umfassend ist die Struktur insofern, als sie möglichst nah die natürliche Sprache und das allgemeine menschliche Verständnis darstellen kann. Der Anspruch von DOLCE besteht hierbei nicht darin, als „universelle“ Standard-Ontologie zu dienen, sondern lediglich eine sinnvolle Erweiterung zu den bereits bestehenden Ontologien wie z.B. WordNet zu schaffen.

Insofern ist DOLCE wie auch die in dieser Arbeit entwickelte Basisontologie eine sinnvolle Erweiterung der bestehenden Landschaft, um auf spezifische Domänen genauer eingehen zu können.

OpenCyc [Cyc04-ol] ist die frei verfügbare Version des Cyc-Projects und die wohl umfassendste Darstellung eines logischen Systems. Cyc enthält Module für die Sprachanalyse, Datenbanken, Ontologiedesign und E-Mail Verarbeitungen. Ihre Datenbank besteht aus mehr als 6000 Klassen. Cyc wurde mit einer neu entwickelten aussagelogischen Sprache namens CycL entwickelt. Diese Sprache ähnelt jedoch sehr dem bereits bekannten KIF-Format. Leider sind die Cyc-Wissensbestände nur zu einem kleinen Teil frei erhältlich (OpenCyc). Über die Cyc-Homepage können (unvollständige) Informationen zu ca. 3.000 Klassen eingesehen werden. Diese „Upper Cyc" genannten Klassen geben Auskunft über Inhalte und Beschaffenheit der allgemeinsten Ebenen der Hierarchie. Durch vielfältige Beziehungen bereits zwischen den Upper-Cyc-Klassen und durch Mehrfachvererbung (eine Klasse kann mehrere Oberklassen besitzen) ist die Top-Level-Hierarchie von Cyc schwer durchschaubar. Zudem ist die direkte Arbeit mit OpenCyc für Nichtbeteiligte schon wegen der eingeschränkten Zugänglichkeit der Wissensbasis unmöglich; auch die Arbeit mit den veröffentlichten Oberklassen ist wegen der „verworrenen" Hierarchie (vgl. [Gua98]) fast unmöglich. Aus diesem Grunde wurde entschieden, nicht mit OpenCyc zu arbeiten.

3.2 Ansätze zur Auflösung von Verständigungsproblemen

Eine zusätzliche Komplexität besitzen Fahrzeuge immer dann, wenn verteilte Lerneffekte mit einbezogen werden sollen. Dies setzt jedoch voraus, dass eine homogene Repräsentation des Wissens zur Verfügung steht, auf Grundlage dessen neue Erfahrungen entstehen bzw. ausgetauscht werden können [VJB+97], [UG96], [Gru93]. Ist die Homogenität der Wissensrepräsentation nicht gegeben, müssen erst Beziehungen zwischen den lokalen Ontologien entwickelt werden. Die am meisten diskutierten Ansätze in diesem Bereich sind das *Mergen* oder *Mappen* von Ontologien [Kle01]. Darüber hinaus existiert eine Methode des *Übersetzens* vieler lokaler über eine globale (Basis-) Ontologie.

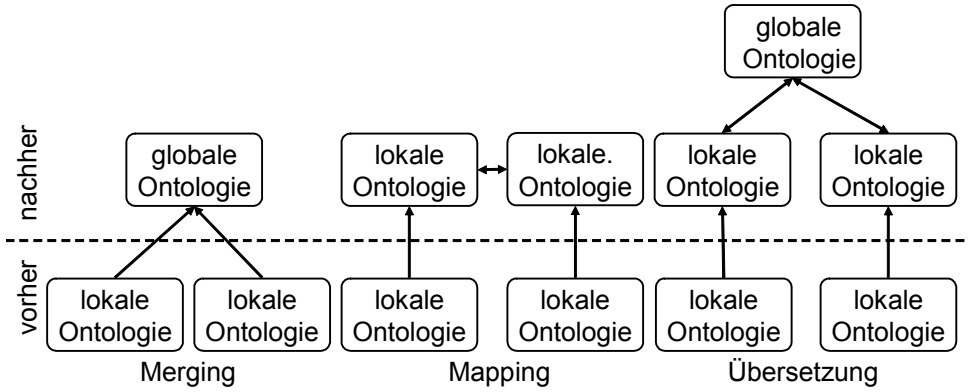


Abbildung 22: Ansätze zur Kommunikation über Ontologien [WVV+01], [Stu03]

Merging: Das Ziel ist die Entwicklung einer globalen Ontologie aus den jeweiligen lokalen Ontologien [NM00], [SM01]. Die globale Ontologie enthält Begriffe aus allen Ontologien, zusammengestellt in einer Taxonomie. Der Vorteil dieser Methode besteht darin, dass alle Relationen sowie Begriffe berücksichtig sind und ein ständiger Abgleich zwischen den lokalen Ontologien im laufenden Prozess nicht notwendig ist. Das Problem besteht jedoch in dem Prozess hin zu einer globalen Ontologie. Gut vorstellbar sind die Schwierigkeiten dieses Prozesses, wenn angenommen wird, dass sich eine definierte Anzahl von Experten auf eine limitierte Anzahl von Begriffen einigen muss. Kommen weitere Experten nach der Erstellung der globalen Ontologie hinzu, die auch ihre Begriffe sowie sämtliche Relationen ihrer lokalen Ontologie berücksichtigt wissen wollen, würde der Prozess noch komplexer. Führt man diese Überlegungen fort, so kann sich die globale Ontologie schnell zu einem unüberschaubaren Netzwerk entwickeln [Stu03].

Mapping: Das Ziel ist die Erstellung einer dauerhaften Beziehung zwischen zwei lokalen Ontologien. Hierzu benötigt man zusätzliches Wissens, so genannte „Mapping-Regeln“ [Stu03], [FNP+99]. Diese Regeln können dazu verwendet werden, Erfahrungen auszutauschen, indem logische Verbindungen zwischen den Begriffen der lokalen Ontologien explizit durch eine Mapping-Regel dargestellt werden [BS02]. Der Vorteil dieser Methode besteht darin, dass das Mappen nur zwischen zwei lokalen Ontologien (Ontologie A und B) stattfindet, andere Ontologien also nicht betrachtet werden. Somit können einmal erstellte „Mappings“ zwischen zwei lokalen Ontologien „für ewig“ halten, falls deren Begriffe sich nicht ändern sollten. Das Problem besteht allerdings darin, dass selten nur zwei lokale Ontologien in einem komplexen System auftreten. Kommen weitere Ontologien hinzu, sind Mapping-Regeln zwischen allen Ontologien aufzustellen. Es würden dadurch $O(n^2)$ Ontologie-Mappings notwendig [Stu03], [DF02].

Übersetzung: Das Ziel ist die Nutzung der Vorteile des Mergings und des Mappings. Die Idee besteht aus einer Zusammentragung von Begriffen aus lokalen Ontologien und die Vereinigung dieser in einer neuen globalen Basisterminologie (Mergen). Darauf basierend wird eine entsprechende Basisontologie entwickelt, deren Begriffe nicht mit denen der lokalen Ontologien übereinstimmen müssen. Entscheidend ist jedoch, dass die Begriffe der lokalen Ontologien durch die Begriffe der Basisontologie beschrieben werden können (Mapping). Der Vorteil dieses Ansatzes liegt in seiner Skalierbarkeit: Neue lokale Ontologien können leicht hinzugefügt werden, ohne die anderen zu beeinflussen, da die lokalen Ontologien (im Gegensatz zum Merging) nicht verändert werden müssen. Gerade im Hinblick auf Selbstoptimierung ist dies ein entscheidender Vorteil, da mit zunehmender Anzahl der Fahrzeuge und einem Austausch von Erfahrungen der Prozess hin zu einer effizienten erfahrungsbasierten Selbstoptimierung beschleunigt werden kann. Ein weiterer Vorteil dieser Methode liegt in der geringen Komplexität, da jeweils nur eine Verbindung von jeder lokalen Ontologie zur globalen Ontologie aufgebaut werden muss. Dadurch werden lediglich $O(n)$ Ontologie-Mappings notwendig [Stu03], [WVV+01]. Aus den genannten Gründen war diese Methode Grundlage der vorliegenden Arbeit.

3.2.1 Ansätze im Bereich des Mergens von Ontologien

Nach Sowa [Sow97] ist Merging der Prozess, Gemeinsamkeiten zwischen zwei verschiedenen Ontologien A und B zu finden und eine neue Ontologie C herzuleiten, die eine Interoperabilität zwischen Computersystemen, die auf den Ontologien A und B basieren, ermöglicht. Die neue Ontologie C ersetzt vollständig A und B.

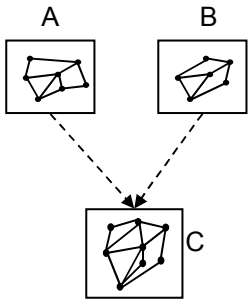


Abbildung 23: Ontology Merging [NM99]

Das Mergen von Ontologien kann notwendig sein, wenn zwei oder mehr Ontologien über denselben Anwendungsbereich existieren und eine allgemeinere benötigt wird, die die vorigen Ontologien ersetzen kann [Unt01-ol]. Die Herausforderung beim Mergen von Ontologien besteht darin, eine Abbildung zwischen ihnen zu entwerfen. [MBD+02].

Die Generierung des Mergens kann fast komplett automatisiert werden. Zum einen werden hierbei oft *Heuristiken*[17] zur Generierung von Abbildungen genutzt. Zum anderen können Abbildungen auch gelernt werden. Insbesondere manuell angebotene Merger-Programme präsentieren Beispiele für einen lernenden Algorithmus, der generalisieren kann und Sub-Mergers vorschlägt. Das anvisierte Ziel in der Wissenschaft ist es, ein System zu bauen, das eine Vielzahl von Techniken einsetzt, um den Anwendern bei der Konstruktion einer Konsens-Ontologie zu unterstützen. Das System wird verschiedene Heuristiken kombinieren, Lernmodule beinhalten und dem Nutzer bei Unsicherheiten Rückmeldungen geben [MBD+02].

Ein oft zitierter Ansatz in diesem Forschungsumfeld ist SIMS [AHK96]. Er beinhaltet eine hierarchische Wissensbasis mit Begriffen, die Objekte, Aktionen und Zustände repräsentieren. Zusätzlich werden diese durch Relationen verknüpft, wodurch ein globales Domänen-Modell entsteht. Jede weitere Informationsquelle muss sich an diesem Modell orientieren und kann durch weitere Relationen in dieses Modell integriert werden.

Das ISI OntoMorph System [Cha00] hat das Ziel, das Mergen und Entwickeln von Ontologien aus Wissensbasen zu vereinfachen. Es kombiniert zwei Mechanismen, um Wissensbasen in Ontologien zu transformieren. Beim ersten Mechanismus, dem syntaktischen

[17] Die Heuristiken basieren oftmals auf der Struktur (z.B. wenn zwei Begriffsformen übereinstimmen, dann passen möglicherweise auch ihre Teilbäume zueinander) oder auf der Namensgebung. Manchmal werden bereichsunabhängige Heuristiken durch spezifischere Heuristiken für bestimmte Anwendungsbereiche ersetzt.

Umschreiben der Wissensbasis, werden die Begriffe zunächst separiert und danach in einer syntaktischen Baumstruktur dargestellt. Kommen neue Begriffe hinzu, kann die Baumstruktur durch die Umschreibe-Regeln ständig erneuert werden. Der zweite Mechanismus beschreibt den semantischen Umschreibprozess. Mit den semantischen Regeln können z.B. Synonyme erkannt bzw. Begriffe aus der syntaktischen Baumstruktur ausgetauscht werden [DF02].

Das wohl wichtigste wissenschaftliche Merging-Tool ist PROMPT. PROMPT (ehemals SMART) [NM99], [NM00] ist ein in Protégé eingebetteter Algorithmus. Mit PROMPT können interaktiv Schritt für Schritt zwei Ontologien zu einer Ontologie zusammengefasst werden. Dabei macht der Algorithmus Vorschläge, erkennt Konflikte und schlägt Strategien zur Behebung der Konflikte vor. Er startet mit der Identifizierung von übereinstimmenden Begriffen, wobei im weiteren Verlauf die schon durchgeführten Schritte sowie Relationen zwischen den schon zusammengefassten Begriffen mit einbezogen werden. Ergebnis dieser Interaktion zwischen Programm und Anwender ist eine neue Ontologie.

Die zur Zeit aktuellste Anwendung im Bereich des Ontology-Mergens ist FCA-Merge [SM01], [Mae03], eine Methode der Ontologienverschmelzung. Sie basiert auf den Techniken der formalen Konzeptanalyse („Formal Concept Analysis“ [GW99]). Dabei wird ein Konzeptgitter anhand von extrahierten Begriffen zweier Dokumente erstellt. Der Mechanismus basiert auf anwendungsspezifischen Begriffen von zwei gegebenen Ontologien O_1 und O_2, die zusammengefügt werden sollen. Der Gesamtprozess der Verschmelzung zweier Ontologien ist in Abbildung 24 dargestellt und besteht aus drei Schritten:

- *Sprachliche Analyse* („Linguistic Processing“): Extraktion von Begriffen und Berechnung zweier formaler Kontexte K_1 und K_2.
- FCA-MERGE: Der Kernalgorithmus, der einen allgemeinen Kontext ableitet und ein Konzeptgitter berechnet („Lattice Exploration“).
- *Halbautomatische Generierung* der endgültigen verschmolzenen Ontologie O_{neu}, die auf dem Konzeptgitter basiert.

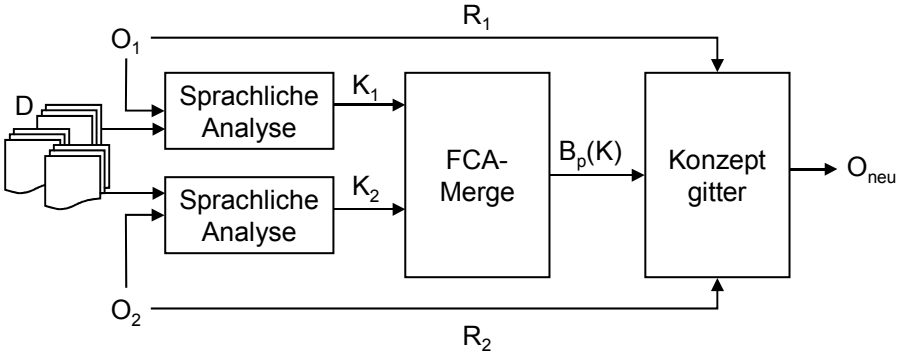


Abbildung 24: Die Methode FCA-Merge [Mae03]

Da es sich bei FCA-Merge und den anderen hier genannten Verfahren um einen Merging-Ansatz handelt, können diese Anwendungen aus den bereits oben erwähnten Gründen nicht für die vorliegende Arbeit verwendet werden.

3.2.2 Ansätze im Bereich des Mappens von Ontologien

Die Aufgabe, heterogene Wissensbasen zu nutzen, ohne ihre Strukturen zu ändern[18], ist wesentlich für komplexe, verteilte Systeme. In diesem Zusammenhang spricht die Literatur von dem Begriff des „Mappings“. Hinter diesem Begriff steht jedoch keine konkrete Methode, sondern lediglich das Phänomen, heterogene Ontologien so miteinander zu verbinden, dass ein eindeutiger Kommunikationsfluss über verschiedene Wissensbasen möglich wird.

Sowa [Sow97] definiert Mapping als eine Abbildung von Begriffen sowie Relationen zwischen zwei Ontologien A und B, für die die partielle Ordnung gilt. Das heißt, es kann viele Begriffe in A oder B geben, die keine Äquivalenzen in der anderen Ontologie haben. Wenn ein Begriff oder eine Relation x der Ontologie A auf einen Begriff oder eine Relation y der Ontologie B abgebildet werden kann, dann nennt man x und y äquivalent. Bevor jedoch zwei Ontologien A und B aufeinander abgebildet werden können, kann es nötig sein, neue Unter- oder Obertypen von Begriffen oder Relationen in entweder A oder B einzuführen, um passende Ziele zur Abbildung anzubieten.

Ein Mapping wird durch ein Regelnetz erzeugt, das Entsprechungen aus den Ontologien zuordnet. Somit werden zwei oder mehrere Ontologien beibehalten und Regeln erstellt, die die entsprechenden Begriffe zuordnen. Dies ist hilfreich, wenn die bestehenden Ontologien noch weiter verwendet werden sollen. Man nutzt diese Art von Integration meist dann, wenn die Ontologien Bereiche beschreiben, die zueinander komplementär sind. So könnte man Teile einer Ontologie über *Person* auf eine Ontologie *Patient* mappen (abbilden), da es nicht unbedingt passend wäre, eine einzige Ontologie zu erstellen, die beide Bereiche abdeckt [Unt01-ol].

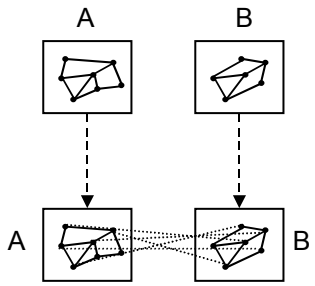


Abbildung 25: Ontology Mapping [NM99]

[18] Während des Mapping Prozesses werden keine Veränderungen an den Axiomen, Definitionen, Beweisen oder Berechnungen in den jeweiligen Ontologien vorgenommen.

Zur Ermittlung der Mapping-Regeln bzw. Beziehungen können sehr unterschiedliche Techniken genutzt werden. So ist es z.B. möglich von Hierarchien auszugehen, wobei Eigenschaften und Wertebereiche der Begriffe berücksichtigt werden können. Es gibt aber auch Ansätze, die ausgehend von einzelnen Begriffen Ähnlichkeiten ermitteln und dabei möglicherweise gemeinsame Begriffe, die in beiden Ontologien vorhanden sind, aufdecken [NM99].

Grundsätzlich lassen sich drei Arten des Mappings ausmachen [Mai03]:

Klasse-zu-Klasse[19] Mapping: Während des Mapping Prozesses werden die beiden Schemata, die „gemappt" werden sollen, in vertikalen Reihen parallel zueinander angezeigt. Wenn zwei Klassen zweier unterschiedlicher Quellen denselben Informationstyp enthalten, kann ein Klasse-zu-Klasse Mapping (z.B. *Automobil* und *Auto*, siehe Abbildung 26 gepunktete Linie) durchgeführt werden.

Slot-zu-Slot Mapping: Ein Slot-zu-Slot Mapping verbindet zwei Slots, vorausgesetzt sie besitzen dieselben Klassen (z.B. *hat_Top-Speed* und *hat_Höchstgeschwindigkeit*, siehe Abbildung 26, gestrichelte Linie.). Ein vorheriges Klasse-zu-Klasse Mapping ist allerdings Voraussetzung dafür.

Slot-zu-Klasse Mapping: Ein weiterer wichtiger Aspekt ist das Mappen von Slots auf Klassen. Das nachfolgende Slot-zu-Klasse Mapping zeigt, dass der Primärschlüssel *ID* von *Automobil* mit dem *Auto* verknüpft wird (siehe Abbildung 26, durchgezogene Linie).

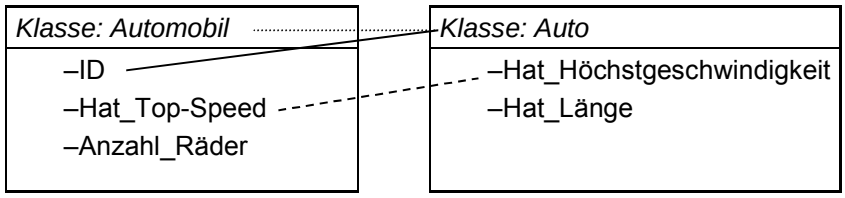


Abbildung 26: Die drei Mapping-Möglichkeiten

Allen bisher existierenden Werkzeugen zum Mapping ist gemein, dass sie einen Eingriff des Nutzers erfordern, der die vorgeschlagenen Beziehungen überprüft und die endgültigen Mapping-Regeln festlegen muss. Die Werkzeuge können lediglich Vorschläge machen, etwa unter Zuhilfenahme von „Natural Language Processing". Die endgültige Festlegung etwa von Äquivalenzbeziehungen muss aber ein Mensch vornehmen. Ansonsten geht die Präzision verloren, die Ontologien gerade auszeichnet [Sch04]. Die Zusammenführung unterschiedlicher Ontologien erfordert also erhebliche menschliche Eingriffe. Diese stellen zudem hohe Anforderungen an den ausführenden Bearbeiter, da er Verständnis für beide

[19] Eine Klasse wird durch einen Begriff bezeichnet (z.B. die Klasse Auto). Die Slots der Klasse dienen zur Bestimmung der Eigenschaften dieser Klasse.

lokale Ontologien und deren Nutzungsumgebungen haben muss, um ein korrektes Mapping durchzuführen.

Es gibt derzeit keine Programme, die diese Arbeit automatisch korrekt vollziehen. Auch in Zukunft ist es wohl unrealistisch, dass es Programme geben wird, die einen solchen Prozess komplett automatisch durchführen können. Es gibt allerdings Systeme, die diesen Prozess halbautomatisch mit Nutzerinteraktion abwickeln. Hierbei finden sich bereits verschiedenste Herangehensweisen in den unterschiedlichen Programmen wieder. Allerdings sind Fragen der Versionierung, also des Umgangs mit Veränderungen in den Ontologien, noch nicht geklärt. Ähnlich wie bei der Ontologieerstellung ist damit zu rechnen, dass die Integration unterschiedlicher Ontologien ein hohen Zeit- und Kostenaufwand verursacht [Sch04].

KRAFT [PHG+01] ist ein interdisziplinäres Forschungsprojekt der Universität von Aberdeen, Cardiff und Liverpool in Kooperation mit BT (British Telecommunications PLC). Es ist ein System, in dem das Mappen zwischen heterogenen Ontologien durch einen Mediator Agenten durchgeführt wird. Dieser Mediator Agent mappt die Ontologien in einem eins-zu-eins Ansatz. Hiermit können sowohl unterschiedliche Klassen als auch Slots in Beziehung zueinander gebracht werden. Dieser Ansatz erlaubt eine große Flexibilität, birgt jedoch auch die Gefahr, dass Inkonsistenzen auftreten können. Zudem legt sich KRAFT auf keine bis dato bestehende Methodik fest und versucht, einen eigenen Regelansatz zu entwickeln [WVV+01]. Neben dem Mappen erweiterten Visser und Tamma das System um die Möglichkeit des Clusterns von Ontologien [VT99], um heterogene Wissensbasen zu verbinden. Das Clustern von Ontologien basiert auf der Ähnlichkeit zwischen Klassen. Dabei ist ein neues Cluster eine Unter-Ontologie, die neue Klassen definiert und dabei die bereits existierenden Klassen der bestehenden Ontologie verwendet [DF02]. Einen ähnlichen Ansatz verfolgt OBSERVER [MIK+96] nur mit dem Unterscheid, dass sich dieses System auf ein definiertes Model der Aussagenlogik stützt.

GLUE [DMD+02], [MBD+02] ist ein System, das an der Universität Washington, USA, entwickelt wird. Es ist darauf ausgelegt, semantische Abbildungen zwischen gegebenen Ontologien zu finden. Dies schafft das System halbautomatisch durch eine Maschinenlerntechnik. Da Taxonomien die zentralen Komponenten in Ontologien sind, wird zunächst der Fokus darauf gerichtet, die korrespondierenden Taxonomien zweier gegebener Ontologien zu finden. Das heißt, für jeden Begriff der einen Taxonomie soll der ähnlichste aus der anderen Taxonomie gefunden werden. Die zentrale Frage hierbei lautet: Wie werden Ähnlichkeiten zwischen Begriffen definiert? GLUE berechnet die *gemeinsame Verteilung* zweier Begriffe und überlässt es dann dem Programm, diese zu nutzen, um ein brauchbares Maß an Ähnlichkeit zu berechnen. Für zwei Begriffe A und B wird $P(A,B)$, $P(\neg A,B)$, $P(A,\neg B)$, $P(\neg A,\neg B)$ berechnet. Der Ausdruck $P(A,\neg B)$ beispielsweise zeigt die Wahrscheinlichkeit an, dass der Begriff A nicht zum Begriff B gehört. Ein Programm kann die Ähnlichkeit als eine Funktion über diese vier Werte definieren. Ein Maß für die Ähnlichkeit ist zum Beispiel der Jaccard-Koeffizient $P(A \cap B)/P(A \cup B)$. Dieser nimmt seinen niedrigsten Wert 0 dann an, wenn A und B disjunkt sind und erreicht seinen höchsten Wert 1, wenn A und

B denselben Begriff darstellen. Um diesen Koeffizienten berechnen zu können, werden die gemeinsamen Wahrscheinlichkeiten benötigt:

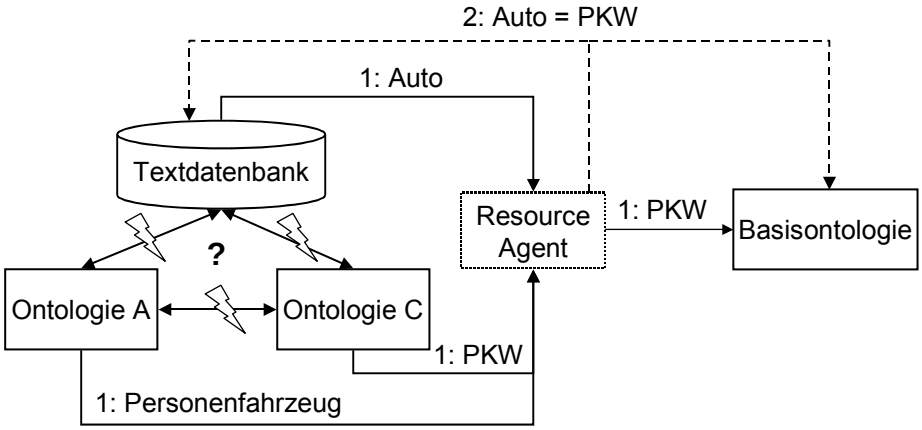
$$Jaccard(A,B) = P(A \cap B) / P(A \cup B) = \frac{P(A \cap B)}{P(A,B) + P(A,\neg B) + P(\neg A,B)}$$

Das Problem bei GLUE besteht darin, dass die Exaktheit des Verfahrens nicht zu 100% gewährleistet werden kann (die besten Ergebnisse zeigen allerdings eine Exaktheit des Mappings von bis zu 97% [DMD+02], was schon erstaunlich gut ist). Somit benötigt das System, wie die meisten anderen Verfahren, ein Feedback von einem Experten [MBD+02].

GLUE, KRAFT und Sowa bauen auf dem Mapping-Ansatz auf, der für ein komplexes Transportsystem mit Fahrzeugen, die heterogene Wissensbasen besitzen, allerdings nicht geeignet ist, da nicht nur zwischen den einzelnen Ontologien selbst, sondern auch zwischen deren Begriffen Mapping-Regeln definiert werden müssen. Dieser Prozess kann bei n beteiligten Ontologien sehr komplex werden. Zudem operieren die Algorithmen mit einer gewissen Unsicherheit. Mapping-Exaktheiten von 100% sind somit nicht zu erreichen, die jedoch für ein sicherheitskritisches System (das Vorlage dieser Arbeit ist) unerlässlich sind.

3.2.3 Übersetzungen mittels einer Basisontologie

Infosleuth ist ein System zur Integration von heterogenen Ontologien, entwickelt von der MCC (Microelectronics and Computer Technology Corporation, Austin, Texas, USA) und kommt dem in dieser Arbeit entwickelten Algorithmus am nächsten. Es unterstützt zunächst die automatische Entwicklung von komplexen Ontologien (Basisontologien) aus Textdatenbanken und kleineren Ontologien bzw. aus einzelnen Komponenten einer Ontologie [DF02], [Hwa99]. In dem in Abbildung 27 dargestellten Fall wird im ersten Schritt der Begriff PKW als Begriffsbedeutung für alle anderen Begriffsformen in die Basisontologie aufgenommen. Zwischen der Begriffsbedeutung innerhalb der Basisontologie und den Begriffsformen der individuellen Ontologien werden danach automatische Mappings erzeugt (siehe Schritt 2 in Abbildung 27). Der Term „Mapping“ ist hier definiert als die Entwicklung von Relationen zwischen Begriffsformen einer individuellen Ontologie mit den Begriffsbedeutungen einer globalen Basisontologie. Alle Mapping-Verfahren zwischen den Ontologien werden hierbei durch eine bestimmte Art von Agenten erledigt. Diese so genannten „Resource Agents“ kapseln alle Mapping-Regeln, die für das Mappen von einer Ontologie zu einer anderen Ontologie notwendig sind. Der Nachteil dieser Methode besteht darin, dass sie nicht zu 100% exakt funktioniert und mit einer Unsicherheit arbeitet, da sowohl die entwickelte Basisontologie als auch die Mappings Fehler aufweisen können [FNP+99]. Insofern benötigt Infosleuth, wie auch alle anderen hier vorgestellten Verfahren, Experten, die das Ergebnis bewerten. Allerdings können die erzeugten Mappings als gute Ausgangsposition für weitere Verbesserungen gesehen werden.



1: Auswahl einer exemplarischen Begriffsform durch den Resource Agenten und Abspeicherung dieser als Begriffsbedeutung in die Basisontologie
2: Mappen der Begriffsbedeutung zu den individuellen Begriffsformen

Abbildung 27: Arbeitsweise von Infosleuth

Die vorliegende Arbeit baut auf den Ansätzen von Infosleuth auf. Allerdings wird in dieser Arbeit die Basisontologie nicht automatisch, sondern manuell erzeugt. Beide Ansätze (Infosleuth und die Ansätze dieser Arbeit) generieren jedoch automatische Mappings zwischen der Basisontologie und den individuellen Ontologien. Bei Infosleuth werden aber nur textuelle Mappings erzeugt; also Ontologie A: PKW = Ontologie C: Auto oder Ontologie A: Schimmel = Ontologie C: weißes Pferd. In dieser Arbeit werden darüber hinaus Mappings zwischen den Regeln der heterogenen Wissensbasen von zwei Fahrzeugen A und C erzeugt; also Wissensbasis A: Fahrzeug(bremsen) :- Radius(Kurve, R), R<125m, Geschwindigkeit(Fahrzeug, G), G>100km/h = Wissensbasis C: Fahrzeug(gleich bleiben) :- Radius(Kurve, R), 50m≤R<125m, Geschwindigkeit(Fahrzeug, G), G>100km/h. So können die Erfahrungen der Fahrzeuge miteinander verglichen werden, was bei Infosleuth nicht möglich ist. Dies hilft um z.B. das Verhalten eines anderen Fahrzeugs richtig antizipieren bzw. die Erfahrungen anderer Fahrzeuge nutzen zu können.

3.3 Ansätze zur Generierung einer erfahrungsbasierten Selbstoptimierung

3.3.1 Explorationsverfahren zur Generierung neuer Erfahrungen

Da diese Arbeit auf symbolischen Erfahrungsdaten basiert, geht sie auch nur auf die Methoden zur Generierung derartiger Erfahrungsdaten ein. Ansätze im Bereich der Neuronalen Netze [Cal03] bzw. des Reinforcement Learnings [SB98] für sub-symbolische Erfahrungsdaten werden demnach nicht betrachtet.

3.3.1.1 Exploration mittels Suchverfahren

Im Allgemeinen werden in der Literatur zwei Arten von Suchverfahren unterschieden: die *uninformierten* und die *informierten Suchverfahren* [Bec03]. In den folgenden Kapiteln werden die wichtigsten Vertreter beider Verfahren vorgestellt.

3.3.1.1.1 Uninformierte Suchverfahren

Realwelt-Probleme werden oft mit Hilfe von Graphen, Suchbäumen oder auch Netzwerken gelöst. Die *uninformierte Suche* generiert dabei systematisch die Knoten des Zustandsraums und überprüft diese auf Zielknoteneigenschaften. Sobald ein Knoten dem Zielzustand entspricht, ist eine Lösung gefunden.

Die uninformierte Suche wird auch „Blind Search" genannt. Sie besitzt keine Informationen über die eigentliche Problemdefinition hinaus, z.B. keine Informationen über die Zustandsübergänge, wie zum Beispiel Pfadkosten oder aber keine Informationen über vorgegebene Restriktionen, wie zum Beispiel bei der Tourenplanung häufig auftretende Kapazitätsrestriktionen. Im Allgemeinen ist bei der uninformierten Suche die optimale Lösung jene, welche auf der höchsten Ebene des Suchbaumes liegt, d.h. mit der kleinsten Anzahl von Schritten erreicht wird [Bec03].

Breitensuche

Bei diesem Suchverfahren wird der Suchbaum von der Wurzel aus „Schicht für Schicht von oben nach unten" [GRS03, S. 133] durchforstet. Zuerst wird der Startknoten expandiert, also die Wurzel des Baumes. Jeder seiner direkten Nachfolgerknoten wird daraufhin überprüft, ob er ein Zielknoten ist und expandiert. Erst nach Abarbeitung aller Knoten innerhalb der Tiefe d=1 werden die Nachfolgerknoten der Tiefe d=2 überprüft und expandiert. Das heißt, dass Knoten der Tiefe d erst expandiert werden dürfen, wenn alle Knoten der Tiefe d-1 expandiert wurden. Dieses Vorgehen endet, sobald ein Zielknoten und somit eine Lösung gefunden wird.

Die Breitensuche ist vollständig. Insofern also eine Lösung in dem Suchbaum existiert, wird die Breitensuche auch eine Lösung finden [RN03]. Damit ist jedoch nicht gesichert, dass der gefundene Zielknoten eine optimale Lösung darstellt. Es wird lediglich ein Knoten gefunden, der die Zieleigenschaften besitzt und in der geringsten Tiefe liegt. Somit liefert die Breitensuche im Vergleich zur Tiefensuche immer die kürzere Lösung [RN03].

Die Anwendungsgebiete der Breitensuche sind vielfältig. So wird sie z.B. bei dem Filesharing-Programm Gnutella zum Lokalisieren von Files verwendet.[20] Eine andere Anwendung findet bei einem „Superoptimierer" zum Optimieren einer

[20] http://ipsi.fraunhofer.de/~risse/seminar2002/Effiziente_Suche_in_Peer-to-Peer-Netzwerken.html

Assemblercodefrequenz[21] statt. Ebenso wird die Breitensuche bei der „roboterbasierten“ Suche verwendet.[22]

Tiefensuche

Wie der Name schon impliziert, durchforstet dieses Suchverfahren den Suchbaum primär in die Tiefe. Angenommen, auf einer Ebene d wurden die Knoten n_1, n_2 und n_3 generiert, dann wird der Knoten n_1 daraufhin überprüft, ob er ein Zielknoten ist. Ist dies nicht der Fall, so wird der genannte Knoten expandiert. Im Gegensatz zur Breitensuche begibt sich die Tiefensuche nun schon auf die Ebene d+1 und überprüft einen der Nachfolger von n_1. Dieser Nachfolger wird dann expandiert, falls er kein Zielknoten ist. Die Tiefensuche verfolgt diesen Pfad so lange, bis er in einer Sackgasse endet, d.h. der zuletzt generierte Knoten hat keine Nachfolger und ist auch kein Zielknoten. Währenddessen verbleiben alle generierten, noch nicht expandierten Knoten der bereits besuchten Ebenen in der Liste der offenen Knoten. Gelangt die Suche in eine Sackgasse, so fährt das Suchverfahren mit dem nächstgelegenen Knoten, für den noch nicht alle Nachfolger expandiert wurden, weiter fort. Dies entspricht einem Backtracking. Nach [GRS03] legt die Vorgehensweise der Tiefensuche – wie bei der Breitensuche – jedoch noch nicht genau fest, in welcher Reihenfolge die Nachfolger eines expandierten Knotens eingefügt werden.

Zur Bewertung der Tiefensuche müssen verschiedene Situationen betrachtet werden. Im Falle von endlichen Suchbäumen ist die Tiefensuche vollständig. Verfolgt die Tiefensuche jedoch einen unendlichen Pfad, welcher keinen Zielknoten enthält, so wird sie nicht terminieren und kann somit keine Lösung liefern. Hier ist die Tiefensuche nicht vollständig. Der ungünstigste Fall für die Tiefensuche wäre ein Suchbaum, in welchem die einzige Lösung am Ende des äußersten rechten Pfades liegt. Hierzu muss die Tiefensuche alle Knoten des Suchbaumes expandieren. Das Gleiche gilt für die Breitensuche [RN03].

Verfahren der uninformierten Suche werden in dieser Arbeit nicht eingesetzt, da der Aspekt der „blinden Exploration“ nicht untersucht wird. Dies hat zwei Gründe: Zum einen machen diese Verfahren aus betriebwirtschaftlicher Sicht in dem hier vorgestellten Szenario keinen Sinn, da durch eine ausschließlich blinde Exploration keine Selbstoptimierung zu erwarten ist, und zum anderen basiert gerade der Grundgedanke dieser Arbeit auf dem Nutzen der Erfahrungen. Deshalb werden in dieser Arbeit nur die informierten Suchverfahren (das iterative Verbesserungsverfahren „Hill Climbing“ für die MFM-Ebene und der A* Algorithmus für die AMS-Ebene) eingesetzt.

[21] http://www.info.uni-karlsruhe.de/lehre/2003SS/Seminar-SSA-Codeerzeugung/Studenten-Ausarbeitungen/Odendahl-Superoptimierer.pdf

[22] http://www.inf-wiss.uni-konstanz.de/suche/tutorial/such_tutorial_anfaenger.html#2.3

3.3.1.1.2 Informierte Suchverfahren

Die *informierten Suchverfahren* verwenden Informationen über die eigentliche Problemdefinition hinaus. Diese Zusatzinformationen können z.B. Kosten-, Entfernungs- oder Kapazitätsangaben sein, die vom spezifischen Suchproblem abhängig sind.

Nach [LC01] wird bei größeren Suchproblemen versucht, den Suchraum über Kriterien oder Heuristiken zu strukturieren oder einzugrenzen, um so zu einem Ergebnis bzw. einer Lösung zu kommen. Eine Strukturierung bedeutet, dass die Abarbeitung der Knoten (Kandidaten) auf der Grundlage dieser zusätzlichen Informationen erfolgt. Bevorzugt werden oftmals solche Knoten, „die nahe am Ziel sind“ [GRS03, S. 139]. Um die Nähe eines Knotens zum Ziel zu bestimmen, wird bei heuristischen Suchverfahren eine heuristische Funktion eingesetzt, welche die Entfernung dieses Knotens zum Ziel schätzt. Diese Funktion schätzt entweder die Kosten oder die Entfernung zum nächstgelegenen Ziel, auf deren Grundlage der als nächster zu expandierende Knoten ausgewählt wird. Eine heuristische Funktion sollte demgemäß zum einen die Nähe zum Ziel richtig wiedergeben, d.h. den Zielknoten mit 0 bewerten, und zum anderen gewährleisten, dass der Aufwand für die Schätzung dieser Nähe in einem guten Kosten-Nutzen-Verhältnis steht, d.h. dass der Schätzaufwand sich für die Ausführung des Algorithmus lohnt [GRS03].

Greedy-Verfahren

Die Greedy Suche wird in der Literatur unterschiedlich bezeichnet, u.a. Best-First-Search, Bestensuche, gierige Suche [RN03], [GRS03]. Es gibt viele Algorithmen, die die Greedy-Methode verwenden oder auf dieser Methode aufbauen. Die Verfahren von Kruskal und Prim [Lev03], der Algorithmus von Dijkstra [Bec03], [Suh02], [Cor01] oder die Bergsteiger-Strategie (Hillclimbing) [BFM+04-ol] tragen die Züge eines Greedy-Verfahrens. Die grundlegende Eigenschaft dieser Verfahren ist es, dass immer die Entscheidung, welche im Moment am besten in Bezug auf die Lösung zu sein scheint, getroffen wird. Dabei berücksichtigt das Verfahren nicht, welche Kosten der beschrittene Weg bereits verursacht hat, sondern richtet sich immer nur nach vorne.

Die Greedy-Verfahren garantieren keine global optimale Lösung. Ein Beispiel dafür ist in Abbildung 28 dargestellt. Hierbei handelt es sich um ein Travelling Salesman Problem. Die Rundreise soll im Knoten A beginnen und auch dort wieder enden, nachdem jeder andere Knoten genau einmal besucht wurde. Hierbei wird durch das Greedy-Verfahren die Rundreise A-B-C-D-A ausgegeben mit einer Gesamtlänge von 2 + 3 + 23 + 5 = 33. Es existiert aber eine noch kürzere Rundreise, nämlich A-C-B-D-A mit einer Gesamtlänge von nur 4 + 3 + 7 + 5 = 19.

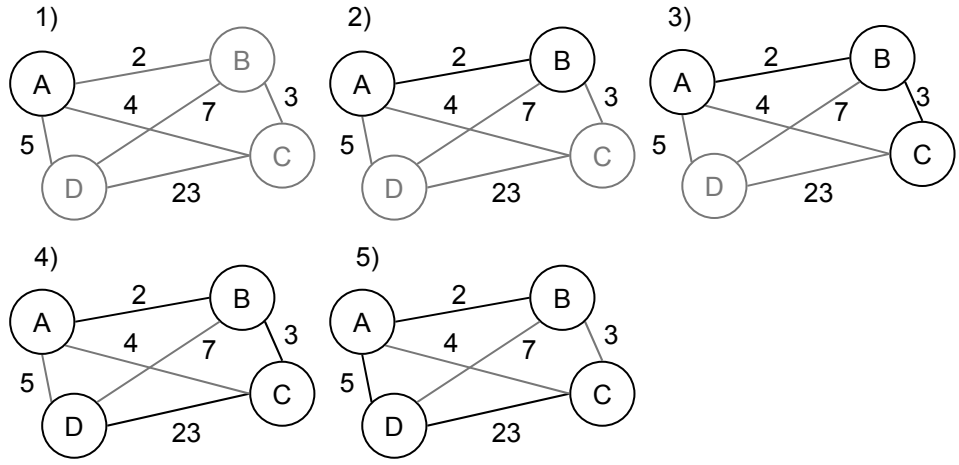


Abbildung 28: Gierige Suche beim TSP

Iterative Verbesserungsverfahren

Viele praktische Problemstellungen sind dadurch gekennzeichnet, dass eine optimale Lösung gesucht wird. Wenn aufgrund der Komplexität der Situation das Optimalitätsziel jedoch aufgegeben wird, bieten sich die iterativen Verbesserungsverfahren an. Bei der so genannten Gradientensuche (oder auch Hill Climbing) [BFM+04-ol] wird, ausgehend von einer vorläufigen Lösung, die Nachbarschaft nach einer Verbesserung durchsucht. Dies kann z.B. durch ein Vertauschen von zwei Komponenten einer Lösung oder einer Veränderung einer Variablen erreicht werden. Die neue Lösung wird bewertet und bei einer Verschlechterung wieder verworfen. Bei einer Verbesserung wird solange in der „positiven“ Richtung gesucht, bis keine Verbesserungen mehr erzeugt werden. Das Hill-Climbing Verfahren wurde in dieser Arbeit auf der MFM-Ebene zur Verbesserung einer Fahrzeug-Vorgabebahn zur Überquerung eines unbekannten Streckenabschnitts eingesetzt. Dieses Verfahren macht es für diese Anwendung so interessant, weil die Fahrzeuge aus betriebswirtschaftlichen Gründen schon nach wenigen Überfahrten über einen Streckenabschnitt eine Verbesserung in ihrem Verhalten erfahren sollen. Verfahren wie z.B. die Neuronalen Netze, Reinforcement Learning oder Neuro-Fuzzy-Systeme wären hier ungeeignet, da zur Trainierung ihrer Netze erst mehrere Tausend Überfahrten erforderlich wären.

Ein wesentlicher Nachteil des Hill-Climbing Verfahrens besteht darin, dass es in einem lokalen Optimum stecken bleiben kann. Daher wurden Erweiterungen entwickelt, die im Laufe der Suche auch schlechtere Lösungen akzeptieren. Eine solche Erweiterung, die auch in dieser Arbeit eingesetzt wurde, ist die Tabusuche (engl.: Tabu Search) [Mic00]. Bei ihm wird innerhalb einer definierten Nachbarschaft nach dem besten Nachbarn gesucht, der u. U. auch eine schlechtere Bewertung als die aktuelle Lösung hat. Um Zyklen zu vermeiden (also das mehrmalige Suchen in einer Nachbarschaft mit schlechten Bewertungen), wird eine Tabuliste eingeführt, die die bereits besuchten schlechten Nachbarschaften abspeichert. Die Tabuliste wird dabei als Ringpuffer mit beschränkter Länge realisiert, so dass die Suche nach n Schritten prinzipiell wieder zu einer alten, damals schlechten Lösung zurückfinden kann. Ein weiteres Verfahren ist das simulierte Ausglühen (engl.: Simulated Annealing) [RN03]. Es ist

ein probabilistisches Verfahren, bei dem mit abnehmender Wahrscheinlichkeit schlechtere Lösungen akzeptiert werden. Die Wahrscheinlichkeit für bessere Lösungen ist immer 1, für schlechtere ist sie abhängig von der Differenz zur vorherigen Bewertung und von der Dauer des Suchverfahrens. Zu Beginn des Verfahrens ist sie groß und verringert sich im Verlaufe der Suche. Ist v_i die Bewertung einer Lösung, dann akzeptiert der Algorithmus eine Lösung mit einer Bewertung v_j mit der Wahrscheinlichkeit

$$P = \frac{1}{1 + \exp\left(-\left(v_i - v_j\right)/t\right)}$$

wobei t der Kontrollparameter Temperatur ist, der während der Suche in mehreren Schritten heruntergesetzt wird. Die Entscheidung, wie schnell die Temperatur verringert wird, hat entscheidenden Einfluss auf die gefundene Lösung. Ein langsames Absenken erhöht die Wahrscheinlichkeit, dass eine bessere Lösung gefunden wird, bedeutet aber auch, dass die Suche länger dauert [Kis03].

A*-Algorithmus

Der A* Algorithmus betrachtet im Unterschied zum Greedy Best First Search zusätzlich zu einer Schätzfunktion *h(n)* die jeweils bereits zurückgelegte Strecke *g(n)*. Die sich daraus ergebende Funktion *f(n)* bildet die Summe der Kosten, die bereits entstanden sind, um zu einem Knoten n zu gelangen, und der geschätzten Kosten zum Zielknoten.

$$f(n) = g(n) + h(n)$$

Der Algorithmus wird den Knoten expandieren, der für den Funktionswert *f(n)* den besten Wert liefert. Des Weiteren speichert der A* Algorithmus alle Werte ab, die für *f(n)* berechnet worden sind. Diese Informationen werden mit abgespeichert, damit der A* Algorithmus immer den Knoten expandiert, welcher den besten Funktionswert *f(n)* hat. Aus dieser Eigenschaft heraus findet der A* Algorithmus immer das globale Optimum (vorausgesetzt, dass die Schätzfunktion zulässig ist, also nicht überschätzt). Des Weiteren kann gesagt werden, dass der Algorithmus vollständig und optimal ist [RN03]. Diese Gründe sprechen für seinen Einsatz auf der AMS-Ebene, auf der das Fahrzeug bestrebt ist, nach seinen derzeitigen Erfahrungen den optimalen Weg von A nach B zu befahren. Würde das Fahrzeug nicht den optimalen Weg befahren, entstünden zusätzliche, unnötige Kosten.

Bei all seinen Vorteilen besitzt der A* Algorithmus allerdings auch Nachteile. Bei der Suche nach dem globalen Optimum merkt sich der Algorithmus alle Funktionswerte der bereits expandierten Knoten, um diese miteinander vergleichen zu können. Diese Eigenschaft verursacht ein Speicherproblem. Zumeist ist der Speicher des A* Algorithmus bereits voll, bevor die Rechenzeit überschritten worden ist und ein Optimum gefunden wurde. Aus diesem Grund ist der A* Algorithmus nicht für sehr große Probleme mit vielen Knoten geeignet. Um den A* Algorithmus dennoch für größere Probleme benutzen zu können, gibt es Varianten des Algorithmus, die mit weniger Speicher auskommen [Bec03].

SMA* (Simplified Memory-bounded A*) ist eine solche Variante und funktioniert wie der A* Algorithmus. Die Ergebnisse nach jeder Expansion eines Knoten werden in einen Speicher geschrieben, bis dieser voll ist. Wenn dieser Punkt erreicht ist, bricht der Algorithmus nicht ab, sondern er wird den Knoten mit dem schlechtesten Funktionswert aus dem Speicher löschen und an seiner Stelle den neuen Funktionswert speichern. Damit die Informationen der Knoten, die gelöscht werden, nicht verloren gehen, werden die Funktionswerte in ihren Vorgängern gespeichert. Diese Teilspeicherung von Daten erspart unnötiges, erneutes Berechnen des Funktionswertes. Demzufolge wird auch der benötigte Speicherplatz reduziert [RN03]. In der vorliegenden Arbeit wurde allerdings ein anderes Verfahren entwickelt, um diese Speicherprobleme zu lösen. Bevor der A*Algorithmus operativ wird, werden zunächst alle Streckenabschnitte des Schienennetzes eliminiert, die nicht befahren werden können oder dürfen. So reduziert sich der Lösungsraum auf eine überschaubare Menge an Verbindungen.

In der Industrie wird der A* Algorithmus von einigen Entwicklern beim Programmieren von PC-Games verwendet[23]. Sein Einsatz wird u.a. bei der hierarchischen Bahnplanung für Industrie-Roboter untersucht[24] und in dem Projekt „Intelligent behavior control and world modeling for virtual humans in virtual environments“[25].

3.3.1.2 Exploration mittels quantitativer Methoden

In der Praxis kommt es in physikalischen Anwendungen häufig vor, dass nur wenige Wertepaare zur Verfügung stehen. Die Werte zwischen, vor und nach den vorgegebenen Punkten müssen dann mit Hilfe einer Funktion interpoliert bzw. extrapoliert werden. Zunächst werden hierfür die gewonnenen Daten in ein Koordinatensystem übertragen, um den Zusammenhang der einzelnen Punkte zu verdeutlichen. Durch Verbinden der Punkte mit Hilfe einer Linie (Regressionsgeraden) oder einer „glatten“ Kurve (Spline) lassen sich die zugehörigen Zwischenwerte ablesen.

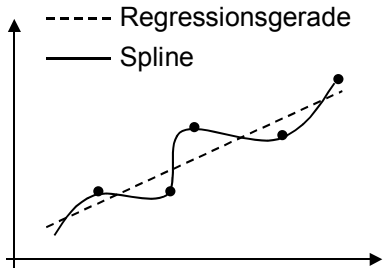


Abbildung 29: Unterschied zwischen Regressionsgerade und Spline

Die Regressionsgerade reicht jedoch im kontinuierlichen Bereich (z.B. Erwärmungsverlauf von Eisen, Bahnverlauf eines Fahrzeugs während der Fahrt) oft nicht aus, um praxisnahe Szenarien darzustellen. Die Spline-Interpolation stellt hierfür eine passendere Methode dar

[23] http://www-cs-students.stanford.edu/~amitp/gameprog.html
[24] http://wwwipr.ira.uka.de/~oster/index.html#download
[25] http://www.publica.fhg.de/documents/N-9293.html

[Vog04-o1]. Obwohl in der vorliegenden Arbeit nur die Splines berücksichtigt werden, geht sie, zur Verdeutlichung der Unterschiede, kurz auf beide Verfahren ein.

Regressionsgerade

Grundlage der quantitativen Prognosemethodik ist die Regressionsgerade. Sie bildet eines der am häufigsten angewandten statistischen Analyseverfahren [BEP+00]. Sie wird zur Analyse von Beziehungen zwischen einer abhängigen Variablen und mehrerer unabhängiger Variablen verwendet [Ste02]. Grundgedanke der Regression ist es, eine Stichprobe von Beobachtungen der Form:

$$(a_1,a_2,a_3..a_n,y)_1,\ (a_1,a_2,a_3..a_n,y)_2,\ldots\ (a_1,a_2,a_3..a_n,y)_m$$

die im euklidischen Raum dargestellt sind, durch eine Trendfunktion zu beschreiben. Bei der multiplen linearen Regression ist dies eine Linie der Form:

$$y = b_0 + b_1*a_1(x) + b_2*a_2(x) + ...b_i*a_i(x)$$

für den n-dimensionalen Raum. $a_i(x)$ bezeichnet hierbei den Wert des i-ten Attributes der Instanz x. Die multiple Regressionsrechnung hat die Aufgabe, den Zusammenhang zwischen mehr als zwei Variablen (Attributen) zu beschreiben und damit zu prognostischen Aussagen für eine als abhängig angesehene Variable y zu gelangen [Mit97]. Geometrisch gibt b_0 (Regressionskonstante) die Schnittebene der Hyperebene mit der Y-Achse an. Die übrigen Koeffizienten b_i (Regressionskoeffizient) geben (bei Beachtung der jeweiligen Vorzeichen) den (positiven oder negativen) Beitrag an, um den sich der Schätzwert für die deterministische Komponente y erhöht [Ste02].

Ein Beispiel [Hub00] soll die Explorationseigenschaften der Regressionsfunktion verdeutlichen. Es soll die elektromotorische Kraft (Spannung im Leerlauf) einer Batterie gemessen werden. Folgende Werte wurden gemessen [Hub00]:

Strom I/mA	1	1,5	2	2,5	3	3,5	4	4,5	5
Spannung U/V	4,73	4,77	4,78	4,81	4,82	4,85	4,86	4,89	4,92

Tabelle 8: Messung der elektromotorischen Kraft einer Batterie [Hub00]

Zur Messung fließen verschiedene Ströme durch die Batterie, so dass die Zelle nicht im Leerlauf ist. Deshalb werden mehrere Messungen mit immer kleineren Strömen durchgeführt, um daraus auf den Strom Null zu extrapolieren.

Es lässt sich eine Tendenz dahingehend erkennen, dass mit abnehmendem Strom auch die Spannung geringer wird. Es besteht also ein Rückgriff (= Regress) vom Strom auf die Spannung. Die graphische Darstellung der Punkte zeigt jedoch, dass die Punkte nicht auf einer Geraden liegen.

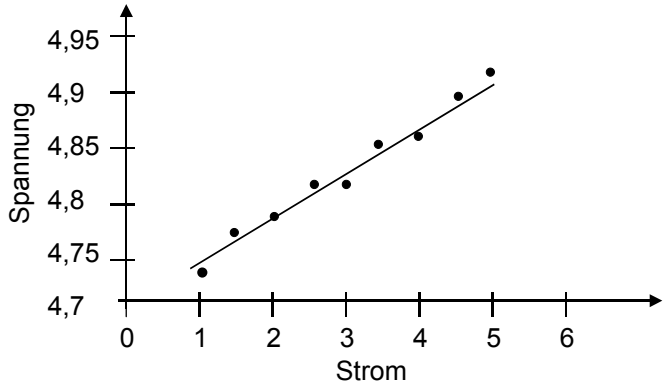


Abbildung 30: Messpunkte der elektromotorischen Kraft einer Batterie [Hub00]

Dies hat den Grund, dass nicht erfasste Einflussgrößen der empirischen Werte sich in Abweichungen von der Regressionsgeraden niederschlagen. Diese Abweichungen werden Residuen e_k genannt und bestimmen sich aus der Differenz des empirischen Wertes y_k zum berechneten Wert $\hat{y}_k$ [BEP+00].

$$y_k - \hat{y}_k = e_k$$

Die Regressionsgleichung muss nun so bestimmt werden, dass die Summe der quadratischen Residuen minimiert wird. Aus diesem Grunde verwendet die Regressionsanalyse bei der Auswahl der besten Geraden die Methode der kleinsten Quadrate (KQ-Methode)[26] [BEP+00].

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \min$$

Dieses Prinzip besagt, dass die Gerade so zu legen ist, dass die Summe der quadratischen Abweichungen der vorhergesagten $\hat{y} - Werte$ von beobachteten *y-Werten* möglichst klein wird [Kah01]. Unter dem Abstand eines Punktes von einer Geraden wird dabei nicht die Länge des Lotes von dem Punkt auf die Gerade, also der senkrechte Abstand, verstanden, sondern die Abweichung der Punkte von der Geraden in y-Richtung [BGG04].

Die notwendigen berechneten Werte werden nach der Gleichung:

$$\hat{y}_i = a + b * x_i$$

mit

[26] Durch die Quadrierung der Abweichungen werden größere Abweichungen stärker bewertet und verhindert, dass sich positive und negative Abweichungen gegeneinander aufheben.

$$b = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)} \text{ und } a = \bar{y} - b\bar{x} \ (\bar{y}, \bar{x} = Mittelwerte)$$

ermittelt.

Für das oben aufgeführte Beispiel ergibt sich:

$a = 4{,}6935; b = 0{,}0439$ und damit $\hat{y} = 4{,}6935 + 0{,}0439 * x$

Mit dieser Regressionsfunktion lässt sich nun jede beliebige Spannung in Abhängigkeit vom Strom extrapolieren.

Spline-Interpolation

Eine „glatte“ Kurve bezeichnet man in der Mathematik als Spline. Der Begriff Spline kommt ursprünglich aus dem Schiffsbau und bedeutet aus dem Englischen übersetzt „Latte“. Betrachtet man einen Schiffsrumpf, so kann man eine gegebene Menge an festgelegten Punkten (so genannten Stützpunkten) durch eine gebogene, elastische Holzlatte miteinander verbinden und erhält so eine glatte Formung der Rumpfkontur [Böh74], [Sch04-o1].

Bei der Spline-Interpolation wird in kubische- und B-(Basis) Splines unterschieden. Die kubische Spline-Interpolation ist eine Methode, um eine „glatte Kurve“ durch eine gegebene Menge von Stützpunkten zu konstruieren. Die kubische Spline-Interpolation setzt sich aus mehreren Teilbereichen mit Funktionen dritten Grades der Form $f_i(y) = a_i x^3 + b_i x^2 + c_i x + d$ zusammen. Das bedeutet, dass bei n Teilbereichen n+1 Stützpunkte vorhanden sein müssen. Zum Erreichen einer „glatten Kurve“ muss die Krümmung an den End-Stützpunkten jedes Teilbereichs bestimmt werden. Die Krümmung errechnet sich aus der zweiten Ableitung und macht es möglich, die einzelnen Teilbereiche stetig miteinander zu verknüpfen [Kno00].

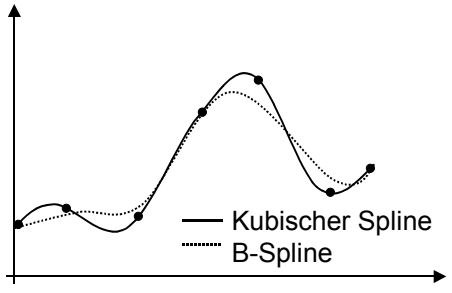


Abbildung 31: Unterschied zwischen kubischem- und B-Spline

B-Splines unterscheiden sich von den kubischen Splines darin, dass ihr Kurvenverlauf nicht durch alle gegebenen Stützpunkte verläuft. Im Allgemeinen geschieht dies bei mehr als drei

Stützpunkten. Ein Vorteil von B-Splines liegt darin, dass sich Änderungen von Werten nur lokal und nicht auf alle Intervalle auswirken [Tuw05-ol].

Ein Beispiel für das Anwendungsgebiet einer Spline-Interpolation ist die Entladekurve eines Kondensators. Die erste Messung erfolgt zum Zeitpunkt t = 0. Alle anderen Messungen erfolgen in gleichen Zeitintervallen. Als Ergebnis werden mehrere punktuelle Messungen vorgenommen, wobei die Stützstellen den Zeitpunkt der Messung und die zugehörigen Stromwerte die Stützwerte darstellen. Durch eine kubische Spline-Interpolation wird die zugehörige Funktion ermittelt, welche die Entladung des Kondensators beschreibt [Vog04-ol]. Mit dieser Funktion kann nunmehr extrapoliert werden, zu welchem Zeitpunkt der Ladezustand des Kondensators Null ist.

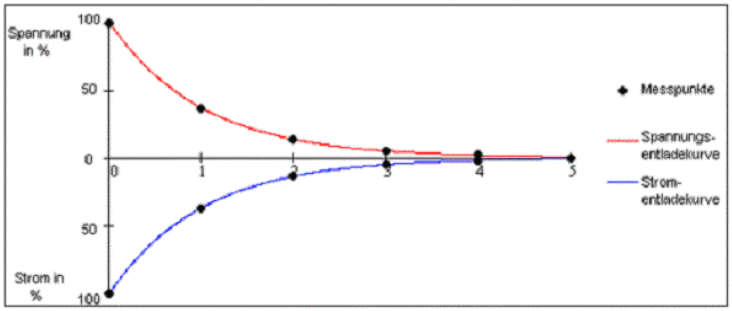


Abbildung 32: Entladekurve eines Kondensators [Vog04-ol]

Die Geowissenschaft verwendet Interpolationsmethoden z.B. zur Bestimmung der Erzstufen in Erzminen. Hierfür müssen die Daten so zusammengestellt werden, dass sie eine kontinuierliche Funktion ergeben, um die topographische Oberfläche einer gewünschten Region realistisch darzustellen. Eines der wichtigen Einsatzgebiete der Interpolation ist hierbei das topographische Darstellen von Höhenunterschieden. Hierfür wird mit Hilfe von Höhenlinien (Isohypsen) eine dreidimensionale Darstellung auf einer zweidimensionalen topographischen Landkarte errichtet. Da immer nur eine geringe Anzahl an Messpunkten zur Verfügung steht, werden die übrigen Messpunkte mit Hilfe der Spline-Interpolation interpoliert und zu kontinuierlichen Isohypsen miteinander verbunden [Vog04-ol]. Ist diese Karte erstellt und wurden Erzstufen gefunden, kann sie zum Explorieren weiterer, potentieller Erzgebiete genutzt werden.

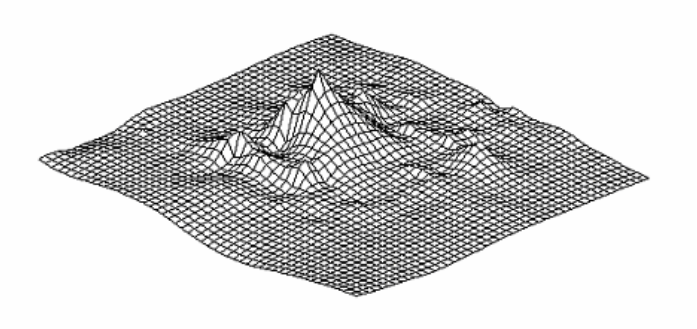


Abbildung 33: Interpolierte Oberfläche [Vog04-ol]

Die Automobilindustrie stellt ein weiteres, wichtiges Anwendungsgebiet für die Spline-Interpolation dar. Im Speziellen kommen Splines bei der Konstruktion von Fahrzeugteilen, wie z.B. einem Auspuff, Bremstrommeln und Radlagern zum Einsatz. Die Konstruktion dieser Teile wird heutzutage mit Computerprogrammen digitalisiert und am Bildschirm drei dimensional dargestellt und nicht an einem Modell plastisch abgebildet [Sch04-ol]. Zudem werden Splines in der Finanzwelt eingesetzt, um die Volatilität eines Aktienkurses zu schätzen und dessen zukünftigen Verlauf zu prognostizieren [Ber04].

3.3.1.3 Exploration mittels genetischer Algorithmen

Die genetischen Algorithmen basieren auf dem natürlichen Vorbild der Evolution und sind auf die Grundlagen von Charles Darwin (1809-1882), dem Begründer der Evolutionstheorie, mit seinem Buch „On the Origin of Species by Means of Natural Selection (1859)“, zurückzuführen. Darwins Theorie der Evolution durch natürliche Selektion besagt im Wesentlichen, dass die Individuen einer Population alle verschieden voneinander sind bzw. unterschiedlichen stochastischen Abweichungen unterliegen, die sich durch unterschiedliche Fitness äußert. Von diesen sind bestimmte Individuen an die herrschenden Umweltbedingungen besser angepasst als andere und haben damit größere Überlebens- und Fortpflanzungswahrscheinlichkeiten. Die genetische Beschaffenheit dieser besser angepassten Individuen wird durch Vererbung an folgende Generationen weitergegeben. Es findet ein Wechselspiel von Mutation (stochastische Abweichung) und Selektion (natürliche Auslese: „Überleben des Stärksten“) statt. Dieser schrittweise und kontinuierliche Prozess bewirkt die Evolution der Arten [Nis97], [SHF96], [GKK04].

Die Darwin`sche Theorie wurde auf die genetischen Algorithmen übertragen (siehe Abbildung 34). Dabei wird versucht, aus einer großen Menge möglicher Lösungen, die durch Individuen in einer Population repräsentiert werden, eine bessere oder sogar die beste Lösung zu generieren. Die Lösungen genetischer Algorithmen werden zufällig bestimmt. Die Zufallsprozesse setzen sich dazu aus der häufig stochastisch gebildeten Ausgangspopulation und den Wahrscheinlichkeiten der einzelnen Operatoren (Selektion, Kreuzung, Mutation) zusammen. Somit ist das Ergebnis eines genetischen Algorithmus nicht vorhersehbar.

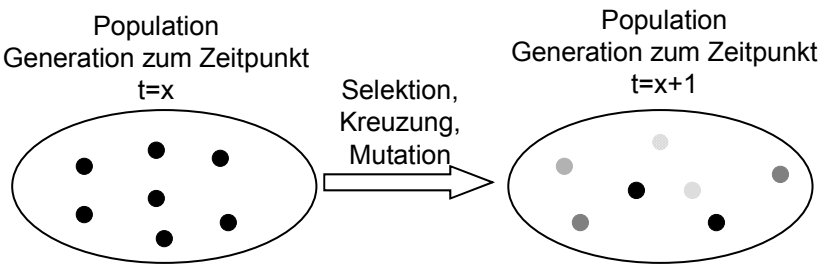


Abbildung 34: Schema eines Generationsprozesses

Die ersten Anwendungen genetischer Algorithmen wurden in intelligente, adaptive Spielprogramme umgesetzt. Die bekanntesten Beispiele sind das vereinfachte und das

minimierte Schachspiel von Bagley sowie das n-Damen Problem von Samuel [RN03]. Des Weiteren wurden von Cavicchio ein Mustererkennungsverfahren und von Rosenberg und Weinberg Verfahren zur Simulation lebender Zellen entwickelt [GKK04].

Aus technischer Sicht bietet sich der Einsatz von genetischen Algorithmen zur Programmierung der Abläufe von Roboterbewegungen an [Ort02-ol]. Dieses Gebiet hat in den letzten Jahren aufgrund steigender Zahlen von Robotern in der Industrie an Bedeutung gewonnen. Ziel ist es, die immer komplexer werdenden Abläufe der Roboterarme (Manipulatorarme) und die entsprechend aufwendige Programmierung mit dem Einsatz von genetischen Algorithmen zu vereinfachen. Eine weitere klassische Anwendung für genetische Algorithmen ist der Einsatz in der Produktionsplanung und -steuerung von Fertigungssystemen [Cla96] oder in der Fahrplanoptimierung der Deutschen Bahn [Gät04]. Bei dieser zentralen Optimierung werden alle möglichen Züge und Verbindungen mit ihren Randbedingungen zur Erstellung eines Gesamtplans betrachtet. In unserem Szenario verbessern sich jedoch die Fahrzeuge nicht nach einem zentralen Gesamtplan, sondern dezentral nach deren individuellen Zielen. Zusätzlich soll aus wirtschaftlichen Gründen nicht stochastisch, sondern statistisch exploriert werden. Diese Überlegungen lassen die genetischen Algorithmen für unser Szenario als ungünstig erscheinen, weshalb sie in dieser Arbeit nicht weiter untersucht wurden.

3.3.2 Ansätze zur erfahrungsbasierten Selbstoptimierung der Verhaltensweisen

Im Bereich der erfahrungsbasierten Selbstoptimierung von intelligenten Systemen wird in kognitive und reaktive Systeme unterschieden [Woo02]. Kognitive Systeme zeichnen sich durch ihre Repräsentationseigenschaften der Umwelt, ihre Kommunikationseigenschaften und ihre Informationsverarbeitung durch Lernfähigkeit und Antizipation aus [GRS03]. Bezogen auf logistische Problemstellungen bedeutet dies, dass kognitive Systeme versuchen, eine Landkarte (eine interne Repräsentation der Umgebung) zu erstellen, ihre Position darin zu bestimmen, Rückschlüsse auf ihr zukünftiges exploratives oder exploitatives Verhalten zu ziehen und dann mit anderen Systemen darüber zu kommunizieren. Reaktive Systeme haben diese Fähigkeiten nicht. Reaktive Systeme nehmen ihre Umwelt nicht wahr, solange nicht von außen auf sie eingewirkt wird. Kommt es zu einer Einwirkung, verhalten sich die Systeme nach fest vorgegebenen Regeln und handeln ohne explizite Denkprozesse. Trotz limitierter Fähigkeiten kann jedoch durch eine Vielzahl reaktiver Systeme das Gesamtsystem intelligente Verhaltensweisen aufweisen, ähnlich einer Ameisenkolonie, die vollkommen ohne jegliche Form von intelligentem Verhalten einer einzelnen Ameise in der Gesamtheit hoch komplexe Aufgaben wie Nestbau, Nahrungssuche etc. durch Kooperation löst [Sch03b-ol]. Diese Arbeit ist jedoch auf die kognitiven Systeme fokussiert, die sowohl deliberative Eigenschaften (Systeme besitzen ein Ziel, das durch geplante Aktionen im Umweltmodell erreicht werden kann) als auch reflektive Eigenschaften (Systeme verfügen über einen eigenen, internen Zustand, der es ermöglicht, Ziele neu zu bestimmen; dies ermöglicht eine Selbstoptimierung) besitzen [Ber03a-ol].

Die Ansätze kognitiver Systeme stützen sich auf die Methoden des maschinellen Lernens. Diese bezeichnen die Automatisierung eines Lernprozesses, der zum Ziel hat, auf einer Maschine (meist ein Computer) eine Aufgabe beim nächsten Mal genauer, dass heißt mit einer geringeren Fehlerwahrscheinlichkeit zu lösen als beim vorherigen Mal. Im Wesentlichen existieren zwei grundverschiedene Arten des Lernens [Mit97]; das induktive und das deduktive Lernen.

- *Induktives Lernen* (Lernen aus Beispielen, vom Einzelfall zum Allgemeinen): Es werden neue Zusammenhänge, die vorher nicht bekannt waren, entdeckt. Voraussetzung ist das Vorhandensein vieler Daten bei geringem Wissen über diese. Anwendungsgebiete für die Verfahren des induktiven Lernens sind z.B. die Analyse von Kundentypen, die Optimierung von Marketingaktionen oder die Modellierung von komplexen Modellen des Devisen- oder Aktienmarktes.
- *Deduktives Lernen* (erklärungsbasiertes Lernen, vom Allgemeinen zum Einzelfall): Das in Wissensbasen oder Datenpools gesammelte Wissen wird analysiert, um es später effizienter oder automatisch einsetzen zu können. Es basiert im Unterschied zum induktiven Lernen auf größeren Wissensbasen und wenigen Erfahrungen. Die Verfahren des deduktiven Lernens eignen sich besonders zur Automatisierung von technischen Regelungsprozessen unter Verwendung von bekannten expliziten Regeln bzw. Expertenwissen. Beispiele hierfür sind die Steuerung von Stahlöfen, Straßenbahnen oder Waschmaschinen [KWZ98].

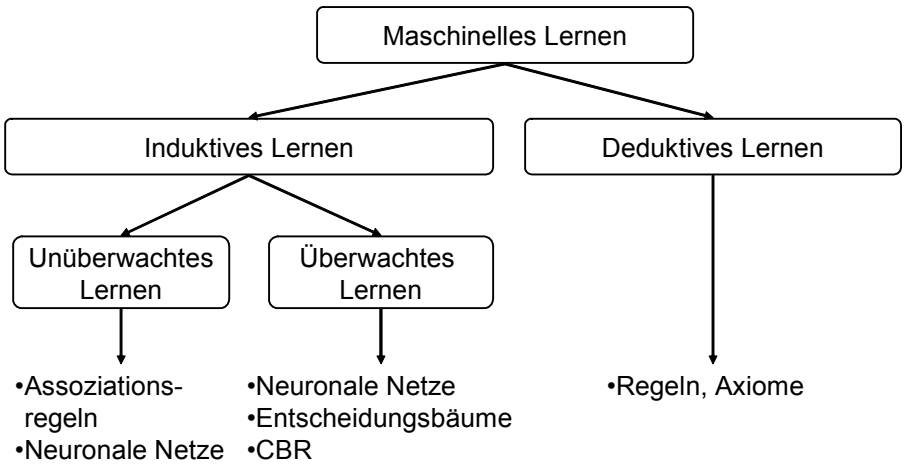


Abbildung 35: Maschinelles Lernen im Überblick

Diese Arbeit beschränkt sich auf die überwachten, induktiven Lernverfahren, wobei die Neuronalen Netze und Entscheidungsbäume ausgeklammert werden; erstere, da sie nicht zu den symbolischen, sondern zu den sub-symbolischen Lernmethoden zählen und letztere, da Entscheidungsbäume den Nachteil aufweisen, dass erst (wie auch im Bereich des Reinforcement Learnings) eine geeignete Anzahl an Erfahrungen vorhanden sein muss, bevor das System verlässlich lernt. Im Bereich des CBR kann durch Ähnlichkeitsschlüsse bereits mit der ersten Erfahrung gelernt werden (siehe Kapitel 3.3.2.2).

Im Bereich der erfahrungsbasierten Selbstoptimierung von Fahrzeugen wurden bereits umfangreiche Untersuchungen unternommen. TOUR [Kui78] wird hier als eine der ersten Anwendungen im Bereich der Logistik angesehen. Es lernt und löst eigenständig Probleme, während es sich durch eine große, städtische Umgebung bewegt. Sein Hauptfokus ist auf den Aufbau einer kognitiven Karte gerichtet, in der das Wissen in fünf Kategorien abgespeichert wird: (1) Routen, (2) topologisches Straßennetzwerk, (3) die relative Position von 2 Orten, (4) Grenzen und (5) Regionen. Das Wissen ergibt sich aus Umweltbeschreibungen, die derzeitige Position und Inferenzregeln. Es kann somit nicht vorausschauend agieren, sondern wird durch eine Sequenz von View-Action-Befehlen gesteuert.

Das Programm TRAVELLER [LZ89] findet in komplexen Netzwerken Wege zwischen vorgegebenen Anfangs- und Zielknoten. Das Modell basiert, im Prinzip wie TOUR, auf der Repräsentation des mentalen Modells der räumlichen Umwelt als Graphennetz und integriert neue Kenntnisse über die Umwelt. Der TRAVELLER findet jedoch nur topologische Relationen zwischen benachbarten Orten und ignoriert die Netzwerkstruktur und die Wegeffektivität, d.h. der TRAVELLER untersucht nicht, welcher Weg der kürzeste oder günstigste ist, sondern findet nur mögliche Pfade zwischen zwei Orten.

NAVIGATOR [GKS89] integriert Aspekte sowohl aus dem Bereich der Psychologie als auch der Künstlichen Intelligenz. Er konzentriert sich auf die Beschreibung der Relationen zwischen verschiedenen Objekten und ihrer Gewichtung durch ein Fahrzeug. Das räumliche Lernen ist als Prozess definiert, bei dem nach der Identifikation von Landmarken (Knoten) Kenntnisse über Routen zwischen den Landmarken gewonnen werden. NAVIGATOR besteht aus zwei Modulen: einer objektiven Umweltbeschreibung und einer individuellen, subjektiven Repräsentation der Umwelt. Die objektive Umwelt wird als Netz aus horizontalen und vertikalen Straßen dargestellt. Die Bewegungen durch das Straßennetz werden vom Nutzer definiert, und durch entsprechende Instruktionen lässt sich das Fahrzeug steuern. Während der Ortsbewegung nimmt das Fahrzeug Informationen über ein begrenztes Gebiet auf. Damit entwickelt es sein mentales Modell der Umwelt für die Beschreibung der Relationen zwischen den verschiedenen Objekten und ihren Gewichtungen. Das wiederholte Auftreten derselben Szene vergrößert die Gewichtung des Objekts. Das bedeutet, wenn ein Fahrzeug ein Objekt wiedererkennt, wird die Gewichtung verstärkt. Wenn die Gewichtung im Laufe der Zeit unter einen bestimmten Schwellenwert sinkt, wird die Information über dieses Objekt im individuellen Speicher gelöscht. Die Gewichtung ist somit abhängig von der Zeit.

NAPS-PC (Network Activity Processing Simulator – PC) [ONe91] besteht aus einem Neuronalen Netz von möglichen Haltestellen (Knoten) und einer textbasierten Liste topologischer Verbindungen zwischen diesen Haltestellen. Die Suche nach einer Route beginnt mit einer maximalen „Stimulierung“ des Anfangs- und Endknotens. Die Suche pflanzt sich von diesen beiden Knoten durch das Neuronale Netz fort, bis sich die Pfade beider Suchaktivitäten schneiden. Knoten, die innerhalb dieser Überschneidung liegen, werden als Unterziele aktiviert. Die Simulation ist allerdings nur für Situationen geeignet, bei der bereits ein umfassendes Wissen über die Umwelt existiert.

Raubals perceptual wayfinding model [Rau01] integriert das Fahrzeug und seine Umwelt innerhalb eines Sense-Plan-Act Framework. Es fokussiert auf Wissen über die Umwelt, um während einer Wegfindungsaufgabe Aktionen zu erklären. Das Modell kann für die Simulation von Wegfindungsverhalten von Menschen genutzt werden und unterstützt Forscher bei der Beantwortung der Frage, wo und warum Menschen Schwierigkeiten bei der Wegfindung haben und was getan werden muss, um dieses zu vermeiden. Das Model wurde am konkreten Beispiel eines Flughafens getestet.

Das Modell von Unger [Ung02] beschreibt das Verhalten eines beliebigen Verkehrsteilnehmers in einer vorgegebenen variablen Umgebung einer Stadt. Hierbei sind die Verkehrsteilnehmer in der Lage, ihre Umgebung wahrzunehmen, sie aktiv zu beeinflussen und sich der Umgebung anzupassen. Jeder Verkehrsteilnehmer sucht für die Erfüllung seiner Aufgabe günstige Wege zwischen verschiedenen Orten und passt dabei sein Verhalten und ggf. seine Aufgabenstellung den eventuellen Änderungen der Umweltbedingungen an. Als Modell für die Entscheidungsfindung eines autonomen Verkehrsteilnehmers dient ein Neuronales Netz mit dem Inputvektor bestehend aus (1) Richtung, (2) Weginformation, (3) Plan und (4) Befahrbarkeit.

Weitere bekannte Systeme sind SPAM [MD84], ELMER [MRS82] und ARIADNE [Eps97], die ähnliche, wie bereits oben beschriebene Ansätze für Lern- und Problemlösungsszenarien in räumlichen Netzwerken haben.

Im Allgemeinen fehlt allen bekannten Verfahren die Hinzunahme eines gezielten Erfahrungsaustausches zwischen mehreren Fahrzeugen, um eine unmittelbare Verbesserung der Entscheidung eines einzelnen Fahrzeugs zu erreichen. Erste Ansätze hierzu wurden im Bereich des verteilten Case-Based Reasonings unternommen, auf das in Kapitel 3.3.2.3 näher eingegangen wird.

3.3.2.1 Instanzbasiertes Lernen

Die Grundlagen des Case-Based Reasonings (fallbasiertes Schließen) bildet das instanzbasierte Lernen. Instanzbasierte Lernverfahren beruhen auf der Annahme, dass nicht nur ein Attribut, sondern eine komplette Erfahrung, die sich aus mehreren Attributen zusammensetzt, zur Lösung eines Problems herangezogen werden muss. Für numerische Attributwerte war das Verfahren in der Statistik schon länger unter dem Begriff des „Nächste Nachbarn“ –Verfahrens [CH67], [Mit97], [BK03] bzw. als lokale, gewichtete Regression bekannt. Das k-Nächste Nachbarn-Verfahren vergleicht die Lage eines Punktes im n-dimensionalen Raum $\Re^n$. Der Abstand dieser Punkte lässt sich mit Hilfe der euklidischen Distanz berechnen. Diese wird definiert durch [Mit97]:

$$\mathrm{dist}_{\mathrm{Euk}}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

beziehungsweise in normierter Form:

$$\text{dist}_{\text{Euk}}(x,y) = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Es ist darüber hinaus möglich, die jeweiligen Attribute zu gewichten und damit deren Wichtigkeit für die entsprechende Erfahrung zu unterstreichen. Hierzu wird der Wert $\alpha \geq 0$ verwendet, so dass gilt: $\Sigma\alpha=1$. Erweitert man die euklidischen Distanzfunktion um den genannten Parameter, so gelangt man zu der Funktion:

$$\text{dist}_{\text{Euk}}(x,y) = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\alpha_i(x_i - y_i)^2}$$

Für die Berechnung der Distanzwerte numerischer Attribute, für die gilt $x_i \in \Re$, müssen jedoch zuvor Wertebereiche festgelegt werden. Angenommen der Wertebereich der Geschwindigkeit eines Fahrzeugs beträgt [0,100], so besitzt dieses eine Minimalgeschwindigkeit v_{min} = 0 km/h und eine Maximalgeschwindigkeit v_{max} = 100 km/h. Diese Werte werden in einen binären Wertebereich umgewandelt, so dass gilt: $v_{min} \rightarrow 0$ und $v_{max} \rightarrow 1$. Da eine Erfahrung grundsätzlich aus mehreren Attributen besteht, definiert sich der binäre Wert der Attribute a_i einer Erfahrung durch eine Matrix W_{ai}: G_i^{min} x $G_i^{max} \rightarrow [0,1]$, wobei G_i^x den minimalen bzw. den maximalen Grenzwert und W_{ai} den daraus resultierenden Wert zwischen 0 und 1 darstellt. Der Distanzwert numerischer Attributwerte definiert sich somit entsprechend der bereits oben erwähnten euklidischen Distanzfunktion.

Als zweites Beispiel wird angenommen, dass es sich um eine Erfahrung mit zweiwertigen (boolschen) Attributen mit den Werten $x_i, y_i \in \{0,1\}$, also um Werte wie wahr, falsch; ja, nein; schnell, langsam etc. handelt. In diesem Fall ist die Distanzberechnung relativ einfach. Die Distanzfunktion zwischen diesen Erfahrungen wird auch als Hamming-Distanz bezeichnet und wird definiert durch [BK03]:

$$\text{dist}_{\text{H}}(x,y) = \sum_{i=1}^{n} | x_i - y_i |$$

beziehungsweise in normierter Form [Ber02]:

$$\text{dist}_{\text{H}}(x,y) = \frac{1}{n}\sum_{i=1}^{n} | x_i - y_i |$$

In Planungssystemen arbeiten Computerprogramme jedoch nicht ausschließlich mit numerischen oder Boolschen, sondern auch mit symbolischen Werten. Häufig müssen diese mit in die Distanzberechnung einbezogen werden. Leider ist ein Computer nicht selbständig in der Lage, eine Distanz zwischen einem BMW und einem Mercedes herzustellen. Nicht

einmal Menschen können hier objektive Klassifikationen vornehmen. Viele Menschen würden vermutlich behaupten, dass ein BMW und ein Mercedes eine geringe Distanz besitzen. Ein Ingenieur von BMW hingegen sähe wohl eine große Distanz zwischen diesen beiden Marken. Bei der Distanzbetrachtung von symbolischen Werten spielt also der subjektive Einfluss der Experten eine große Rolle. Die lokalen Distanzen von symbolischen Werten befinden sich dabei ebenfalls im Intervall dist(x,y) = [0,1].

Für die Betrachtung der lokalen Distanzen numerischer und symbolischer Attribute ist es unerlässlich, die Art der Dynamik zu spezifizieren. Es ist entscheidend zu wissen, wie sich die Distanzen zueinander verhalten. Steigen sie linear, degressiv oder progressiv? Gibt es einen Threshold, ab dem eine Distanz erst sinkt oder steigt? Grundsätzlich unterscheidet Bergmann [Ber02] in vier verschiedene Distanzverläufe:

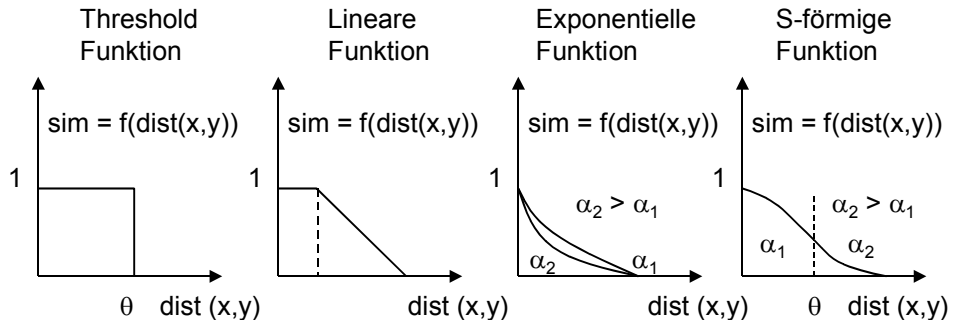


Abbildung 36: Basisverläufe für Ähnlichkeitsfunktionen [Ber02]

Bei der *Threshold Funktion* nimmt ab einer bestimmten Distanz (Threshold) zweier Erfahrungen der Distanzwert schlagartig ab. Bei der *linearen Funktion* wird bis zu einer minimalen Distanz ($dist_{min}$) der Distanzwert stets 1. Ab diesem Punkt nimmt der Wert linear mit der Erweiterung der Distanz ab, bis die maximale Distanz ($dist_{max}$) erreicht ist. Möglich sind auch *exponentielle* oder *s-förmige* (sigmoide) *Distanzfunktionen*. Ein exponentieller Verlauf bedeutet, dass schon ein geringer Unterschied in der Distanz eine große Auswirkung auf den Distanzwert hat. Entscheidend in diesem Zusammenhang ist der zu wählende Parameter α. Da der Distanzwert mit zunehmender Distanz abnimmt, liegt α im Intervall [0,-1]. Je größer sein Wert ist, desto stärker wirken sich Änderungen auf die Distanzwerte aus. Sigmoide Funktionen werden eingesetzt, wenn sich ab einem gewissen Threshold eine signifikante Änderung der Distanz ergibt. Wie bei den exponentiellen Funktionen kann auch hier der Kurvenverlauf mit dem α-Wert beeinflusst werden. Je kleiner dieser Wert ist, desto steiler ist der Abfall der Distanzwerte beim Threshold [Ber02].

In vielen Fällen wird nicht von Distanzen, sondern von Ähnlichkeiten gesprochen. Die Beziehung zwischen beiden Begriffen ist leicht hergestellt und lautet:

$$dist\ (x,y) := 1 - sim\ (x,y)$$

In der vorliegenden Arbeit wird jedoch der Begriff der „Distanz“ weiter beibehalten.

Instanzbasierte Lernverfahren gehören zu der Klasse der „lazy learners“, da sie immer erst dann aktiv werden, wenn eine neue Instanz (Erfahrung) klassifiziert werden soll. Ansonsten besteht die Arbeit des Instanzbasierten Lernens lediglich aus dem Abspeichern der Erfahrungen [GRS03]. Der Vorteil dieser Methode liegt darin, dass keine globale Zielfunktion für den gesamten Erfahrungsraum ständig neu berechnet werden muss, sondern nur für jede spezielle Erfahrung neu angestoßen wird.

3.3.2.2 Case-Based Reasoning

Für komplexe Erfahrungen mit *symbolischen und numerischen Werten* entstand in den letzen zehn Jahren das fallbasierte Schließen (Case-Based Reasoning) [Ber02], [CP02]. Im Wesentlichen baut dieses Verfahren auf denselben, oben erwähnten Prozessen auf. Es wird nach geeigneten nächsten Nachbarn einer neuen Erfahrung gesucht. Die nächsten Nachbarn sind historische Erfahrungen, die bereits einmal zur Anwendung gekommen sind. Auf diesen Erfahrungen aufbauend können neue, komplexe Erfahrungen generiert werden. Nach Generierung dieser Erfahrungen werden diese in der entsprechenden Klasse abgespeichert.

Die Ursprünge des fallbasierten Schließens liegen in drei unterschiedlichen Bereichen der Künstlichen Intelligenz [Ber02], [Ric03], [Kol93]:

- der kognitiv orientierten Forschung,
- dem analogen Schließen und
- dem Einsatz von wissensbasierten Methoden.

Das Ziel des fallbasierten Schließens ist es, das Gedächtnis von Experten oder ein reales Objekt in seiner Gesamtheit zu simulieren. Im täglichen Leben lösen die Menschen Probleme, indem sie auf gemachte Erfahrungen zurückgreifen. So verhält es sich zum Beispiel mit einem Arzt, der sich während einer Behandlung an einen früheren Patienten mit ähnlichen Symptomen erinnert und davon ausgehend eine entsprechende Therapie empfiehlt. Menschliche Experten lösen Probleme oft mit Hilfe von bekannten Fallbeispielen. Sie erinnern sich bei der Bewältigung des neuen Problems an eine vergleichbare frühere Situation und versuchen, diese Erfahrung gewinnbringend zur Lösung der neuen Aufgabe einzusetzen. Das fallbasierte Schließen wurde somit aus der kognitiven Sicht des Menschen dargestellt. Das bedeutet: Ähnliche Probleme haben ähnliche Lösungen [Wes96].

Aus dieser Aussage lässt sich der Kernpunkt des fallbasierten Schließens ableiten: nämlich dass das fallbasierte Schließen spezifische Erfahrungen über eine Domäne voraussetzt. In der Literatur [GRS03], [RN03], [Ber02], [CP02] werden diese Erfahrungen auch als Fälle

bezeichnet, wobei sich ein Fall aus der Beschreibung der Problemsituation und der Lösung dieser Situation zusammensetzt.

Vereinfacht ist eine Fallbasis eine Liste, ein Feld oder eine serielle Datei, die keine weitreichenden Strukturen aufweist. So wird bei der Abfrage jeder Fall durchlaufen und hinsichtlich der Ähnlichkeit überprüft. Eine solche flache Organisationsform hat den Vorteil, dass

- durch den Durchlauf der gesamten Fallbasis der oder die besten Fälle gefunden werden.
- durch die einfachen Strukturen neue Fälle einfach in die Fallbasis integriert werden können.

Der wesentliche Nachteil besteht darin, dass die Suche bei einer großen Fallbasis sehr zeitaufwendig und ineffektiv sein kann. Bei großen Fallbasen ist somit eine flache Struktur indiskutabel. Damit die Suche effektiver ist und die Vorteile der seriellen Fallbasis genutzt werden, bedient die Forschung sich der hierarchischen Strukturierung [Wes96]. Die hierarchische Strukturierung der Fallbasis hat allerdings die Nachteile, dass

- diese mehr Speicherkapazität benötigt als die flache Struktur und
- die Integration der neuen Fälle in die Fallbasis sorgfältig durchzuführen ist.

Der Vorteil des wesentlich effizienteren Verfahrens wiegt jedoch die Nachteile auf.

In der Literatur werden mehrere Modelle des fallbasierten Schließens mit verschiedenen Phasen vorgestellt. Im europäischen Raum wird hauptsächlich auf das Modell von Aamodt und Plaza [AP94] zurückgegriffen.

Nach diesem Model besteht der Prozess aus folgenden Phasen:

- Abrufen (Retrieve) des ähnlichsten Falles über ein Ähnlichkeitsmaß,
- Wiederverwendung (Reuse) der Informationen des genannten Falles für das aktuelle Problem durch Adaption dieses Falles,
- Durchführung (Revise) der vorgeschlagenen Lösung und
- Lernen (Retain) aus den erhaltenen Ergebnissen für die Zukunft.

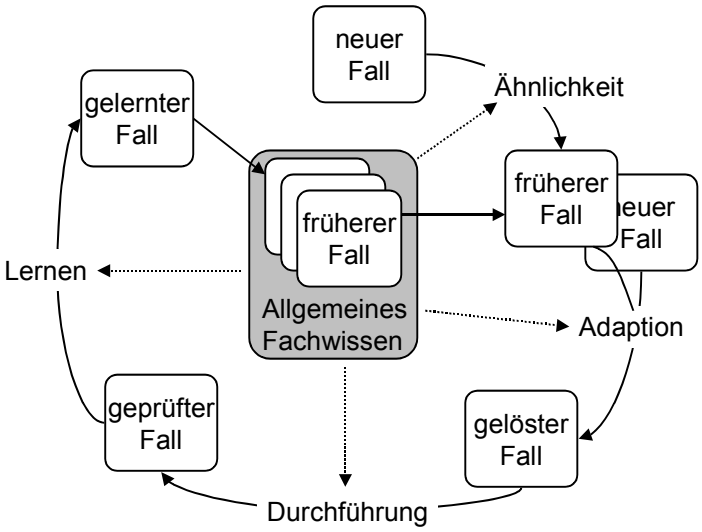


Abbildung 37: Zyklus des fallbasierten Schließens nach Aamodt und Plaza [AP94]

So ist das Lernen im Prozess des fallbasierten Schließens fester und wesentlicher Bestandteil. Neben der Beschreibung des Falles müssen dabei die Problembeschreibung, Lösung und gegebenenfalls auch der Prozess, der zu der Lösung oder Interpretation geführt hat, gespeichert werden.

Eine interessante Anwendung ist die von McGinty und Smyth [MS01a], [MS01b], [MS01c]. In ihr verwenden sie ein CBR-System zur verteilten Routenplanung. Erfahrungen über gute Routen können somit über mehrere Fahrzeuge ausgetauscht werden (siehe Kapitel 3.3.2.3). Im kommerziellen Bereich sind besonders die Help-Desk Anwendungen hervorzuheben. Hier gab es in den letzten Jahren eine starke Entwicklung. CBR-Systeme in Help-Desk Anwendungen können Anbietern von technischen Geräten durch die verbesserte Kundenunterstützung Marktvorteile verschaffen. Darüber hinaus können CBR-Systeme bei der Erfahrungsakquisition für die Diagnose helfen, um

- die Unterstützung nach dem Verkauf mittels Help-Desk Systemen zu verbessern (Telefon-Hotline),
- Diagnostik- und Fehleranalyse-Werkzeuge zu entwickeln und
- die Erfahrung von Spezialisten zu extrahieren und wiederzuverwerten.

Auch im Internetbereich werden diese Systeme als virtuelle Assistenten eingesetzt. Der Vorteil besteht darin, dass dem Kunden sogar bei unscharfen und unvollständigen Anfragen noch sinnvolle Ergebnisse und Alternativen genannt werden können [Web94]. Eine Übersicht weiterer interessanter Anwendung bietet Tabelle 9.

Anwendungsgebiet	System	Autor
Medizinische Diagnose	CASEY	Koton
Fehler-Diagnose	PATEX/2	Weiß
Verhandlung	MEDIATOR	Kolodner&Simpson
Verpflegung	Julia	Kolodner
Rechtssprechung	Judge	Bain
Rezepte	CHEF	Hammond

Tabelle 9: Beispielhafte CBR-Anwendungen [RS89]

3.3.2.3 Verteiltes Case-Based Reasoning

Der Begriff „verteiltes Case-Based Reasoning“ wird in zwei verschiedenen Ausprägungen verwendet. Zum einen kann hiermit ein Single-Agenten-Ansatz gemeint sein, in dem eine Fallbasis auf viele einzelne Fallbasen aufgeteilt ist. Lediglich ein Agent hat Zugang zu ihnen und selektiert je nach Anwendungsbereich die entsprechenden Basen [MS01a]. Erste Ansätze hierzu wurden von Branting & Aha [BA95] und Smyth & Cunningham [SC96] veröffentlicht. Sie beschreiben den Aufbau eines CBR-Systems als ein hierarchisches System, das Problemfälle auf unterschiedlichen Abstraktionsebenen speichert. Diese Arbeiten wurden von Bergmann [Ber96], [Ber02] erweitert. Der zweite Ansatz im Bereich des verteilten Case-Based Reasoning geht davon aus, dass Wissen nicht nur in einem Agenten, sondern verteilt über mehrere Agenten gespeichert ist. Ansätze zu dieser Methode wurden von Prasad, Lesser & Lander [PLL96], Plaza & Prasad [PP96] und McGinty & Smyth [MS01a], [MS01b], [MS01c] untersucht.

Prasad, Lesser & Lander [PLL96] entwickelten CBR-TEAM, das eine Anzahl heterogener, kooperativer Agenten zur Lösung einer Designaufgabe benutzt [MS01a]. Jeder Agent ist für eine bestimmte Komponente zum Design eines Dampfkondensators zuständig. Die Agenten sind der Motor-Agent, der Pumpen-Agent und der Keilriemen-Agent [PP96].

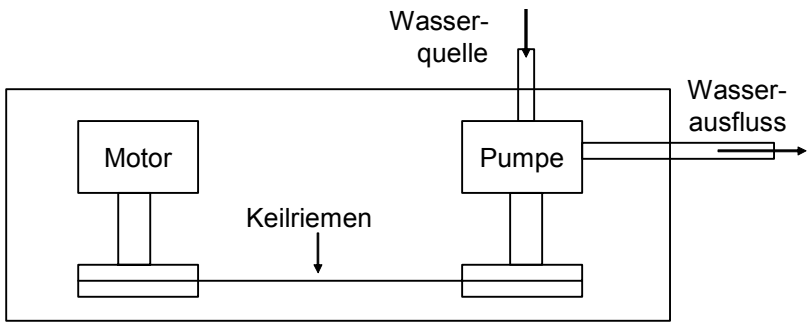


Abbildung 38: Dampfkondensator erstellt durch CBR-TEAM [PLL96]

Der Benutzer des Systems formuliert zunächst eine Problembeschreibung, die aus der minimalen Kopfgröße der Pumpe für das geforderte Design besteht. Die Agenten $A = \{A_{Motor}, A_{Pumpe}, A_{Keilriemen}\}$ rufen hiernach ein geeignetes Design aus einer Bibliothek mit herstellerspezifischen Modellen ab und gelangen durch Verhandlung untereinander zu einem allgemein

akzeptablen Design. Sollten innerhalb dieses Prozesses Bedingungen des Benutzers verletzt werden oder kommt es zu Unstimmigkeiten zwischen gemeinsam genutzten Parametern, wird durch den Austausch von Informationen das bis dato bestehende Design re-designed. Während der initialen Phase haben die Agenten nur grobe Informationen über die Voraussetzungen der anderen Komponenten. Deshalb wählt jeder Agent die zunächst kostengünstigsten Komponenten aus. Sollte dies zu einem Konflikt mit anderen Komponenten führen, verhandeln die beteiligten Agenten so lange, bis eine insgesamt kostengünstigste Variante gefunden ist. Alle Agenten tauschen hierzu ihre Constraints aus, die der jeweils andere Agent in seiner Wissensbasis abspeichert. Die Agenten durchlaufen so iterativ mehrere Runden, indem sie die bereits gespeicherten Informationen (Constraints) und neue Informationen von anderen Agenten miteinander vergleichen, um immer geeignetere Komponenten zu finden, die immer weniger (und am Ende gar keine) Konflikte hervorrufen [PLL96].

Plaza & Prasad [PP96] entwickelten hierzu den Ansatz des *Federated Peer Learnings (FPL)*, der sich in die Bereiche des Distributed-CBR und des Collective-CBR unterteilen lässt (siehe Abbildung 40). Im ersten Ansatz übermittelt ein Agent A_i sein Problem und die zu lösende Aufgabe an einen anderen Agenten A_j. Agent A_j löst die Aufgabe, indem er auf seine eigene Wissensbasis zurückgreift. Nach dem Lösen der Aufgabe sendet er die Ergebnisse an A_i zurück. Sollte die Aufgabe nicht zufrieden stellend gelöst worden sein oder ist A_j gar nicht in der Lage, die Aufgabe zu lösen, wird eine Fehlermeldung erzeugt und der nächste Agent gefragt. Dieser Vorgang dauert so lange, bis die Aufgabe zufriedenstellend gelöst wurde. Im zweiten Ansatz übermittelt der Agent A_i das Problem, die Aufgabe und zusätzlich die Methode, mit der die Aufgabe gelöst werden soll, zu einem Agenten A_j. Hierbei schließt sich der gleiche oben bereits erwähnte Prozess an, mit dem Unterschied, dass jetzt nur eine Methode zum Auffinden einer Lösung erlaubt ist (z.B. eine spezielle Suchstrategie – Breitensuche, Tiefensuche, Hill Climbing etc.).

McGinty & Smyth [MS01a], [MS01b], [MS01c] entwickelten den Ansatz des Collaborative-CBR (siehe Abbildung 39), eine Form des Federated Peer Learnings. Alle Agenten besitzen die gleichen Fähigkeiten mit unterschiedlichem Wissen. Sollte das Wissen des Agenten A_i nicht ausreichen, um eine Aufgabe *p* zu lösen, wendet er sich an andere Agenten wie A_j oder A_k. Hierbei werden nur die Agenten berücksichtigt, die A_i am ähnlichsten sind. A_i wird am Ende nur die Lösung *s* mit der höchsten Qualitätszahl berücksichtigen. Im Unterschied zu dem ursprünglichen FPL behalten die Agenten ihre Problemlösungsautorität. Es kann also einem Agenten nicht vorgeschrieben werden, mit welchen Methoden ein Problem zu lösen ist. Zudem stellt dieser Ansatz die Ähnlichkeit der Agenten in den Vordergrund. Im FPL-Ansatz werden die Agenten noch wahllos gefragt. Bei einer schlechten Antwort wird einfach ein nächster Agent gefragt.

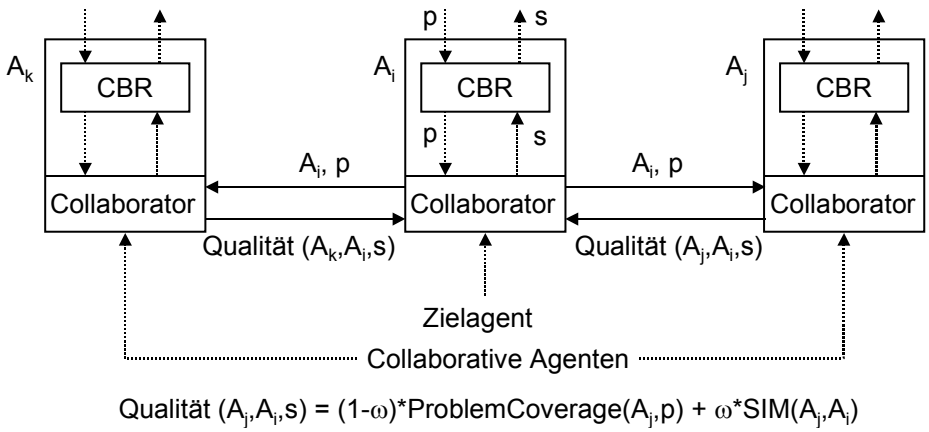


Abbildung 39: Top-level Collaborative-CBR Agenten-Algorithmus [MS01a]

Die vorliegende Arbeit verfolgt noch einen weitergehenden Ansatz, der Cooperative CBR genannt wurde [DS04], bei dem vom anfragenden Agenten noch ein Feedback an den antwortenden Agenten übermittelt wird. Dadurch kommt es zu einer zusätzlichen Beschleunigung des Lernprozesses (siehe Abbildung 40).

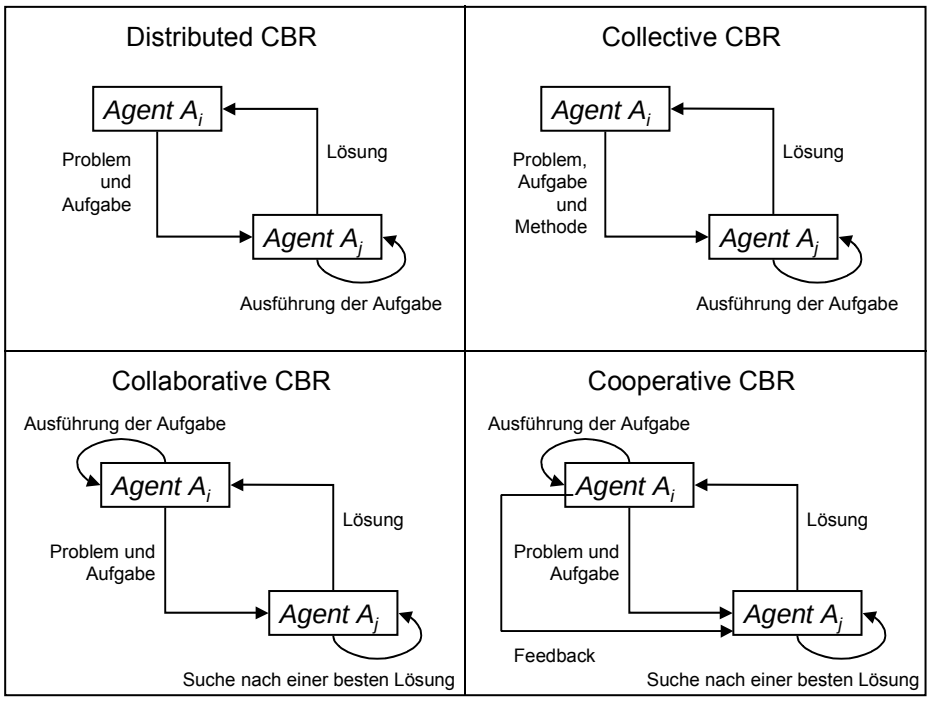


Abbildung 40: Ablauf verschiedener Formen des verteilten CBR

4 Zielsetzung

4.1 Entwicklung einer Basisontologie zur Wissensrepräsentation einer Domäne

Eine *Wissensrepräsentation* wird zunächst durch die vollständige Darstellung der *Domäne* durch eine Terminologie bestimmt. Eine Terminologie ist dabei eine systematische Sammlung und Beschreibung von Begriffen einer Domäne (Objekte, Attribute, Parameter etc.). Aus diesen Begriffen können Begriffssysteme entworfen werden, die die Verbindung der Begriffe untereinander verdeutlichen (vgl. [Sch99]). Das Begriffssystem (Ontologie) ist so aufzubauen, dass möglichst viele logische Verbindungen zwischen den Begriffen entstehen. In der vorliegenden Arbeit wären z.B. das Fahrzeug, die Schiene und der Benutzer drei verschiedene Begriffe, die in Verbindung zueinander stehen, da der Benutzer in einem Fahrzeug sitzt und das Fahrzeug auf einer Schiene fährt.

In dieser Arbeit soll die zu entwickelnde Basisontologie im Wesentlichen eine eindeutige Kommunikation zwischen den Fahrzeugen sicherstellen, in dem sie eine allgemeingültige Wissensrepräsentation der Domäne liefert, über die die Fahrzeuge ihre individuellen Erfahrungen austauschen können.

4.2 Übertragung von Erfahrungen aus heterogenen Wissensbasen

Tauschen Fahrzeuge mit unterschiedlichen Zielen, Verhaltensweisen und unterschiedlichem Domänenwissen Erfahrungen untereinander aus, so müssen Methoden entwickelt werden, die in der Lage sind, diese Heterogenität zu überwinden. Klassische Methoden beschränken sich hierbei auf die rein semantische Übersetzung von Erfahrungen. Der vorliegende Ansatz geht darüber hinaus und stellt sowohl eine Übersetzung der Semantik als auch eine Übersetzung der Verhaltensweisen und der Ziele, dargestellt durch Regeln, her.

Für die Untersuchung der Erfahrungsverbreitung über mehrere Fahrzeuge ist die Entwicklung eines Algorithmus notwendig, der zwei heterogene Wissensbasen miteinander vergleicht. Jede Wissensbasis besitzt eine andere Semantik, zwischen denen es zu Konflikt- und Korrespondenzproblemen kommen kann. Zudem bestehen die heterogenen Wissensbasen aus unterschiedlichen Verhaltensweisen und Zielen, die in Regeln dargestellt sind. Hierbei wird vorausgesetzt, dass die Prämissen der Regeln jeweils mit der Semantik der Basisontologie definiert werden.

4.3 Generierung einer erfahrungsbasierten Selbstoptimierung

Damit sich Fahrzeuge selbständig verbessern können, müssen neue Situationszustände exploriert werden. Die so erlangten Erfahrungen sind in die jeweilige Wissensbasis zu integrieren, so dass sie für exploitative Lernprozesse zur Verfügung stehen. Insgesamt wird ein Fahrzeug dadurch in die Lage versetzt, sich selbständig in dynamischen, unbekannten Umgebungen zu verbessern. Um diesen Anpassungsprozess zu beschleunigen, müssen

Fahrzeuge miteinander kooperieren bzw. kommunizieren und die so fremd erlangten Erfahrungen für die eigenen Lernprozesse nutzbar machen. Hierfür leisten Kapitel 5.1 und 5.2 die Grundlagen.

Um die Effizienz der Wissensbasis, die durch die Masse der gesammelten Erfahrungen schnell anwächst, zu gewährleisten, müssen Methoden entwickelt werden, die die Erfahrungen in einen Regelsatz oder eine Funktion transformieren. So kann ein schnelles Abfragen der Wissensbasis gewährleistet werden.

5 Konzeption zur erfahrungsbasierten Selbstoptimierung einer Menge technisch homogener fahrerloser Fahrzeuge

Innerhalb der vorliegenden Arbeit wird davon ausgegangen, dass ein Fahrzeug sein Verhalten sowohl durch eigene, explorativ erlangte Erfahrungen als auch kommunikativ durch die Erfahrungen anderer Fahrzeuge in Verbindung mit den in dieser Arbeit entwickelten Algorithmen kontinuierlich verbessern (also selbstoptimieren) kann (siehe Kapitel 5.3). In diesem Zusammenhang wird die Problematik der heterogenen Wissensbasen einzelner Fahrzeuge hinsichtlich der Bewertung von Erfahrungen untersucht (siehe Kapitel 5.2). Die Lösung dieser Problematik erfolgt über eine Basisontologie, die das Wissen über die Domäne umfassend repräsentiert (siehe Kapitel 5.1).

5.1 Wissensrepräsentation der Domäne

Die Basisontologie orientiert sich an der Struktur von SUMO [NP02], einer Basisontologie, die im Rahmen der IEEE Standard Upper Ontology Working Group entwickelt wurde (siehe Kapitel 3.1.3). Die Ontologie von SUMO wie auch die hier beschriebene Basisontologie bietet eine offene Struktur für eine technische Ontologie, die beliebig erweitert werden kann. Somit erhebt die hier entwickelte Basisontologie nicht den Anspruch, das weltweite technische Wissen darzustellen, sondern eine *Strukturierung von selbstoptimierenden Systemen* zu entwickeln.

Die Darstellungsform der Basisontologie erfolgt in dieser Arbeit im FRAME-Format. Frames clustern Instanzen um Klassen herum. Die Instanzen einer Klasse können mit den Instanzen anderer Klassen durch Relationen und Prozeduren (Regeln und Funktionen) verbunden werden. Dadurch ergibt sich eine vernetzte Begriffswelt, die als *objektorientierte Repräsentation semantischer Netze* gesehen werden kann. Frames bieten jedoch den entscheidenden Vorteil, dass sie „procedurale attachments“ erlauben. Hierdurch können Regeln und Funktionen innerhalb einer Klasse dargestellt und ausgeführt werden.

Es sei darauf hingewiesen, dass der Bereich der Basisontologie sehr komplex ist, da eine Ontologie aus einer Vielzahl von Relationen, Regeln, Funktionen und Klassen besteht. Zur Übersichtlichkeit werden diese in unterschiedlichen Kapiteln einzeln ausführlich behandelt. Beispielhaft werden die Relationen, Regeln, Funktionen und Klassen zur Demonstration eines Ausschnitts aus der Basisontologie zusammengefügt. Die vollständige Basisontologie ist zusätzlich im Anhang unter Kapitel A aufgeführt.

In Kapitel 5.1.1 werden zunächst die Slots (Relationen, Regeln, Funktionen) der Basisontologie genauer spezifiziert. Dabei ist es unerheblich, ob es sich um proprietäre oder geerbte Slots handelt. In Kapitel 5.1.2 werden die Slots in die Klassen der Basisontologie integriert.

5.1.1 Entwicklung der Slots

Slots besitzen stets einen Namen, einen Typ (Integer, Float, Instanz, Klasse etc.), eine Kardinalität (multiple oder single) und eine Ausprägung (entweder als Referenz zu einer anderen Klasse oder als Instanz oder als spezieller Wert; Näheres siehe Anhang C).

Beispielhaft sei hier die Klasse Fahrzeug dargestellt (siehe Tabelle 10). Sie besitzt (hier nur exemplarisch) vier Slots. Der erste Slot ist ein absoluter deskriptiver Slot, der besagt, dass hier nur eine (da die Kardinalität single vorliegt) textuelle Bezeichnung (Typ: String) für das Fahrzeug, z.B. Siemens 305, eingetragen werden kann. Der zweite, ein relationaler Slot, besagt, dass die Höhe eine Länge ist, die bereits an anderer Stelle innerhalb der Basisontologie durch eine eigene Klasse spezifiziert wurde. Die Höhe übernimmt somit ihre Eigenschaften. Der dritte Slot ist ebenfalls ein relationaler Slot, der besagt, dass sich ein Fahrzeug auf einer bestimmten Schiene befindet. Der vierte, auch ein relationaler Slot, besagt, dass mehrere Fahrgäste in einem Fahrzeug sitzen können. Dabei besitzt der Fahrgast die Eigenschaften eines Menschen, der bereits an anderer Stelle durch eine Klasse in der Basisontologie definiert wurde. Weitere relationale deskriptive Slots könnten die Abfahrtszeit und die Ankunftszeit sein. Beide beziehen sich auf die Klasse Zeit, die bereits an anderer Stelle durch eine Klasse innerhalb der Basisontologie definiert wurde.

Klasse: Fahrzeug				
Slot	**Name**	**Typ**	**Kardinalität**	**Ausprägung**
Slot 1	Bezeichnung	String	Single	
Slot 2	Höhe	Instanz	Single	Klasse = (Länge)
Slot 3	Fährt_auf	Instanz	Single	Klasse = (Schiene)
Slot 4	Transportgut	Instanz	Multiple	Klasse = (Fahrgast)

Tabelle 10: Ausschnitt aus der Klasse Fahrzeug

Zusätzlich zu den deskriptiven und relationalen Slots können prozedurale Slots definiert werden. Prozedurale Slots stellen Funktionen bzw. Regeln dar.

In den folgenden Kapiteln werden sämtliche Slotarten näher beschrieben.

5.1.1.1 Relationale Slots

Zur Bewertung verschiedener Instanzen einer Klasse (z.B. Fahrzeug A ist schneller als Fahrzeug B) müssen eindeutige temporäre, räumliche und quantitative relationale Slots definiert werden. Insbesondere die temporären und räumlichen Slots weisen eine interne Struktur auf, die einzelne Systeme und deren Verhaltensweisen in einem Raum in Beziehung zueinander setzen. Dabei ist die Zeit z.B. ein wesentlicher Bestandteil innerhalb der Anwendungen aus dem Bereich des „Job-Shop-Scheduling“ [RN03]. Diese benötigen eine komplette Darstellung der Systemverhaltensweisen, welche aus einer Sequenz von Aktionen besteht, in der jede Aktion eine bestimmte Dauer und Ressource benötigt. Das Problem besteht häufig darin, einen Plan zu entwickeln, der die minimal notwendige Zeit bei gegebenen Ressourcen berechnet, um ein komplettes Verhalten auszuführen.

Temporäre relationale Slots

Grundsätzlich wird bei *temporären relationalen Slots* nach dem Zeitpunkt t^P (Datum des Geburtstags, Produktionsstart etc.) und der Zeitdauer t^D (Reisedauer, Produktionsdauer, Wartezeit etc.) unterschieden. Mittels t^P und t^D können Systeme und deren Verhaltenweisen in eine eindeutige temporäre Beziehung zueinander gesetzt werden. Fahrzeug A fährt beispielsweise zuerst Bahnhof B und dann Bahnhof Z an. Zwischen den beiden Bahnhöfen besteht demnach eine „früher-später-Relation ($t_B \prec t_Z$)“ [EH04-ol]. Eine „später-früher-Relation“ erhält dementsprechend das Symbol $\succ$. Geschieht etwas zum selben Zeitpunkt, wird dies durch ein „=“ Symbol ausgedrückt. Mit diesen drei Basissymbolen lassen sich alle Verhaltensweisen X, Y temporal definieren. X ist hierbei definiert als Verhalten V_x, das aus einer Anfangszeit ($t^P_{Vx}(t_0)$), einer Endzeit ($t^P_{Vx}(t_1)$) und einer Dauer t^D_{Vx} besteht. Entsprechendes gilt für Y.

$$V_x=\{t^P_{Vx}(t_0);\ t^P_{Vx}(t_1);\ t^D_{Vx}\} \qquad V_y=\{t^P_{Vy}(t_0);\ t^P_{Vy}(t_1);\ t^D_{Vy}\}$$

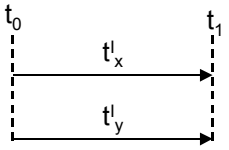
Slot	Symbol	Inverser-Slot	Symbol
X gleich Y	$X = Y$	Y gleich X	$Y = X$
X vor Y	$X \prec Y$	Y nach X	$Y \succ X$
X direkt vor Y	$X \prec_d Y$	Y direkt nach X	$Y \succ_d X$
X überlappt mit Y	$X \prec_o Y$	Y überlappt mit X	$Y \succ_o X$
X startet zusammen mit Y	$X =_s Y$	Y startet zusammen mit X	$Y =_s X$
X läuft innerhalb von Y	$X =_i Y$	Y beinhaltet X	$Y =_i X$
X endet zusammen mit Y	$X =_e Y$	Y endet zusammen mit X	$Y =_e X$

Tabelle 11: Temporäre relationale Slots

Definition: X gleich Y, Y gleich X

Dieser Slot wird für Verhaltensweisen verwendet, die gleichzeitig zu einem definierten Zeitpunkt starten und enden.

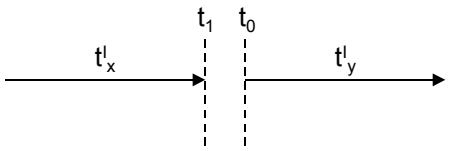
$$\forall V_x\ V_y\ [t^D_{Vx} = t^D_{Vy} \wedge t^P_{Vx}(t_0) = t^P_{Vy}(t_0) \wedge t^P_{Vx}(t_1) = t^P_{Vy}(t_1)]$$



Definition: X vor Y, Y nach X

Dieser Slot wird für Verhaltensweisen verwendet, bei der X irgendwann vor Y begonnen hat und bereits abgeschlossen ist, wenn Y beginnt.

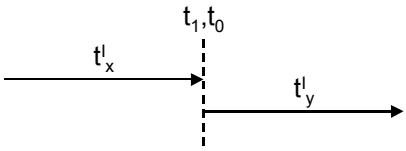
$$\forall V_x\ V_y\ [t^P_{Vx}(t_1) < t^P_{Vy}(t_0)]$$



Definition: X direkt vor Y, Y direkt nach X

Dieser Slot wird für Verhaltensweisen verwendet, bei denen die Endzeit von X mit der Anfangszeit von Y identisch ist.

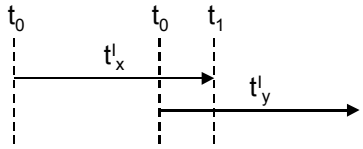
$\forall V_x\ V_y\ [t^P{}_{Vx}(t_1) = t^P{}_{Vy}(t_0)]$



Definition: X überlappt mit Y, Y überlappt mit X

Dieser Slot wird für Verhaltensweisen verwendet, bei denen die Endzeit von X in die Zeitdauer von Y fällt.

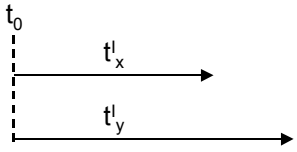
$\forall V_x\ V_y\ [t^P{}_{Vx}(t_0) < t^P{}_{Vy}(t_0) \wedge t^P{}_{Vx}(t_1) > t^P{}_{Vy}(t_0)]$



Definition: X startet zusammen mit Y, Y startet zusammen mit X

Dieser Slot wird für Verhaltensweisen verwendet, bei denen die Anfangs- jedoch nicht die Endzeit von X und Y identisch ist.

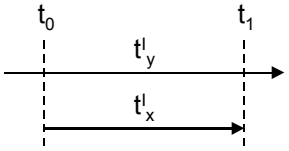
$\forall V_x\ V_y\ [t^P{}_{Vx}(t_0) = t^P{}_{Vy}(t_0) \wedge t^D{}_{Vx} \neq t^D{}_{Vy}]$



Definition: X läuft innerhalb von Y, Y beinhaltet X

Dieser Slot wird für Verhaltensweisen verwendet, bei denen die Zeitdauer von X komplett innerhalb der Zeitdauer von Y läuft.

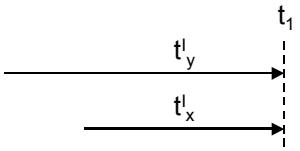
$$\forall V_x\ V_y\ [t^P{}_{Vx}(t_0) > t^P{}_{Vy}(t_0) \wedge t^P{}_{Vx}(t_1) < t^P{}_{Vy}(t_1)]$$



Definition: X endet zusammen mit Y, Y endet zusammen mit X

Dieser Slot wird für Verhaltensweisen verwendet, bei denen die End- jedoch nicht die Anfangszeit von X und Y identisch ist.

$$\forall V_x\ V_y\ [t^P{}_{Vx}(t_1) = t^P{}_{Vy}(t_1) \wedge t^D{}_{Vx} \neq t^D{}_{Vy}]$$



Räumliche relationale Slots

Entscheidend für die Position eines Systems im Raum ist die eindeutige Festlegung seines *absoluten* und auch seines *relativen Standortes*. Der absolute Standort ergibt sich durch den Standort im Netzwerk (bzw. in einem vergleichbaren Koordinatensystem). Der relative Standort ergibt sich durch die Position des Systems in Relation zu anderen Systemen. Die relative Position wird durch die Slots: *DirektVor, DirektHinter, Vor, Hinter, RechtsVon, LinksVon, Auf* und *Unter* beschrieben. Mit diesen räumlichen relationalen Slots können sämtliche relative Raumzustände eines Systems beschrieben werden (siehe Abbildung 41).

Definition: DirektVor, DirektHinter

Ein System A (S^A) befindet sich direkt hinter einem System B (S^B), wenn die aktuelle Prozessrichtung $\overrightarrow{\theta^A} = \begin{pmatrix} x \\ y \end{pmatrix}$ von S^A auf S^B gerichtet ist, der Abstand ($D^{\overrightarrow{AB}} = |d| \cdot \overrightarrow{\theta^A}$) der absoluten Positionen P ($P=\{x,y\}$) beider Systeme größer als Null ist und sich keine weiteren Systeme S^X zwischen den genannten in Prozessrichtung befinden.

$$DirektHinter(S^A, S^B) :- D^{\overrightarrow{AB}} > 0 \wedge D^{\overrightarrow{AX}} > D^{\overrightarrow{AB}}$$
$$DirektVor(S^A, S^B) :- D^{\overrightarrow{AB}} < 0 \wedge D^{\overrightarrow{AX}} < D^{\overrightarrow{AB}}$$

Definition: Vor, Hinter

Ein System A (S^A) befindet sich hinter einem System Y (S^Y), wenn aus der Menge aller Prozessrichtungen $R^A = \{ \overrightarrow{\theta_i^A} \}$ eines gesamten Prozesses eine Prozessrichtung von S^A auf S^Y gerichtet ist und der Abstand ($D^{\overrightarrow{AY}} = |d| \cdot \overrightarrow{\theta_i^A}$) der absoluten Positionen P ($P=\{x,y\}$) beider Systeme größer als Null ist.

$$Hinter(S^A, S^Y) :- D^{\overrightarrow{AY}} > 0$$
$$Vor(S^A, S^Y) :- D^{\overrightarrow{AY}} < 0$$

Definition: RechtsVon, LinksVon

Ein System C (S^C) befindet sich links von einem System A (S^A), wenn der Ortsvektor $\vec{\upsilon}^A = \begin{pmatrix} x \\ y \end{pmatrix}$ von S^A auf S^C einen Winkel $\beta^{\theta\upsilon}$ im Uhrzeigersinn zwischen 180 und 360 Grad zur Prozessrichtung $\vec{\theta}^A = \begin{pmatrix} x \\ y \end{pmatrix}$ in der x,y-Ebene aufspannt. Im anderen Falle befindet sich das System rechts von System A.

$$LinksVon(S^C,S^A) :- 180° < \beta^{\theta\upsilon} < 360°$$
$$RechtsVon(S^C,S^A) :- 0° < \beta^{\theta\upsilon} < 180°$$

Definition: Auf, Unter

Ein System C (S^C) befindet sich auf einem System A (S^A), wenn der Ortsvektor $\vec{\upsilon}^A = \begin{pmatrix} x \\ y \end{pmatrix}$ von S^A auf S^C einen Winkel $\psi^{\theta\upsilon}$ im Uhrzeigersinn von 270 Grad zur Prozessrichtung $\vec{\theta}^A = \begin{pmatrix} x \\ y \end{pmatrix}$ in der x,z-Ebene aufspannt. Im Falle von 90 Grad befindet sich das System C unter System A.

$$Auf(S^C,S^A) :- \psi^{\theta\upsilon} = 270°$$
$$Unter(S^C,S^A) :- \psi^{\theta\upsilon} = 90°$$

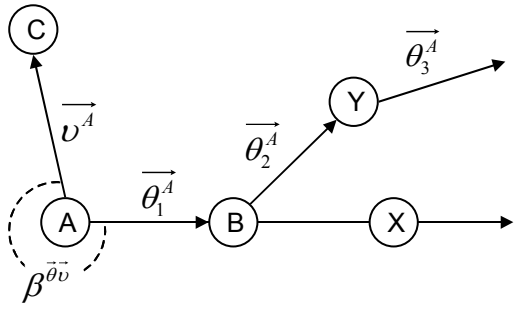


Abbildung 41: Graphische Darstellung der räumlichen relationalen Slots

Die räumlichen relationalen Slots werden auf alle Systeme dieses Szenarios bezogen. So kann ein Fahrzeug *DirektVor* einem anderen Fahrzeug oder *Vor* einem Bahnhof stehen oder sich *Auf* einer Schiene befinden.

Quantitative relationale Slots

Die quantitativen relationalen Slots setzen Zahlen (z.B. Messwerte) in Relation zueinander. Die Basissymbole sind <, ≤, =, ≥, >. Wie bereits für die temporären und die räumlichen relationalen Slots werden hier nicht die Symbole, sondern die Begriffe als Slots in die Basisontologie aufgenommen. Sie lauten: *Kleiner, KleinerGleich, Gleich, GrößerGleich* und *Größer*.

Mit den temporären, räumlichen und quantitativen relationalen Slots können die Beziehungen z.B. zwischen zwei Fahrzeugen an einem einfachen Beispiel (Abbildung 42[27]) eindeutig spezifiziert werden. In diesem Beispiel befindet sich das Fahrzeug Siemens X2 *DirektVor* dem Fahrzeug ABB FE5.

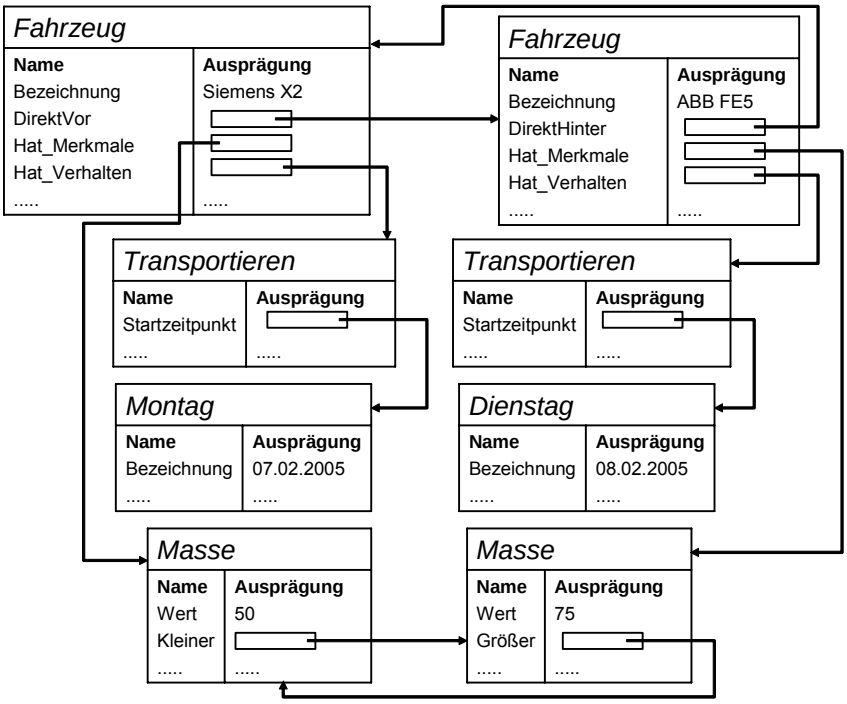


Abbildung 42: Beispiel von Slot-Beziehungen zwischen mehreren Klassen

5.1.1.2 Deskriptive Slots

Die *deskriptiven Slots* beschreiben die Eigenschaften eines Systems, z.B. seine Herkunft, seine Bezeichnung (siehe Abbildung 42) oder seinen Hersteller. Diese Art der deskriptiven Slots wird auch Eigenschafts-Slots (Eigenschaftskanten [Reu04]) genannt. Eigenschaften

[27] Aus Übersichtlichkeitsgründen wurden die Spalten Typ und Kardinalität innerhalb der Klassen aus der Graphik entfernt. Zudem wurden nicht sämtliche Slots der Klassen aufgeführt, was durch die Punkte innerhalb der Klassen symbolisiert werden soll.

können sowohl in der Klasse selbst durch einen deskriptiven Slot beschrieben werden (z.B. Bezeichnung = Siemens X2) als auch sich auf bereits definierte Merkmale in anderen Klassen beziehen (Hat_Merkmale = Masse). Erstere werden in dieser Arbeit als *absolute deskriptive Slots* und letztere als *relationale deskriptive Slots* bezeichnet. Ein Beispiel für relationale und absolute deskriptive Slots bietet Abbildung 43 .

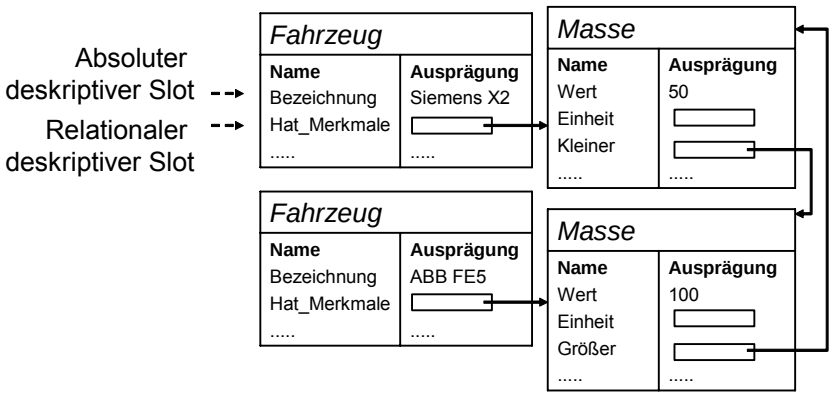


Abbildung 43: Darstellung deskriptiver Slots

5.1.1.3 Prozedurale Slots

Die *prozeduralen Slots* beinhalten eine Funktion oder Regel mit dem Ziel, eine Verhaltensweise oder spezielle kalkulatorische Aussagen bezüglich einer Instanz einer Klasse zu bekommen.

Ein Beispiel für eine Funktion sei die Formel des Trägheitsmoments eines Vollzylinders. Die Formel besitzt die mathematische Form:

$$I = \frac{1}{2} m_{ges} r^2 \quad \text{(m = Masse, r = Außenradius)}$$

Diese Art der Darstellung kann für einen Rechner (und damit auch für ein Fahrzeug) nicht gelesen werden. Hierzu bedarf es der Umwandlung der Funktion in die Standard-Prozedur-Form eines Frames.

<Trägheitsmoment
type: float
exec: If-added
*proc: (1 / 2 * Masse * Außenradius^2)>*

In die Klasse Trägheitsmoment integriert ergibt sich folgendes Bild:

Trägheitsmoment			
Name	**Typ**	**Kardinalität**	**Ausprägung**
Wert	Float	Single	
Formelzeichen	String	Single	
Parameter	Instanz	Multiple	Klasse = (Größe)
Formel	Float (Prozedur)	Single	Value = {1/2*Masse*Außenradius^2}

Tabelle 12: Darstellung eines prozeduralen Slots in der Klasse Vollzylinder

Neben Formel können über prozedurale Slots auch *Regeln* dargestellt werden. Regeln werden entweder im Laufe der Zeit vom Fahrzeug erlernt (siehe Kapitel 5.3) oder durch Menschen vorab bestimmt. Dabei entwickeln sich die Regeln aus der allgemeinen Physik, aus dem Erfahrungswissen der Experten oder aus dem gemeinsamen Verständnis einer Gruppe. Dass ein Auto an einer roten Ampel zu warten hat, ist eine vorgegebene Regel aus dem Straßenverkehr. Dass Wasser bei unter 0°C gefriert, ist eine allgemeine Regel aus der Physik. Dass sich eine Maschine bei einer Temperatur von über 1000°C automatisch abschaltet, ist eine erfahrungsbasierte Regel, die von Experten in mehreren Tests als richtig ermittelt wurde. Regeln können sowohl Zustände, Situationen oder Wahrnehmungen beschreiben als auch Verhaltensweisen vorgeben.

Regel für Zustandsbeschreibungen:

<Zustand_Wasser
type: String
exec: if_added
proc: (if Temperatur > 100 then kocht)>

Es wird der String „kocht“ in den Wertebereich des Slots *Zustand_Wasser* geschrieben.

<Straßenauslastung
type: String
exec: if_added
proc: (if Anzahl Autos < 100 und Streckenlänge = 1 km then gering)>

Es wird der String „gering“ in den Wertebereich des Slots *Straßenauslastung* geschrieben.

Regel für Verhalten:

<Verhalten
type: Lisp-Prozedur
exec: If_added
proc: (If Straßenauslastung = gering then Fahrzeug beschleunigen)>

Als Ausgabe kann demnach auch eine Lisp[28]- oder Prolog -Prozedur angestoßen werden.

[28] Lisp (**Lis**t **P**rogramming) ist neben Prolog die wichtigste Programmiersprache für den KI-Bereich.

Grundsätzlich gehen den Verhaltensregeln die Zustandsregeln voraus, da zunächst die Umwelt bewertet werden muss, bevor ein Verhalten ausgewählt werden kann. Diese Bewertungen beruhen auf der Wissensbasis, die bei jedem Fahrzeug unterschiedlich sein kann. So kann das eine Fahrzeug einen Zustand als „gut“ bewerten, bei dem ein anderes die Bewertung „schlecht“ trifft. Aufgrund dieser unscharfen Bewertungen kann es zu falschen bzw. unerwarteten Verhaltensweisen kommen, die das Gruppenverhalten der Fahrzeuge und damit das Gesamtsystem gefährden können. Eine Lösung dieses Problems bietet Kapitels 5.2.

5.1.2 Entwicklung der Klassen und deren Taxonomie

Eine genaue Bestimmung der *Klassen* sowie die Entwicklung einer *Klassenhierarchie* H^c (auch Taxonomie genannt) werden in den folgenden Kapiteln vorgenommen. Dabei lehnt sich die Taxonomie der Basisontologie an SUMO an, die aus zwei Basisklassen (*körperliche Entität, abstrakte Entität*) besteht. Zusätzlich werden in dieser Arbeit die Klassen *örtliche Entität* und *situative Entität* entwickelt.

Die Pfeile innerhalb der Taxonomie symbolisieren eine hierarchische „ist_eine“ Beziehung zwischen den Klassen. Dies bedeutet, dass die Unterklassen System und Material zur Klasse körperliche Entität die Slots dieser Klasse „erben“. Somit müssen die Slots für die Unterklassen nicht wiederholt aufgeführt werden. Gleiches gilt für sämtliche hierarchischen Darstellungen.

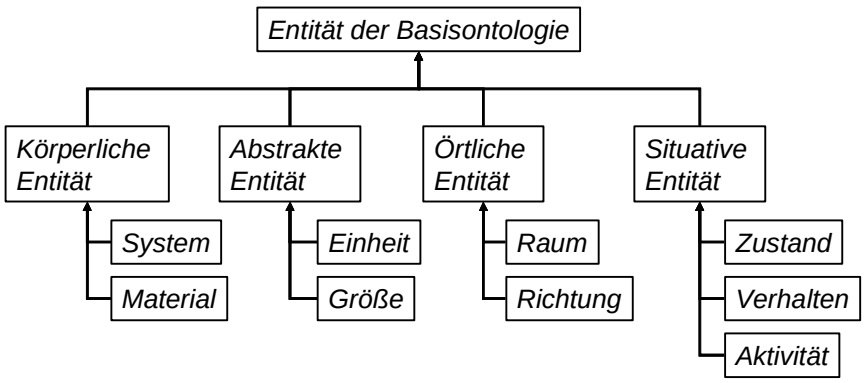


Abbildung 44: Die ersten zwei Klassen-Hierarchiestufen der Basisontologie

Die Klasse körperliche Entität beschreibt die Struktur eines Systems (z.B. Motor Teil_von Fahrzeug; Fahrzeug Teil_von Vernetztes Mechatronisches System; Fahrzeug A vor Fahrzeug B; Fahrzeug A hinter Fahrzeug B) und die verwendeten Materialen (z.B. Motor besteht_aus Stahl, Schraube besteht_aus Aluminium). Für eine spezifische Beschreibung der Eigenschaften einer körperlichen Entität bedarf es der abstrakten, örtlichen und situativen Entitäten.

Über die abstrakten Entitäten werden die nicht zeit- und raumabhängigen Merkmale der körperlichen Entitäten beschrieben. So besitzt ein Fahrzeug als nicht zeit- und

raumabhängiges Merkmal eine Höhe, eine Länge, ein Leergewicht, eine Leistung etc. die durch die abstrakten Entitäten Meter, Zentimeter, Kilogramm und Kilowatt/Stunde näher beschrieben werden. Gleiches gilt für die Materialien. So besitzt Stahl eine gewisse Bruchfestigkeit bzw. Aluminium einen spezifischen Leitwert.

Die zeitliche und räumliche Dimensionierung der körperlichen Entitäten geschieht über die örtlichen und situativen Entitäten. So befindet sich ein Fahrzeug (Unterklasse zu Klasse System) zu einer bestimmten Zeit (15:00 Uhr) an einem bestimmten Ort (z.B. Paderborn) mit einem bestimmten Zustand (z.B. funktionsfähig) in einem bestimmen Verhalten (z.B. Bremsverhalten) und führt eine bestimmte Aktivität aus (Bremsen).

Zur Beschreibung einer körperlichen Entität werden die Slots *Hat_Merkmale*, *Hat_Ort*, *Hat_Zustand* und *Hat_Verhalten* (Aktivitäten werden über das Verhalten gesteuert) benötigt, die auf die entsprechenden Klassen verweisen.

Körperliche Entität			
Name	**Typ**	**Kardinalität**	**Ausprägung**
Bezeichnung	String	Single	
Dokumentation	String	Single	
Hat_Merkmale	Instanz	Multiple	Klasse = (Abstrakte Entität)
Hat_Ort	Instanz	Single	Klasse = (Örtliche Entität)
Hat_Zustand	Instanz	Multiple	Klasse = (Zustand)
Hat_Verhalten	Instanz	Multiple	Klasse = (Verhalten)

Tabelle 13: Darstellung der Klasse Körperliche Entität

In den folgenden Kapiteln werden sämtliche Klassen näher beschrieben.

5.1.2.1 Die Klasse System

Die Klasse System ist zentraler Bestandteil dieser Basisontologie. Sie untergliedert sich in die Klassen technisches System und organisches System. Sämtliche Systeme besitzen Slots, die das Verhalten, die Struktur, die Ziele, die Zustände, die Richtung, das Material, die absolute und die relative Position sowie weitere Eigenschaften (*ähnlich_zu, identisch_zu*) beschreiben (siehe Abbildung 45).

Die Struktur eines Systems wird mit der hierarchischen Verknüpfung (Slot *Teil_von*) der System-Klassen beschrieben. Das Verhalten eines Systems wird mit der Verknüpfung der Verhaltens-Klassen (siehe Kapitel 5.1.2.9) durch den Slot *Hat_Verhalten* beschrieben. Der Zustand eines Systems wird mit der Verknüpfung der Zustands-Klassen durch den Slot *Hat_Zustand* beschrieben. Die Ziele eines Systems können sich sowohl auf die Geschwindigkeit als auch auf einen gewünschten Zustand beziehen. Aus diesem Grunde verweist der Slot *Hat_Ziel* auf alle möglichen Entitäten (Ziel kann auch ein Ort sein). Die relative und absolute Position des Systems wird über die Slots *Hat_Ort* bzw. *RechtsNeben*, *Unter*, *Vor*, *LinksNeben* etc. beschrieben. Weitere Slots beschreiben die Gleichheiten (*identisch_zu*) bzw. die Ähnlichkeiten (*ähnlich_zu*) von Systemen.

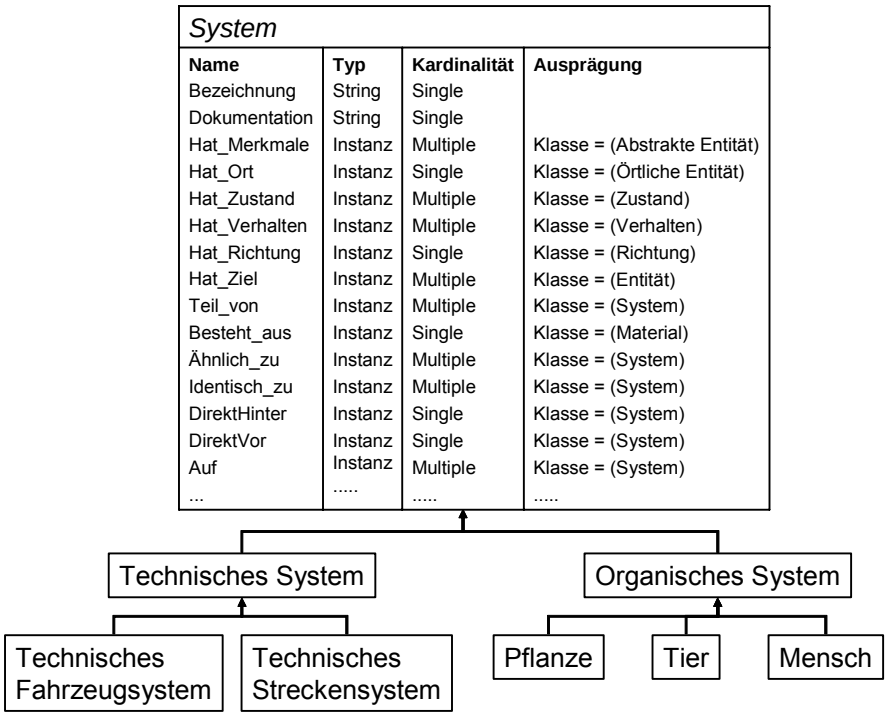


Abbildung 45: Darstellung der Klasse System mit ihren Unterklassen

Technisches System

Unterhalb der Klasse System befindet sich die Klasse technisches System. In ihr werden sowohl die Fahrzeug- als auch die Streckensysteme näher beschrieben. So ist ein Dämpfer ein Bauteil und ein Depot ein Knotenpunktsystem.

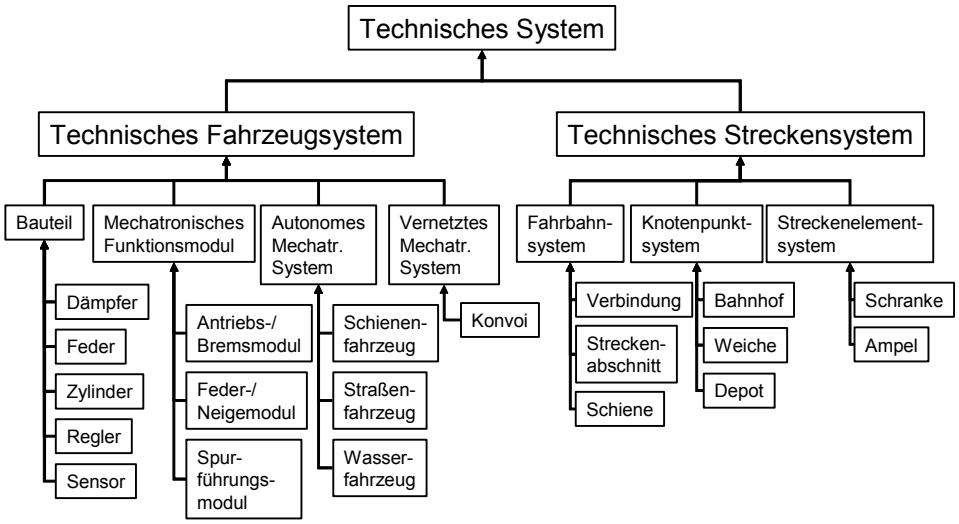


Abbildung 46: Darstellung der Klasse Technisches System mit ihren Unterklassen

Technisches Fahrzeugsystem

Die technischen Fahrzeugsysteme (siehe Abbildung 46) untergliedern sich in die Bauteile, in die aus den Bauteilen zusammengesetzten Mechatronischen Funktionsmodule (MFM), in die

aus den Mechatronischen Funktionsmodulen zusammengesetzten AMS und die wiederum in die aus den AMS zusammengesetzten VMS. Über den Slot *Teil_von* können die einzelnen Systeme zu einem Gesamtsystem zusammengefügt werden (siehe Abbildung 47).

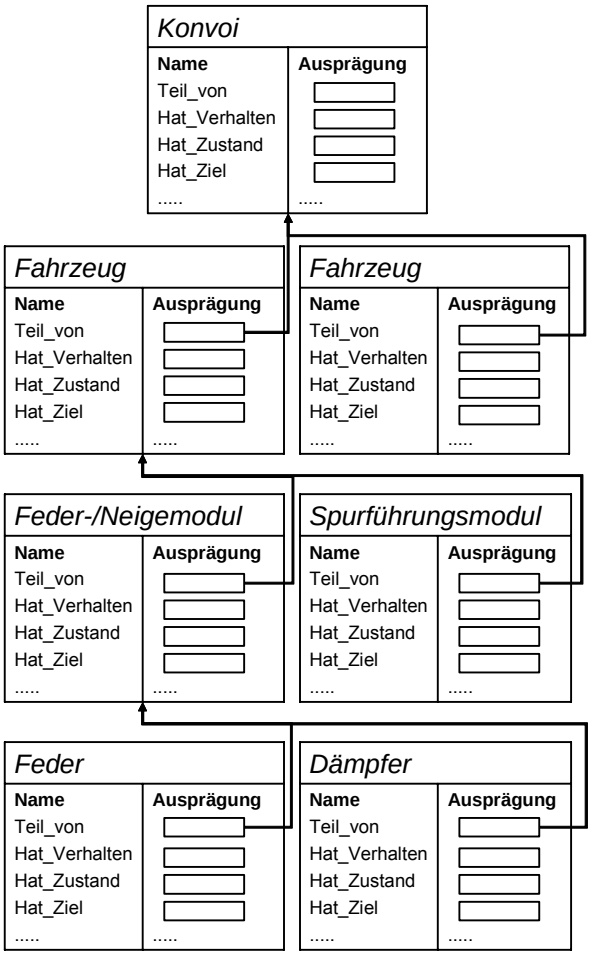


Abbildung 47: Vernetzung eines Technischen Fahrzeugsystems

Technisches Streckensystem

Die technischen Streckensysteme (siehe Abbildung 46) untergliedern sich in die Fahrbahnsysteme, die Knotenpunktsysteme und die Streckenelementsysteme. Sie können entsprechend der Fahrzeugsysteme über den Slot *Teil_von* zu einem Gesamtsystem zusammengefügt werden.

Durch die Verknüpfung der Strecken- mit den Fahrzeugsystemen über die Slots *Auf*, *Hinter*, *DirektVor* etc. kann jedes System relativ lokalisiert werden (siehe Abbildung 48). So befindet sich ein Fahrzeug *DirektVor* einem bestimmten Bahnhof und *Auf* einem bestimmten Streckenabschnitt, der wiederum ein *Teil_von* einer Verbindung ist.

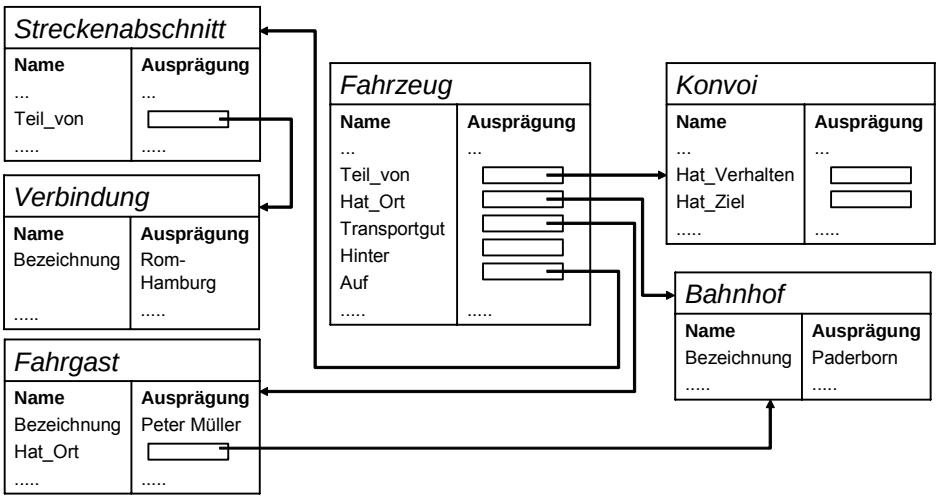


Abbildung 48: Verknüpfung der Klasse Fahrzeug mit anderen Klassen

Organisches System

Die organischen Systeme übernehmen die bereits in der Klasse System definierten Slots. Zu den organischen Systemen gehören zum einen die Menschen (Fahrgäste) und zum anderen die Tiere (Fauna) und die Pflanzen (Flora). Auf die letzt genannten Begriffe wird in dieser Arbeit jedoch nicht weiter eingegangen, da sie keine weitere Bedeutung für die Untersuchungen haben. Dem Fahrgast hingegen kommt bei der Planung der optimalen Verbindung und des Komforts eine besondere Bedeutung zu, da er die Rahmenbedingungen (Constraints) festlegt, unter denen das Fahrzeug zu operieren hat (z.B. die Festlegung einer Route oder eines bestimmtes Komfortwertes; Näheres hierzu in Kapitel 5.3).

5.1.2.2 Die Klasse Material

Da sich die vorliegende Arbeit auf reine Festkörper bezieht, grenzt sie das weite Feld der Materialien etwas ein, d.h. es werden keine Gase und Flüssigkeiten bzw. biologische Naturstoffe (z.B. Milch, Zucker) betrachtet. Damit bleiben in einer ersten Strukturierung die aus dem täglichen Leben bekannten Metalle (Eisen, Aluminium, Kupfer, Blei etc.), Nichtmetalle (Glas, Porzellan, Graphit, Keramik etc.) und (nicht-biologische) Naturstoffe (Holz, Kohle, Gummi, Sand, Kies etc.). Da diese Arbeit nicht tiefer in die Materialwissenschaft vordringen möchte, werden Aspekte wie z.B. die Struktur der Materialen nicht weiter behandelt. Für die vorliegende Arbeit sei es ausreichend, dass die Materialien bestimmte Größen (Bruchfestigkeit, Zugfestigkeit, Härte, Duktilität etc.), bestimmte Verhaltensweisen (Spannungsverhalten, Dehnungsverhalten etc.) und bestimmte Zustände (fest, flüssig, gasförmig, gebraucht, beschädigt, weich, ausgehärtet) besitzen. Die Klasse Material übernimmt vollständig die in der Klasse Körperliche Entität definierten Slots.

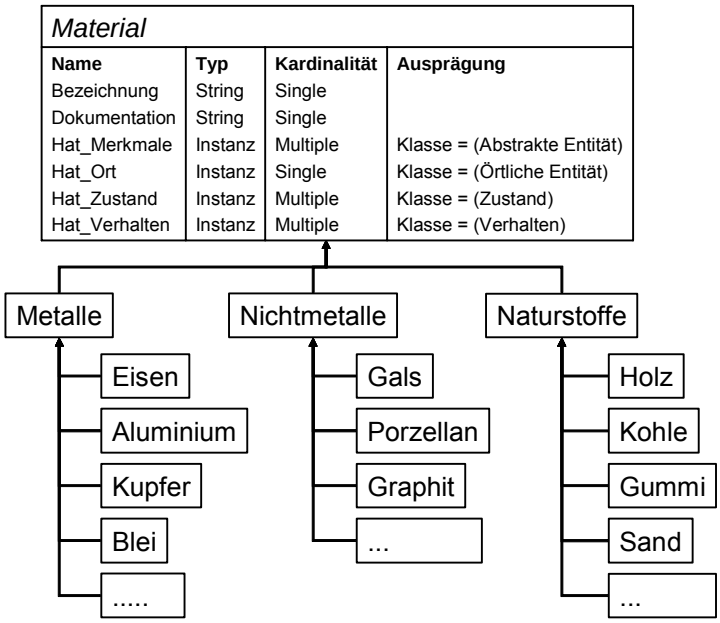


Abbildung 49: Darstellung der Klasse Material mit ihren Unterklassen

5.1.2.3 Die Klasse Einheit

Die Klasse Einheit ist eine Unterklasse der Klasse abstrakte Entität und untergliedert sich in alle Einheiten, die zur Beschreibung der Größen eines Systems notwendig sind. Dies sind die physikalischen und die monetären Einheiten. Jede dieser Klassen besitzt als Slot eine *Abkürzung* (z.B. m für Meter) eine *Referenzeinheit* (z.B. Sekunde für Minute) und eine auf die Referenzeinheit bezogene *Beziehung* (z.B. 1 Gramm = 0,001 Kilogramm).

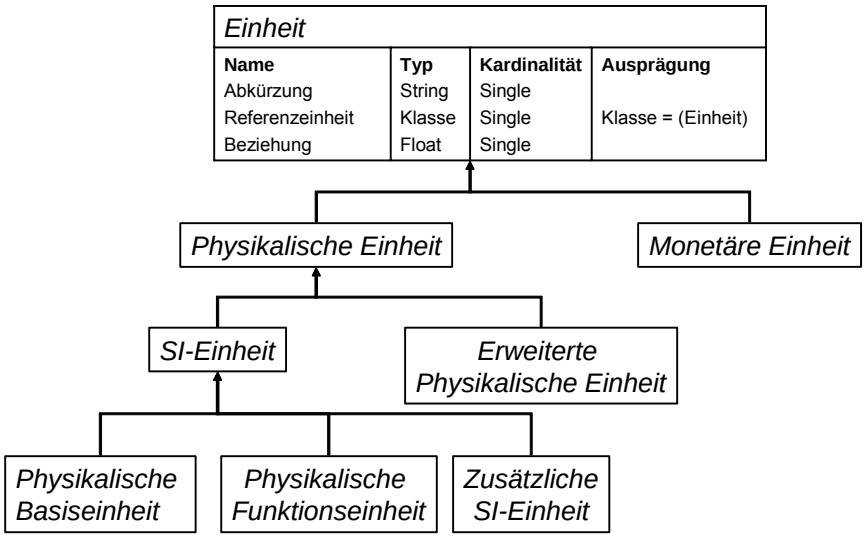


Abbildung 50: Darstellung der Klasse Einheit mit ihren Unterklassen

5.1.2.3.1 Die Klasse Physikalische Einheit

Die physikalischen Einheiten müssen sowohl für das Fahrzeug als auch für die Entwickler verständlich dargestellt werden. Für die Fahrzeuge ist es wichtig, dass die physikalischen Einheiten in maschinenlesbarer Form beschrieben werden können, da sie insbesondere auf der MFM-Ebene Erfahrungen über die möglichst optimalen Parametereinstellungen ihrer mechatronischen Funktionsmodule austauschen. Hieraus ergibt sich die Anforderung nach einer formalen Spezifizierung und Strukturierung sowie einer Vollständigkeit der physikalischen Einheiten, um die Strukturen, Zustände, Ziele und Verhaltensweisen der MFM-, AMS- als auch der VMS-Ebene umfassend und eindeutig beschreiben zu können.

Die Klasse Physikalische Basiseinheit

Da die Festlegung der physikalischen Basiseinheiten grundsätzlich willkürlich ist, wurde im Jahre 1960 auf der internationalen Generalkonferenz für Maß und Gewicht die allgemeine Verwendung des Internationalen Einheitensystems oder des SI-Systems (Système International d'Unités; ISO 1000, DIN 1301) empfohlen [BWC+03-ol], [Ber03b-ol], [IFC03-ol]. Das internationale Einheitensystem baut auf den folgenden sieben physikalischen Basisgrößen (siehe Kapitel 5.1.2.4) und physikalischen Basiseinheiten auf:

Physikalische Basisgrößen und Physikalische Basiseinheiten			
Physikalische Basisgröße	*Formel-zeichen*	*Physikalisches Basiseinheiten*	*Einheitenzeichen*
Länge	*l*	Meter	m
Zeit	*t*	Sekunde	s
Masse	*m*	Kilogramm	kg
Elektrische Stromstärke	*I*	Ampere	A
Temperatur	*T*	Kelvin	K
Lichtstärke	*L*	Candela	cd
Stoffmenge	*ν*	Mol	mol

Tabelle 14: SI-Basisgrößen und Basiseinheiten [Win03-ol]

Entsprechend der ontologischen Darstellung ergibt sich hieraus die Klasse physikalische Basiseinheit.

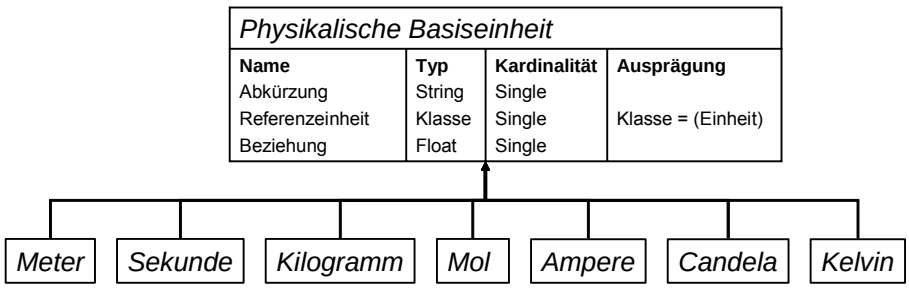


Abbildung 51: Ontologische Darstellung der Klasse Physikalische Basiseinheit

Die Klasse Physikalische Funktionseinheit

Innerhalb des SI-Systems wurden neben den physikalischen Basiseinheiten auch die physikalischen Funktionseinheiten festgelegt. Diese sind in Tabelle 15 [Sto95] dargestellt.

Physikalische Funktionsgrößen und Physikalische Funktionseinheiten		
Physikalische Funktionsgröße	*Formelzeichen*	*Funktionseinheiten*
Frequenz	*f*	s^{-1} = Hz (Hertz)
Kraft	*F*	kg m s^{-2} = N (Newton)
Druck	*p*	kg m^{-1} s^{-2} = N m^{-2} = Pa (Pascal)
Arbeit, Energie	*W*, *E*	kg m^2 s^{-2} = N m = J (Joule)
Leistung	*P*	kg m^2 s^{-3} = W (Watt)
Elektrische Ladung	*Q*	A s = C (Coulomb)
Elektrische Spannung	*U*	kg m^2 A s^{-3} = W A^{-1} = V (Volt)
Elektrischer Widerstand	*R*	kg m^2 A^{-2} s^{-3} = V A^{-1} = Ω (Ohm)
Kapazität	*C*	s^4 A^2 kg^{-1} m^{-2} = C V^{-1} = F (Farad)
Magnetische Induktion	*B*	kg A^{-1} s^{-2} = T (Tesla)
Induktivität	*L*	kg m^2 s^{-2} A^{-2} = H (Henry)
Energiedosis	*D*	m^2 s^{-2} = Gy (Gray)
Aktivität	*A*	s^{-1} = Bq (Bequerel)

Tabelle 15: SI-Funktionsgrößen und Funktionseinheiten [Sto95]

Aus den Funktionseinheiten ergibt sich ein der Abbildung 51 entsprechendes Bild.

Die Klasse Zusätzliche SI-Einheit

Weitere Einheiten wären Quadratmeter (m^2), Kubikmeter (m^3), Grad (°), Meter/Sekunde (m/s), Newtonmeter (Nm) und Prozent (%). Insgesamt ergibt sich somit für die Klasse SI-Einheit die in Abbildung 52 dargestellte Ontologie.

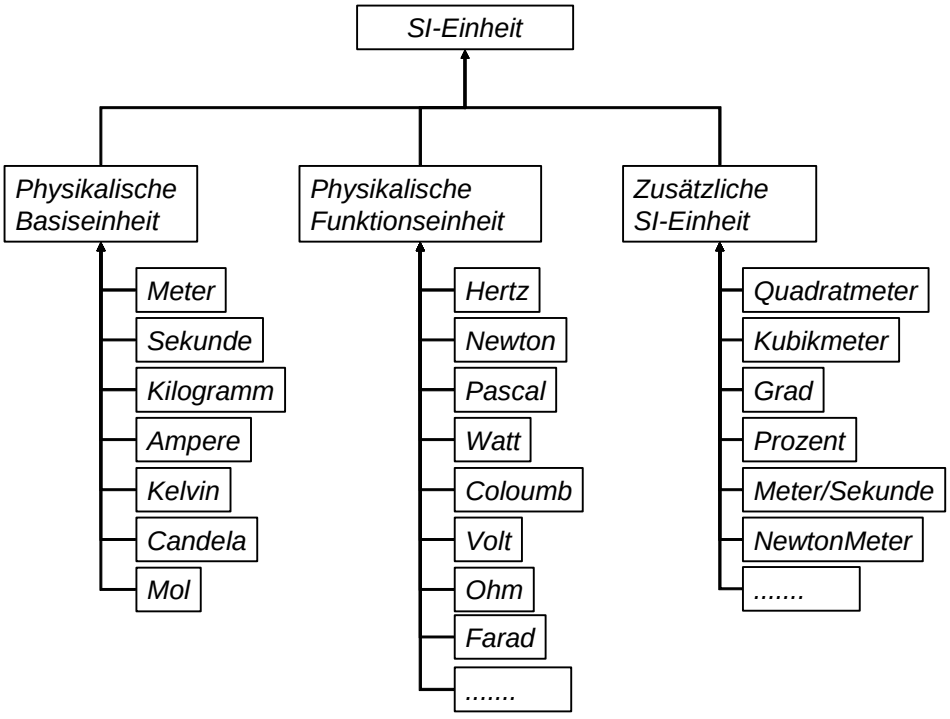


Abbildung 52: Darstellung der Klasse SI-Einheit mit ihren Unterklassen

Die Klasse Erweiterte Physikalische Einheit

Neben den SI-Einheiten werden in der Literatur weitere Einheiten verwendet, die sich jedoch alle auf die SI-Einheiten zurückführen lassen. Die Verwendung weiterer Einheiten liegt zum einen daran, dass die Zahlenwerte von physikalischen Größen üblicherweise im Bereich 999,9 bis 0,1 liegen. Dies wird dadurch erreicht, dass die Einheiten mit Vorsätzen versehen werden, die die Zehnerpotenzen der Einheiten bedeuten. So gibt es neben der Einheit Meter z.B. noch die Einheiten Kilo-, Milli-, und Zentimeter. Zum anderen liegt die Verwendung anderer Einheiten an den historisch gewachsenen Einheiten spezieller Domänen (z.B. die Seemeile in der Schifffahrt) bzw. an den unterschiedlichen metrischen Systemen (z.B. die Meile). All diese Einheiten werden unterhalb der Klasse erweiterte physikalische Einheit strukturiert (siehe Abbildung 53).

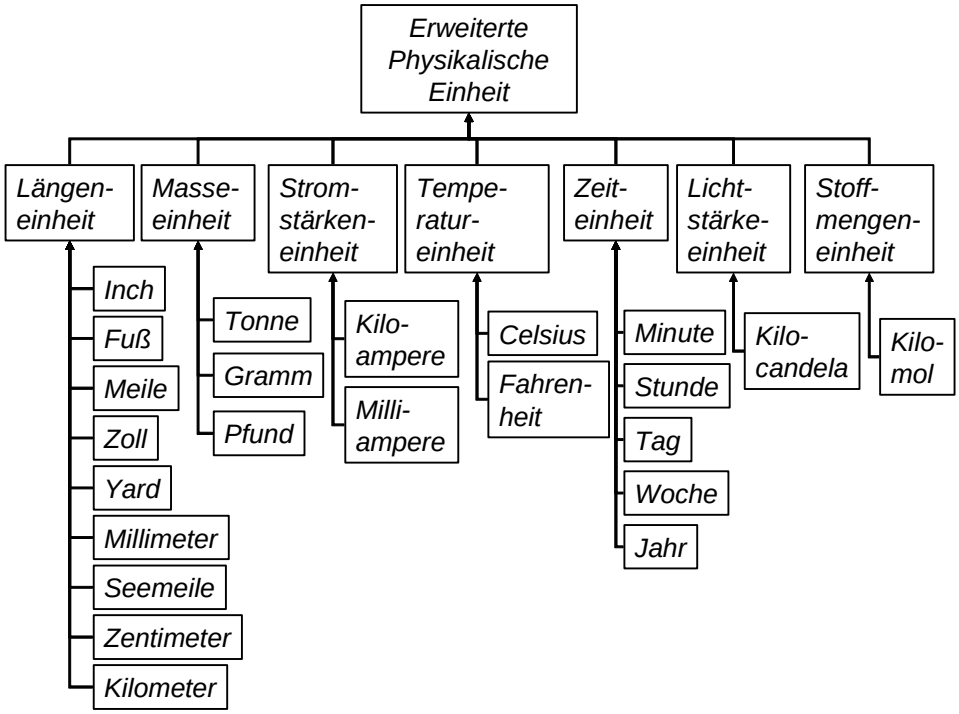


Abbildung 53: Ontologische Darstellung der Klasse erweiterte physikalische Einheit

Für das Rechnen mit unterschiedlichen Einheiten wurde der Slot *Referenzeinheit* eingeführt. So bezieht sich die Klasse Kilometer auf die *Referenzeinheit* Meter. Das Verhältnis von Kilometer zur *Referenzeinheit* Meter wurde in einem zusätzlichen Slot *Beziehung* beschrieben. Er ermöglicht einen quantitativen Vergleich zweier Einheiten derselben Größe. (siehe Abbildung 54). So ist die Beziehung zwischen Kilometer und Meter gleich 1000.

Kilometer	
Name	**Ausprägung**
Referenzeinheit	Meter
Beziehung	1000

=

Meter

=

Zentimeter	
Name	**Ausprägung**
Referenzeinnheit	Meter
Beziehung	0,1

Abbildung 54:Vergleich der Klassen Kilometer, Meter, Zentimeter

5.1.2.3.2 Die Klasse Monetäre Einheit

Die Klasse der Monetären Einheiten wird unterteilt in die Klassen Euro, US-Dollar, Britisches Pfund und Schweizer Franken (weitere Währungen sind ohne weiteres hinzuzufügen). Zur Umrechnung und zum Vergleich der einzelnen Währungen besitzt jede Klasse wie die physikalischen Einheiten die Slots *Beziehung* und *Referenzeinheit*. Als Basiseinheit wird der Euro angenommen.

5.1.2.4 Die Klasse Größe

Die Klasse Größe ist eine Unterklasse der Klasse abstrakte Entität und untergliedert sich in alle Größen, die zur Beschreibung der Merkmale und Ziele eines Systems und Materials benötigt werden. Dies sind die monetären, die logistischen und die physikalischen Größen.

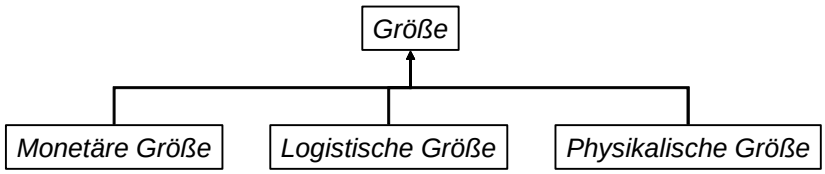


Abbildung 55: Die Klasse Größe mit ihren Unterklassen

Jede Größen (z.B. die physikalische Größen Geschwindigkeit) kann endlos viele Instanzen (z.B. 3 km/h, 4 km/h, 5 km/h) besitzen. Diese Instanzen werden durch ihre relationalen (*Parameter*, *Einheit, Kleiner, Kleiner Gleich, Gleich, Größer Gleich, Größer*) prozeduralen (*Formel*) und deskriptiven Slots (*Wert, Formelzeichen*) näher beschrieben. Dementsprechend ergibt sich folgendes Bild der Klasse Größe.

Größe			
Name	**Typ**	**Kardinalität**	**Ausprägung**
Wert	Float	Single	
Formelzeichen	String	Single	
Formel	Float (Prozedur)	Single	Value = ( )
Parameter	Instanz	Multiple	Klasse = (Größe)
Einheit	Klasse	Single	Klasse = (Einheit)
Gleich	Instanz	Multiple	Klasse = (Größe)
Größer	Instanz	Multiple	Klasse = (Größe)
GrößerGleich	Instanz	Multiple	Klasse = (Größe)
Kleiner	Instanz	Multiple	Klasse = (Größe)
KleinerGleich	Instanz	Multiple	Klasse = (Größe)

Tabelle 16: Darstellung der Klasse Größe mit ihren Slots

Die Klasse Monetäre Größe

Die monetären Größen beistzen die Besonderheit, dass sie sowohl positiv (Einnahmen) als auch negativ (Ausgaben) zu bewerten sind. So sind die Streckengebühren für das Fahrzeug negativ (belasten den Gewinn) und für die Strecke positiv (erhöhen den Gewinn). Weitere monetäre Größen sind z.B. der Fahrtpreis oder die Transferzahlungen zwischen den Fahrzeugen.

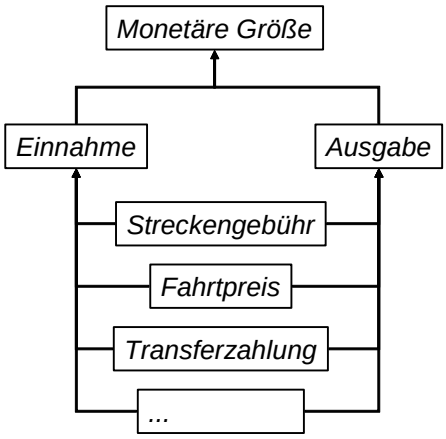


Abbildung 56: Die Klasse Monetäre Größe mit ihren Unterklassen

Die Klasse Logistische Größe

Zu den logistischen Größen zählen z.B. die Auslastung einer Verbindung, die Pünktlichkeit des Eintreffens eines Fahrzeugs oder die Kapazität einer Verbindung. Die logistischen Größen dienen zur Bewertung einer bestimmten Situation. So bedeutet eine Auslastung einer Verbindung von 95% für das Fahrzeug, dass hier in nächster Zeit ein Stau zu erwarten ist. Ist das Fahrzeug unter Zeitdruck, sollte es diesen Bereich umfahren.

Die logistischen Größen sind in die logistischen Fahrzeuggrößen und die logistischen Verbindungsgrößen unterteilt.

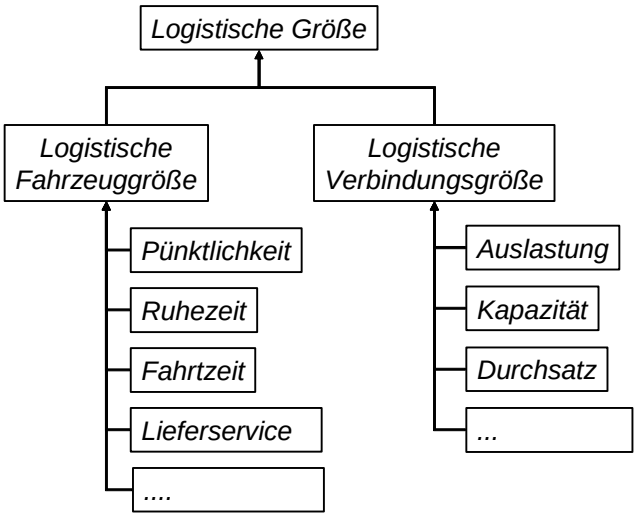


Abbildung 57: Die Klasse Logistische Größe mit ihren Unterklassen

Die Klasse Physikalische Größe

Die physikalischen Größen lassen sich in die physikalischen Basisgrößen und Funktionsgrößen unterteilen. Physikalische Basisgrößen sind Größen, für die keine Definition im Sinne einer Zurückführung auf schon bekannte Größen gegeben werden kann [Sto95]. Physikalische Funktionsgrößen sind Größen, die sich direkt aus anderen Größen, wie z.B. den Basisgrößen, entwickeln lassen.

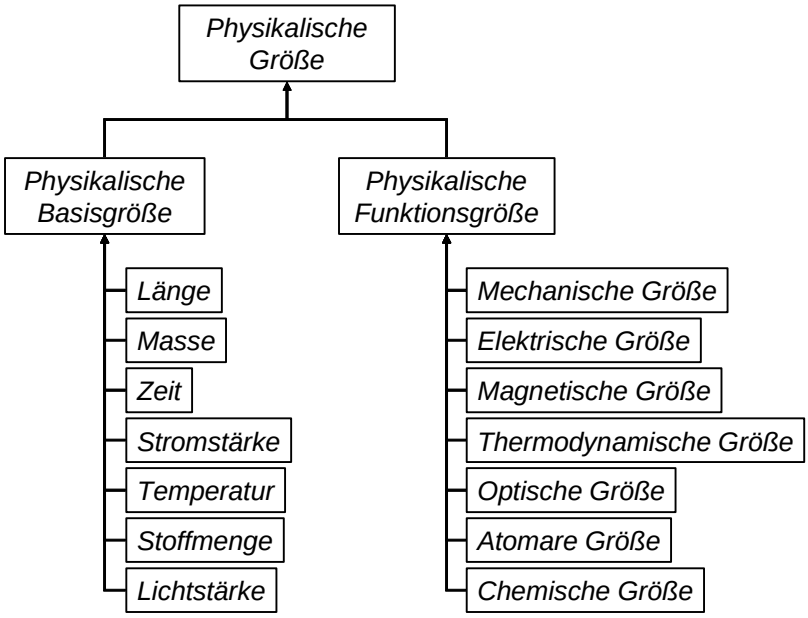


Abbildung 58: Die Klasse Physikalische Größe mit ihren Unterklassen

Eine genauere Darstellung der physikalischen Funktionsgrößen leistet Abbildung 59.

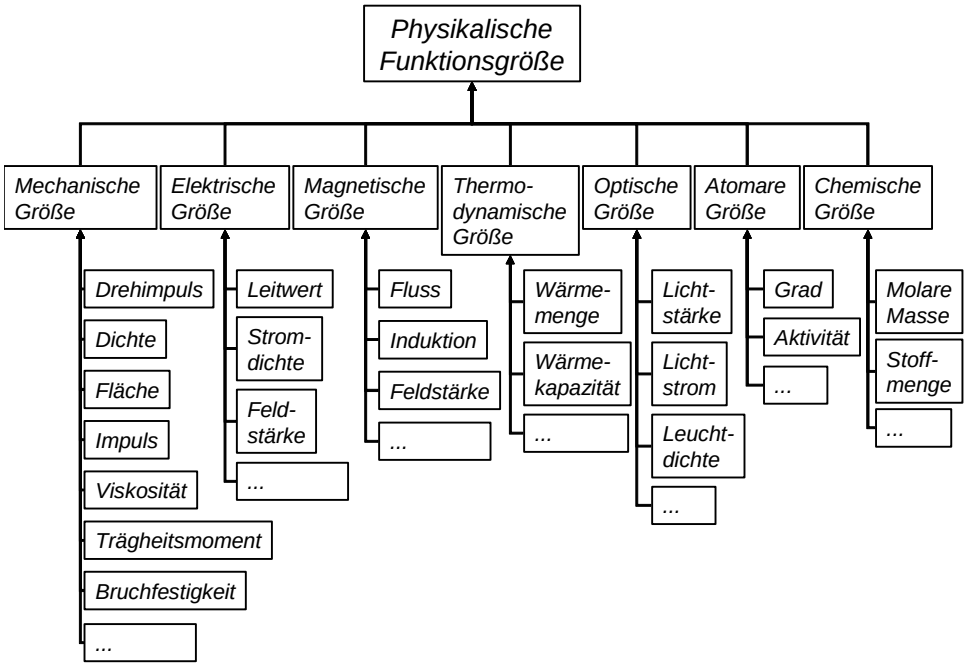


Abbildung 59: Die Klasse Physikalische Funktionsgröße mit ihren Unterklassen

Abbildung 60 beschreibt den Zusammenhang zwischen den System-Klassen, den Größen-Klassen und den Einheiten-Klassen.

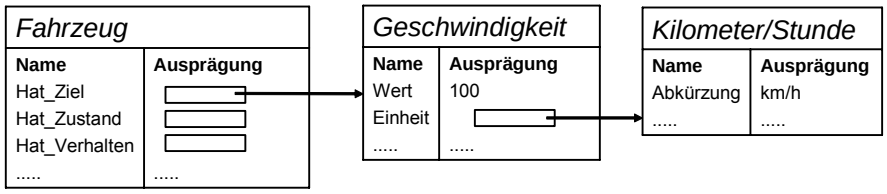


Abbildung 60: Verbindung der System-Klassen zu den Größen- und Einheiten-Klassen

5.1.2.5 Die Klasse Raum

Über die Klasse Raum kann ein System seine absolute Lage im Raum definieren. So befindet sich das System Bahnhof oder das System Fahrzeug z.B. im Raum Paderborn (Stadt) im Graphen (x,y). Weitere Systeme können entsprechend dieser Darstellung beschrieben werden. Zusätzlich bietet die Ontologie die Möglichkeit, eine horizontale oder ein vertikale Lage eines Systems zu bestimmen.

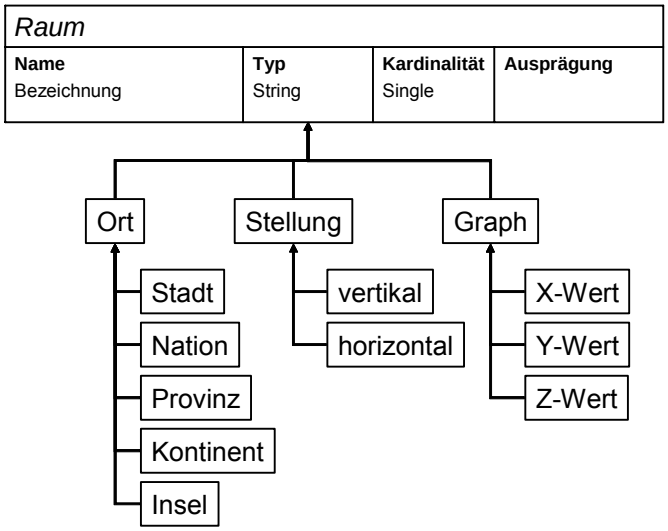


Abbildung 61: Darstellung der Klasse Raum mit ihren Unterklassen

5.1.2.6 Die Klasse Richtung

Die Klasse Richtung definiert sowohl die Ausrichtung als auch die Bewegungsrichtung eines Systems. So zeigt die Fahrtrichtung des Systems Fahrzeug nach Norden und die Bewegungsrichtung nach links, um z.B. einen Überholvorgang einleiten zu können. Die Verbindung der Systemklassen zu den Richtungsklassen erfolgt über den Slot *Hat-Richtung*.

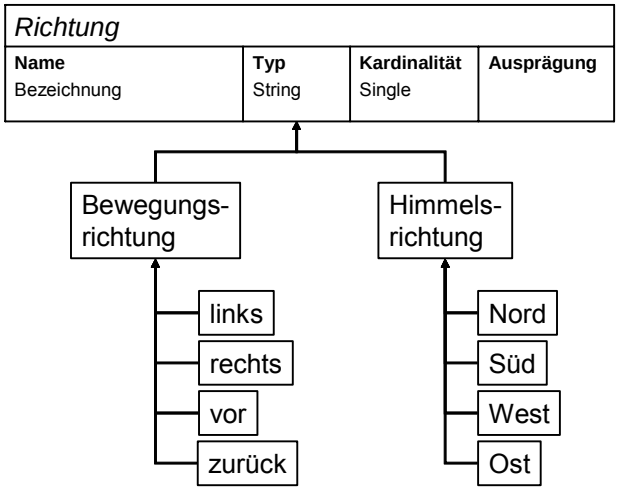


Abbildung 62: Darstellung der Klasse Richtung mit ihren Unterklassen

5.1.2.7 Die Klasse Zustand

Der Zustand eines Systems wird über die Zustandsklassen definiert. Die Verbindung wird durch den Slot *Hat_Zustand* innerhalb der Systemklassen erreicht.

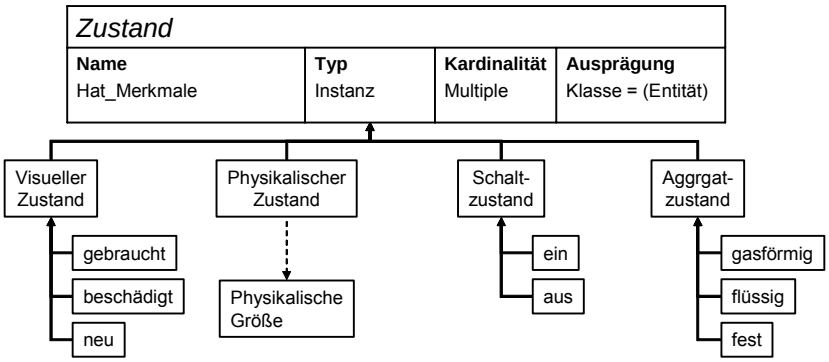


Abbildung 63: Darstellung der Klasse Zustand mit ihren Unterklassen

Die Klasse Zustand untergliedert sich in die visuellen Zustände, die physikalischen Zustände, die Schalt- und die Aggregatzustände. Die visuellen Zustände beziehen sich auf die Oberfläche eines Systems. So kann die Oberfläche beschädigt, porös, matt, glänzend oder neu sein. Die physikalischen Zustände beziehen sich auf die physikalischen Größen, die bereits in Kapitel 5.1.2.4 definiert wurden. Aus diesem Grunde wird hier lediglich ein Verweis zu den bereits definierten Klassen gezogen. Die Schaltzustände beziehen sich meist auf elektrotechnische Anlagen und zeigen an, ob ein System momentan operative ist oder nicht. Der Aggregatzustand definiert, ob sich ein Material in einem festen, flüssigen oder gasförmigen Zustand befindet.

Jeder Zustand besitzt spezifische Merkmale, die durch entsprechende Größen spezifiziert werden können. So besitzt ein gebrauchter Stahl eine andere Bruchfestigkeit (Unterklasse der physikliaschen Funktionsgrößen) als ein neuer Stahl. Diese Spezifizierung erfolgt über den Slot *Hat-Merkmale*, der bereits in der Klasse System verwendet wird (einmal definierte Slots können für mehrere Klassen verwendet werden).

5.1.2.8 Die Klasse Aktivität

Jedes System besitzt eine oder mehrere spezifische Aktivitäten. Diese untergliedern sich in die kognitiven, sozialen und motorischen Aktivitäten. Kognitive Aktivitäten sind immer datengetriebene Aktivitäten, die ohne einen Partner durchgeführt werden können, z.B. das Explorieren eines Raumes oder das Suchen einer Verbindung. Sollte ein Partner dazukommen, ist eine soziale Aktivität notwendig. Dies kann die Kommunikation oder auch die Kooperation mit diesem Partner sein.

Die sozialen und kognitiven Aktivitäten generieren zunächst nur eine Verarbeitung der vorhandenen Daten. Das Suchen einer Verbindung bedeutet nicht, dass das Fahrzeug diese Verbindung fährt. Die Bereitschaft zur Kooperation schließt nicht auf eine konkrete Aktivität. Diese, z.B. das eigentliche Fahren einer Verbindung bzw. der kooperative Zusammenschluß zu einem Konvoi, wird über die motorischen Aktivitäten erreicht. So kann das Fahrzeug beschleunigen, sich neigen, bremsen, sich verbinden oder sich trennen, es kann Fahrgäste transportieren oder von einem Roboter repariert werden.

Für eine Aktivität muss bestimmt werden:

- zu welchem Zeitpunkt sie beginnt und zu welchem Zeitpunkt sie enden soll
- mit welcher Intensität diese Aktivität ausgeführt werden soll (z.B. Beschleunigen mit $5m/s^2$ oder $50m/s^2$)
- mit welcher Methode die Aktivität durchgeführt werden soll (z.B. Suchen mit Hill-Climbing oder A*-Algorithmus)
- welches System die Aktivität durchführen soll (Feder- und Neigemodul, Spurführungsmodul etc.) und
- welche Kosten durch die Aktivität entstehen.

Aus diesen Anforderungen erfgibt sich für die Klasse Aktivität die folgende Tabelle:

Aktivität			
Name	**Typ**	**Kardinalität**	**Ausprägung**
Bezeichnung	String	Single	
Startzeitpunkt	Instanz	Single	Klasse = (Zeiteinheit)
Endzeitpunkt	Instanz	Single	Klasse = (Zeiteinheit)
Intensität	Instanz	Single	Klasse = (Physikalische Größe)
Methode	Instanz	Single	
Wird_ausgeführt_durch	Instanz	Single	Klasse = (System)
Kosten	Instanz	Single	Klasse = (Monetäre Einheit)
Vor_	Instanz	Multiple	Klasse = (Aktivität)

Nach_	Instanz	Multiple	Klasse = (Aktivität)
Direkt_Vor	Instanz	Single	Klasse = (Aktivität)
Direkt_Nach	Instanz	Single	Klasse = (Aktivität)
Startet_Zusammen_Mit	Instanz	Multiple	Klasse = (Aktivität)
Endet_Zusammen_Mit	Instanz	Multiple	Klasse = (Aktivität)
Überlappt_Mit	Instanz	Multiple	Klasse = (Aktivität)
Läuft_Innerhalb	Instanz	Multiple	Klasse = (Aktivität)
Gleich	Instanz	Multiple	Klasse = (Aktivität)

Tabelle 17: Darstellung der Klasse Aktivität mit ihren Slots

Entsprechend der ontologischen Darstellung ergibt sich folgende Struktur.

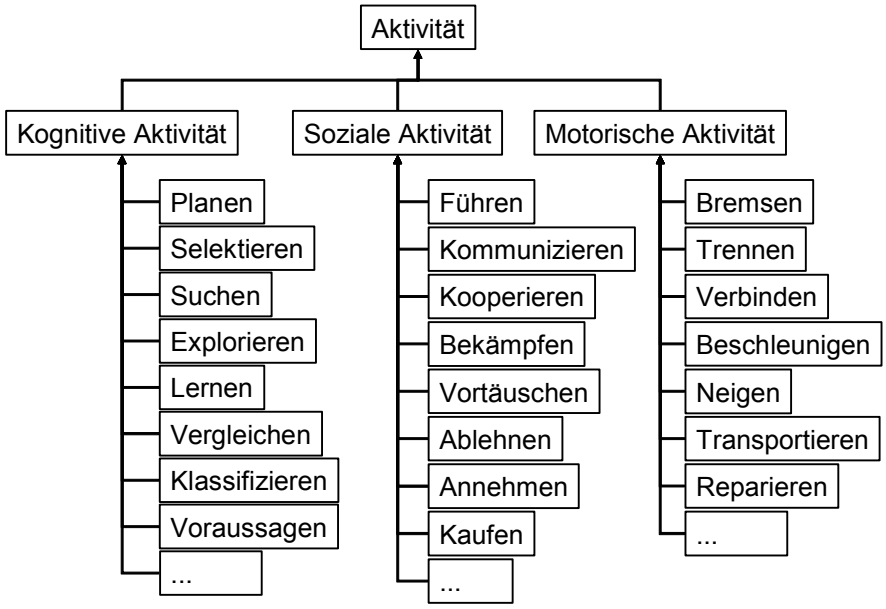


Abbildung 64: Darstellung der Klasse Aktivität mit ihren Unterklassen

5.1.2.9 Die Klasse Verhalten

Laut Definition 2-6 wird dann von einem Verhalten eines Systems gesprochen, wenn eine Veränderung des Zustandes bzw. der Zustandsgrößen des Systems auf der Makroebene beobachtet werden kann. Diese Veränderung des Zustands wird innerhalb der Klasse Verhalten durch die Slots *Ausgangszustand* und *Endzustand* mit dem Verweis zu den Zustands-Klassen (Kapitel 5.1.2.7) beschrieben. So kann der Ausgangszustand von Eisen fest sein. Wird das Eisen jedoch erhitzt, geht es in einen flüssigen Zustand über.

Der Anstoß zu einer Veränderung des Zustands bzw. der Zustände kann:

- ohne Einfluss von außen erfolgen oder
- mit einem Einfluss von außen zusammenhängen.

Ein Einfluss (Eingaben in ein System) kann sowohl ein Materie-, Energie- sowie Informationsfluss eines anderen Systems sein. Gleiches gilt für die Wirkungsmöglichkeiten

(Ausgaben eines Systems). So kann die Ausgabe eines Verhaltens ein Materie-, Energie – sowie Informationsfluss für ein anderes System sein. Grundsätzlich sind alle Kombinationen von Einfluss und Wirkungsmöglichkeiten denkbar. (Beispiel 1: Beim Bremsen eines Autos an der Ampel bewirkt ein Informationsfluss (Einfluss: Ampel rot) einen Energiefluss (Abgabe von Energie durch das Bremsen); Beispiel 2: Beim Laden eines Elektroautos bewirkt ein Energiefluss (Einfluss: Elektrizität) einen Materiefluss (Bezahlung mit Geld)). Unabhängig von den Einflüssen und den Wirkungsmöglichkeiten wird während eines Verhaltens stets Energie (sogenannte Entropie) freigesetzt. Dieser Energieabfluss zwingt Systeme dazu, immer neue Energieflüsse aufzunehmen.

Neben einer Einfluss–Wirkungsmöglichkeit-Beziehung kann es möglich sein, dass mit einem Einfluss von außen keine Wirkungsmöglichkeit hervorgerufen wird. Dies ist dann der Fall, wenn das System den Einfluss auf Grund fehlender Strukturen nicht weiterleiten oder verarbeiten kann. Im Falle eines Informaionsflusses wäre dies die rote Ampel, die vom System nicht gedeutet werden kann oder die Nachricht eines anderen Fahrzeugs, die aufgrund unterschiedlicher Ontologien nicht verstanden werden kann (eine Lösung hierzu bietet das folgende Kapitel). Allerdings kann es auch vorkommen, dass der Einfluss von außen zwar erkannt wird, aber so schwach ist, dass die vom System gegebene Reizschwelle nicht überschritten wird (um zu verhindern, dass auf alle Einflüsse reagiert wird) oder dass das System so träge ist, dass es im Beobachtungszeitraum praktisch nicht reagiert. Neben der Art des Einflusses bestimmt demnach die Stärke des Einflusses die Wirkungsmöglichkeiten eines Verhaltens. So erzeugt ein starker Energiefluss als Einfluss (z.B. starker Gegenwind beim Fahren) einen entsprechend starken Energiefluss als Wirkungsmöglichkeit (z.B. hoher Benzinverbrauch). Ein entsprechend schwacher Einfluss kann, wie bereits oben erwähnt, auch zu gar keiner Wirkungsmöglichkeit führen.

In der vorliegenden Arbeit werden die Einflüsse durch den Slot *Einfluss* definiert. Dieser Einfluss kann laut Abbildung 1 durch sämtliche Systemteilnehmer (das Umfeld, den Benutzer, das eigene System, die eigene Struktur) erfolgen. So könnte ein Einfluss die Temperatur der Bremsen, die Bruchfestigkeit des Aufbaus aber auch die Geschwindigkeit eines vorausfahrenden Fahrzeugs sein. Entsprechendes gilt für die *Wirkungsmöglichkeit.* Die Energieabgabe wird über den Slot *Energieabgabe* definiert.

Ein Verhalten wird nicht gekapselt für sich, sondern „aus Sicht der Informationsverarbeitung wird Systemverhalten durch eine Menge von Prozessen, die als Abfolge von Aktivitäten verstanden werden, beschrieben“ [FGK+04]. Um diese Abfolge von Aktivitäten beschreiben zu können, müssen sie in eine temporäre Beziehung zueinander gebracht werden. Dies bedeutet, dass innerhalb eines Verhaltens:

- parallele Aktivitäten und
- sequenziellen Aktivitäten durchgeführt werden können.

Die Definition der spezifischen Aktivitäten geschieht über den Slot *Aktivität.* Die temporären Beziehungen zwischen den Aktivitäten werden innerhalb der Klasse Aktivität definiert.

Letzendlich muss definiert werden, zu welchem Zeitpunkt ein Verhalten von wem ausgeführt wird. Dieses wird durch die Slots *Wird_ausgeführt_durch, Startzeitpunkt, Endzeitpunkt* und *Dauer* erreicht.

Aus den oben beschriebenen Anforderungen für ein Verhalten lässt sich die Klasse Verhalten wie folgt definieren.

Verhalten			
Name	**Typ**	**Kardinalität**	**Ausprägung**
Bezeichnung	String	Single	
Einfluss	Instanz	Multiple	Klasse = (System)
Wirkungsmöglichkeit	Instanz	Multiple	Klasse = (System)
Energieabgabe	Instanz	Single	Klasse = (Physikalische Größe)
Ausgangszustand	Instanz	Multiple	Klasse = (Zustand)
Endzustand	Instanz	Multiple	Klasse = (Zustand)
Wird_ausgeführt_durch	Instanz	Single	Klasse = (System)
Startzeitpunkt	String	Multiple	Klasse = (Zeiteinheit)
Endzeitpunkt	Instanz	Multiple	Klasse = (Zeiteinheit)
Dauer	Instanz	Multiple	Klasse = (Zeiteinheit)
Kosten	Instanz	Single	Klasse = (Monetäre Einheit)
Aktivität	Instanz	Single	Klasse = (Aktivität)

Tabelle 18: Darstellung der Klasse Verhalten mit ihren Slots

Typische Verhaltensweisen lassen sich in die kognitiven, die sozialen, die motorischen und die physikalischen Verhaltensweisen unterteilen.

Die kognitiven Verhaltensweisen besitzen stets als Einfluss einen Informationsfluss, können in ihren Wirkungsmöglichkeiten jedoch sowohl einen Materiefluss als auch einen Informations- oder Energiefluss bewirken. So deutet ein aggressives Suchverhalten auf eine hohe Energieabgabe hin, da sich das Fahrzeug schnell im Raum bewegt. Zudem wird Energie über die Reibung der Räder an den Boden übertragen. Weitere Verhaltensweisen wären ein bestimmtes Lern- oder Explorationsverhalten.

Die sozialen Verhaltensweisen besitzen als Einfluss sowohl einen Informations- als auch einen Materie- und einen Energiefluss. So können Informationen an ein System übertragen werden, ein anderes System angestoßen werden (Energiefluss) oder „böswillig“ Materie an ein System übertragen werden (Pistolenschuß). Alle drei Einflüsse führen zu entsprechenden Wirkungsmöglichkeiten und einem eher aggressivem oder freundlichem sozialen Verhalten.

Die motorischen Verhaltensweisen (Bremsverhalten, Beschleunigungsverhalten, Fahrverhalten etc.) besitzen ebenfalls alle drei Einflussgrößen und Wirkungsmöglichkeiten. So kann eine Information ein bestimmtes Bremsverhalten hervorrufen, das wiederum als Wirkungsmöglichkeit einen Energiefluss erzeugt.

Bei den physikalischen Verhaltensweisen (Spannungsverhalten, Korrosionsverhalten etc.) existiert als Einfluss kein Informationsfluss. Entweder kommt es zu Materieflüssen (z.B. eine ätzende Säure auf Stahl) oder zu Energieflüssen (Strom, Hitze). Diese können die entsprechenden Materie- und Energieflüsse in ihren Wirkungsmöglichkeiten hervorrufen.

Insgesamt zeigt das in dieser Arbeit untersuchte System ein offenes, dynamisches, kontinuierliches, stochastisches und stabiles Verhalten: Offen, da die Zustandsgrößen des Systems mit der Umgebung in Wechselwirkung stehen, dynamisch, da sich die Zustandsgrößen im Laufe der Zeit verändern, kontinuierlich, da sich die Zustandsgrößen kontinuierlich - in beliebig kleinen Zeitabschnitten – verändern, stochastisch, da die Zustandsgrößen bei identischen Wiederholungen nur durch Wahrscheinlichkeiten beschreibbar sind, die in dieser Arbeit duch den Konfidenzfaktor *cf* abgebildet werden (siehe Kapitel 5.3.3) und stabil, da das System aufgrund von Sicherheitsmechanismen bei „normalen“ Änderungen von Parametern nicht „kippt“. In der vorliegenden Arbeit wird vom System das Verhalten verstärkt, das sich als erfolgreich erwiesen hat. Erfolgreich bedeutet hier, dass ein Verhalten zu minimalen Kosten unter Zielerreichung des Systems ausgeführt wurde.

Abschließend
Durch die Spezifizierung sämtlicher Klassen wurde eine allgemein-gültige Ontologie (Basisontologie) erzeugt, über die alle individuellen Ontologie miteinander kommunizieren können. Voraussetzung hierfür ist, dass jede individuelle Ontologie vorher mit ihren Klassen und Slots auf die Basisontologie referenziert wird.

Derzeit existiert noch kein System, das diese Referenzen automatisch erzeugen kann (siehe hierzu Kaptiel 3.2), wodurch diese Arbeit manuell ausgeführt werden muss. Einmal richtig zugewiesen können jedoch beliebig viele individuelle Ontologien mit dem nachfolgenden Algorithmus über die Basisontologie Erfahrungen austauschen.

5.2 Auflösung von Konflikten bei heterogenen Wissensbasen

Das vorliegende Szenario dieser Arbeit stellt sich als ein offenes Transportsystem dar. Dies bedeutet, dass jederzeit Fahrzeuge sowohl in das Transportsystem ein- als auch austreten können. Hierbei ist zu beachten, dass sich die Fahrzeuge hinsichtlich ihrer Wissensbasis unterscheiden. Die gewonnenen Erfahrungen und die daraus resultierenden Verhaltensweisen sind demnach abhängig vom jeweiligen Fahrzeug und müssen deshalb zum Zweck ihrer Kommunikation übersetzt werden. Hierfür wurde der **SCOUT**- (**S**olving **C**onflicts in an **O**ntology **U**sed **T**ranslation) **Algorithmus** entwickelt, der eine Übersetzung individueller Erfahrungen in die jeweilige Wissensbasis anderer Fahrzeuge leistet. Hierzu wird die entwickelte Basisontologie aus Kapitel 5.1 verwendet.

In Kapitel 2.2.1.1 wurde bereits eine Definition einer Basisontologie und einer spezifischen Ontologie aufgestellt. Sie unterscheiden sich im Wesentlichen dadurch, dass die

Basisontologie innerhalb der hier bearbeiteten Domäne übergreifend Verwendung findet, während sich die spezifische Ontologie auf die Strukturierung der Wissensbasis eines bestimmten Fahrzeugs beschränkt. Die Basisontologie nimmt somit die Rolle eines Referenzmodells der Domäne ein und hat die Funktion, eine einheitliche, konsistente Bezugsebene zu bilden, d.h. jeder neue Begriff und jede neue Regel einer spezifischen Ontologie muss mit Begriffen und Regeln aus der Basisontologie definiert werden können. Eine Basisontologie ermöglicht so heterogenen Wissensbasen einen effizienten Datenaustausch und eine individuelle Datenspeicherung.

Neben den unterschiedlichen Terminologien innerhalb der verschiedenen Wissensbasen kommt den unterschiedlichen, gelernten Regeln eine besondere Bedeutung zu. Dabei müssen die Regeln zum Zwecke ihrer Übersetzung im Bereich der Prämissen (siehe Abbildung 65) durch eine für alle Fahrzeuge geltende Basisontologie mit deren Klassen (C^B) und Slots (S^B) ausgedrückt werden. Die Variablen und der Wertebereich können hierbei jedoch individuell definiert werden.

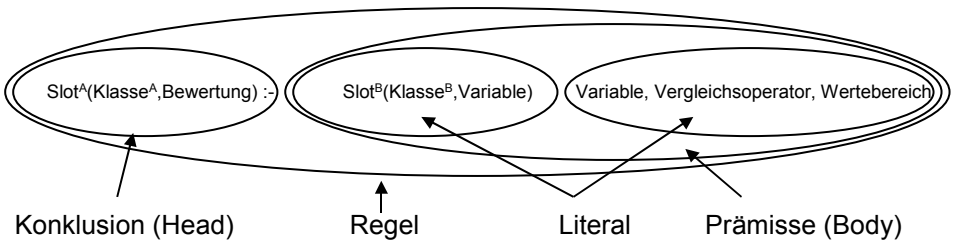


Abbildung 65: Definition individueller Wahrnehmungen mittels einer Basisontologie

Die nachfolgenden Ansätze werden in Prolog[29], einer deklarativen Sprache, mit der symbolisches Wissen verarbeitet werden kann, dargestellt. Mit Prolog kann auf die intentionale Definition von Termen mittels Regeln, Fakten und Abfragen zurückgegriffen werden [Bra00] (bei der *intentionalen Definition* werden meist nach Nennung des nächst höheren Begriffs, der den zu definierenden Begriff beinhaltet, charakteristische Eigenschaften des zu definierenden Begriffs oder Wortes genannt [Bay99]).

5.2.1 Auflösung der Integrationsproblematik

Per Definition geben Ontologien das gemeinsame Verständnis unterschiedlicher Akteure für Begrifflichkeiten wieder. Dies schließt allerdings nicht aus, dass Bewertungen der Begrifflichkeiten unterschiedlich ausfallen (z.B. böse Katze oder gefährliche Katze) bzw. zusätzliche individuelle Begrifflichkeiten verwendet werden (Hund oder Köter). Um diesen Konflikt zu lösen, bedarf es Mechanismen, die unterschiedliche Begrifflichkeiten und Bewertungen über eine Basisontologie eindeutig beschreiben und zuordnen können. Neben der reinen Übersetzung von Begrifflichkeiten (Terminologien) müssen auch Regeln

[29] PROgramming in LOGic, weitere Informationen siehe Kapitel E

übersetzbar sein. Sie bieten erst die Grundlage für den Aufbau eines intelligenten Systems, da Regeln über das reine Faktenwissen hinausgehen und Verhaltensweisen und Ziele eines Fahrzeugs erst erklärbar machen.

Zur Verdeutlichung der Integrationsproblematik sind zwei Szenarien denkbar:

Szenario 1: Übersetzen von Begriffen Das Fahrzeug A möchte von C eine Information über die *Geschwindigkeit* erhalten, die es auf einer bestimmten Strecke zu fahren hat. C findet das Wort *Geschwindigkeit* in seiner Ontologie nicht. Bevor es diese Nachricht jedoch als „false" an A zurück schreibt, fragt es die *Basisontologie*, wo dieser Term einzuordnen ist. Diese Basisontologie identifiziert die *Geschwindigkeit* als *Speed*, indem sie erkennt, dass die Klasse *Geschwindigkeit* der Ontologie A mit der Klasse *Speed* aus der Basisontologie verknüpft ist. Die Basisontologie liefert demnach die Klasse *Speed* als Referenzklasse an C zurück. C ist mit der Klasse *Speed* durch die Klasse *Schnelligkeit* verknüpft. Dadurch weiß C, dass es sich bei dem Wort *Geschwindigkeit* um die *Schnelligkeit* handelt und kann die gewünschten Informationen schicken.

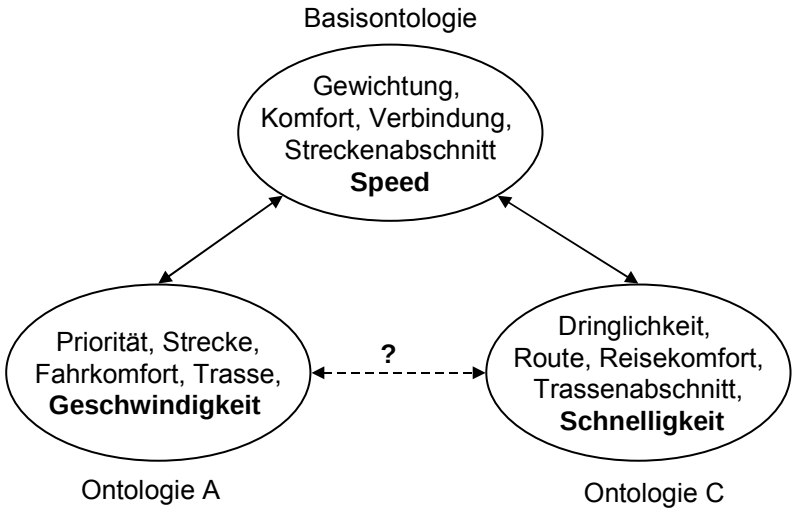


Abbildung 66: Übersetzung von Begriffen zwischen den Ontologien *A* und *C*

Diese recht einfachen Übersetzungs-Funktionen (*uf*) werden explizit durch eine semantische Verknüpfung dargestellt, wobei O^A eine Klasse der Ontologie A, O^B eine Klasse der Basisontologie und O^C eine Klasse der Ontologie C symbolisiert:

$$uf\ (O^A\text{: Geschwindigkeit (X)} \ :\!-\ O^B\text{: Speed (X))}$$
$$uf\ (O^C\text{: Schnelligkeit (X)} \ :\!-\ O^B\text{: Speed (X))}$$

$$\Rightarrow O^A\text{: Geschwindigkeit (X)} = O^C\text{: Schnelligkeit (X)}$$

Szenario 2: Abfrage von Verhaltensweisen A hat keine Vorstellung (keine Regel) von möglichen Verhaltensweisen in einer unbekannten Umweltsituation. Aus diesem Grund fragt er C nach seiner gelernten Regel. Angenommen A sind zwar die Parameter *Art des Transportguts* (Fahrgast), *Priorität des Fahrgastes* (hoch), *Kosten der Beförderung* (5000 Euro) und die *eigene monetäre Situation* (10 Euro) bekannt, jedoch nicht das daraus resultierende Verhalten. Hierzu fragt er C, der die gelernte Regel besitzt:

df (O^C: Auftrag (Fahrzeug,ablehnen) :- O^B: Transportgut (Fahrzeug,K), K=Fahrgast, Priorität (Fahrgast,P), P=hoch,P=mittel, Kosten (Beförderung,C), C>=1000, Geld (Fahrzeug,G), G<=50)

Als Konsequenz lehnt A den Auftrag ab (es sei denn, er entscheidet sich explorativ eigene Erfahrungen zu machen). Weitere Beispiele sind in Tabelle 19 beschrieben.

5.2.2 Grundlagen des SCOUT-Algorithmus

Der in der vorliegenden Arbeit entwickelte Algorithmus stellt ein Verfahren zur Auflösung von Integrationskonflikten dar, welche auf heterogene Wissensbasen zurückzuführen sind. Dadurch ist es einem Fahrzeug, auch in einer für ihm fremden Umwelt, möglich, Erfahrungen mit anderen Fahrzeugen auszutauschen und somit z.B. einen optimalen Weg oder eine optimale Komforteinstellung zu finden.

Die Mächtigkeit des entwickelten Algorithmus wird insgesamt durch sechs Bedingungen beschrieben:

1. **Vergleich von Begriffen (Klassen) und Regeln:** Neben der reinen Übersetzung von Begriffen (Klassen) ermöglicht der Algorithmus einen Vergleich von Regeln (Verhaltensweisen).

2. **Berücksichtigung der Werte:** Es ist eine Analyse von numerischen und nicht-numerischen (symbolischen) Werten möglich. Hierbei können Slots auch negative Werte zugewiesen werden (z.B. kann der Wert der Temperatur auch kleiner 0 sein).

3. **Funktionen:** Der Algorithmus erlaubt eine Untersuchung von einfach ineinander verschachtelten Literalen - wie bei der Benutzung von Funktionen. Unter einer Funktion wird in diesem Zusammenhang „minus, plus, multipliziert, dividiert“ verstanden. Ein solches Literal wäre beispielsweise:

   minus(Einnahmen(Fahrzeug, E), Ausgaben(Fahrzeug, A), Z)

4. **Vollständigkeit:** Es erfolgt nur eine Übersetzung, wenn sämtliche Slots im Prämissen-Bereich einer Regel von A vollständig durch gleichnamige Slots von C

abgedeckt werden. Hierbei müssen auch die zu den Slots gehörenden Klassen übereinstimmen (siehe Tabelle 19, Beispiel 1, 2).

5. **Übersetzung:** Sofern zu einer Regel von A keine 1-1-Übersetzung zu einer Regel von C erzeugt werden kann, ist eine Verkettung von mehreren Regeln von C herzustellen. Hierbei wird entsprechend der Werte zwischen einer eindeutigen[30] und einer partiellen[31] Übersetzung unterschieden. Bei einer existenten 1-1-Übersetzung darf die Regel von C im Prämissen-Bereich mehr Literale enthalten als die von A. (siehe Tabelle 19, Beispiel 3 - 6).

6. **Gegenseitiger Ausschluss:** Es existiert eine in jeder Situation sichere Übersetzung, d.h., dass bei den partiellen Übersetzungen keine disjunkten Wertebereiche bei den gemeinsamen Slots auftreten dürfen. Des Weiteren wird von vornherein vermieden, dass Regeln mit aufgenommen werden, die nicht feuern können. (siehe Tabelle 19, Beispiel 7, 8).

Tabelle 19 verdeutlicht anhand von beispielhaften Übersetzungen die oben aufgeführten Bedingungen. Dabei beziehen sich die Beispiele 1, 2, 3, 5 und 6 auf die Übersetzung von Begriffen (Klassen) und die Beispiele 4, 7 und 8 auf die Übersetzung von Regeln (Verhaltensweisen).

Beispiel 1: 1 – 1 – Übersetzung *df (O^A: A() :- O^B: a(B), b(B), c(B))* *df (O^C: C() :- O^B: a(B), b(B), c(B))* **Übersetzung**: A() = C(), da Bedingung 4 erfüllt ist.	Beispiel 2: 1 – 1 – Übersetzung *df (O^A: A() :- O^B: a(B), b(B), c(B))* *df (O^C: C_1() :- O^B: a(D), b(B), c(B))* *df (O^C: C_2() :- O^B: a(B), b(B))* **Übersetzung**: keine Übersetzung, da Bedingung 4 verletzt ist.
Beispiel 3: Eindeutige Übersetzung *df (O^A: A() :- O^B: a(B), b(B), c(B))* *df (O^C: C_1() :- O^B: a(B), b(B))* *df (O^C: C_2() :- O^B: c(B))* **Übersetzung**: A() = $C_1() \wedge C_2()$, da Bedingung 5 erfüllt ist.	Beispiel 4: Partielle Übersetzung *df (O^A: A() :- O^B: a(B), b(B), c(B, w), w>2, w<8)* *df (O^C: C_1() :- O^B: a(B), b(B), c(B, w), w>0, w<6)* *df (O^C: C_2() :- O^B: a(B), b(B), c(B, w), w>4, w<9)* **Übersetzung**: A() = $(C_1() \vee C_2())$, da Bedingung 5 erfüllt ist, aber die Wertebereiche von C sich mit denen von A überlappen. Eine eindeutige „und“ Übersetzung ist somit nicht möglich.
Beispiel 5: 1 – 1 – Übersetzung *df (O^A: A() :- O^B: a(B), b(B), c(B))* *df (O^C: C_1() :- O^B: a(B), b(B), c(B), d(B))* *df (O^C: C_2() :- O^B: a(B), b(B))* *df (O^C: C_3() :- O^B: c(B))*	Beispiel 6: Eindeutige Übersetzung *df (O^A: A() :- O^B: a(B), b(B), c(B))* *df (O^C: C_1() :- O^B: a(B), b(B))* *df (O^C: C_2() :- O^B: c(B), d(B))*

[30] Eindeutige Übersetzung: Übereinstimmung in Slot und Klasse, der Wertebereich von Ontologie C ist immer mindestens so groß wie der von Ontologie A.

[31] Partielle Übersetzung: Übereinstimmung in Slot und Klasse, der Wertebereich von Ontologie C besitzt zumindest immer eine Überschneidung mit dem von Ontologie A. Kennzeichnend hierfür sind die um die Regeln von Ontologie C befindlichen Klammern. D.h., dass es von den tatsächlich auftretenden Werten abhängt, ob bzw. welche Regeln feuern (siehe Tabelle 19 Beispiel 4 und 8).

Übersetzung: A() = C_1(), da Bedingung 5 für C_1 erfüllt ist und diese eindeutig ist, so dass C_2 und C_3 nicht zum Übersetzen herangezogen werden (obwohl dies beim Fehlen von C_1 der Fall wäre (siehe Beispiel 3)).	**Übersetzung**: keine Übersetzung, da Bedingung 4 verletzt ist.
Beispiel 7: Gegenseitiger Ausschluss *df (O^A: A() :- O^B: a(B, w),w>2,w<8,b(B),c(B))* *df (O^C: C_1() :- O^B: a(B, w),w>0,w<5,b(B))* *df (O^C: C_2() :- O^B: a(B, w),w>5,w<9,c(B))* **Übersetzung**: keine Übersetzung, da Bedingung 6 nicht erfüllt ist.	Beispiel 8: Gegenseitiger Ausschluss *df (O^A: A() :- O^B: a(B, w), w>2, w<8, b(B), c(B))* *df (O^C: C_1() :- O^B: a(B, w), w>4, w<8, b(B))* *df (O^C: C_2() :- O^B: a(B, w), w>3, w<7, c(B))* *df (O^C: C_3() :- O^B: a(B, w), w>0, w<2)* **Übersetzung**: A() = (C_1() $\vee$ C_2()), da Bedingung 6 erfüllt ist.

Tabelle 19: Beispielhafte Übersetzungen

Im Algorithmus werden zum Vergleich der einzelnen Ontologien die *Prämissen* der Regeln von A und C in mehrere 6-Tupel $T:=\{S, K, O^{min}, W^{min}, O^{max}, W^{max}\}$ zergliedert. Hierbei ist:

- S= Slot der Basisontologie
- K= Klasse der Basisontologie
- O^{min}= Operator des minimalen Wertes
- W^{min}= minimaler Wert
- O^{max}= Operator des maximalen Wertes
- W^{max}= maximaler Wert

Die separierten 6-er Tupel von A und C werden im Algorithmus schrittweise miteinander verglichen:

Schritt 1: In der ersten Iteration des Algorithmus werden von der ersten Regel von Ontologie A die zugehörigen Literale analysiert. Dies erfolgt sowohl hinsichtlich ihrer Beziehung zueinander (z.B. gehören Temperatur(Luft, T), T>-5, T<15 zusammen) als auch hinsichtlich ihrer Ausprägungen. So wird aus jeder dieser Konstellationen ein 6er-Tupel zum späteren Vergleich erzeugt (hier S=Temperatur, K=Luft, W^{min}=-5, O^{min}=>, W^{max}=15, O^{min}=<).

Schritt 2: Anschließend werden von der ersten Regel von Ontologie C die ersten zusammengehörigen Literale analysiert. Dies erfolgt analog zum ersten Schritt (z.B.: bei Temperatur(Luft,T), T>0, T<10 wären dies: S=Temperatur, K=Luft, W^{min}=0, O^{min}=>, W^{max}=10, O^{min}=<).

Schritt 3: Der Algorithmus vergleicht das in Schritt 2 generierte 6er-Tupel sukzessive mit denen von Ontologie A (Schritt 1) anhand des Slots und der Klasse.

Schritt 4: Sofern im vorherigen Schritt eine Übereinstimmung festgestellt wird, erfolgt eine Analyse der Werte (W^{min}, O^{min}, W^{max}, O^{max}).

Schritt 5: Abhängig vom Ergebnis aus Schritt 4 erfolgt entweder die weitere Analyse der aktuellen Regel und ggf. eine Aufnahme in die Menge der gefundenen Übersetzungen oder die Betrachtung der nächsten Regel von Ontologie C.

Schritt 6: Nach der Untersuchung der letzten Regel von Ontologie C erfolgt die Vermeidung des gegenseitigen Ausschlusses und eine Analyse auf eine bestehende 1-1-Übersetzung.

5.2.3 Umsetzung des SCOUT-Algorithmus

Nachdem der SCOUT-Algorithmus grob skizziert wurde, erfolgt in diesem Kapitel eine detailliertere Betrachtung anhand des Pseudo-Codes. Aufgrund der Komplexität des Verfahrens wird bewusst von einem einzigen durchgängigen Beispiel abgesehen. Stattdessen wird an entsprechender Stelle der Ablauf anhand einzelner, voneinander unabhängiger Beispiele illustriert. Der Algorithmus besteht aus einem Hauptalgorithmus sowie sieben Teilalgorithmen:

- ***Hauptalgorithmus***: Er übernimmt die Kontrolle über den Ablauf. Aus ihm lassen sich bei Bedarf die entsprechenden Teilalgorithmen aufrufen. Des Weiteren wird in ihm die Ausgabe verwaltet.

- ***Separation***: Dieser Teilalgorithmus extrahiert aus den Informationsträgern die Klassen und die Slots und bereitet darüber hinaus die Informationen zur Analyse der enthaltenen Werte auf. Zudem wird die Betrachtung von Funktionen ermöglicht.

- ***Detaillierung***: Dieser Teilalgorithmus vollzieht die Transformation der Werteangaben in analysierbare Größen.

- ***Prädikatanalyse***: Dieser Teilalgorithmus untersucht eine Übereinstimmung der Slots und der Klassen der Ontologien A und C.

- ***Vertauschung***: Beim Auftreten der Funktionen „dividiert“ und „minus“ ist zur Analyse der Werte die Reihenfolge innerhalb des geschachtelten Literals entscheidend. Diese wird bei Bedarf entsprechend dem Aufbau der Ontologie A für die Ontologie C umgestellt.

- ***Intervallanalyse***: Bei diesem Teilalgorithmus werden nach einem erfolgreichen Slot- und Klassenvergleich die Werte auf eine Überschneidung der Wertebereiche hin analysiert. Nur wenn mindestens eine Überschneidung existiert, kann überhaupt eine Übersetzung erfolgen.

- ***Intervallverhältnis***: Dieser Teilalgorithmus untersucht die Art der Überschneidung der Wertebereiche. Sofern der Wertebereich von Ontologie C mindestens den von

Ontologie A umfasst, liegt eine eindeutige Übersetzung vor, ansonsten eine partielle Übersetzung.

- ***Validierung***: Dieser Teilalgorithmus dient zur Vermeidung des gegenseitigen Ausschlusses, d.h., dass dieser Teilalgorithmus vorrangig auf Basis der partiellen Übersetzungen operiert. Zudem untersucht er die gefundenen Regeln von Ontologie C auf eine 1-1-Übersetzung und eliminiert partielle Übersetzungen.

Grundlage des Algorithmus (die genaue Beschreibung der Variablen des SCOUT-Algorithmus ist im Anhang unter Kapitel B dargestellt) sind zwei Mengen der Regeln M^A und M^C. Diese bestehen aus den jeweiligen Beziehungen zwischen den individuellen Ontologien von A bzw. C zu der Basisontologie. Ziel des Verfahrens ist es, zu jedem a_k (Konklusion einer Regel aus Ontologie A) aus M^A, sofern vorhanden, ein (bzw. mehrere) c_i*(´s)* (Konklusion einer Regeln aus Ontologie C) aus M^C zu finden, so dass am Ende die Übersetzung gilt: $a_1 = c_2 \wedge c_3$; $a_2 = (c_1)$ etc.

Es sei:

a_k :- $m^A_{k,l}$ *(mit k = 1…n und l = 1…m)*
c_i :- $m^C_{i,j}$ *(mit i=1…s und j=1…t)*

$$M^A = \sum_{k=1}^{n}\sum_{l=1}^{m} [\, a_k :- m^A_{k,l} \,]$$

$$M^C = \sum_{i=1}^{s}\sum_{j=1}^{t} [\, c_i :- m^C_{i,j} \,]$$

Initialisierungen:

d = 1
e = 1
i = 1
id = 1
j = 1
ljInkrement = 0
k = 1
l = 1
p = 1
r = 1

In jeder Iteration des Algorithmus wird eine Regel aus der Regelmenge M^A analysiert. Hierzu werden zusammengehörige Literale gesucht und zur weiteren Auflösung mit einer eindeutigen Kennzeichnung an die Teilalgorithmen SEPARATION und DETAILLIERUNG übergeben. Als zusammengehörig gelten hierbei zwei bzw. drei Literale, wobei nur in einem Literal eine Klasse der Basisontologie vorhanden sein darf. Diese wird hinsichtlich. ihrer

Werte durch die nachfolgenden Literale konkretisiert (z.B. sei in diesem Beispiel M^A: Bedingungen (Wetter, schlecht) :- Temperatur(Luft, T), T>0, T<15, wobei a_k = Bedingungen (Wetter, schlecht) und $m_{k,l}$ = Temperatur(Luft, T), T>0, T<15 ist). Grundlage eines jeden Vergleichs bildet die Zuweisung der Variablen zum 6er-Tupel. Hierbei ist:

- Slot der Basisontologie – $prädikat^{typ}_{id}$,
- Klasse der Basisontologie – $basisOnt^{typ}_{id}$,
- minimaler Wert – wg^{typ}_{id},
- Operator des minimalen Wertes – tg^{typ}_{id},
- maximaler Wert – wk^{typ}_{id}
- Operator des maximalen Wertes – tk^{typ}_{id}.

```
1.  FOR M^A DO
2.    WHILE a_k ≠ Ø
3.      prädikatAnzA = 0
4.      WHILE m_k,l ≠ Ø
5.          ljInkrement = 0
6.          detaillierungsliteral = m_k,l
7.          prädikatAnzA = prädikatAnzA + 1
8.          IF (∈ Vergleichsoperator) ∈ m_k,l+1
9.              detaillierungsliteral = detaillierungsliteral ∧ m_k,l+1
10.             ljInkrement = ljInkrement + 1
11.             IF (∈ Vergleichsoperator) ∈ m_k,l+2
12.                 detaillierungsliteral = detaillierungsliteral ∧ m_k,l+2
13.                 ljInkrement = ljInkrement + 1
14.         SEPARATION(A, id, detaillierungsliteral)
15.         prädikatVereinigungA = prädikatVereinigungA ∧ prädikat^typ_id
16.         id = id + 1
17.         l = l + ljInkrement + 1
```

Das Ergebnis des ersten Teils ist die Umwandlung der Operatoren „<“ und „>“ in Zahlen. Statt „T>0“ wird jetzt „T, 2, 0“ im Algorithmus gespeichert. Dies unterstützt die Vergleichbarkeit der Wertebereiche verschiedener Regeln. Zudem geschieht die Zuweisung des Slots *Temperatur* zu $prädikat^A_1$ sowie der Klasse *Wetter* zu $basisOnt^A_1$. Dies ist für den späteren Vergleich der Regeln von Ontologie A zu denen der Ontologie C wichtig.

Nach der Analyse aller Literale einer Regel von M^A wird von M^C die erste Regel betrachtet (z.B. sei hier M^C: Zustand (Wetter, schlecht) :- Temperatur(Luft, T), T>-10, T<5). Auch hier erfolgt zuerst die Suche nach zusammengehörigen Literalen, welche nach demselben Verfahren wie bei A an die Teilalgorithmen SEPARATION und DETAILLIERUNG zur Bildung eines 6er-Tupels übergeben werden.

```
18.   FOR M^C DO
19.       WHILE c_i ≠ Ø
20.           aufnahmeArt = AND
21.           prädikatAnzB = 0
22.           bodyIdentität = TRUE
23.           WHILE m_i,j ≠ Ø
24.               ljInkrement = 0
25.               detaillierungsliteral = m_i,j
26.               IF (∈ Vergleichsoperator) ∈ m_i,j+1
```

```
27.             detaillierungsliteral = detaillierungsliteral ∧ m_{i,j+1}
28.             ljInkrement = ljInkrement + 1
29.             IF (∈ Vergleichsoperator) ∈ m_{i,j+2}
30.                 detaillierungsliteral = detaillierungsliteral ∧ m_{i,j+2}
31.                 ljInkrement = ljInkrement + 1
32.         SEPARATION(C, 1, detaillierungsliteral)
```

Im Anschluss wird mit Hilfe des Teilalgorithmus PRÄDIKATANALYSE nach einer Übereinstimmung der Slots und der Klassen gesucht. Dazu werden sukzessiv, aber nur bis zu einer erfolgreichen Untersuchung, die 6er-Tupel der Regel von M^A genutzt.

```
33.         id = 1
34.         treffer = FALSE
35.         WHILE (treffer = FALSE ∧ prädikat^A_id ≠ Ø)
36.             treffer = PRÄDIKATANALYSE(id)
37.             id = id + 1
```

Im Falle einer erfolgreichen Suche erfolgt nun die Betrachtung der Intervalle. Hierzu werden im ersten Schritt die Wertebereiche hinsichtlich einer Überschneidung analysiert. Wenn eine solche Überschneidung festgestellt werden kann, gilt es deren Art zu ermittlen und der Slot der Regel von M^C festzuhalten. Ansonsten wird die Variable *depot* geleert und mit j = 0 die aktuelle Regelmenge von M^C verlassen.

Konnte keine Übereinstimmung zwischen den Slots und den Klassen festgestellt werden, erfolgt die Analyse des nächsten 6-Tupels der aktuellen Regel von M^C. Zusätzlich wird die Variable *bodyIdentität* auf „FALSE“ gesetzt. Dadurch werden partielle Übersetzungen mit nicht in beiden Regeln vorhandenen Slots vermieden (Siehe Tabelle 19, Beispiel 6).

```
38.         IF treffer = TRUE
39.             id = id - 1
40.             wertIdentität = INTERVALLANALYSE(id)
41.             IF wertIdentität = TRUE
42.                 prädikatAnzB = prädikatAnzB + 1
43.                 IF aufnahmeArt ≠ OR
44.                     aufnahmeArt = INTERVALLVERHÄLTNIS(id)
45.                 depot= depot ∧ prädikat^C_1
46.                 j = j + ljInkrement + 1
47.             ELSE
48.                 depot = Ø
49.                 j = -1
50.                 prädikatAnzB = 0
51.         ELSE
52.             bodyIdentität = FALSE
53.             j = j + ljInkrement + 1
```

Nachdem alle 6er-Tupel einer Regel von M^C betrachtet wurden, erfolgt eine Überprüfung der Größe dieser Regel. Eine Regel von M^C darf nur dann mehr Slots als eine Regel von M^A umfassen, wenn letztere hinsichtlich ihrer Slots vollständig in der Regel von M^C enthalten ist (siehe Tabelle 19, Beispiel 5). Ansonsten wird das *depot* geleert.

Sofern das *depot* nicht leer ist, werden die darin enthaltenen Slots und die Konklusion der Regel entsprechend der ermittelten Aufnahmeart den für die Ausgabe und Validierung notwendigen Variablen zugewiesen. Im Anschluss daran erfolgt die Analyse der nächsten Regel von M^C.

```
54.     IF prädikatAnzB ≠ prädikatAnzA ∧ bodyIdentität = FALSE
55.         depot = Ø
56.     IF depot ≠ Ø
57.         bodyGröße^P = prädikatAnzB
58.         headCollection^P = c_i
59.         prädikatCollection^P = depot
60.         p = p + 1
61.         IF aufnahmeArt = AND
62.             ausgabeAND^d = c_i
63.             prädikatmengeAND^d = depot
64.             d = d + 1
65.         ESLE
66.             ausgabeOR^e = c_i
67.             prädikatmengeOR^e = depot
68.             e = e + 1
69.     i = i + 1
70.     j = 1
71.     depot = Ø
```

Nach der Analyse aller Regeln von M^C erfolgt mit der VALIDIERUNG eine Routine zur Vermeidung des gegenseitigen Ausschlusses und hinsichtlich einer existenten 1-1-Übersetzung.

Sind nach dieser Prüfung alle Slots der Regel von M^A in der Menge der Slots einer (oder mehrerer) Regel(n) von M^C enthalten, so ist eine zulässige Übersetzung gefunden, die mit der Anweisung RETURN(M^T) ausgegeben wird.

```
72.  VALIDIERUNG()
73.  prädikatVereinigungB = prädikatmengeAND ∧ prädikatmengeOR
74.  IF prädikatVereinigungA ⊆ prädikatVereinigungB
75.  ausgabe = ausgabeAND^1 ∧ ... ∧ ausgabeAND^d ∧ (ausgabeOR^1 ∨ ∨ ausgabeOR^e )
76.      M^T = M^T ∧ (a_k = ausgabe)
77.  prädikatVereinigungA = prädikatVereinigungB = ... = Ø
78.  p = ... = 1
79.  k = k + 1
80. RETURN(M^T)
```

SEPARATIONS-Algorithmus

Beim Aufruf des Teilalgorithmus SEPARATION werden zuerst Default-Werte gesetzt. Hierbei wird zunächst das größtmögliche Intervall zugrunde gelegt, d.h., dass sich dieses von „>-∞“ bis „< +∞“ erstreckt. Darüber hinaus gilt es, aus den übergebenen Literalen den Slot und die Klasse zu ermitteln.

```
81. SEPARATION(typ, id, detaillierungsliteral)
82. wg^typ_id = -∞, tg^typ_id = 2, tk^typ_id = -1, wk^typ_id = ∞
83. prädikat^typ_id = Slot
84. basisOnt^typ_id = Klasse
```

85. m_r = *r-te Literal vom detaillierungsliteral*
86. *IF* $m_{r+1} = \emptyset$
87. *DETAILLIERUNG(typ, id,* m_r*)*
88. *ESLE*
89. *DETAILLIERUNG(typ, id,* m_{r+1}*)*
90. *IF* $m_{r+2} \neq \emptyset$
91. *DETAILLIERUNG(typ, id,* m_{r+2}*)*
92. *IF* $prädikat^{typ}_{id} \in$ *Funktion*
93. *prädikatA = erster innerer Slot in Funktion*
94. *basisOntA = Klasse vom ersten inneren Slot*
95. *prädikatB = zweiter innerer Slot in Funktion*
96. *basisOntB = Klasse vom zweiten inneren Slot*
97. *IF typ = A*
98. $prädikat^{typ}_{id} = prädikat^{typ}_{id}$*/ prädikatA/ basisOntA/ prädikatB/ basisOntB*
99. *IF typ = C*
100. *prädikatAlternativeA =* $prädikat^{typ}_{id}$*/ prädikatA/ basisOntA/ prädikatB/ basisOntB*
101. *prädikatAlternativeB =* $prädikat^{typ}_{id}$*/ prädikatB/ basisOntB/ prädikatA/ basisOntA*

Sofern die Variable *detaillierungsliteral* aus mehr als einem Literal besteht, erfolgt mittels des Teilalgorithmus DETAILLIERUNG eine Beschränkung der Intervallgrenzen. Die Aufgaben des Teilalgorithmus DETAILLIERUNG sind die Transformation eines Vergleichsoperators in einen numerischen Wert und die Begrenzung des initialen Intervalls.

102. *DETAILLIERUNG(typ, id, literal)*
103. *operator = Vergleichsoperator aus literal*
104. *ausprägung = Wert aus literal*
105. *IF operator = "<="*
106. $tk^{typ}_{id} = -4$, $wk^{typ}_{id} = ausprägung$
107. *IF operator = "<"*
108. $tk^{typ}_{id} = -1$, $wk^{typ}_{id} = ausprägung$
109. *IF operator = "="*
110. $tg^{typ}_{id} = 10$, $tk^{typ}_{id} = -10$, $wg^{typ}_{id} = wk^{typ}_{id} = ausprägung$
111. *IF operator = ">"*
112. $tg^{typ}_{id} = 2$, $wg^{typ}_{id} = ausprägung$
113. *IF operator = ">="*
114. $tg^{typ}_{id} = 3$, $wg^{typ}_{id} = ausprägung$

Im zweiten Teil des SEPARATIONS-Algorithmus (ab Zeile 92) erfolgt eine Untersuchung von einfach ineinander verschachtelten Literalen, z.B. „minus(Einnahmen(Fahrzeug, E), Ausgaben(Fahrzeug, A), Z)". Hierbei muss berücksichtigt werden, dass trotz einer unterschiedlichen Reihenfolge der inneren Klassen und Slots eine Gleichheit vorliegen kann. So sind beispielsweise „minus(Einnahmen(Fahrzeug,E), Ausgaben(Fahrzeug,A),Z), Z>=10, Z<20" und „minus(Ausgaben(Fahrzeug,A), Einnahmen(Fahrzeug,E), Z), Z<=-10, Z>-20" äquivalent.

PRÄDIKATENANALYSE

Die PRÄDIKATANALYSE untersucht die 6er-Tupel hinsichtlich einer Äquivalenz zwischen den Slots und den Klassen. Wenn eine Übereinstimmung unter Beteiligung der Funktion minus oder dividiert aufgetreten ist und eine unterschiedliche Reihenfolge vorliegt, muss dies auch in dem 6er-Tupel von M^C Berücksichtigung finden. In diesem Fall wird der Teilalgorithmus VERTAUSCHUNG angestoßen.

```
115. PRÄDIKATANALYSE(id)
116. IF prädikat^C_1 ∉ Funktion
117.    IF (prädikat^A_id = prädikat^C_1 ∧ basisOnt^A_id = basisOnt^C_1)
118.          return TRUE
119.    ESLE
120.          return FALSE
121.  ELSE
122.       IF   prädikat^A_id = prädikatAlternativeA
123.          prädikat^C_1 = prädikatAlternativeA
124.          return TRUE
125.       IF  prädikat^A_id = prädikatAlternativeB
126.          IF prädikat^C_1 ∈ {minus, dividiert}
127.               VERTAUSCHUNG()
128.          prädikat^C_1 = prädikatAlternativeA
129.          return TRUE
130.       ELSE
131. return FALSE
```

Abhängig von der *Funktion* werden für die Werte zunächst entweder das Produkt mit (-1) oder der Kehrwert gebildet. Danach erfolgt der Tausch dieser Größen und eine Neubelegung der Vergleichsoperatorausprägungen. Dadurch wird aus „c_i :- minus(Ausgaben(Fahrzeug,A), Einnahmen(Fahrzeug,E), Z), Z<=-10, Z>-20“ ein „c_i :- minus(Einnahmen(Fahrzeug,E), Ausgaben(Fahrzeug,A),Z), Z>=10, Z<20“ und kann nun mit „a_k :- minus(Einnahmen(Fahrzeug,E), Ausgaben(Fahrzeug,A),Z), Z>=9, Z<25“ verglichen werden.

```
132. VERTAUSCHUNG()
133. tempA = wg^C_1
134. tempB = wk^C_1
135. if prädikat^C_1 = minus
136.    wg^C_1 = tempB * (-1)
137.    wk^C_1 = tempA * (-1)
138. if prädikat^C_1 = dividiert
139.    wg^C_1 = (tempB)^-1
140.    wk^C_1 = (tempA)^-1
141. tempA = tg^C_1
142. tg^C_1 = tk^C_1
143. tk^C_1 = tempA
144. IF tg^C_1 = -4
145.    tg^C_1 = 3
146.  IF tg^C_1 = -1
147.    tg^C_1 = 2
148. ELSE
149.    tg^C_1 = 10
150. IF tk^C_1 = 2
151.    tk^C_1 = -1
152. IF tk^C_1 = 3
153.    tk^C_1 = -4
154. ELSE
155.    tk^C_1 = -10
```

INTERVALLANALYSE

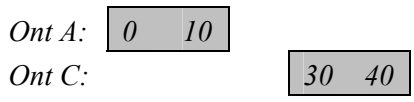
Eine erfolgreiche Übersetzung zwischen zwei Regeln kann nur dann erfolgen, wenn mindestens eine Überschneidung der Wertebereiche existiert. Bei dieser Überprüfung wird zwischen nicht-numerischen und numerischen Werten unterschieden.

Bei den nicht-numerischen, symbolischen Werten wird genau dann eine Überschneidung festgestellt, wenn alle vier Werte (Slot, Klasse, Variable, Wertebereich) identisch sind. Eine Betrachtung der Vergleichsoperatoren ist nicht erforderlich.

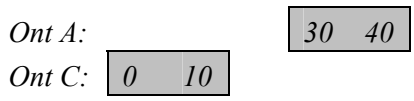
Beispiel 1: *Durch die Darstellung Ontologie A: Hersteller(Fahrzeug, Siemens) und Ontologie C: Hersteller(Fahrzeug, H), H = Siemens wird eine Gleichheit diagnostiziert.*

Im Gegensatz dazu müssen bei den numerischen Werten auch die Vergleichsoperatoren berücksichtigt werden. Zur Ermittlung einer Überschneidung werden hierzu jeweils unterschiedliche Differenzen zwischen den Werten und den, in Zahlen transformierten Vergleichsoperatoren gebildet.

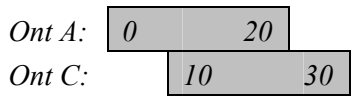
Beispiel 2: *Es sei Ontologie A: Temperatur(Luft, T), T>0, T<10 und Ontologie C: Temperatur(Luft, T), T>30, T<40. Zwischen den einzelnen Wertebereichen existiert keine Überschneidung, d.h. dass die INTERVALLANALYSE als Ausgabe „FALSE“ zurückgibt. Somit ist keine Übersetzung möglich.*



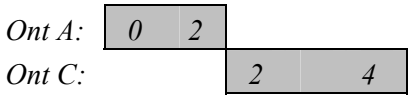
Beispiel 3: *Es sei Ontologie A: Temperatur(Luft, T), T>30, T<40 und Ontologie C: Temperatur(Luft, T), T>0, T<10. Zwischen den einzelnen Wertebereichen existiert keine Überschneidung, d.h. dass die INTERVALLANALYSE als Ausgabe „FALSE“ zurückgibt. Somit ist keine Übersetzung möglich.*



Beispiel 4: *Es sei Ontologie A: Temperatur(Luft, T), T>0, T<20 und Ontologie C: Temperatur(Luft, T), T>10, T<30. Zwischen dem Wertebereich von Ontologie A und Ontologie C existiert eine Überschneidung. Aufgrund dessen gibt die INTERVALLANALYSE „TRUE“ zurück. Eine Übersetzung ist möglich.*



Beispiel 5: *Es sei Ontologie A: Temperatur(Luft, T), T>=0, T<2 und Ontologie C: Temperatur(Luft, T), T>=2, T<5. Zwischen den einzelnen Wertebereichen existiert keine Überschneidung, d.h. dass die INTERVALLANALYSE als Ausgabe „FALSE“ zurückgibt. Somit ist keine Übersetzung möglich.*



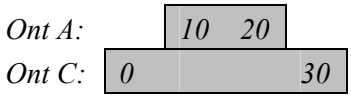

```
156. INTERVALLANALYSE(id)
157. IF ((wg^A_id = wk^A_id) ∧ (wg^C_1 = wk^C_1))
158.    IF wg^A_id = wg^C_1
159.          return TRUE
160.    ELSE
161.          return FALSE
162. ELSE
163.    diffGK = wg^A_id – wk^C_1
164.    diffKG = wk^A_id – wg^C_1
165.    ausrGK = tg^A_id – tk^C_1
166.    ausrKG = tk^A_id – tg^C_1
167.    IF ((diffGK > 0 ∨ (diffGK = 0 ∧ (ausrGK ≠ 7 ∨ ausrGK <= 12))) ∨
            (diffKG < 0 ∨ (diffKG = 0 ∧ (ausrKG ≠ -7 ∨ ausrKG >= -12))))
168.          return FALSE
169.    ELSE
170.          return TRUE
```

INTERVALLVERHÄLTNIS

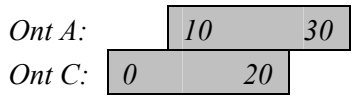
Während mit Hilfe der INTERVALLANALYSE die Existenz einer Überschneidung nachgewiesen werden kann, erfolgt mit dem Teilalgorithmus INTERVALLVERHÄLTNIS die Analyse hinsichtlich der Art („AND“ oder „OR“). Stehen alle Literale zweier Regeln zueinander in einem „AND“-Verhältnis, so existiert eine eindeutige Übersetzung, ansonsten eine partielle Übersetzung.

Wie bei der INTERVALLANALYSE wird bei der Untersuchung zwischen nicht-numerischen und numerischen Werten differenziert. Beim Auftreten von nicht-numerischen Werten wird als Ausgabe ein „AND“ zurückgegeben, bei numerischen Werten werden unterschiedliche Differenzen zwischen den Werten und den, in Zahlen transformierten Vergleichsoperatoren gebildet:

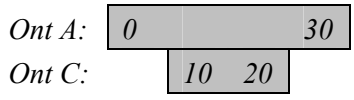
Beispiel 6: *Es sei Ontologie A: Temperatur(Luft, T), T>=10, T<=20 und Ontologie C: Temperatur(Luft, T), T>=0, T<=30. Das Intervall von Ontologie C umfasst vollständig das von Ontologie A. Insofern gibt der Teilalgorithmus als Ausgabe „AND“ zurück.*



Beispiel 7: *Es sei Ontologie A: Temperatur(Luft, T), T>=10, T<=30 und Ontologie C: Temperatur(Luft, T), T>=0, T<=20. Das obere Intervall wird nicht vollständig durch das untere Intervall abgedeckt, d.h. dass eine „OR"-Aufnahme vorliegt.*



Beispiel 8: *Es sei Ontologie A: Temperatur(Luft, T), T>=10, T<=30 und Ontologie C: Temperatur(Luft, T), T>=10, T<=20. In diesem Beispiel liegt eine „OR"-Aufnahme vor. Nur wenn die tatsächlich auftretenden Werte zwischen 10 und 20 liegen kann die Regel feuern.*



171. INTERVALLVERHÄLTNIS(id)
172. IF $wg^A_{id} = wk^A_{id} = wg^C_1 = wk^C_1$
173. return AND
174. ELSE
175. $diffGG = wg^A_{id} - wg^C_1$
176. $diffKK = wk^A_{id} - wk^C_1$
177. $ausrGG = tg^A_{id} - tg^C_1$
178. $ausrKK = tk^A_{id} - tk^C_1$
179. $IF((diffGG>0 \vee (diffGG=0 \wedge (ausrGG=-1 \vee ausrGG=0 \vee ausrGG=7))) \wedge (diffKK<0 \vee (diffKK=0 \wedge (ausrKK=-6 \vee ausrKK=0 \vee ausrKK=3))))$
180. return AND
181. ESLE
182. return OR

VALIDIERUNG

In dem Teilalgorithmus VALIDIERUNG erfolgt abschließend eine Überprüfung hinsichtlich einer existenten 1-1-Übersetzung. Dies hat aufgrund des inhärenten Prioritätensystems eine höhere Relevanz als eine Übersetzung mit verketteten Regeln (siehe Tabelle 19 Beispiel 5). Wenn ein solches Prioritätssystem vorhanden ist, werden alle anderen Konklusionen aus den Mengen *ausgabeAND*, bzw. *ausgabeOR* und alle Slots aus *prädikatmengeAND*, bzw. *prädikatmengeOR* gelöscht.

Beispiel 9: *Es sei:*

O^A: Bedingungen (Wetter, schlecht):- O^B: Temperatur(Luft, T), T>=-15, T<15, Menge(Regen, N), N>=0, Geschwindigkeit(Wind, W), W>=0

O^C_1: Zustand(Wetter, kalt) :- O^B: Temperatur(Luft, T), T>-30, T<10, Menge(Regen, N), N>=0, Geschwindigkeit(Wind, W), W>=0

O^C_2: Zustand (Wetter, schön):- O^B: Temperatur(Luft, T), T>=15, T<30, Menge (Regen, N), N=0

Ohne die VALIDIERUNG wird eine Regelverkettung nach dem Muster $A()= (C_1() \wedge C_2())$ erzeugt. Da jedoch die Regel C_1 bereits komplett die Regel A darstellt, wird die Regel C_2 wieder aus der Lösungsmenge gelöscht. Als Ausgabe erfolgt $A()=(C_1())$. Die Klammer um C_1 bleibt erhalten, da nicht in allen Fällen eine eindeutige Übersetzung existiert (z.B. nicht bei T=14).

Im Anschluss daran erfolgt eine Überprüfung der Vermeidung des gegenseitigen Ausschlusses bei Übersetzungen mit mehreren Regeln aus der Menge M^C (siehe Tabelle 19, Beispiel 7, 8). Ein solcher gegenseitiger Ausschluss ergibt sich beispielsweise:

Beispiel 10: *Es sei*

O^A: Bedingungen (Wetter, schlecht):- O^B: Temperatur(Luft, T), T>=-15, T<15, Menge(Regen, N), N>=0, Geschwindigkeit(Wind, W), W>=0

O^C: Zustand(Wetter, kalt) :- O^B: Temperatur(Luft, T), T>-30, T<0, Geschwindigkeit(Wind, W), W>=0

O^C: Zustand (Wetter, warm) :- O^B: Temperatur(Luft, T), T>=0, T<20, Menge (Regen, N), N >=0

Ohne Vorkehrungen zur Vermeidung des gegenseitigen Ausschlusses würde der Algorithmus hier eine partielle Übersetzung erzeugen. Da jedoch „Zustand (Wetter, kalt)“ und „Zustand (Wetter, warm)“, bedingt durch die Angaben zu „Temperatur“, disjunkte Regeln sind, können beide Regeln niemals gleichzeitig feuern. Insofern ist diese Übersetzung unzulässig.

Bei der VALIDIERUNG wird zunächst die Menge der zu überprüfenden Slots gebildet, die *prüfmenge*. In ihr ist jeder Slot genau einmal vorhanden. Abgezogen werden davon alle bereits in der Prädikatmenge *prädikatmengeAND* enthaltenen Slots. Diese sind aufgrund ihrer „AND“-Aufnahme bereits in der Übersetzung enthalten und brauchen so nicht überprüft zu werden. Ausgehend von dieser Menge erfolgt die Überprüfung dann in zwei Schritten:

Schritt 1: Im ersten Schritt werden von der *prüfmenge* die zu einer Regel von M^C gehörenden Slots aus der *prädikatmengeOR* abgezogen. Dies soll den Fall darstellen, dass aufgrund der Ausprägungen diese Regel nicht feuern kann. Daraus resultiert eine Prädikatmenge, die *restmenge*, welche im Weiteren zu überprüfen ist.

Schritt 2: Im zweiten Schritt wird nun sukzessiv untersucht, ob mit den in der *prädikatmengeOR* noch vorhandenen Slots eine erfolgreiche Übersetzung durchgeführt werden kann. Hierzu wird schrittweise überprüft, ob die jeweils zu einer Regel gehörenden Slots der *prädikatmengeOR* noch vollständig in der *restmenge* enthalten sind. In diesem Fall werden diese von der *restmenge* abgezogen.

Anderenfalls gilt es zu überprüfen, ob zwischen der von der *prüfmenge* abgezogenen Regel und zwischen der aktuellen Regel eine Schnittmenge existiert. Die Notwendigkeit dieses Schrittes erläutert folgendes Beispiel:

Beispiel 11: *Es sei:*

O^A*: Bedingung (Wetter, schlecht) :-* O^R*: Temperatur(Luft, T), T>=-5, T<15, Menge (Regen, N), N >=0, R<15, Geschwindigkeit (Wind, W), W >=0*

O^C*: Zustand(Wetter, kalt) :-* O^B*: Temperatur(Luft, T), T>-15, T<10, Geschwindigkeit (Wind, W), W >=0*

O^C*: Zustand(Wetter, warm):-* O^B*: Temperatur(Luft, T), T>=-1, T<=20, Menge (Regen, N), N >=0*

Anders als beim ersten Beispiel liegt bei dieser „OR"-Verkettung keine Disjunktion bzgl. der Angaben zu „Temperatur" vor, d.h. dass unter bestimmten Bedingungen, wenn die tatsächliche Temperatur zwischen -1 und 10 liegt, eine zulässige Übersetzung vorliegt. Würden jedoch lediglich die o. g. Schritte durchgeführt, so würden beide Regeln, und somit eine zulässige Übersetzung, gelöscht.

```
183. VALIDIERUNG()
184. if prädikatAnzA ∈ bodyGröße
185.    p = 1
186.    while bodyGröße^P ≠ Ø
187.       if bodyGröße^P < prädikatAnzA
188.          (ausgabeAND ∨ ausgabeOR)\ headCollection^P
189.          (prädikatmengeAND ∨ prädikatmengeOR)\ prädikatCollection^P
190.       p = p + 1
191. prüfmenge = prädikatmengeOR^1 ∪ ... ∪ prädikatmengeOR^e
192. prüfmenge = prüfmenge\ (prädikatmengeAND^1 ∪ ... ∪ prädikatmengeAND^d)
193. sa = 1
194. WHILE prädikatmengeOR^sa ≠ Ø
195.    restmenge = prüfmenge \ prädikatmengeOR^sa
196.    sb = 1
197.    WHILE prädikatmengeOR^sb ≠ Ø
198.       IF prädikatmengeOR^sb ⊆ restmenge
199.          restmenge = restmenge \ prädikatmengeOR^sb
200.       ELSE
201.             If prädikatmengeOR^sa ≠ prädikatmengeOR^sb
202.                schnittmenge = prädikatmengeOR^sa ∩ prädikatmengeOR^sb
203.                   wertidentität = TRUE
204.                   q = 1
205.                   while schnittmenge^q ≠ Ø ∧ wertidentität = TRUE
206.                   schnittA =detaillierungsliteral mit schnittmenge^q aus prädikatmengeOR^sa –Regel
207.                   schnittB =detaillierungsliteral mit schnittmenge^q aus prädikatmengeOR^sb –Regel
208.                   SEPARATION(A, 1, schnittA)
209.                   SEPARATION(B, 1, schnittB)
210.                   wertidentität = INTERVALLANALYSE(1)
211.                   q = q + 1
212.                if wertidentität = TRUE
213.                   prädikatmengeOR^sb = prädikatmengeOR^sb\ schnittmenge
214.                   if prädikatmengeOR^sb ⊆ restmenge
215.                   restmenge = restmenge\ prädikatmengeOR^sb
```

216. *sb = sb + 1*
217. *IF restmenge ≠ Ø*
218. *prädikatmengeOR = prädikatmengeOR \ prädikatmengeOR^{sa}*
219. *ausgabeOR = ausgabeOR \ ausgabeOR^{sa}*
220. *sa = sa + 1*

Nach der VALIDIERUNG erfolgt die Ausgabe der gefundenen Übersetzungen. Die Erfahrungen (Information, Verhaltensweisen, Ziele) können nun von einem Fahrzeug in das andere übertragen werden.

Mit Hilfe der Basisontologie (Kapitel 5.1) sowie des SCOUT-Algorithmus (Kapitel 5.2) lassen sich die Erfahrungen nun so verarbeiten, dass sich ein Fahrzeug erfahrungsbasiert in Kooperation mit anderen Fahrzeugen selbstoptimieren kann.

5.3 Generierung einer erfahrungsbasierten Selbstoptimierung

5.3.1 Allgemeiner Ansatz

5.3.1.1 Gesamtkostenfunktion

Zur Erlangung einer erfahrungsbasierten Selbstoptimierung ist es erforderlich, dass das Fahrzeug ständig neue Erfahrungen generiert. Dies kann nur über geeignete Explorationsstrategien erreicht werden. Zur Definition der Explorationsstrategien ist es notwendig die Parameter zu definieren, die exploriert werden sollen. Hierzu werden die Transportkosten verwendet, die es zu minimieren gilt. Die Transportkosten für den einfachen Weg von einer Quelle (Start) zu einer Senke (Ziel) setzen sich in dem vorliegenden Szenario zusammen aus den Komponenten Zeitkosten d^e zum Befahren eines Streckenabschnitts, den Tarifkosten m^e eines Streckenabschnitts und den Komfortkosten k^S eines Fahrzeugs zum Befahren eines Streckenabschnitts (je unebener der Streckenabschnitt, desto höher die Komfortkosten). Während die Tarife bereits originär in monetären Einheiten gemessen werden, erfolgt die Berechnung der Zeitkosten durch Multiplikation der Zeit t [Einheit: h] zum Befahren eines Streckenabschnitts mit einem fixen Betriebsstundenkostensatz γ^S [Einheit: €/h] eines Fahrzeugs. Die Komfortkosten eines Streckenabschnitts k^e können durch Multiplikation des Integrals der Vertikalbeschleunigung $\ddot{z}^A$ [Einheit: m/s] des Fahrzeugaufbaus mit einem fixen Komfortkostensatz κ^S [Einheit: €/m/s] berechnet werden.

$$d^e = t \cdot \gamma^S$$
$$k^e = \ddot{z}^A \cdot \kappa^S$$

Somit lassen sich die Kosten eines Streckenabschnitts $c^e{}_i$ zu einem bestimmten Zeitpunkt t folgendermaßen darstellen:

$$c_i^e(t,e) = m^e(t) + d^e(t) + k^e(e)$$
$$c_i^e(t,e) = m^e(t) + t(t) \cdot \gamma^S + \ddot{z}^A(e) \cdot \kappa^S$$

Da sich die einzelnen Komponenten dieser Formel auf unterschiedliche Ebenen des hierarchisch aufgebauten Fahrzeugs beziehen und unterschiedliche Einflussvariablen haben (so werden die monetären und temporären Kosten von der Tageszeit und die Komfortkosten vom Streckenabschnitt beeinflusst), werden die Explorationsstrategien für die MFM- (Komfortkosten) und die AMS- (temporäre und monetäre Kosten) Ebene in getrennten Kapiteln behandelt. Zudem wurden die Untersuchungen der MFM- und der AMS-Ebene auf unterschiedlichen Testplattformen durchgeführt. Stand für die MFM-Ebene ein Demonstrator im Labor zur Verfügung, musste im AMS-Bereich erst eine Simulation entwickelt werden.

5.3.1.2 Identifikation der Explorationsstrategien

Grundsätzlich sind drei verschiedene Explorationsstrategien möglich:

1. **Blinde Exploration:** Das Fahrzeug besitzt keinerlei Erfahrungen über die bestmöglichen nächsten Explorationsschritte. Somit wird wie bei einer Tiefen- oder Breitensuche ein Pfad solange zufällig exploriert, bis ein definiertes Ziel gefunden oder die entsprechend benötigte Ressource verbraucht worden ist. Diese Suche ist jedoch sehr kosten- und zeitintensiv und für ein System, das unter marktwirtschaftlichen Aspekten operieren und schnelle Verbesserungen ermitteln soll, nicht geeignet.
2. **Exploration aufgrund von Erfahrungen und Abschätzungen (Exploitation):** Das Fahrzeug operiert mit einer bestimmten Menge an Erfahrungen bzw. vorher gespeicherten Informationen. Die Exploration kann dadurch aufgrund von Abschätzungen (Heuristiken) erfolgen. So wird z.B. die Dauer des Befahrens einer Strecke als Funktion aus Länge der Strecke, Typ der Strecke (z.B. Hochgeschwindigkeitsstrecke) und der Durchschnittsgeschwindigkeit des Fahrzeugs abgeschätzt. Diese Abschätzungen werden mit der Zeit durch Erfahrungen ausgetauscht, die eine zunehmende Sicherheit in der Ausführung von Verhaltensweisen zu bestimmten Zuständen bieten. Aufgrund der Zustandsänderung der Umwelt (neue Wege, Sperrung eines Weges) oder des Fahrzeugs selbst (z.B. neue Technik) kommt es zu einer kontinuierlichen Exploration neuer Verhaltensweisen.
3. **Exploration aufgrund der Erfahrungen anderer Fahrzeuge:** Das Fahrzeug fragt aktiv nach neuen, ihm noch unbekannten Zuständen. Ab einem bestimmten Wert df_i, der sich aus den Kosten und einem Konfidenzfaktor zusammensetzt (siehe weiter unten), übernimmt es diese „externen Erfahrungen“ und exploriert wie unter Punkt 2.

Da die Exploration nicht blind erfolgen soll, startet das Fahrzeug mit vorab eingegebenen Informationen über den Aufbau des Schienennetzwerkes. Dieses wird als gerichteter Graph G modelliert. Jeder Knoten v_i symbolisiert einen Bahnhof, ein Depot oder eine Weiche; jede Kante e_j steht für einen Streckenabschnitt. Sowohl die Kanten als auch die Knoten können mit n Attributen beschrieben werden. Eine Verbindung π von Knoten v^s (Startbahnhof) zu Knoten v^z (Zielbahnhof) ist demnach definiert als eine Sequenz aus zusammengefügten Streckenabschnitten (z.B. e_1, e_3, …). Dabei ist e_i ein Element aus der Menge aller Kanten

E(G), v_i ein Element aus der Menge aller Knoten V(G) und π ein Element aus der Menge aller Verbindungen Π(G).

An den Kantenenden befinden sich jeweils die Knoten, so dass für einen zusammenhängenden Graphen gilt:

$$e_i.v_{Ende}=e_{i+1}.v_{Anfang},;\ i = 1..n\text{-}1$$

Aus den Definitionen ergibt sich Π(G). Es kennzeichnet die Menge aller Pfade in G:

$$\Pi(G) \cong \left\{ (e_0, e_1, ..., e_n) \,\middle|\, \begin{array}{l} e_i \in E(G), i = 1..n \wedge \\ e_j.v_{Ende} = e_{j+1}.v_{Anfang}, j = 1..n-1 \end{array} \right\}$$

Innerhalb dieses Graphen definiert ein Fahrgast einen Auftrag, indem er von einem Startbahnhof $v^s \in V$ zu einem Zielbahnhof $v^z \in V$ reisen möchte. Hierzu definiert er weitere, spezifische Anforderungen seiner gewünschten Reise, wie z.B. einen gewünschten Startzeitpunkt t^c, einen Faktor ω^c zur Festlegung für eine entweder möglichst billige oder eine möglichst schnelle Verbindung ($\omega^c = 0$ bedeutet, dass die Kosten entscheidend sind und die Zeit irrelevant ist. Demnach sucht der Fahrgast eine möglichst billige Verbindung.), ein Reisezeitlimit d^c, maximale Kosten m^c, einen bestimmten Komfortwert k^c, eine Menge an Präferenzen bezüglich des Fahrzeugtyps typ^c (z.B. Standard, Luxus, Güter) oder besondere Anforderungen an die lokalen Bedingungen der Streckenabschnitte L^c (z.B. keine Nebenlinien, keine Tunnels, nur Panoramastrecken). Je restriktiver die Anforderungen vom Fahrgast definiert werden, umso weniger Lösungsmöglichkeiten gibt es und damit besteht die „Gefahr“, kein passendes Angebot von einem Fahrzeug zu bekommen. Die Summe der Anforderungen eines Fahrgastes definieren die Menge der Constraints *C*, die ein Fahrzeug zu berücksichtigen hat.

$$C^c \equiv \{t^c, v^s, v^z, \omega^c, d^c, m^c, k^c, typ^c, L^c\}$$

Die Constraints des Benutzers werden in lokale und globale Constraints unterschieden. Aus der Sicht der Fahrzeuge können diese in die dynamischen und statischen Parameter untergliedert werden, wodurch vier mögliche Klassen entstehen.

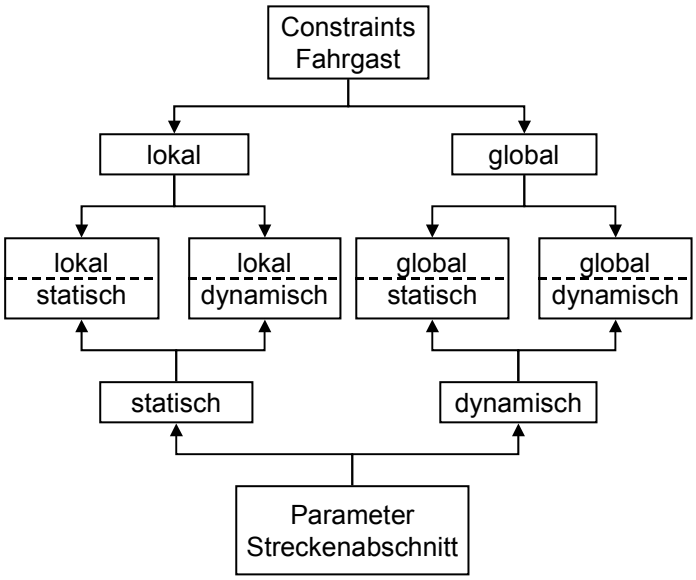


Abbildung 67: Klassifikation verschiedener Constraint Typen

Lokale Constraints werden direkt mit den Parametern der einzelnen Streckenabschnitte verglichen. Ein Streckenabschnitt, der nur eines dieser Parameter verletzt, wird aus der möglichen Lösungsmenge eliminiert. Global Constraints beziehen sich auf Parameter einer vollständigen Verbindung π, wie z.B. die Reisezeit oder die Reisekosten (d^c und m^c). Statische Parameter verändern sich nicht oder nur sehr langsam im Laufe der Zeit (z.B. die maximale Geschwindigkeit eines Streckenabschnitts oder die Beschaffenheit einer Schiene). Dynamische Parameter hingegen passen sich ständig den gegebenen Umweltbedingungen an (z.B. die Zeitdauer, um einen Streckenabschnitt zu passieren in Abhängigkeit von der Tageszeit).

Um nicht den gesamten Graphen nach einer geeigneten Lösung durchsuchen zu müssen und später eventuell festzustellen, dass der gefundene Pfad lokale Constraints verletzt, benutzt das Fahrzeug einen Graph-Reduzierungsalgorithmus. Er löscht alle Pfade, die die lokalen Constraints des Fahrgastes verletzten, indem alle ungültigen Streckenabschnitte vom Graphen entfernt werden. Dadurch wird sichergestellt, dass die danach gefundene Lösung keine weiteren lokalen Constraints verletzt.

FOREACH edge $e_i \in$ E(G):
  FOREACH local constraint $l_c \in L_c$
    IF (edge e_i violates constraint l_c) THEN
      remove edge e_i from Graph G

Um entscheiden zu können, ob ein Streckenabschnitt für einen speziellen Auftrag nicht berücksichtigt zu werden braucht, müssen Erfahrungen über diesen Streckenabschnitt gesammelt werden. Diese Erfahrungen werden sowohl hinsichtlich des Komforts (MFM-Ebene; Kapitel 5.3.2) als auch hinsichtlich der Kosten und der Zeit generiert (AMS-Ebene; Kapitel; 5.3.3).

5.3.2 Erfahrungsbasierte Selbstoptimierung der MFM-Ebene

In Kombination mit verteilten Systemen und wissensbasierten Ansätzen wurde ein Szenario für die Exploration von Führungsgrößen für schienengebundene Fahrzeuge entwickelt, das die Struktur eines Multiagenten-Systems nutzt, so dass eine Selbstoptimierung nicht nur auf ein einzelnes Fahrzeug beschränkt bleibt, sondern durch den Erfahrungsaustausch zwischen den Fahrzeugen unter Einbeziehung der Streckenabschnitte beschrieben ist. Als Anwendungsbeispiel wird hier das Feder- und Neigemodul des Fahrzeugs betrachtet. Ebenso wären das Spurführungs- oder das Antriebs- und Bremsmodul denkbar. Als Optimierungsgröße für eine Exploration wird der Fahrkomfort ausgewählt.

Definition 5-1 Fahrkomfort Der *Fahrkomfort* bedeutet die Abstimmung der Aufbaufederung und -dämpfung zur Erreichung einer minimalen Belastung des Ladeguts bei ausreichender Fahrsicherheit [Ril02].

Zunächst wird das Modell (die Struktur, die Parameter) für den Fahrkomfort entwickelt. Die wesentlichen Eigenschaften zur Optimierung dieses Modells (siehe Abbildung 68) sind die Parameter m_A (Masse des Aufbaus), m_R (Masse der Radlast), c_A (Aufbaufedersteifigkeit), d_A (Dämpfer des Aufbaus), c_R (Radlastfedersteifigkeit), z_A (Auslenkung der Aufbaumasse), z_R (Auslenkung der Radlast) und z_S (die Straßenanregung).

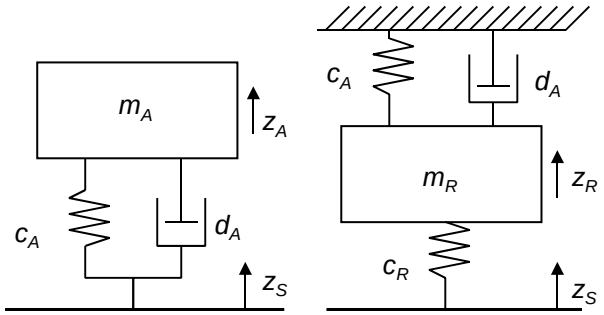


Abbildung 68: Einfache Vertikaldynamik-Modelle [Ril02]

Aus mathematischer Sicht ist der Fahrzeugaufbau bezogen auf die *Aufbaubeschleunigung* optimal (d.h., dass die Auslenkungen z_A und die Beschleunigungen $\ddot{z}_A$ des Aufbaus und die Schwankungen der dynamischen Radlast $P_{dyn} = c_R * z_R$ minimal sind), wenn die Parameter m_A (Masse des Aufbaus), m_R (Masse der Radlast), c_A (Aufbaufedersteifigkeit), d_A (Dämpfer des Aufbaus), c_R (Radlastfedersteifigkeit), z_A (Anfangsauslenkung der Aufbaumasse) und z_R (Anfangsauslenkung der Radlast) aus den Forderungen nach Komfort $\varepsilon^2_{G_K}$ [Ril02]:

$$\varepsilon^2_{G_K} = z^2_{A_0} \frac{c_A}{m_A} \frac{1}{2} \left[\frac{d_A}{m_A} + 2 \frac{c_A}{d_A} \right] \rightarrow Minimum$$

und Sicherheit $\varepsilon_{G_S}^2$

$$\varepsilon_{G_S}^2 = z_{R_0}^2 c_R^2 \frac{1}{2}\left[\frac{d_A}{c_A + c_R} + \frac{m_R}{d_A}\right] \rightarrow Minimum$$

bestimmt werden [Ril02].

Eine geringe Aufbaufedersteifigkeit $c_A \rightarrow 0$ oder große Aufbaumassen $m_A \rightarrow \infty$ gewährleisten demnach einen hohen Komfort. Beides ist im Extrem jedoch wenig ratsam. Entscheidend ist es, den richtigen Mix zu finden, wobei sich durch Differenzieren der Komfortformel nach d_A der optimale Dämpfungsparameter ergibt:

$$(d_A)_{opt} = \sqrt{2 c_A m_A}$$

Parallel bedeutet dies für die Sicherheit, dass geringe Reifenfedersteifigkeiten $c_R \rightarrow 0$, eine harte Aufbaufederung $c_A \rightarrow \infty$ oder kleine Radmassen $m_R \rightarrow 0$ die Sicherheit erhöhen. Im Sinne des Ausschwingverhaltens kann durch Differenzieren der Sicherheitsformel nach d_A der optimale Dämpfungsparameter ermittelt werden:

$$(d_A)_{opt} = \sqrt{(c_A + c_R) m_R}$$

Aus mathematischer Sicht ist das Fahrzeug im Sinne des Fahrkomforts damit eindeutig beschrieben. Im vorliegenden Szenario wird jedoch davon ausgegangen, dass sich die Umwelteinflüsse kontinuierlich ändern und dass sich die Optima durch das Zusammenspiel der einzelnen Funktionsmodule nicht „so einfach“ mathematisch berechnen lassen, sondern durch Erfahrungen im Betrieb explorativ ermittelt werden müssen. So bedarf es zunächst der Beschreibung des Vertikaldynamik-Modells mittels einer individuellen Ontologie (Abbildung 69 liefert einen Ausschnitt dieser Ontologie).

Da in diesem Beispiel der Sicherheitsaspekt nicht berücksichtigt wird, werden die Radgrößen nicht weiter untersucht[32]. Das Szenario konzentriert sich somit ausschließlich auf die Größen c_A, d_A, m_A und z_A des Feder- und Neigemoduls. Da c_A und m_A während des online-Betriebs nicht veränderbar sind, ist eine Verbesserung des Komforts nur mit den Größen d_A und z_A möglich. Die Größe d_A wird durch eine aktive Federung so eingestellt, dass z_A minimal wird.

[32] Für die Sicherheit eines Fahrzeugs sind die Radgrößen deshalb wichtig, weil die Räder den Boden nicht verlassen dürfen. Wäre dies der Fall, könnte ein schienengebundenes Fahrzeug aus der Schiene springen oder ein Auto von der Fahrbahn abkommen.

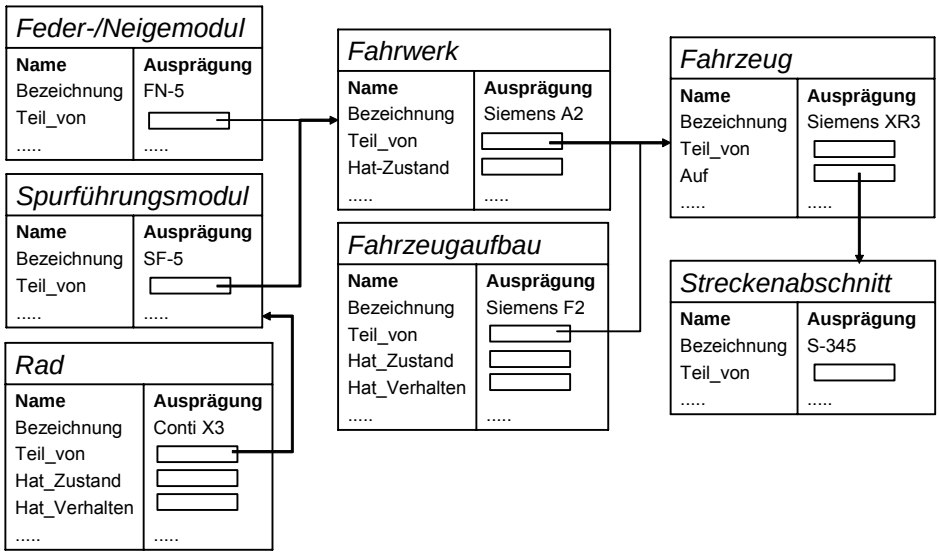


Abbildung 69: Ausschnitt aus der individuellen Ontologie eines Fahrzeugs

5.3.2.1 Der Explorationsalgorithmus

Das Grundkonzept der Exploration basiert auf der Vorgabe von Führungsdaten für die Regelung des Fahrwerks für einen bestimmten Streckenabschnitt und der Bewertung dieser Vorgaben durch das Fahrzeug. Dabei ist es nicht das Ziel, optimale Führungsdaten für einen bestimmten Zustand zu erzielen (was aufgrund der Komplexität und der Dynamik der Umwelt sowie aufgrund der weichen Echtzeit, unter dem das Fahrzeug operiert, unmöglich erscheint), sondern eine ständige Verbesserung der Führungsdaten zu erreichen.

Sind die Fahrzeuge in ihrer Dynamik sehr ähnlich oder gleich, was hier vereinfacht angenommen wird, so ist eine Bewertung des Führungsverlaufs durch verschiedene Fahrzeuge möglich. Werden die Führungs- und Bewertungsdaten durch eine der Strecke zugeordneten Instanz optimiert und verwaltet, ergibt sich daraus eine verteilte Architektur, die eine Exploration über viele Fahrzeuge erlaubt.

Um die Exploration zu lokalisieren und das Gesamtproblem aufzuteilen, wird eine Verbindung in mehrere Streckenabschnitte unterteilt. Dabei ist ein Streckenabschnitt genau einer Bereichskontrolle zugeordnet, die als Optimierer ausgebildet ist. Die Bereichskontrolle führt die Optimierung nur für den eigenen Streckenabschnitt durch. Des Weiteren hat die Bereichskontrolle die Aufgabe, die Führungsgrößen an das Fahrzeug zu senden, Bewertungsdaten entgegenzunehmen und das Fahrzeug an die nächste Bereichskontrolle weiterzuleiten.

Abbildung 70 zeigt die Grundstruktur der verteilten Exploration.

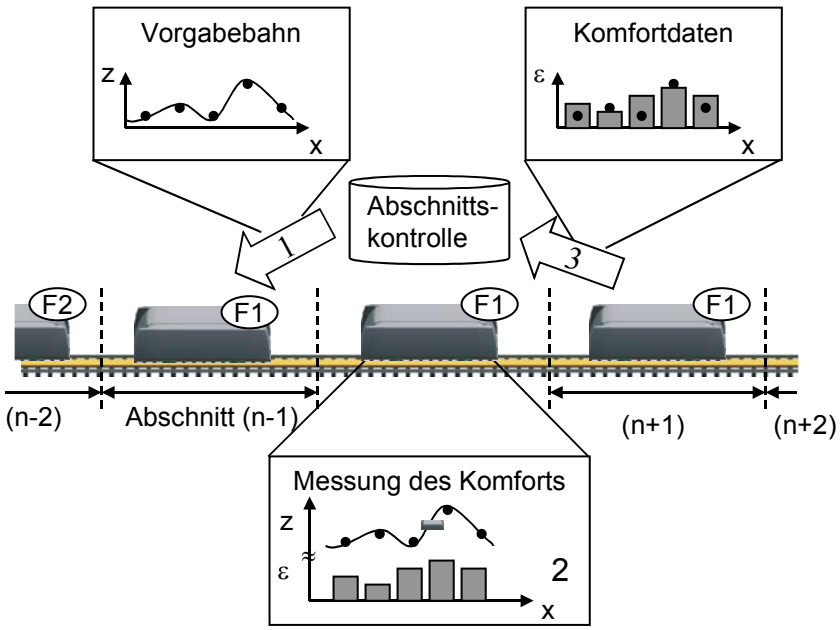


Abbildung 70: Verteilte Optimierung der Führungsvorgabe

Ein einfahrendes Fahrzeug *F1* meldet sich vor der Durchfahrt durch einen Bereich bei der jeweiligen Bereichskontrolle an. Die Bereichskontrolle sendet als Antwort die Führungsgröße für die Aufbauregelung in Form von Stützstellen eines Splines zurück. Nach der Durchfahrt sendet das ausfahrende Fahrzeug *F1* Bewertungsdaten, in diesem Fall die Bewertung des Komforts, an die Bereichskontrolle zurück. Diese berechnet mit Hilfe eines Optimierungs- bzw. Explorationsverfahrens auf der Basis der vorgegebenen Führungsgrößen und der korrespondierenden Bewertung eine neue, verbesserte Führungsgröße für ein nachfolgendes Fahrzeug *F2*[33].

Die eigentliche Optimierung des Komforts erfolgt aufgrund der gemessenen Daten über den Ist-Verlauf des Aufbauweges z_A und der Ist-Aufbaubeschleunigung $\ddot{z}_A$. Es gilt den Komfort zu maximieren, der in der Fahrzeugtechnik häufig als Bewertung des Ausschwingverhaltens [HO03] definiert wird. Mit z_A und $\ddot{z}_A$ lässt sich eine einfache Bewertungsfunktion als Ersatzformel der weiter oben beschriebenen Formel von [Ril02] aus der quadratischen Fehlerfläche (mit $\varepsilon^2_{G_K}$ als Komfortmaß und g_1 und g_2 als Gewichtungsfaktoren) gegenüber der Ruhelage in z-Richtung in Abhängigkeit vom Ort x definieren.

$$\varepsilon^2_{G_K} = \int_{x=0}^{x=x} \left\{ \left(g_1 z_A\right)^2 + \left(g_2 \ddot{z}_A\right)^2 \right\} dx \rightarrow \min$$

Im einfachsten Fall kann der Komfortgewinn als Reduzierung der Aufbaubeschleunigung beschrieben werden ($g_1 = 0$, $g_2 = 1$), wenn die relative Auslenkung nicht herangezogen werden muss.

[33] Um einen sicheren Betrieb zu gewähleisten, ist die Fahrzeugregelung in der Lage, bei Ausfall der Kommunikation auch ohne Vorgabedaten zu arbeiten [HM004].

Somit ergibt sich als Komfort:

$$\varepsilon_{G_K}^2 = \int_{x=0}^{x=x} \ddot{z}_A^2 dx$$

Über diesen Komfortwert können die Stützpunkte ausgewählt werden, die hinsichtlich des Komforts verbessert werden müssen. Für die Bewertung der einzelnen Stützpunkte werden dafür diejenigen Integrale der quadratischen[34] Beschleunigung $\ddot{z}_A^2$ betrachtet, die in einem gewissen Bereich um den Stützpunkt S_i liegen. Für einen Stützpunkt S_i ergibt sich der Komfortwert aus den Intervallbereichen $I_{i-2} = [S_{i-2}, S_{i-1}]$, $I_{i-1} = [S_{i-1}, S_i]$, $I_i = [S_i, S_{i+1}]$ und $I_{i+1} = [S_{i+1}, S_{i+2}]$(siehe auch Abbildung 71).

Hierbei ist zu beachten, dass bei einer Verschiebung des Stützpunktes S_i in die direkten Nachbarbereiche I_{i-1} und I_i die größten Änderungen auftreten. In den erweiterten Nachbarbereichen I_{i-2} und I_{i+1} sind die Änderungen weniger ausgeprägt. Für alle weiteren Bereiche wird angenommen, dass die Auswirkungen der Verschiebung so gering sind, dass sie vernachlässigt werden können. Daraus ergibt sich für einen Stützpunkt S_i der Komfort[35] als:

$$Komfort(S_i) = \frac{g_1 \cdot \varepsilon_{G_K}^2(I_{i-2}) + \varepsilon_{G_K}^2(I_{i-1}) + \varepsilon_{G_K}^2(I_i) + g_2 \varepsilon_{G_K}^2(I_{i+1})}{2 + g_1 + g_2}$$

g_1 und g_2 sind Gewichtungsfaktoren der erweiterten Nachbarbereiche (hier $g_1 = g_2 = 0.5$). Mit der Division durch $2 + g_1 + g_2$ wird das Ergebnis normiert. Das ist nötig, da bei Randpunkten nicht alle vier Bereiche existieren. Falls S_i ein Randpunkt ist, werden nur Messdaten der existierenden Bereiche summiert und die Normierung dementsprechend angepasst. Die Auswahl des schlechtesten Punktes erfolgt aufgrund der zuvor ermittelten Komfortwerte der Stützpunkte. Dabei werden alle regulären Stützpunkte durchlaufen, und es wird derjenige Stützpunkt ausgewählt, der den schlechtesten Komfortwert[36] aufweist.

Für eine iterative Verbesserung gilt es, einen Zusammenhang zwischen Vorgabebahn und vertikaler Beschleunigung zu ermitteln und die Vorgabebahn entsprechend zu variieren. Ob eine Verbesserung eingetreten ist, kann durch Vergleich der aktuellen Beschleunigungsdaten mit vorher aufgezeichneten Daten erreicht werden. Dazu wird ein Streckenabschnitt, wie in Abbildung 71 gezeigt, in Intervalle (I_i) zerlegt und die quadratische Abweichung zwischen vorheriger Beschleunigungskurve $\ddot{z}_{A,P}$ und aktueller Beschleunigungskurve $\ddot{z}_{A,N}$ berechnet.

[34] Die Beschleunigung wird quadriert, um positive Ergebnisse zu erhalten.

[35] Es wurden lediglich vier Sektionen betrachtet, da eine Verschiebung des Stützpunktes nur diese Sektionen in ihren Komfortwerten entscheidend beeinflusst.

[36] Da der Komfortwert aus dem Integral der Beschleunigung ermittelt wird, bedeutet eine Verbesserung des Komforts eine Verringerung der Maßzahl.

Die Bewertung der Komfortveränderung lautet hierbei:

$$\Delta\varepsilon^2_{G_K}(I_i) = \int_{S_i}^{S_{i+1}} \ddot{z}^2_{A,P}\,dx - \int_{S_i}^{S_{i+1}} \ddot{z}^2_{A,N}\,dx$$

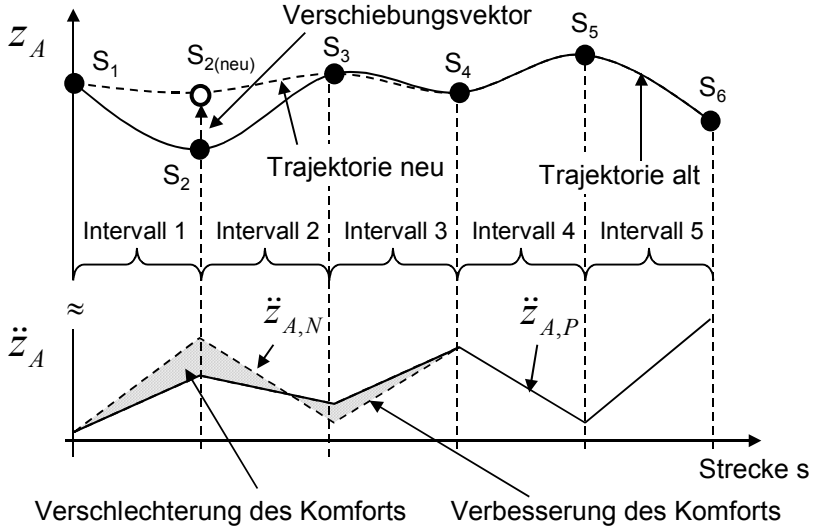


Abbildung 71: Beschleunigungsveränderung

Das Vorzeichen des Ergebnisses ist ein Kriterium dafür, ob der letzte Schritt erfolgreich war oder nicht. Ein positives Vorzeichen bedeutet Erfolg, ein negatives bedeutet Misserfolg. Die Größe der Änderung kann für die Berechnung der Folgeschrittweite der neuen Vorgabebahn genutzt werden. Hierbei ist der Schritt Δz_N eine Funktion der Fehlerfläche und der vorherigen Schrittweite Δz_P.

$$\Delta z_N = \mathrm{sgn}\left(\Delta\varepsilon^2_{G_K}\right) \cdot q \cdot \Delta\varepsilon^2_{G_K} \cdot \Delta z_P$$

q ist lediglich ein über Erfahrungen zu optimierender Gewichtungsfaktor, der in Abhängigkeit von der Anzahl der erfolgreichen Schritte variiert werden kann. $\mathrm{sgn}(\Delta\varepsilon^2_{G_K})$ bestimmt durch sein Vorzeichen die Änderungsrichtung und $\Delta\varepsilon^2_{G_K}$ die Schrittweite.

Während der Optimierung kann die Situation eintreten, dass die Verschiebung eines Stützpunktes, obwohl dieser die schlechtesten Komfortwerte aufweist, nicht zu einer (signifikanten) Verbesserung der Vorgabebahn führt. In diesem Fall würde dieser Punkt immer wieder ausgewählt und verschoben mit der Konsequenz, dass die anderen Stützpunkte von dem Verfahren unberührt blieben. Um eine solche Situation zu vermeiden, wurde eine Tabuliste [GL93] eingeführt. Zu Beginn der Optimierung ist die Tabuliste leer. Wird ein Stützpunkt als schlechtester ausgewählt und verschoben, so wird das Ergebnis dieser Verschiebung zu Beginn des darauf folgenden Durchlaufs ausgewertet. Für diesen „veränderten“ Punkt steht nun fest, ob seine Verschiebung erfolgreich war. Falls die Verschiebung nicht zu einer Verbesserung geführt hat, wird der Punkt „angezählt“. Das

bedeutet, dass der Zähler[37] für den Stützpunkt um eins inkrementiert wird. Kam es durch die Verschiebung zu einer Verbesserung der Vorgabebahn, wird der Zähler um einen Wert c_P erhöht, wobei $0 \leq c_P \leq 1$ gelten sollte. Ist $c_P > 0$, so wächst der Zähler auch bei einer positiv ausgewerteten Verschiebung des Punktes. Übersteigt der Zähler eines Stützpunktes S_i einen festen Wert t_{max}, wird er in die Tabuliste eingetragen. Dazu wird in einem Tabufeld an der Position S_i eine 1 eingetragen. Bei jedem weiteren Optimierungs-Durchlauf werden die Zähler aller auf der Tabuliste eingetragenen Punkte um eins inkrementiert. Übersteigt dabei der Zähler eines Stützpunktes S_i den Wert t_{free}, wird die Tabuliste und der Zähler auf 0 zurückgesetzt. S_i ist somit von der Tabuliste gelöscht. Der entsprechende Optimierungsalgorithmus lautet:

1. Berechne Startvorgabebahn
2. Sende Stützpunkte S_i der Startvorgabebahn an das Fahrzeug
3. Empfange Messdaten (neue Stützpunkte S_i) vom Fahrzeug
4. Für alle Stützpunkte S_i
    a. Bestimme Komfortwerte für I_i nach $\varepsilon^2_{G_K}$
    b. Bestimme Komfortwert für S_i nach Komfort(S_i)
    c. Setze $Gesamtkomfort_{aktuell} = Gesamtkomfort_{aktuell}$ + Komfort(I_i)
5. Falls $Gesamtkomfort_{alt} \leq Gesamtkomfort_{aktuell}$
    a. Multipliziere den Verschiebungsvektor des schlechtesten Punktes mit Δz_N
    b. Erhöhe den Tabuwert des schlechtesten Punktes um 1
6. Ansonsten multipliziere den Verschiebungsvektor des schlechtesten Punktes mit Δz_N
7. Falls der Tabuwert des schlechtesten Punktes größer ist als 3, trage den Punkt in die Tabuliste ein
8. Lösche alle Punkte, die länger als zehn Durchläufe „tabu“ waren, von der Tabuliste
9. Finde den aktuell schlechtesten Punkt und verschiebe ihn
10. Setze $Gesamtkomfort_{alt} = Gesamtkomfort_{aktuell}$
11. Setze $Gesamtkomfort_{aktuell} = 0{,}0$
12. weiter mit 2

5.3.2.2 Ergebnisse des Explorationsalgorithmus

Abbildung 72 zeigt einen virtuellen Streckenabschnitt der Teststrecke. Dieser Abschnitt ist 50m lang und durch 50 Stützpunkte definiert. Ausgehend von einer „geraden“ Vorgabebahn wurde jeweils in der Mitte eines Streckenabschnitts eine konstante Störung aufaddiert. Die Aufschaltung hatte eine unterschiedliche Länge (10, 20 und 30 cm) und Intensität (3, 6 und 10cm). Diese Maße erscheinen zwar recht groß für eine realistische Schienenstörung, waren aber in dieser Höhe notwendig, da sich das Fahrzeug in der Testumgebung mit einer Geschwindigkeit von nur 3,6 km/h bewegt.

[37] Für alle Stützpunkte existiert eine solche Zählvariable, die zu Beginn der Optimierung mit 0 initialisiert wird.

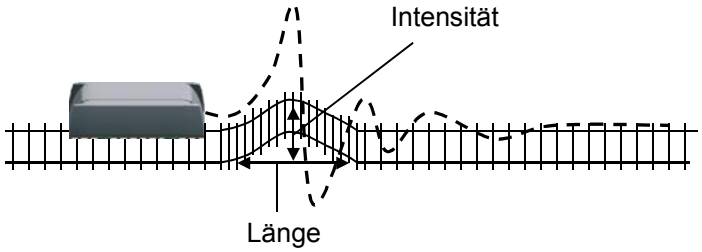


Abbildung 72: Länge und Intensität einer Störung

Abbildung 73 zeigt die Ergebnisse des Optimierungsverfahrens bei einer Simulation nach dem oben beschriebenen Szenario. Auf der x-Achse sind die einzelnen Durchfahrten des Fahrzeugs auf dem Streckenabschnitt abgetragen. Die drei Abschnitte wurden jeweils 200-mal überfahren. Auf der y-Achse sind die Komfortwerte abgetragen, welche in einem Durchgang für einen entsprechenden Streckenabschnitt erreicht wurden. Die jeweiligen Peaks haben zwei Ursachen: Einerseits führt nicht jeder Schritt zu einer Verbesserung des Komforts („Explorations“ - Prinzip), andererseits wird bei der Verschiebung von Stützpunkten der Stellweg der Feder teilweise überschritten, was zu einer erheblichen Verschlechterung des Komforts führt („Ruckeln“).

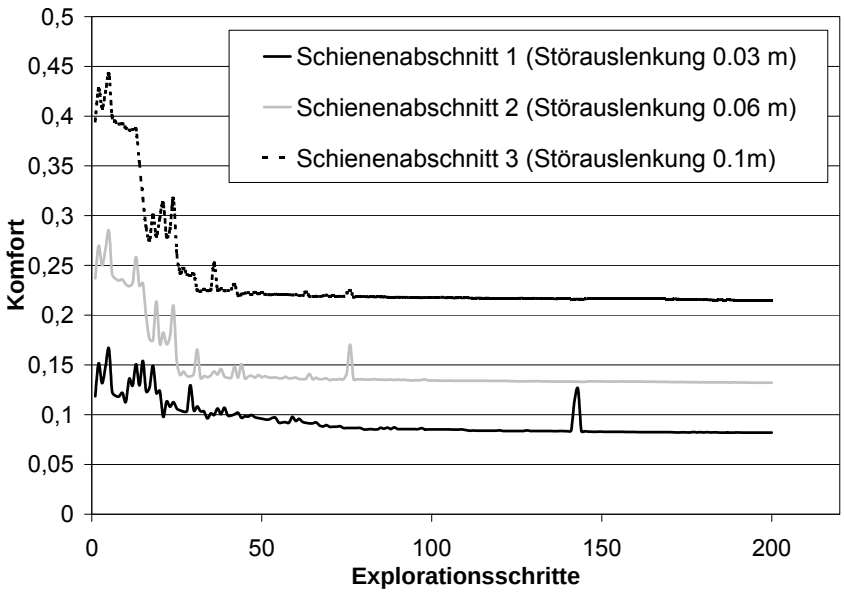


Abbildung 73: Komfortverbesserung nach 200 Durchläufen

Es ist zu erkennen, dass sich der Komfort für die einzelnen Tracks je nach Störung nach etwa 30 bis 50 Durchgängen um bis zu 50% verbessert. Danach sind die Änderungen an den Spline-Funktionen nur gering.

Dass sich die Werte nicht annähernd auf 0 optimieren lassen, ist auf die Struktur der Störung zurückzuführen. Um der Schienenanregung optimal entgegenzuwirken, müsste eine Aufschaltung erzeugt werden, die, aufsummiert auf die Anregung, eine konstante Funktion

erzeugt. Eine Spline-Funktion kann aber diese Struktur nicht genau erzeugen, sondern sich ihr nur annähern.

Ein weiteres Problem ergibt sich bei einer relativ hohen Frequenz der Anregung. Wird die Störung nur durch einen kurzen und wenig intensiven Stoß verursacht, ist die Antwort des Systems eine kurzwellige Schwingung. Da die Stützpunkte der Spline-Funktion aber einen Meter voneinander entfernt sind, können keine Störungen mit einer höheren Frequenz absorbiert werden. Dies ist allerdings auch nicht die Kernaufgabe der Vorgabebahn. Im endgültigen System verhindern Luftfedern die Übertragung hochfrequenter Schwingungen auf den Aufbau des Fahrzeugs.

Der Algorithmus wurde an einem zweiten Beispiel, einem nicht geraden Streckenabschnitt, getestet. Bei einem nicht geraden Streckenabschnitt (z.B. Kurve; Abbildung 74) erfährt ein überfahrendes Fahrzeug eine zusätzliche horizontale Beschleunigung, die explorativ minimiert werden soll. Die Vorgehensweise zur Verbesserung der horizontalen Beschleunigung ist der zuvor vorgestellten Methode zur Optimierung der vertikalen Beschleunigung ähnlich. Auch hier wird eine Spline-Kurve, gegeben durch eine konstante Zahl von Stützpunkten, benutzt, um eine Vorgabebahn zu beschreiben.

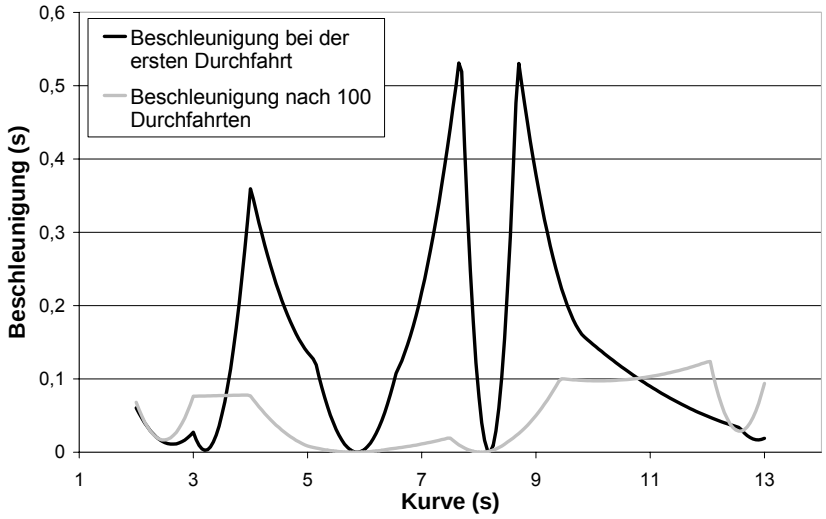


Abbildung 74: Beschleunigung des Fahrzeugs im Kurvenbereich

Betrachtet man die horizontale Beschleunigung des Fahrzeugs, greift der obige Ansatz (vertikale Beschleunigung) jedoch nicht mehr ganz. Ein extremes Beispiel hierfür ist eine Kreisfahrt, bei der die x- sowie die y-Dimension nicht monoton ist. Daher lässt sich dieses Szenario nicht durch eine Funktion mit einem Parameter x oder y darstellen. Hier werden zweidimensionale Splines benutzt, die auf der Grundlage von zwei linearen Spline-Funktionen harmonische Verläufe in der Ebene beschreiben können. Eine solche Funktion ist eindeutig über die Strecke s, d.h. zu jeder zurückgelegten Strecke existiert ein eindeutiges Tupel (x,y), das die Soll-Position des Aufbaus beschreibt. Die beiden linearen Splines sind dabei zwei Funktionen $f_x(s)$ und $f_y(s)$, die jeweils die x bzw. die y Position des Splines

zurückliefern. Damit ist es möglich, alle denkbaren Vorgabebahnen in horizontaler Ebene zu definieren.

Mit diesen Voraussetzungen kann der oben beschriebene Verbesserungsalgorithmus hier analog eingesetzt werden. Zu Berechnung der Kurve werden die absoluten Werte der Beschleunigungskurven von $f_x(s)$ und $f_y(s)$ addiert.

5.3.3 Erfahrungsbasierte Selbstoptimierung der AMS-Ebene

Intelligente, autonome Fahrzeuge sind durch ihre automatisierten Berechnungen von optimierten Pfaden, die Forderung nach Flexibilität, das Vorhandensein von jederzeit aktuellen Informationen und verteilt asynchronen Algorithmen für die Kontrolle und die Koordination mit anderen Fahrzeugen gekennzeichnet [GL00]. Dies versetzt sie in die Lage, flexibel und individuell ihr Verhalten auf die Bedürfnisse des Fahrgastes (Startbahnhof, Zielbahnhof, Route, max. Kosten, Reisezeit etc.) anzupassen, indem dem Fahrgast ein möglichst optimales, maßgeschneidertes Angebot unterbreitet wird.

Dieses Kapitel bietet einen Ansatz für flexible, intelligente, dezentrale, bedarfsgesteuerte Fahrzeuge, die miteinander kommunizieren, dadurch Wissen austauschen und so kooperieren. Es konzentriert sich auf die AMS-Ebene des hierarchischen Transportsystems, in der die Basis die individuelle Ontologie eines Fahrzeugs ist, die in Abbildung 75 ausschnittsweise dargestellt ist.

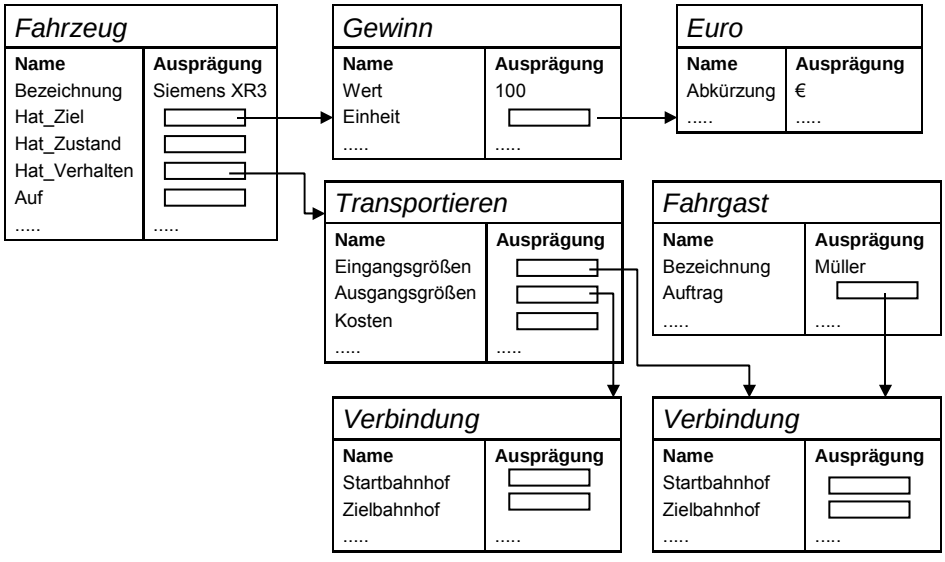


Abbildung 75: Ausschnitt aus der individuellen Ontologie des Fahrzeugs

5.3.3.1 Der Explorationsalgorithmus

Da die Exploration nicht blind erfolgen soll, werden für die Exploration eines Streckennetzes die Kosten einer Verbindung c^{π}, die sich aus der Summe der Kosten eines Streckenabschnitts c^{e} ergeben, herangezogen.

Diese gilt es zu minimieren:

$$c^{\pi} = \min \sum_{e \in \pi} c^{e}$$

Die Kosten eines Streckenabschnitts werden auf der AMS-Ebene als gewichtete Summe (ω^c wird durch Benutzer definiert und liegt im Intervall [0,1]) der monetären m_i^e und der temporären d_i^e Kosten in Abhängigkeit von der Zeit t berechnet.

$$c_i^e(t) = (1 - \omega^c) \cdot m_i^e(t) + \omega^c \cdot d_i^e(t)$$

Die Werte für m_i^e und d_i^e können sich einerseits aus den Erfahrungen über die Zeit heraus ergeben ($m_i^e(t), d_i^e(t)$), lassen sich andererseits aber auch, wenn diese noch nicht vorhanden sind, da jedes Fahrzeug anfänglich keine Erfahrungen über die Streckenabschnitte besitzt, rechnerisch (über Heuristiken) ermitteln ($m_i^{heu,e}, d_i^{heu,e}$).

Zur Bestimmung der heuristischen Kosten $c_i^{heu,e}$ eines Streckenabschnitts (die nicht zeitabhängig sind, da noch nicht gesagt werden kann, ob morgens oder abends andere Verhältnisse existieren) werden für die monetären Kosten die durchschnittlichen Streckengebühren $toll_i^{heu,e}$ und für die temporären Kosten die Länge des Streckenabschnitts $length^e$ dividiert durch die Durchschnittsgeschwindigkeit des Fahrzeugs v^a multipliziert mit den Betriebsstundenkostensatz γ^S berechnet:

$$c_i^{heu,e} = (1 - \omega^c) \cdot toll^{heu,e} + \omega^c \cdot \frac{length^e}{v^a} \cdot \gamma^S$$

Die rechnerische bzw. die heuristische Lösung ermöglicht nur eine ungefähre Bestimmung der tatsächlichen Kosten. Aus diesem Grunde ist die Nutzung der tatsächlich gemachten Erfahrungen vorzuziehen (soweit diese vorhanden sind). Hierbei können die tatsächlichen Kosten je nach Tageszeit und Wochentag stark schwanken. So können zu Hauptverkehrszeiten deutlich längere Reisezeiten entstehen oder die Streckengebühren deutlich angehoben werden (um z.B. eine gleichmäßige Verteilung des Verkehrs zu erreichen). Zudem kann ein Stop-and-Go Verkehr zu höheren Energie- und Verschleißkosten führen. Aus diesem Grunde erstellt das Fahrzeug ein zeitabhängiges Profil für jeden Streckenabschnitt. Dieses Profil besteht aus vier Funktionen ($m_i^e(t), d_i^e(t), sf_i^m(t), sf_i^d(t)$), die sich jeweils über die Zeit von 168 Stunden (t=1,...,168) einer Woche erstrecken. $sf_i^m(t)$ und $sf_i^d(t)$ sind die zu der monetären und der temporären Kostenfunktion gehörenden **Support-Funktionen**. Die Support-Werte beschreiben die Anzahl der sicheren Erfahrungen, die zur Berechnung der monetären und der temporären Kosten herangezogen werden. Ein Beispiel soll die Funktionsweise der Support-Funktion verdeutlichen. Wird eine neue Erfahrung x_k (temporäre und monetäre Kosten) zum Zeitpunkt t_k auf einem bestimmten Streckenabschnitt generiert, bewegt der Lernalgorithmus die Funktion $m_i^e(t_k)$ in Richtung des neues Messpunktes, während der Wert von $sf_i^m(t_k)$ um 1 (also um eine zusätzliche Erfahrung) erhöht wird (siehe Abbildung 76).

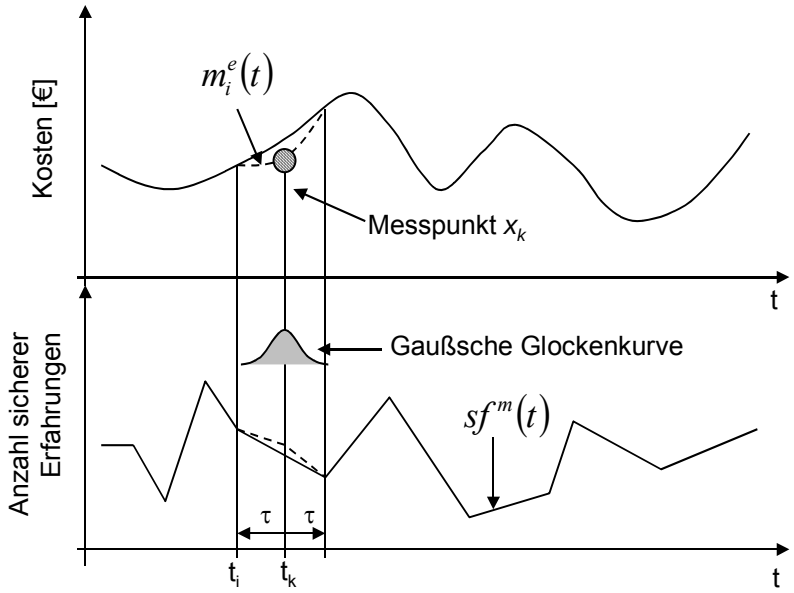


Abbildung 76: Monetäre Funktion (oben) mit zugehöriger Support-Funktion (unten)

Zur Vereinfachung wird hier lediglich die monetäre Kosten- und Support-Funktion erläutert, da die Berechnung der temporären Kosten- und Support-Funktion identisch ist.

Der neue Messpunkt x_k beeinflusst jedoch nicht nur den exakt gemessenen Zeitpunkt, sondern auch benachbarte Zeitpunkte. Hierfür wird der Faktor υ_j eingeführt, der sich im Intervall [0,1] befindet und sich an die Dichtefunktion der Gauß'schen Normalverteilung ($y=ae^{-b*t*t}$) anlehnt [EH04], [Sac04], [Kre03] (siehe Abbildung 77). Dabei beträgt die Höhe der Glockenkurve immer 1 (a = 1), da immer genau eine neue Erfahrung gemacht wird. Der Faktor für die Form der Kurve (flacher oder steiler Berg) ist $b = 1/(2*\sigma^2)$. Da eine möglichst gute Abdeckung der Einflüsse erreicht werden soll, wird ein Intervallbereich gewählt, der 99,73 Prozent aller Nachbarwerte beeinflusst. Dies gilt, wenn τ (also der Einflussbereich des Messpunktes) = 3σ. Setzt man τ in b ein, folgt daraus $b=9/(2*\tau^2)$. Um den Einfluss des Zeitpunkts t_k (Nullpunkt der Gauß'schen Glockenkurve) auf die Nachbarwerte t_j zu berechnen, wird statt der Variablen *t* der Term (t_j-t_k) in die Formel eingesetzt. Somit gilt:

$$\upsilon_j(t_j) = e^{-9(t_j-t_k)^2/2\tau^2}$$

Ist der Faktor τ beispielsweise auf 30 Minuten gestellt, so beeinflusst der Messwert alle Werte, die sich im Intervall [t_k-0,5, t_k+0,5] befinden, jedoch nicht zu gleichen Teilen. So steigt der Support-Wert bei t_k um den Wert 1 und nimmt zu beiden Seiten hin ab (je weiter also t_j von t_k entfernt ist, desto unsicherer wird die Übertragbarkeit der Erfahrung bei t_k auf den Zeitpunkt t_j).

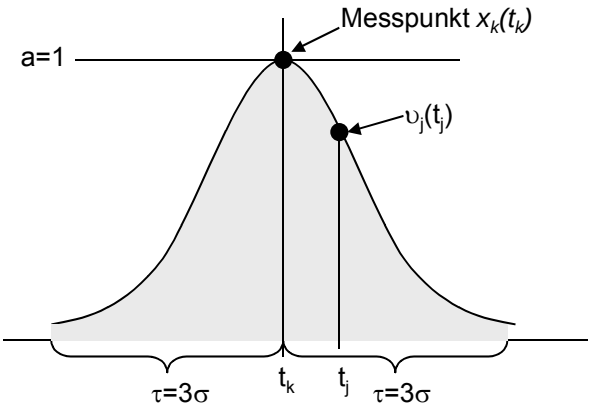


Abbildung 77: Gauß'sche Glockenkurve

Der Support-Wert wird nun an der Stelle t_j um den Faktor υ_j erhöht. Daraus ergibt sich der neue Wert:

$$sf_{i,n+1}^{m}(t_j) = sf_{i,n}^{m}(t_j) + \upsilon_j(t_j)$$

Das Problem bei der Support-Funktion besteht darin, dass diese bis jetzt nur neue Erfahrungen aufsummiert und damit bei hohen Erfahrungswerten sehr resistent gegenüber neuen, unerwarteten Erfahrungen wird (da mit zunehmender Anzahl der Erfahrungen der anteilsmäßige Einfluss einer neuen Erfahrung abnimmt). Weichen jedoch die neuen Erfahrungen von den bisherigen in starkem Maße ab, muss dies eine sofortige Auswirkung auf den Support-Wert haben. Aus diesem Grunde wird eine **Unsicherheitsfunktion** $U(t_j)$ mit in die Support-Funktion aufgenommen, die bei Ausnahmeerfahrungen den Wert der Support-Funktion sofort reduziert.

$$sf_{i,n+1}^{m}(t_j) = \left(sf_{i,n}^{m}(t_j) + \upsilon_j(t_j)\right) \cdot U(t_j)$$

Zur Berechnung der Unsicherheitsfunktion werden die Faktoren κ, ρ und φ benötigt. φ beschreibt die Grenze, bis zu der bei abweichenden Messungen kein Support-Verlust eintritt. φ ist ein Wert zwischen 0 und 1 und beschreibt diese Grenze als Prozentgröße der Bandbreite der monetären Funktion. Bei φ = 0,05 und einer Funktion, die sich zwischen 0 und 100 bewegt, liegt die Grenze somit bei ±5 um den Erwartungswert. Standard im Lernalgorithmus ist $\varphi = 0{,}1$.

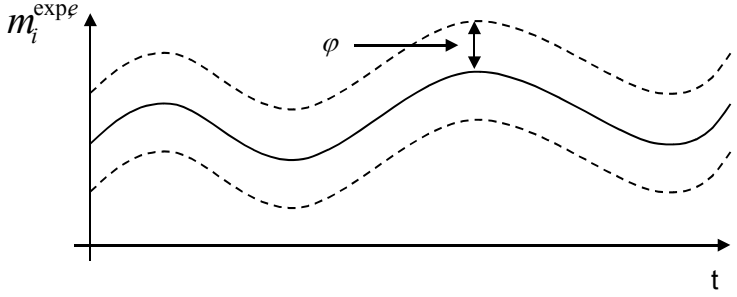


Abbildung 78: Abweichungsfaktor φ

ρ beschreibt, wie stark der Vergessenseffekt bei Abweichungen jenseits der durch φ festgelegten Grenze ausfällt. ρ liegt im Intervall zwischen 0 und 1. Je höher ρ ist, desto heftiger reagiert der Algorithmus auf Abweichungen. Standard im Lernalgorithmus ist ρ=0,3. φ und ρ werden in eine e-Funktion eingebunden, über die die Abnahme des Supports berechnet wird.

$$e^{\rho^2-\left(\rho\cdot\frac{\Delta}{\varphi}\right)^2}$$

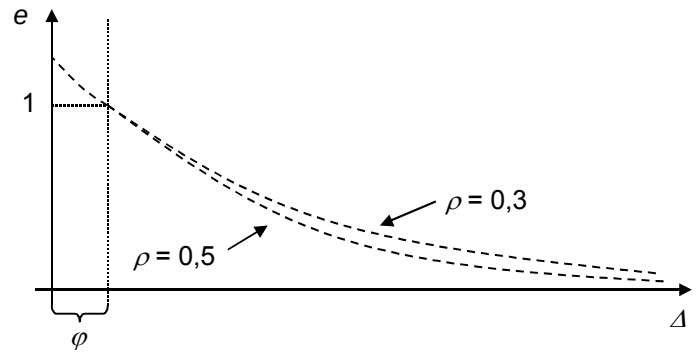


Abbildung 79: Vergessensfaktor ρ

κ liegt ebenfalls zwischen 0 und 1. (1-κ) legt fest, welcher Anteil des Support-Wertes durch eine einzige abweichende Erfahrung maximal vernichtet werden kann. Standard im Tool ist κ=0,6, d.h., dass 60% der Erfahrungen immer beibehalten werden, auch wenn die neuen Erfahrungswerte den alten vollkommen widersprechen (Δ→∞).

Aus den oben beschriebenen Faktoren ergibt sich die Unsicherheitsfunktion als:

$$U(t)=\kappa+(1-\kappa)e^{\rho^2-\left(\rho\cdot\frac{\Delta}{\varphi}\right)^2}$$

Drei Beispiele sollen die Arbeitsweise dieser Funktion verdeutlichen:

1) $\Delta = \varphi$, d.h., dass die Abweichung des Erfahrungswertes gleich dem Threshold ist. In diesem Fall ist $e^0 = 1$. Der Wert für κ ist egal, da die Funktion immer zu 1 wird. Es tritt also keine Unsicherheit auf (bzw. U = 1).
2) $\Delta < \varphi$, d.h., dass die Abweichung des Erfahrungswertes minimal ist. In diesem Fall ist $e^{(0,09;0)} > 1$. Dies stellt allerdings ein Problem dar, da dieser Fall der Normalfall sein sollte. Ohne weitere Einschränkungen würde die Support-Funktion schlagartig nach oben schnellen. Aus diesem Grunde werden durch die min(1,*U*)-Funktion Werte über 1 immer auf 1 reduziert. Somit gilt auch hier, dass U = 1.
3) $\Delta > \varphi$, d.h., dass die Abweichung des Erfahrungswertes größer als der Threshold ist. In diesem Fall ist $e^{(0;-\infty)} < 1$. Der Wert κ bestimmt nun, wie stark der Unsicherheitswert in die Support-Funktion mit eingehen soll. Bei κ = 0 könnte bei einem einzigen heftigen Ausreißer die Konfidenz schlagartig auf 0 sinken. Deshalb sind κ-Werte um die 0,6-0,8 sinnvoll. Es gilt U = (0,1)

Insgesamt ergibt sich somit folgende Support-Funktion:

$$sf^m_{i,n+1}(t_j) = \left(sf^m_{i,n}(t_j) + \upsilon_j(t_j)\right) \cdot \min\left(1, \left(\kappa + (1-\kappa)e^{\rho^2 - \left(\rho \cdot \frac{\Delta}{\varphi}\right)^2}\right)\right)$$

Die Support-Funktion hat natürlich Auswirkungen auf die monetäre Funktion. Bei niedrigem Support (also bei einer hohen Unsicherheit bezüglich der Erfahrungswerte) werden neue Erfahrungen stärker gewichtet. Dadurch wird eine Anpassung an sich ändernde Umstände sehr schnell erreicht. Die monetäre Funktion ist somit die durchschnittlich gewichtete Summe aus der historischen Kurve multipliziert mit der Support-Funktion und einem neuen Wert multipliziert mit dem Abstand zum gemachten Erfahrungspunkt x_k. Es gilt somit:

$$m^e_{i,n+1}(t_j) = \left(sf^m_{i,n}(t_j) \cdot m^e_{i,n}(t_j) + \upsilon_j(t_j) \cdot x_k\right) / \left(sf^m_{i,n}(t_j) + \upsilon_j(t_j)\right)$$

bzw. entsprechend für die temporäre Funktion:

$$d^e_{i,n+1}(t_j) = \left(sf^d_{i,n}(t_j) \cdot d^e_{i,n}(t_j) + \upsilon_j(t_j) \cdot x_k\right) / \left(sf^d_{i,n}(t_j) + \upsilon_j(t_j)\right)$$

Da jedes Fahrzeug neben den Kosten auch einen Gewinn erwirtschaften soll, muss zu den errechneten Kosten noch ein Gewinnaufschlag erfolgen. Hierzu besitzt das Fahrzeug zusätzliche Konfidenz- ($cf^m_e(t)$ für die monetären und $cf^d_e(t)$ für die temporären Kosten) und Pufferfaktoren (bf^m für die monetären und bf^d für die temporären Kosten). Die Konfidenzfaktoren normieren die Support-Werte auf die Werte zwischen 0 und 1 (in Abhängigkeit vom Schwankungsfaktor γ, der sich ebenfalls im Intervall von 0 bis 1 bewegt).

$$cf_e^m(t) = 1 - \gamma^{sf_{i,n}^m(t_i)}$$
$$cf_e^d(t) = 1 - \gamma^{sf_{i,n}^d(t_i)} \quad |0<\gamma<1$$

Ist der Schwankungsfaktor hoch (also schwanken die Werte über die Zeit sehr stark), ist der Konfidenzfaktor niedrig. Der Schwankungsfaktor γ wird individuell von jedem Fahrzeug vergeben. Schwanken die Werte für die monetären und die zeitlichen Kosten über die Zeit nur gering, hat er einen niedrigen Wert, bei starken Schwankungen hat er einen hohen Wert.

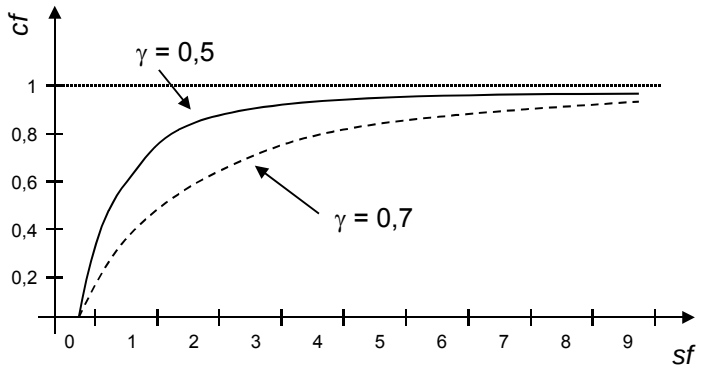


Abbildung 80: Schwankungsfaktor γ

Die Pufferfaktoren sind notwendig, um die Varianz der erwarteten Kosten aufzufangen. Diese Faktoren können sich je nach dem internem Zustand *Z* und dem Ziel *T* des Fahrzeugs ändern. Bei einem risikofreudigen Fahrzeug können die Puffer geringer sein als bei einem gewinnorientierten Fahrzeug. Der Sinn der Puffer ist darin begründet, dass bei Nichteinhalten der Constraints des Fahrgastes das Fahrzeug Kompensationszahlungen leisten muss. Bei großem Pufferfaktor hat das Fahrzeug somit einen höheren Spielraum.

Um den Puffer- und auch den Konfidenzfaktor in die Kostenfunktion zu integrieren, wird der Faktor λ eingeführt. Er beschreibt den Aufschlag auf die tatsächlichen monetären bzw. temporären Kosten einer errechneten Lösung. Grundsätzlich verstärkt sich dabei der Einfluss des Pufferfaktors auf den Faktor λ bei geringer Konfidenz und umgekehrt. λ bewegt sich innerhalb des Intervalls [1;+∞]. Für den Faktor λ gilt:

$$\lambda = 1 + bf\left(1 + \left(1 - cf\right)\right)$$

Umgerechnet für die monetären und temporären Kosten ergibt sich:

$$\lambda_e^m = 1 + bf^m(Z,T)\left(2 - cf_e^m(t)\right)$$
$$\lambda_e^d = 1 + bf^d(Z,T)\left(2 - cf_e^d(t)\right)$$

In die Kostenfunktion eingesetzt ergibt sich für die erfahrungsbasierte Lösung:

$$c_i^e(t) = \left(1 - \omega^c\right) \cdot \lambda_e^m m_i^e(t) + \omega^c \cdot \lambda_e^d d_i^e(t)$$

Mit dieser Kostenfunktion kann das Fahrzeug mit Hilfe des A*-Algorithmus [RN03] [Bec03] oder der Methode des Case-Based Reasonings [Ber02] einen aufgrund der derzeitigen Erfahrungen optimalen Weg finden. Die Entscheidung, welche Methode vom Fahrzeug angewendet wird, hängt hierbei vom Schwellenwert α ab, der den niedrigst zulässigen kumulierten Konfidenzfaktor einer Verbindung (Summe (1..n) aller Streckenabschnitte *e*) beschreibt. Für ihn gilt:

$$\alpha(t_j) = \sum_{i=1}^{n} \left(cf_e^m(t_j) + cf_e^d(t_j) \right) / 2n$$

Für Verbindungen mit α>0,5 wird die Methode des Case-Based Reasonings verwendet, da in diesem Falle bereits ausreichend Erfahrungen zur Auswahl einer Strecke vorliegen (Tabelle 20).

Betrachtet man zuerst den A*-Algorithmus, dann wird zusätzlich eine Schätzfunktion $h(n',v^z,t_{n'})$ zur Ermittliung der minimalen Kosten einer Verbindung unter Berücksichtigung von *n* (momentaner Standort) und dem Start- und Zielbahnhof verwendet. Bis zum Punkt *n* werden die exakten, tatsächlichen Kosten berechnet (also $c^e{}_i(t)$). Danach werden die Kosten zum Zielbahnhof v^z über den Folgepunkt *n'* zur Zeit $t_{n'}$ geschätzt (z.B. per Luftlinie).

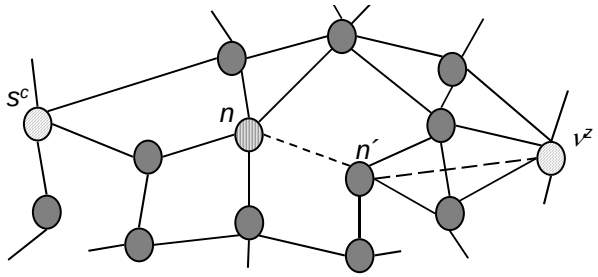


Abbildung 81: A*-Algorithmus

Da die Kostenfunktion abhängig von der Zeit ist, werden zu den Knoten die entsprechenden Zeiten t^c (Startzeitpunkt) und $t_{n'}$ (Zeit bei Erreichung des nächsten Knotens) mit berücksichtigt. Hieraus ergibt sich die Bewertungsfunktion:

$$c^{A^*}(n',t^c) = c(n',t^c) + h(n',v^z,t_{n'})$$

Diese Gleichung besagt: Vorausgesetzt, das Fahrzeug ist um t^c gestartet, wie hoch sind dann die tatsächlichen Kosten ($c(n',t^c)$) bis zum Punkt *n'* und wie hoch sind die zu erwartenden Restkosten ($h(n',v^z,t_{n'})$), wenn der Punkt n' zum Zeitpunkt $t_{n'}$ erreicht wird?

Die tatsächlichen Kosten für die beste Route bis zum Knoten *n'* werden durch die folgende Funktion berechnet:

$$c(n',t^c) = c(n,t^c) + c_{(n,n')}(t_n)$$

Diese Gleichung besagt: Vorausgesetzt, das Fahrzeug ist um t^c gestartet, wie hoch sind die tatsächlichen Kosten bis zum Punkt *n*, und wie hoch sind die tatsächlichen Kosten zwischen *n* und *n'* unter der Voraussetzung, dass das Fahrzeug zum Zeitpunkt t_n weiterfährt.

Das Verhalten von A* hängt stark von der heuristischen Funktion $h(n',v^z,t_{n'})$ ab. Falls diese keine korrekte Darstellung der minimalen Kosten von *n'* nach v^z abbildet, kann der Algorithmus frühzeitig abbrechen oder liefert nur eine suboptimale Lösung. Aus diesem Grunde wird die folgende Annahme zur Findung des Optimums getroffen:

$$h(n',v^z) \leq c_\pi \wedge \pi \in \Pi(n',v^z)$$

Zur Berechnung von *h* werden die Luftliniendistanz (Euklidische Distanz), die minimalen durchschnittlichen Kosten pro Kilometer $m_e^{avg}(t)$ und die Zeit $d_e^{avg}(t)$, jeweils bezogen auf die relevante Zeit $[t_{n'}, t^c+d^c]$, benötigt. Somit lautet die Definition für die heuristische Funktion $h(n',v^z,t_{n'})$:

$$\Delta(n',v^z) = \sqrt{(x_{n'} - x_{v^z})^2 - (y_{n'} - y_{v^z})^2}$$

$$h(n',v^z,t_{n'}) = \Delta(n',v^z)\left[(1-\omega_c)\cdot \min_{[t_{n'},t_c+d_c]}(m_e^{avg}) + \omega_c \cdot \min_{[t_{n'},t_c+d_c]}(d_e^{avg})\right]$$

Da die Findung der besten Route durch den A*-Algorithmus einige Zeit in Anspruch nehmen kann, hat das Fahrzeug die Möglichkeit, schnell auf erfahrene Verbindungen zurückzugreifen. Hierzu speichert das Fahrzeug häufig befahrene Verbindungen als Summe aller Streckenabschnitte in einer Fallbasis ab. „Häufig" bedeutet in diesem Fall ein kumulierter Konfidenzfaktor α oberhalb von 0,5.

π		e_i						α	c_i^π	df_i
ν_{12}	ν_{54}	S-46	S-34	S-56	S-12	S-23	S-122	0,5	100	0,25
ν_{12}	ν_{54}	S-35	S-56	S-1	S-34			0,8	120	0,34
ν_{12}	ν_{54}	S-344	S-345	S-23	S-45	S-48		0,9	110	0,41

Tabelle 20: Auswahl bekannter Verbindungen

In diesem Fall wählt das Fahrzeug die dritte Verbindung (grau hinterlegt) vom Startbahnhof ν_{12} zum Zielbahnhof ν_{54} aus.

Wählt das Fahrzeug diese Verbindung, werden dieselben Algorithmen und Funktionen zum Speichern der Erfahrungen, die bereits oben erstellt wurden, verwendet. Sollten dabei Abweichungen im Bereich der monetären und der temporären Kosten auftreten, die den kumulierten Konfidenzwert unter die 0,5 Grenze drücken, wird die entsprechende Verbindung aus der Fallbasis gelöscht.

Falls wenig oder gar keine Erfahrungen für eine Verbindung von A nach B zu einem bestimmten Zeitpunkt vorliegen, hat jedes Fahrzeug die Möglichkeit, neben der

eigenständigen heuristischen Berechnung einer Explorationsrichtung (siehe Kapitel 5.3.3.1) andere Fahrzeuge nach einer möglichst optimalen Route zu fragen. Diese Entscheidung, ob gefragt wird, hängt ab vom Schwellenwert α. Ist α kleiner als 0,1 (dieser Faktor wurde in der Anwendung so festgelegt, kann aber durch weitere Lernalgorithmen auch erfahrungsbasiert ermittelt werden), werden andere Fahrzeuge gefragt.

Die Frage nach einer besseren Verbindung kann in beiden Fällen willkürlich oder gezielt gestellt werden (hier dargestellt). *Gezielt* bedeutet, dass die Selektion der befragten Fahrzeuge nach einer Distanzfunktion erfolgt. Die Distanz zu den zu befragenden Fahrzeugen wird über spezifische Attribute ausgedrückt. Diese sind:

Typ =T = {Güterfahrzeug | Luxusfahrzeug}
Baujahr = B = {1980, 1981, 1982 ...}
Modell = M = {Siemens X5, ABB Z5,...}
Region = R = {Bayern, Hamburg, Sachsen}

Es werden nur die Fahrzeuge befragt, die einen Distanzwert unterhalb eines Schwellenwertes β besitzen (bei einem Distanzwert von 0 existiert eine 100%-ige Übereinstimmung des gefundenen Fahrzeugs mit dem befragten Fahrzeug). Die Abfrage dieser Werte geschieht über einen zentralen Servicebereich. In ihm sind innerhalb eines Case-Based Reasoning Systems alle Fahrzeuge nach den objektiven Attributen *Typ, Baujahr, Modell* oder *Region* gespeichert. Dabei werden die Distanzwerte von *Typ*, *Baujahr* und *Region* über die Hamming-Distanz-Funktion und das *Modell* (da ein Modell numerische Attribute wie die Drehzahl besitzt) über die Euklidische-Distanz-Funktion berechnet. Zusätzlich erfolgt eine individuelle Gewichtung der Distanzwerte von jedem Fahrzeug. So kann es sinnvoll sein, in einer bestimmten Region den Distanzwert der Region stärker zu gewichten, da ortskundige Fahrzeuge, ungeachtet welchen Baujahrs, Modells oder Typs sie sind, befragt werden sollen. Somit ergibt sich für β:

$$\beta = \xi \cdot dist_H^T\left(t_i, y_i^t\right) + \psi \cdot dist_H^B\left(b_i, y_i^b\right) + \chi \cdot dist_E^M\left(m_i, y_i^m\right) + \theta \cdot dist_H^R\left(r_i, y_i^r\right)$$

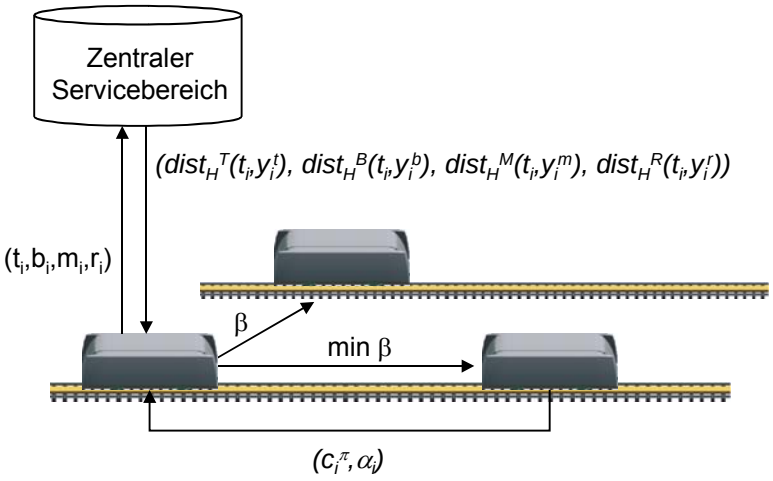


Abbildung 82: Auswahl des ähnlichsten Fahrzeugs

Hierbei sind ξ, ψ, χ und θ die Gewichtungen der Distanzwerte. $dist_H^T(t_i,y_i^t)$ ist der Distanzwert des Typs, $dist_H^B(b_i,y_i^b)$ der Distanzwert für das Baujahr, $dist_E^M(m_i,y_i^m)$ der Distanzwert für das Modell und $dist_H^R(r_i,y_i^r)$ der Distanzwert für die Region. Die Distanzberechnungen werden über den zentralen Servicebereich durchgeführt und sind somit unabhängig vom anfragenden Fahrzeug.

Die Antwort des befragten bzw. der befragten Fahrzeuge nach einer kostengünstigen Verbindung von A nach B beinhaltet neben den Kosten einer Verbindung c_i^π auch die entsprechenden Konfidenzfaktoren α_i, die zusammen die Bewertungsfunktion df_i ergeben:

$$ef_i = \left(\sum_{i=1}^{n} c_i^\pi \Big/ n \cdot c_i^\pi \right)^r \cdot \alpha_i^{\frac{1}{r}}$$

$$df_i = ef_i \Big/ \sum_{i=1}^{n} ef_i$$

ef_i formuliert den Kompromiss zwischen Kosten und Konfidenz. Der erste Faktor (in Klammern) repräsentiert die relative Attraktivität der ermittelten Kosten. Hierzu werden die Kosten aller Lösungen aufsummiert und durch die Multiplikation der jeweiligen Kosten und der Anzahl der Lösungen dividiert. Dadurch wird ermittelt, ob die Kosten über- oder unterdurchschnittlich attraktiv sind. Der zweite Faktor ist die Konfidenzsumme der Verbindung π. Mit dem Faktor r>1 werden die Kosten und mit r<1 wird die Konfidenz höher bewertet. Als endgültigen Vergleichswert der Lösungen erhält man so den Faktor df_i wie er in Tabelle 21 dargestellt (r = 1) ist.

Lösung	c_i^π	α_i	ef_i	df_i
S_{lokal}	22	0,6	0,64	0,23
S_1	28	0,5	0,42	0,16
S_2	24	0,8	0,78	0,29
S_3	32	0,9	0,66	0,25
S_4	12	0,1	0,19	0,07

Tabelle 21: Bewertung der Lösungen

In diesem Fall wählt das Fahrzeug die vorgeschlagene Verbindung des Fahrzeugs S_2.

5.3.3.2 Ergebnisse des Explorationsalgorithmus

Es wurde eine leistungsfähige Simulation implementiert, die individuell konfigurierbar ist. An jeder beliebigen berechenbaren Funktion kann der Algorithmus getestet werden. Das Umschalten zwischen verschiedenen Funktionen ist während des laufenden Betriebs problemlos möglich. Mehrere Algorithmen können gleichzeitig getestet und somit verglichen werden.

In Abbildung 83 ist die GUI der Simulation zu sehen. Es sind vier Fahrzeuge nebeneinander angeordnet. Das Fahrzeug auf der linken Seite kommuniziert nicht mit anderen Fahrzeugen.

Die anderen drei Fahrzeuge kommunizieren miteinander und tauschen ihre Erfahrungen aus. Wie an diesem Beispiel sehr gut zu erkennen ist, haben die drei miteinander kommunizierenden Fahrzeuge die zu lernende Funktion (z.B. reale Kostenfunktion) in kurzer Zeit gut approximiert, wohingegen beim einzelnen Fahrzeug die Funktion noch nicht erkennbar ist.

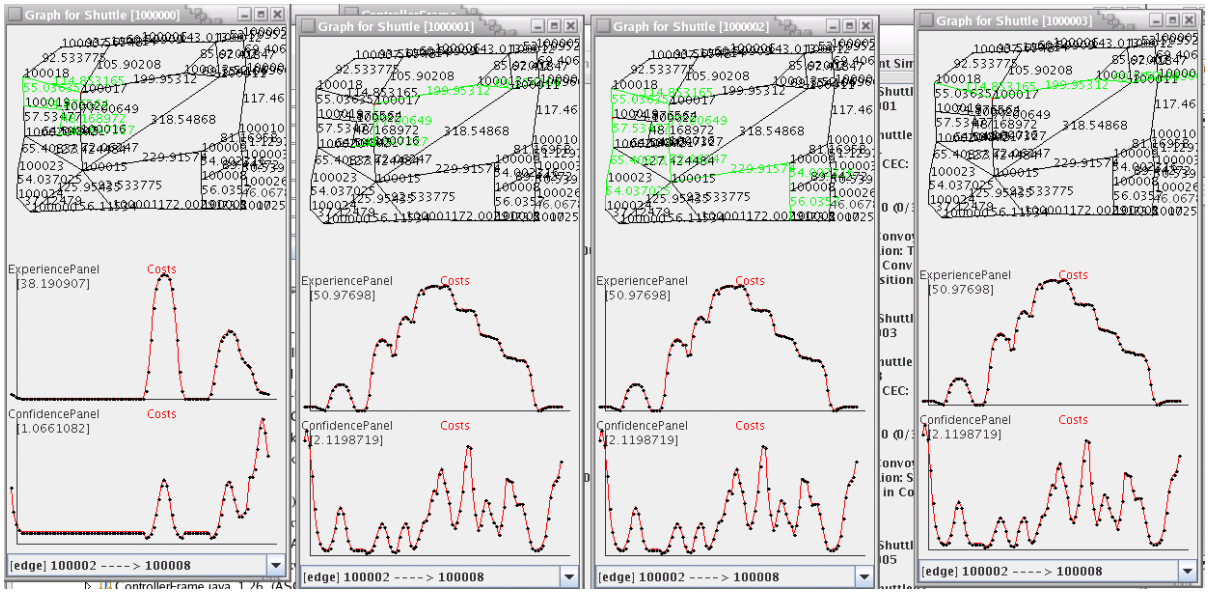


Abbildung 83: Verteiltes versus individuelles Lernen

Ein zweiter Test wurde mit dem Approximationsalgorithmus selbst vorgenommen. Hierzu zeigt Abbildung 84 die graphische Oberfläche der Testanwendung. Die obere Hälfte zeigt die zu approximierende Funktion (gepunktete Linie) und die errechnete Funktion (durchgezogene Linie) des Algorithmus. In der unteren Hälfte werden die zu den Daten korrelierenden Support-Daten angezeigt. In der Testanwendung lässt sich jederzeit die zu approximierende Funktion abändern. Dies kommt im Szenario einer, an einem Streckenabschnitt plötzlich auftretenden Änderung des Fahrverhaltens, z.B. durch eine Baustelle, gleich. Ebenfalls beliebig erweiterbar ist die Menge der auswählbaren Funktionen. Beliebige Funktionen können in die Testanwendung eingegeben werden, die der Algorithmus dann approximieren soll. Damit ist es möglich, Stärken und Schwächen des Algorithmus herauszufinden.

Insgesamt zeigte sich, dass sich der Algorithmus bereits nach fünf Durchläufen der „neuen Realität“ gut annähern konnte.

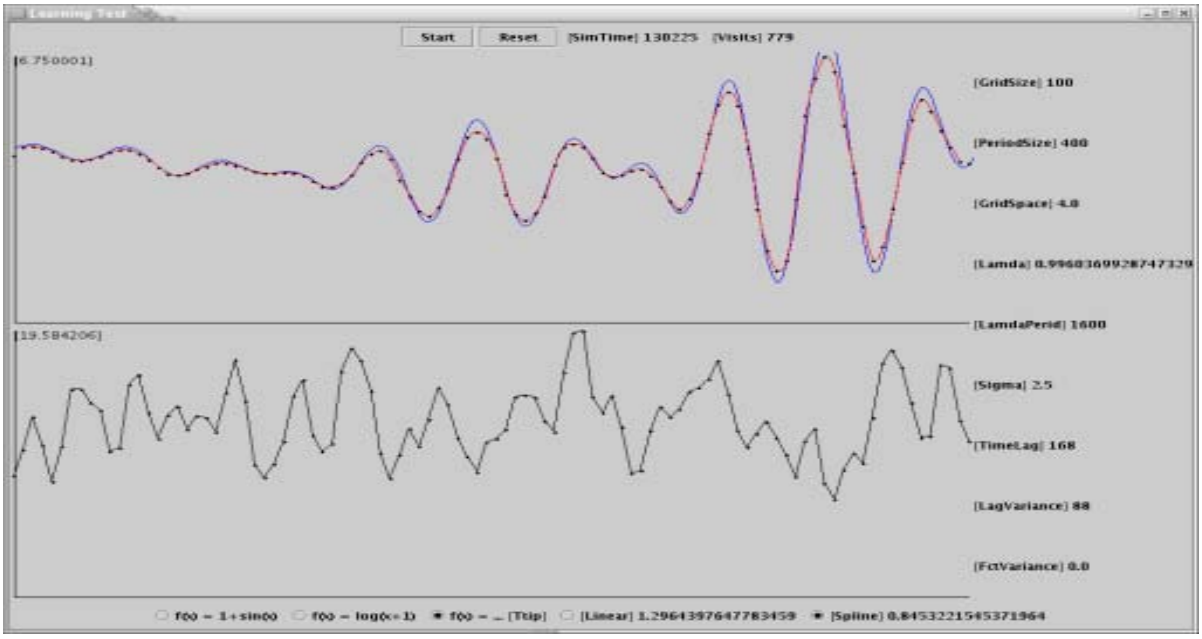


Abbildung 84: Ergebnisse des Approximationsalgorithmus

5.3.4 Bewertung der Explorationsstrategien

Die entwickelten Ansätze haben zwei unterschiedliche Betrachtungsweisen. In Kapitel 5.3.2 bestand das Ziel darin, eine Vorgabebahn und damit das Bewegungsverhalten eines technischen Systems innerhalb einer dynamischen Umwelt ständig anpassen und damit verbessern zu können. Die Umwelt (Beschaffenheit der Schiene, Wetterverhältnisse etc.) wurde hierbei als Black-Blox betrachtet. In Kapitel 5.3.3 bestand das Ziel darin, eine dynamische Umwelt möglichst realistisch darzustellen und dadurch das intentionale Verhalten ständig anpassen und verbessern zu können. Die unterschiedlichen Herangehensweisen sind damit begründet, dass das Bewegungsverhalten lediglich auf die dynamische Umwelt bestmöglich reagieren möchte, wohingegen das intentionale Verhalten seine Umwelt vorhersagen bzw. verstehen möchte.

Mit den in Kapitel 5.3.2 entwickelten Algorithmen ist es möglich, jeden beliebigen Bahnverlauf über eine Strecke erfahrungsbasiert zu verbessern. Voraussetzung hierfür ist, dass es sich bei den Szenarien um Streckenintervalle handelt, die unendlich oft wiederholt werden können. In dem hier untersuchten Szenario ist dies ein bestimmter Streckenabschnitt, der unendlich oft von den Fahrzeugen befahren werden kann. Der Streckenabschnitt besitzt hierbei ein abgeschlossenes Streckenintervall (z.B. 1000m). Weitere Beispiele, die gleiche Bedingungen aufweisen, sind zum einen ein Transportroboter mit Greifarm oder ein Einlagerer im Hochregallagersystem. Mit dem hier entwickelten Verfahren können ihre Transportbahnen erfahrungsbasiert verbessert werden.

Mit dem in Kapitel 5.3.3 entwickelten Algorithmus ist es möglich, jede Umweltsituation schnell zu approximieren. Voraussetzung hierfür ist, dass es sich bei dem Szenario um Zeitintervalle handelt, die sich unendlich oft wiederholen. In dem hier untersuchten Szenario ist dies die Tageszeit (24 Stunden eines Tages), die sich ständig wiederholt. So ist es möglich, Regelmäßigkeiten festzustellen (z.B. Stau immer um 9:00 Uhr) und entsprechende Konsequenzen daraus zu ziehen. Über die Approximation hinaus ist der Algorithmus in der Lage, Wahrscheinlichkeiten für das Eintreffen einer bestimmten Umweltsituation berechnen und Änderungen der Umwelt schnell berücksichtigen zu können. Dabei unterscheidet der Algorithmus in „natürliche“ und „unnatürliche“ Veränderungen. „Natürliche Veränderungen“ ergeben sich bei kleinen Änderungen zum Erwartungswert. Ob demnach das Befahren einer Strecke 25 oder 27 Minuten dauert, führt zu keinen Änderungen in der Wahrscheinlichkeitsberechnung des Eintretens einer bestimmten Situation. Treten jedoch „unnatürliche Änderungen“ auf (z.B. 2 Stunden statt 25 Minuten), hat dies großen Einfluss auf die Wahrscheinlichkeit, da die Unsicherheit der Aussage, dass das Befahren dieser Strecke 25 Minuten dauert, zunimmt. Grundsätzlich lässt sich der Algorithmus auf alle planerisch-explorativen Tätigkeiten mit fest definierten, sich wiederholenden Zeitintervallen zur Vorhersage einer bestimmten Wahrscheinlichkeit übertragen.

6 Realisierung

Die Realisierung wurde an zwei verschiedenen Umgebungen durchgeführt. Für die MFM-Ebene stand hierfür der Demonstrator „Feder- und Neigemodul“ zur Verfügung. Für die AMS-Ebene musste eine Simulationsumgebung entwickelt werden.

6.1 MFM-Ebene

Ein Fahrzeugmodell sowie ein Modell des Track[38]-Operators wurden mit der Entwurfsumgebung CAMeL[39] entworfen. Der Track-Operator entspricht der Bereichskontrolle und führt die Optimierung der Vorgabebahn für seinen Streckenabschnitt durch. Dazu erhält er vom Operator des Fahrzeugs eine Nachricht mit Beschleunigungs-Messdaten. Aufgrund dieser Daten kann die Güte der Vorgabebahn bewertet werden.

Abbildung 85 zeigt die Spezifikation des Track-Operators, also der Einheit, die der Bereichskontrolle entspricht. Der Track-Operator besitzt vier Parameter. Über den *splinePort* wird der Verbindungskanal für die Übertragung von Spline-Nachrichten festgelegt. Analog bestimmen *splineReqPort* und *estimationPort* die Kanäle für Anfrage-Nachrichten bzw. Komfort-Nachrichten. Der Parameter *basepoints* gibt die Anzahl der zu betrachtenden Stützpunkte der Vorgabebahn an.

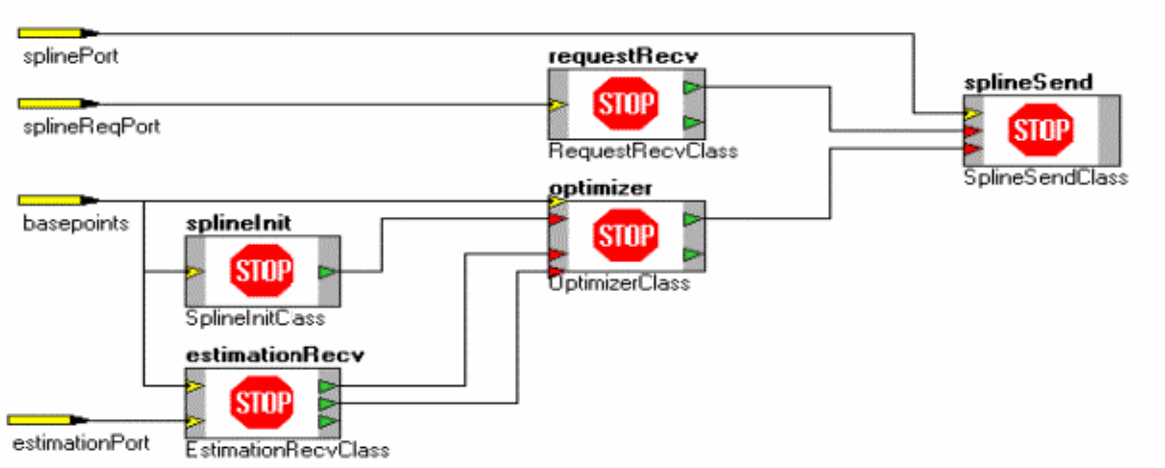


Abbildung 85: Track-Operator

Für jeden Streckenabschnitt wird eine Instanz dieses Programms gestartet. Die einzelnen Instanzen unterscheiden sich nur aufgrund ihrer *PUB-ID*, welche über die PUB-Bibliothek allen Prozessen eindeutig zugewiesen wird. Damit kann in dem Block *splineInit* eine individuelle Sollbahn (siehe Abbildung 87) für jeden Streckenabschnitt, abhängig von der PUB-ID, zugewiesen werden.

Die Kommunikation zwischen den Track-Operatoren und dem Fahrzeug-Operator verläuft über Nachrichten. Über die PUB-Funktion *bsp_sendmsg* kann eine Nachricht an einen

[38] Track = Streckenabschnitt
[39] http://www.ixtronics.de/Deutsch/CAMeLViewD.htm

anderen Prozess (bzw. an ein PUB-Programm) geschickt werden, wobei in der Nachricht zusätzlich ein Port angegeben werden kann. Über diesen Port kann der Typ der Nachricht kodiert werden. Das Senden der Nachrichten erfolgt innerhalb eines so genannten Supersteps, an dessen Ende eine Synchronisation aller Prozesse durchgeführt wird. Zu Beginn des nächsten Supersteps ist sichergestellt, dass alle zuvor verschickten Nachrichten beim Empfänger angekommen sind.

Die einzelnen PUB-Programme können echt-parallel auf verschiedenen Rechnern ausgeführt werden. Der Versand von Nachrichten wird über das TCP/IP-Protokoll realisiert. Zur Kommunikation enthält der Track-Operator drei Input bzw. Output Blöcke: *requestRecv* empfängt alle an den jeweiligen Track gerichteten Nachrichten, die über den *splineReqPort* verschickt werden. Die ID des Absenders wird aus dem Kopf der Nachricht ersichtlich. Der Block kann daraufhin die Sender-ID sowie die Empfangszeit über seine Ausgabeports ausgeben. Wurde in einem Simulationsschritt keine Nachricht empfangen, werden die Ausgabeports mit einem ungültigen Wert (hier -1.0) beschrieben.

Ganz ähnlich ist auch der Empfang von Komfortdaten realisiert. In einer Komfort-Nachricht sind aber zusätzlich Daten enthalten, die den Messwerten des Komforts in den einzelnen Intervallen zwischen den Stützpunkten entsprechen. Der Block *estimationRecv* empfängt diese Nachrichten, extrahiert daraus die Komfortdaten und speichert sie in einem Array ab. Da CAMeL im Wesentlichen nur In- und Outputs vom Typ Skalar oder Vector unterstützt, wird nicht das Array selbst, sondern ein Zeiger auf das Array auf den Ausgabeport gelegt. Dazu wird ein Union-Typ *__commit* deklariert (Listing 1, Zeilen 1 bis 4), in dessen Komponenten sowohl ein Zeiger, als auch ein *double* Wert gespeichert werden kann. Weist man nun der Zeiger-Komponente den Zeiger auf das Array zu (10), so kann dieser als Zahlenwert „maskiert“ an einen Out-Port übergeben werden (11).

Listing 1: union-Typ zur Übergabe von Zeigern

```
1.  typedef union {
2.      double dbl;
3.      void * ptr;
4.  } __committ ;
5.  [...]
6.  __committ comm;
7.  double out;
8.  double komf\_array[100];
9.  [...]
10. comm.ptr = komf_array;
11. out = comm..dbl;
```

Neben den beiden Blöcken zum Empfangen von Nachrichten existiert ein weiteres I/O-Modul zum Senden der Spline-Daten an das Fahrzeug. Der Block *splineSend* bekommt als Eingaben die PUB-ID des anfragenden Fahrzeugs. Ist diese gültig, dann wurde im selben

Simulationsschritt eine *splineRequest* Nachricht empfangen. Die aktuellen Spline-Daten werden daraufhin in ein Array kopiert und über den *splinePort* an die PUB-ID des Fahrzeugs verschickt. Im nächsten Superstep der Simulation stehen die Daten dann für das Fahrzeug zur Verfügung. Das Paket der Spline-Daten hat dabei immer eine festgelegte Form. Der erste Wert des Arrays (array[0]) wird als Anzahl der gültigen Datenwerte im Array gedeutet. Bei einem kubischen Spline mit sechs Stützstellen werden je sechs x- und y-Werte sowie die Steigungswerte im Start- und Endpunkt des Splines benötigt. Daraus ergibt sich, dass die ersten 15 Werte des Arrays gültig sind[40]. An zweiter und dritter Position des Arrays stehen die Start- bzw. Endsteigung, gefolgt von den einzelnen Stützpunktkoordinaten in der Form x_1, y_1, x_2, y_2, ..., x_n, y_n. Die Anzahl *n* der Stützstellen ergibt sich aus (array[0]-3)/2.

Das zentrale Element des Track-Operators ist der Block *optimizer*. Dieser bekommt als Eingabe eine individuelle Soll-Vorgabe, sowie jeweils die Komfortdaten des Fahrzeugs, das den zugehörigen Streckenabschnitt als letztes passiert hat. Daraus wird eine neue Vorgabebahn berechnet und diese dem Block *splineSend* zur Verfügung gestellt. Die Optimierung der Bahnvorgabe verläuft dabei nach demselben Prinzip wie im Kapitel 5.3.2 beschrieben.

Bei der Modellierung des Fahrzeugs wurde darauf geachtet, die OCM[41]-Struktur [HOG04] einzuhalten (siehe Abbildung 86). Auf unterster Ebene steht dabei der *Controller*. Hier findet die eigentliche Regelung der Stellsignale unter harten Echtzeitbedingungen statt. Ein Merkmal dieser Ebene ist die Verarbeitung von (quasi-) kontinuierlichen Daten. Die Eingabe des Reglers ist im Fall des Fahrzeugs z.B. die Absolutposition des Aufbau-Schwerpunkts. Aus diesen Vorgaben werden Stellgrößen der physischen Komponenten berechnet und über eine Messeinrichtung Rückführungsgrößen ausgegeben. Diese Rückführungsgrößen sind zum einen die gemessenen Beschleunigungsdaten (hier: $\ddot{z}_A(x)$) und zum anderen die Absolutposition des Fahrzeugs auf der Schiene (hier: x). Ein Controller kann sich aus mehreren Reglern zusammensetzen, zwischen welchen zur Laufzeit umgeschaltet werden kann.

Über den so genannten Reflektorischen Kreis ist der *Controller* mit dem *Reflektorischen Operator* gekoppelt. Auf dieser Ebene werden teilweise diskrete Daten unter Echtzeitaspekten verarbeitet. Die wichtigste Aufgabe des Reflektorischen Operators in diesem Szenario ist die Berechnung der Stellgröße u_{akt} als Eingabe der Regler. Dazu muss die Summe der Vorgabebahnen an der relativen Position des Fahrzeugs gelöst werden. Da als Sollbahn eine konstante Funktion $z_{ref}(x) = c$ ($z_{ref}(x) = 0 \forall x$) vorgegeben ist, genügt es die Bahn der Störgröße $z_S(x)$ zu betrachten. Die Sollbahn stellt hierbei diejenige Bahn dar, an der das Fahrzeug geführt werden soll. Das Fahrzeug unterliegt aber zusätzlich einer Schienenanregung, die aus der physischen Beschaffenheit der Schiene resultiert. Diese ist im Regelfall unbekannt. Schaltet man auf die Sollbahn eine so genannte Störgröße, die der Schienenanregung entgegenwirkt, auf, so kann im Ergebnis die eigentliche Sollbahn (Ist-Bahnverlauf) entwickelt werden (siehe Abbildung 87).

[40] 12 Werte für Stützpunktkoordinaten, zwei Steigungswerte und die Längenangabe selbst
[41] Operator-Controller-Modul

Zunächst muss aus der Absolutposition des Fahrzeugs eine Relativposition zum betrachteten Bereich berechnet werden. Da nur Streckenabschnitte gleicher Länge *l* betrachtet werden, wird die Relativposition einfach durch $x_{rel} = x \bmod l$ bestimmt. Die Regelgröße u_{akt} kann nun durch Lösen der Spline-Funktion an der Stelle x_{rel} unter Echtzeitbedingungen berechnet werden.

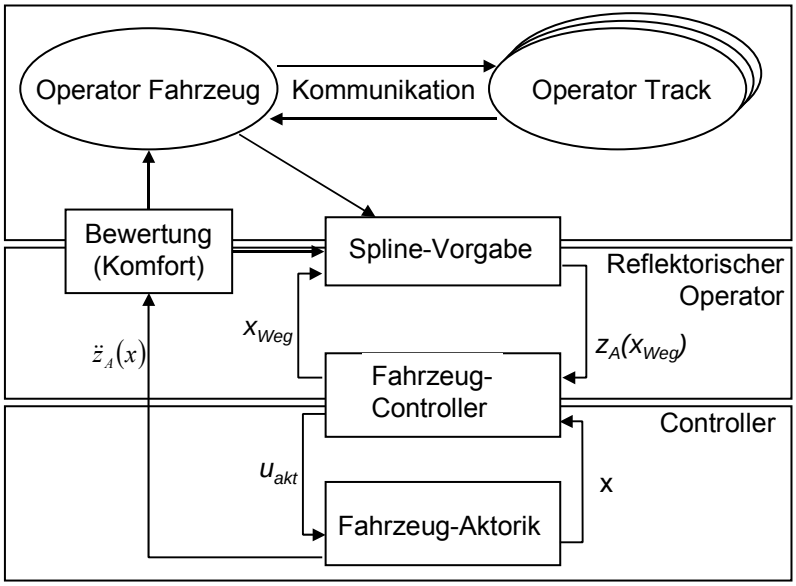


Abbildung 86: OCM-Struktur

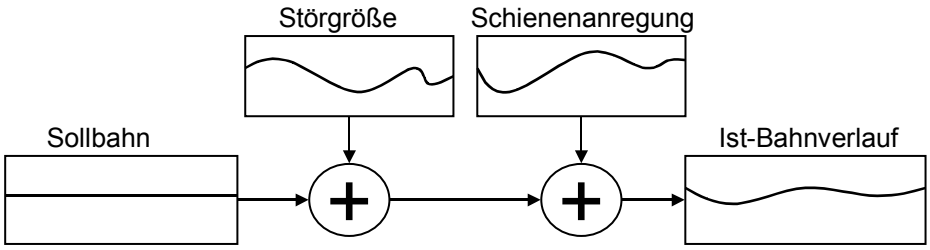


Abbildung 87: Sollbahn mit Störgrößenaufschaltung und Schienenanregung

Problematisch ist allerdings das Umschalten zwischen zwei Spline-Funktionen. Da das Lösen eines Splines erheblich rechenintensiver als die Berechnung der Spline-Funktion an einer Stelle *x* ist, muss dieses Lösen im Vorfeld erfolgen. So hält der Operator immer mindestens zwei Spline-Funktionen vor, die des aktuellen sowie die des nachfolgenden Bereichs. Bei einem Wechsel zwischen zwei Streckenabschnitten muss demnach zwischen den bereits gelösten Splines umgeschaltet werden.

Abbildung 88 zeigt, wie das Modell des Fahrzeugs in CAMeL aufgebaut ist. Der Block *xtest* gibt die Absolutposition des Fahrzeugs auf der Schiene aus. Es handelt sich dabei um eine konstante Funktion *f(t)=dx*t*, für $t \geq 0$. Der Parameter *dx* entspricht dabei der Stellgröße *m/sec*.

Diese absolute Position wird als Eingabe für das Fahrzeug sowie für die Schienenanregung benötigt. Die Umsetzung der Schienenanregung ist durch den Block *disturbance* realisiert. Abhängig von der Position wird in regelmäßigen Abständen etwa in der Mitte eines Streckenabschnitts eine Anregung ausgegeben. Bei betrachteten drei aufeinander folgenden Tracks und einer Tracklänge von 50m wurde alle 25, 75 und 125mfür eine gewisse Strecke eine Rechteckanregung aufgeschaltet[42].

$$x_{rel} = x \bmod 150$$

$$y = \begin{cases} 0.2\,für\, 25.0 \le x_{rel} \le 25.2 \\ 0.4\,für\, 75.0 \le x_{rel} \le 75.3 \\ 0.6\,für\, 125.0 \le x_{rel} \le 125.1 \\ 0.0\,sonst \end{cases}$$

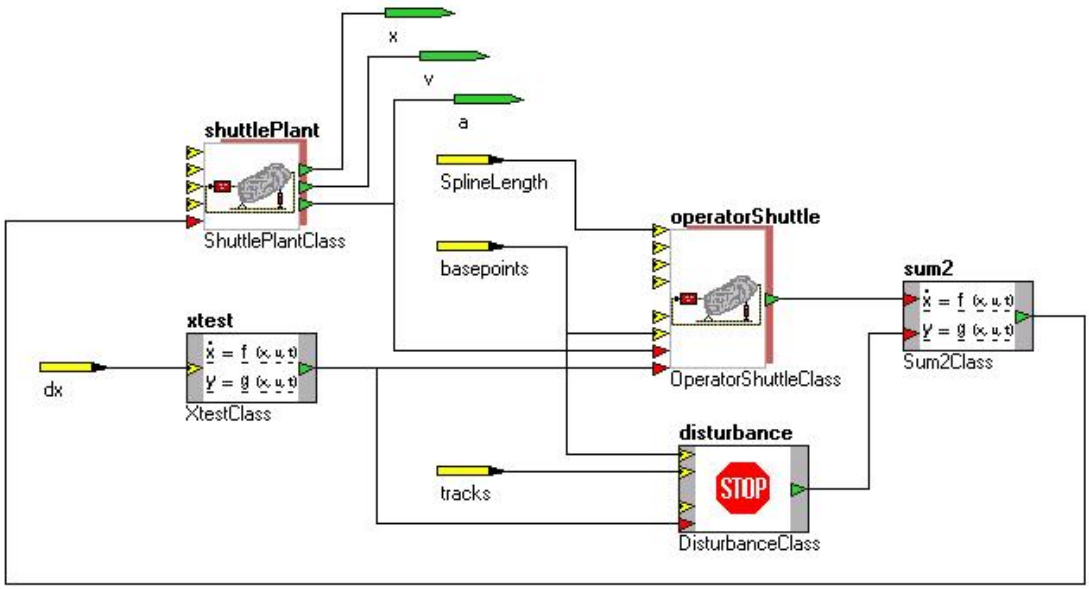


Abbildung 88: Aufbau des Fahrzeug-Modells

Der Block *operatorShuttle* ist ein hierarchisches Element. Die interne Struktur ist in Abbildung 89 dargestellt. Die Blöcke *splineRecv*, *splineSolve*, *actualSpline* und *splineEval* verarbeiten die vom Track-Operator gesendeten Vorgabebahn-Daten und ermitteln am Ende die Stellgröße für den Regler. In *splineRecv* werden die Spline-Daten empfangen und in einen Vektor kopiert. Eine Referenz auf diesen Vektor wird dem folgenden Block *splineSolve* übergeben. Dieser extrahiert daraus die Anzahl der Stützpunkte, die Anfangs- und Endsteigung sowie die Koordinaten der Stützpunkte. Daraus kann nun der Spline gebildet (gelöst) und als Referenz an *actualSpline* übergeben werden. Dieses rechenintensive Lösen der Spline-Funktion kann also während der Überfahrt des vorherigen Streckenabschnitts erfolgen.

[42] Bei weiteren Tests ist es problemlos möglich, eine strukturell andere Anregung, etwa eine Sinusfunktion, aufzuschalten. „0.2 sin(x_{rel}-69.0) für 69.0 ≤ xrel ≤ (69.0 + 4π)“ wäre eine solche Anregungsaufschaltung.

Der Block *splineSolve* ist durch das Umschalten zwischen zwei Streckenabschnitten getriggert. Tritt dieser Event auf, werden die Parameter des zuvor gelösten Splines in den Speicher des aktuellen Splines kopiert. Dieser Spline hält also zu jedem Zeitpunkt diejenige Spline-Funktion vor, die zum Befahren des aktuellen Streckenabschnitts benötigt wird. Der Spline wird als Pointer dem folgenden Block, *splineSolve*, übergeben. Abhängig von der relativen Position auf dem Streckenabschnitt wird der Spline an der Stelle x_{rel} gelöst. Das Ergebnis ist die absolute Strecke der Auslenkung und dient als Ausgabe des hierarchischen Blocks.

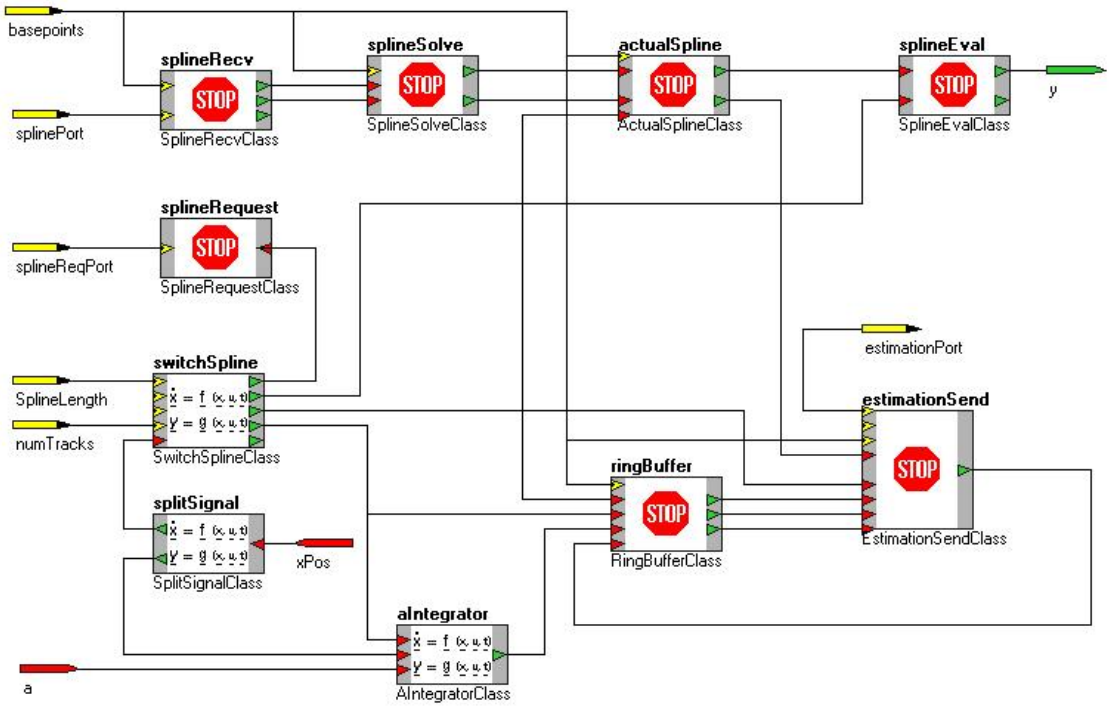


Abbildung 89: Aufbau des Fahrzeug-Operators

Ein wichtiges Element des Fahrzeug-Operators stellt der Block *switchSpline* dar. Hier wird, abhängig von der Absolutposition auf der Strecke (Eingangsport xPos), die relative Position x_{rel} berechnet. Des Weiteren produziert *switchSpline* Events jeweils zu den Zeitpunkten, wenn ein neuer Streckenabschnitt beginnt und wenn ein neuer Stützpunkt erreicht wird. Neben dem Umschalten zwischen zwei Spline-Funktionen (wie oben beschrieben) sind diese Events besonders für die Aufzeichnung der Messdaten (Komfort-Daten) wichtig.

Der Block *aIntegrator* ermittelt aus der Beschleunigung des Fahrzeug-Aufbaus (Eingangsport a) die Komfort-Daten. Dazu wird, solange sich das Fahrzeug zwischen zwei Stützstellen bewegt, das Beschleunigungssignal aufintegriert. Beim „Überqueren“ eines Stützpunktes wird das gebildete Integral

$$\int_{i}^{i+1} \ddot{z}$$

(mit i und i+1 aufeinander folgenden Stützstellen) in einen Speicher kopiert und die Integrationssumme zurückgesetzt. Die Speicherverwaltung übernimmt dabei der Block *ringBuffer*. Dieser Ring-Buffer ist ein Speicherbereich fester Größe, in dem die Messdaten von mindestens drei Streckenabschnitten gespeichert werden können (Abbildung 90).

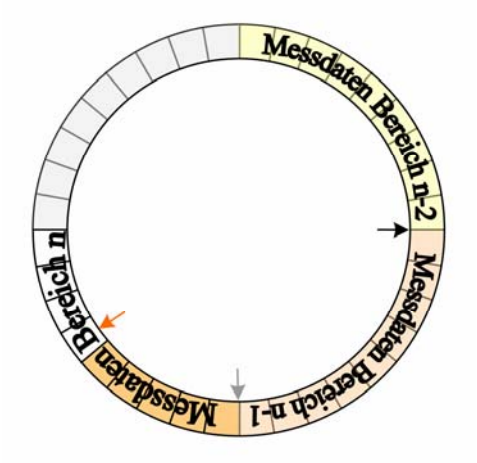


Abbildung 90: Ring-Buffer

Einem Spline mit *n* Stützstellen werden *n-1* Messdaten für die Bereiche zwischen den einzelnen Punkten zugeordnet. Der Ring-Buffer besitzt einen Zeiger, der die erste freie Speicherstelle anzeigt und einen weiteren Zeiger, der die erste noch nicht übermittelte Speicherstelle kennzeichnet. Erkennt der Block *ringBuffer* den Event „Neuer Stützpunkt“, übernimmt er den aktuellen Wert des Beschleunigungs-Integrators und speichert ihn an der aktuellen Schreibposition (Abbildung 90, linker Pfeil). Danach wird der Zeiger um eine Position verschoben. Überschreitet der Speicher das Ende des zugeordneten Bereichs, wird er vom Anfang her weiter beschrieben.

Der Event „Neuer Spline“ verarbeitet der Block *estimationSend*, indem er eine Nachricht mit Komfortdaten an den Operator des zuvor befahrenen Streckenabschnitts sendet. Dazu erhält er vom *ringBuffer* einen Pointer auf den Speicher der Komfortdaten und die Position des ersten noch nicht gesendeten Elements. Um die Grenze des Speicherbereichs nicht zu überschreiten, muss auch dieser Block die Größe des Speichers kennen. Wurde die Grenze zwischen Bereich *n-1* und *n* passiert, übernimmt der Block *estimationSend* die Werte beginnend mit dem Anfang des „ungesendeten“ Bereichs (Abbildung 90, rechter Pfeil) bis zu der Grenze des neuen Bereichs (grauer Pfeil). Die ID des letzten befahrenen Abschnitts wird vom Block zwischengespeichert; die ID des aktuellen Bereichs erhält er vom *actualSpline*. Die so erhaltenen Komfortdaten kapselt der *estimationSend* in eine Nachricht und schickt sie an den entsprechenden Track Operator. Anschließend teilt er dem Block *ringBuffer* mit, dass Daten versandt wurden, woraufhin dieser den Zeiger des nicht gesendeten Bereichs verschiebt.

6.2 AMS-Ebene

Es wurde eine Simulation entwickelt, um die Arbeitsweise der Algorithmen zu testen. Diese besteht aus einem konfigurierbaren Simulator, der simulierte Daten an die Algorithmen weiterleitet.

Abbildung 91 zeigt den schematischen Aufbau der Testanwendung. Der Simulator generiert kontinuierlich neue Erfahrungen. Diese werden von den Algorithmen verarbeitet und die daraus resultierenden Daten von der GUI angezeigt. Die wohldefinierten Schnittstellen sorgen dafür, dass das Arbeitspaket im Fahrzeug wieder verwendet werden kann.

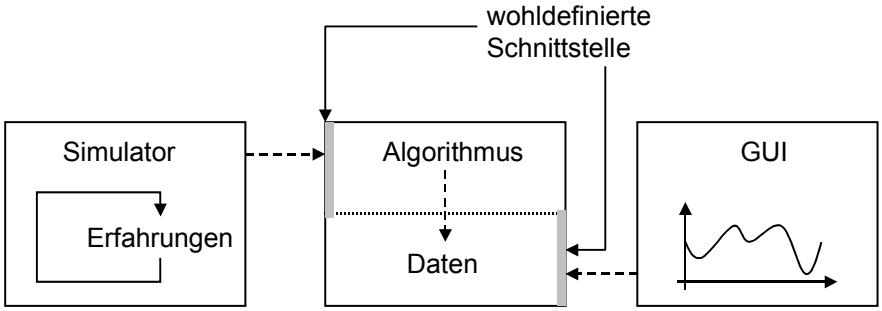


Abbildung 91: Aufbau der Testanwendung

Bei der Implementierung wurde darauf geachtet, dass wohl definierte Schnittstellen nach außen getragen werden, über die sich die Funktionalität der Algorithmen erschließt. Somit konnte das entwickelte Algorithmenpaket bei der Fahrzeugimplementierung wieder verwendet werden. Darüber hinaus ließen sich die Algorithmen dadurch leichter warten und auf ihre Richtigkeit hin überprüfen.

Die Testanwendung wurde unter den folgenden Anforderungen entwickelt:

Allgemein	
(S100)	Verschiedene Fahrzeugtypen
(S110)	Modularer Aufbau, um Algorithmen leicht austauschen zu können
(S120)	Verarbeitung von Nachrichten durch Modulsets
Auftragsverarbeitung	
(S200)	Angebotserstellung
(S201)	Routenplanung
(S202)	Modifizierter A*
(S210)	Auftragsverarbeitung
(S211)	Plandurchführung
(S212)	Nachbearbeitung/ Entgeldliche Abrechnung
Wissensverarbeitung	
	Datenstrukturen
(S300)	Streckennetz als gerichteter Graph
(S301)	Statische Constraints

(S302)	Spline-Verarbeitung für sich periodisch verändernde Constraints
(S310)	FahrzeugIDs den Fahrzeugtypen zuordnen
(S320)	Daten-Pools zum Datenaustausch
(S330)	Aufzeichnen von Messdaten
	Funktionell
(S340)	Kommunikation mit anderen Fahrzeugen zum Wissensaustausch
(S350)	Feedback erstellen
(S351)	Streckendaten-Recording
(S352)	Analyse gefahrener Strecken (Wissensgenerierung)
(S360)	Feedback von anderen Fahrzeugen verarbeiten
(S361)	Spline-Verarbeitung
(S370)	Laden bzw. Speichern der Fahrzeugwissensbasis

Tabelle 22: Anforderungen an das Fahrzeug

Um den Anforderungen (S100) und (S110) gerecht zu werden, wurde ein Framework entwickelt, das in mehrere Layer unterteilt ist (siehe Abbildung 92).

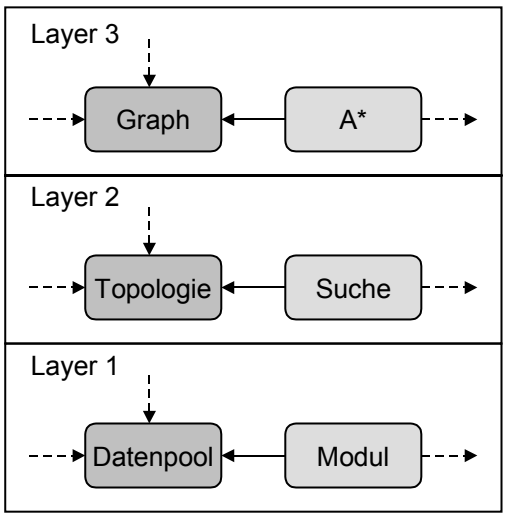


Abbildung 92: Das Framework der Simulation

Der unterste Layer definiert Typen, die grundlegende Funktionen zum Erfahrungsaustausch und zur Erfahrungsverarbeitung besitzen. Dabei wird hauptsächlich zwischen zwei Typen unterschieden: Modul und Daten-Pool. Ein Modul ist eine reagierende Einheit. Sie besitzt Schnittstellen, um Nachrichten zu empfangen, zu verarbeiten und neue Nachrichten zu versenden. Ein Daten-Pool ist ganz allgemein ein Container, der für Module Daten bereithält (S320). Er besitzt Schnittstellen, mit denen sich Module bei ihm registrieren können. Registrierte Module werden dann über Veränderungen an den Daten des Pools informiert. Im zweiten Layer werden die Fahrzeugkomponenten durch diverse Interfaces konkretisiert, so dass die einzelnen Module des Fahrzeugs über wohldefinierte Schnittstellen verfügen. In Abbildung 92 werden im zweiten Layer eine Suche und eine Topologie definiert. Dadurch, dass es nun verschiedene Implementierungen derselben Schnittstelle geben kann, werden die

Typen austauschbar. Diese sind im dritten Layer zusammengefasst. Im Beispiel wird die Suche zum A*-Algorithmus konkretisiert und die Topologie als Graph implementiert.

Abbildung 93 zeigt das Fahrzeug in der Layer 2 Ansicht. Die durchgezogenen Vierecke stellen Daten-Pools dar. Die gestrichelten Vierecke symbolisieren die Module. Der IOManager ist grau hinterlegt, da er sich zu keinem der beiden Typen zählen lässt. Er sorgt für das Zustellen der Simulationsnachrichten an die Module. Jedes Modul registriert sich bei dem IOManager für bestimmte Simulationsnachrichten. Daraus erstellt der Manager für jede Nachricht Modulsets. Empfängt das Fahrzeug nun eine Nachricht, leitet der IOManager die Nachricht an alle Module aus dem entsprechenden Set weiter (S120). Dadurch wird auch klar, dass die Module voneinander getrennt sein müssen. Es ist nämlich nicht vorhersagbar, in welcher Reihenfolge die Module die Nachricht zur Bearbeitung erhalten.

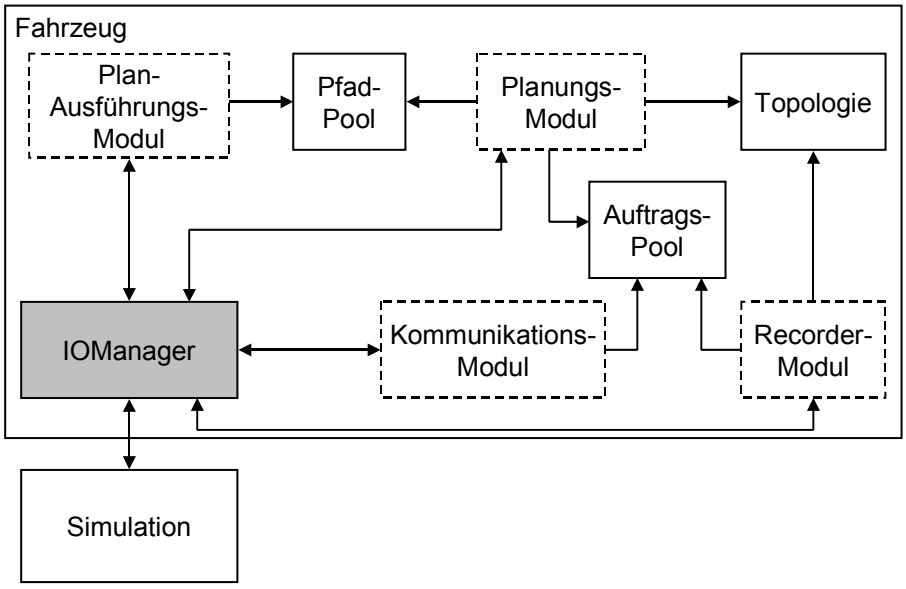


Abbildung 93: Fahrzeug-Framework

Die *Topologie* enthält einen gerichteten Graph (S300), der das Streckennetz repräsentiert. An den einzelnen Kanten sind die Profile gespeichert. Das Profil setzt sich dabei aus den dynamischen und den statischen Parametern zusammen. Da die Implementierung der Topologie nicht nach außen getragen wird, besitzt dieser Pool noch einen eigenen Suchalgorithmus. Dies ist ein modifizierter A*-Algorithmus (S202). Er wurde dahingehend verändert, dass er auch mit Constraints des Fahrgastes bzw. des Fahrzeugs umgehen kann (S301). Seine Gewichtungsfunktion ist so gebaut, dass Kanten auch abgelehnt werden können (z.B. aufgrund nicht erfüllter lokaler Constraints). Es ist, als würden diese Kanten im Graph nicht existieren.

Der *Pfad-Pool* enthält den Weg, den das Fahrzeug abfährt. Ein Weg besteht aus zusammenhängenden Streckenabschnitten und gegebenenfalls Aktionen an den Knotenpunkten zwischen den Abschnitten. Diese Aktionen beinhalten das Be- und Entladen, das Warten oder auch das Reparieren des Fahrzeugs.

Der *Auftrags-Pool* enthält alle zu bearbeitenden Aufträge. Der Pool ist so konstruiert, dass er mehrere Ansichten desselben Auftrages nach außen trägt, je nachdem welches Modul gerade auf den Pool zugreift. Zum Beispiel sind die vom *Recorder-Modul* aufgezeichneten Daten für das *Planungs-Modul* unerheblich und daher in seiner Ansicht nicht vorhanden.

Das *Plan-Ausführungs-Modul* arbeitet den Plan aus dem Pfad-Pool ab und reagiert auf eingehende Ankunftsnachrichten des Simulationskernels. Es schaut nach, ob an dem angekommenen Knoten eine Aktion durchzuführen ist. Wenn ja, wird dies der Simulation mitgeteilt und gewartet, bis die Aktion durchgeführt ist, bevor das Fahrzeug weiterfährt (S211).

Das *Planungs-Modul* übernimmt die Funktionen (S200), (S201) und (S210). Erhält das Fahrzeug einen neuen Auftrag, für den geboten werden muss, erstellt das Planungs-Modul ein Angebot aufgrund der eigenen Wissensbasis (S200) – falls Wissen vorhanden ist. Der potentielle Auftrag wird im Auftrags-Pool angelegt und mit den entsprechenden Daten versehen. Erhält das Fahrzeug nun den Zuschlag für das abgegebene Angebot, muss die Route neu geplant werden (S201). Die neu berechnete Route wird dann in den *Pfad-Pool* geschrieben. Ab diesem Zeitpunkt fährt das Fahrzeug die neue Route. Ist das Fahrzeug am Ziel eines Auftrages angekommen, wird von der Simulation automatisch das Konto des Fahrzeugs um die im Auftrag festgelegten Geldeinheiten aktualisiert (S212).

Das *Recorder-Modul* zeichnet während der Fahrt Daten auf ((S330) und (S351)). Diese werden im Auftrags-Pool den gerade aktiven Aufträgen zugeordnet. Diese Zuordnung dient später der Feedback-Generierung und auch zur Bildung der Fallbasis. Ist ein Auftrag abgeschlossen, analysiert das Recorder-Modul die im Auftrag gespeicherten Daten und bewertet diese. Es generiert also Wissen (S352). Dieses wird in die eigene Datenbasis eingepflegt (S212) und als Feedback an die anderen Fahrzeuge gesendet ((S340) und (S350)).

Das *Kommunikations-Modul* kommuniziert mit anderen Fahrzeugen. Es speichert die Attribute der anderen Fahrzeuge (S310), um eine Ähnlichkeitsbewertung (SIM) vornehmen zu können. Um zu verhindern, dass die Fahrzeugnachrichten den Durchsatz der Datenleitungen sprengen, wird bei der Angebotserstellung nur bei einem bestimmten Prozentsatz an Fahrzeugen angefragt. Diese Fahrzeugmenge wird im Auftrag abgespeichert, damit an genau diese Menge später das Feedback versendet werden kann. Kommt Feedback von anderen Fahrzeugen an, wird die Nachricht analysiert und das Wissen in die Topologie geschrieben (S360).

Die Kernel-Anbindung geschieht mittels einer vom Kernel bereitgestellten API und wohl definierten Nachrichtenprotokollen. Da der Kernel (zur Verfügung gestellt von der AG Schäfer[43]) ereignisbasiert arbeitet und Manipulationen desselben weitestgehend ausgeschlossen werden sollen, geschieht der Erfahrungsaustausch mit dem Kernel nur über Nachrichten. Das proprietäre Protokoll weist einige Besonderheiten auf. So wird vom Kernel in regelmäßigen Abständen ein so genannter „Dead Man-Switch“ verschickt. Auf diese

[43] http://wwwcs.uni-paderborn.de/cs/ag-schaefer/Lehre/

Nachricht muss das Fahrzeug innerhalb einer gewissen Zeit antworten. Tut es dies nicht, nimmt der Kernel an, dass das Fahrzeug in einem nicht arbeitsfähigen Zustand ist. Ein solcher Zustand kann z.B. durch eine Endlosschleife hervorgerufen werden. Der Kernel schließt dann das entsprechende Fahrzeug aus der Simulation aus.

Um bei einer Simulation nicht immer wieder von vorne starten zu müssen, ist es über die API (Application Programming Interface) möglich, die Topologie dazu zu veranlassen, den Datenbestand auf die Festplatte zu schreiben bzw. einen bestehenden Datenbestand von der Festplatte zu lesen (S370). Dazu wurde ein proprietäres Dateiformat auf XML Basis entwickelt.

7 Zusammenfassung und Ausblick

Für die erfahrungsbasierte Selbstoptimierung eines Fahrzeugs ist es notwendig, dass dieses Erfahrungen über sich und sein Umfeld abspeichert und daraus auf Handlungsempfehlungen für zukünftiges Verhalten schließen kann. Hierbei besteht die Problematik darin, dass sich das Fahrzeug zu früh auf ein „gutes, bereits mehrfach erprobtes“ Verhalten festlegt und das Optimum dadurch gar nicht gefunden wird. Zur Vermeidung dieses Zustands wurden Explorationsstrategien entwickelt, die neue, noch nicht erprobte Zustände untersuchten. Diese Explorationsstrategien wurden sowohl für die MFM- (am Beispiel des Feder- und Neigemoduls) als auch für die AMS-Ebene (am Beispiel der Streckenexploration) entwickelt.

Die Explorationsstrategie der MFM-Ebene bestand aus der Verschiebung der Stützpunkte einer Vorgabebahn für das Feder- und Neigemodul. Folgte das Feder- und Neigemodul dieser Bahn, so erfuhr das Fahrzeug (bzw. der Aufbau des Fahrzeugs) beim Befahren eines Streckenabschnitts eine bestimmte vertikale Beschleunigung. Über das Integral der Beschleunigung konnten die Komfortwerte berechnet werden. Die Komfortwerte wurden an den befahrenen Streckenabschnitt übermittelt, der aus diesen Werten eine neue Explorationsrichtung für die Stützpunkte der Vorgabebahn berechnete. Diese Bahn wurde an das nächste sich nähernde Fahrzeug übermittelt, das wiederum die erfahrenen Komfortwerte nach dem Befahren an den Streckenabschnitt schickte. So entstand eine verteilte Optimierung der Komfortwerte, die bereits nach 30 Durchläufen eine signifikante Verbesserung des Komforts bewirkte.

Auf der AMS-Ebene wurden mehrere Explorationsstrategien untersucht. Am Anfang war jedes Fahrzeug bei der Erkundung des Transportnetzes auf sich allein gestellt. So mussten die ersten Explorationserfahrungen teilweise blind und teilweise heuristisch gewonnen werden. Mit zunehmenden Erfahrungen baute sich im Fahrzeug für jeden Streckenabschnitt eine stundengenaue Kostenfunktion über die Woche auf, mit dessen Hilfe das Fahrzeug seine Gesamtkosten für eine Verbindung in Abhängigkeit der Zeit immer genauer nach realen Bedingungen berechnen konnte. Die Kostenfunktion bestand aus den monetären (z.B. Energieverbrauch, Streckengebühr etc.) und den temporären Kosten (Dauer des Befahrens des Streckenabschnitts). Der Sinn der ermittelten Kostenfunktion bestand darin, die Bedingungen (Constraints) des Fahrgastes mit den eigenen Zielen zu einer bestimmten Zeit möglichst optimal zu verbinden. Wurde z.B. das Ziel der Gewinn-Maximierung verfolgt, wurde den Gesamtkosten ein Pufferfaktor aufgeschlagen, der jedoch nicht so hoch sein durfte, dass der Auftrag des Fahrgastes nicht mehr bedient werden konnte. Allerdings durfte er auch nicht zu niedrig gewählt werden, um nicht unnötig Gewinn zu verschenken.

Waren insgesamt wenige Erfahrungen vorhanden, konnte eine Exploration auch mit den Erfahrungen anderer Fahrzeuge durchgeführt werden. Hierzu wurden deren Kosten- und Konfidenzwerte für bestimmte Verbindungen abgefragt und mit den eigenen verglichen. Waren die externen Kosten- und Konfidenzwerte günstig für das Fahrzeug, übernahm es die externen Explorations-Empfehlungen.

Sowohl für die MFM- als auch für die AMS-Ebene wurde bei der Exploration auf die Erfahrungen anderer Fahrzeuge zurückgegriffen. Besaßen alle Fahrzeuge eine identische Wissensbasis, stellte dies kein größeres Problem dar. In diesem Szenario wurde jedoch davon ausgegangen, dass die Fahrzeuge unterschiedliche Wissensbasen besitzen. Dies führte dazu, dass Situationen unterschiedlich wahrgenommen wurden. So ist für das eine Fahrzeug eine Kurvenfahrt bei 100 km/h gefährlich. Für ein anderes Fahrzeug beginnt diese Klassifizierung erst ab einer Geschwindigkeit von 200 km/h. Diese unterschiedlichen Wahrnehmungen führten zu unterschiedlichen Verhaltensweisen. Während z.B. das erste Fahrzeug vor der Kurve abbremste (da es 120 km/h fuhr), fuhr das andere Fahrzeug mit unverminderter Geschwindigkeit (120 km/h) weiter. Ohne das Wissen über die Wahrnehmung des anderen Fahrzeugs würde das zweite Fahrzeug somit einen Auffahrunfall verursachen (es sei denn, es kommt mit einer Vollbremsung noch rechtzeitig zum Stehen, was bei einem Personenfahrzeug allerdings nicht vorkommen sollte). Da es aber mit Hilfe des in dieser Arbeit entwickelten Algorithmus die Wahrnehmung des anderen „verstehen“ konnte, konnte es den Bremsvorgang des anderen antizipieren und frühzeitig bremsen. Der entwickelte Algorithmus löste in diesem Zusammenhang noch ein zweites Problem: die Kommunikation über unterschiedliche Terminologien. Werden Worte falsch interpretiert, kann es leicht zu gefährlichen Situationen kommen. Aus diesem Grunde leistete der entwickelte Algorithmus eine Übersetzung der heterogenen Terminologien mittels einer Basisontologie. So konnten Wahrnehmungen auch in der eigenen Terminologie übermittelt werden.

Ausblick

In dieser Arbeit wurde ein erster Ansatz für einen fahrerlosen, schienengeführten, bedarfsgesteuerten, sich selbstoptimierenden Transportverkehr geleistet. Die Fahrzeuge dieses Transportverkehrs sind in der Lage, ihr Verhalten den dynamischen Anforderungen der Umwelt eigenständig und online über Erfahrungen anzupassen.

Im Bereich der mechatronischen Funktionsmodule führte das Explorationsverfahren gemessen an der prozentualen Verbesserung der Komfortwerte einzelner Streckenabschnitte zu guten Ergebnissen. Es zeigte sich aber auch, dass eine große Menge an Stützpunkten (zum Beispiel 50 im Szenario) in der Regel zu vielen Optimierungsschritten führte. Jeder dieser Schritte verursacht im endgültigen System Kosten, da ein Schritt eine Fahrt eines Fahrzeugs über einen Streckenabschnitt bedeutet. Es ist daher wichtig, die Effizienz des Verfahrens zu steigern. Eine Möglichkeit besteht in der weiteren Partionierung des Streckenabschnitts. Dies bedeutet, dass mehrere oder sogar alle Stützpunkte gleichzeitig optimiert werden können. Ein solches Verfahren wurde bereits testweise am Demonstrator des Feder- und Neigemoduls implementiert und führte zu guten Ergebnissen. Ein großes Problem stellt hierbei die Auswertung eines Optimierungsschrittes dar. So kann nicht eindeutig ermittelt werden, welche Stützpunktverschiebung welchen Anteil an der Verbesserung des Komforts besitzt. Hier ist es erforderlich weitere Heuristiken zu finden, um die Komfortdaten besser auswerten und damit die Exploration verbessern zu können.

Im Bereich der autonomen mechatronischen Systeme ist das hier vorgestellte Fahrzeug in der Lage, einen möglichst optimalen Weg von A nach B zu planen, ohne jedoch bereits im

Vorfeld optionale Pläne zu berücksichtigen. Es wäre somit vorstellbar, dass das Fahrzeug bereits im Vorfeld mehrere optionale Pläne mit bestimmten Wahrscheinlichkeiten entwirft. Dies hätte den Vorteil, dass bei unerwarteten Ereignissen sofort der „nächst beste“ Plan zur Verfügung stünde. In der vorliegenden Arbeit plant das Fahrzeug erst einen neuen, optionalen Weg, wenn das Ereignis bereits eingetroffen ist. Hier wird Zeit vergeudet, die in Ruhephasen, in denen das Fahrzeug z.B. in einem Bahnhof steht, genutzt werden könnte. Um den Planungsbaum nicht zu groß werden zu lassen, muss zudem überlegt werden, ab wann ein optionaler Plan „abgeschnitten“ werden kann oder ob der erste Gesamtplan die komplette Verbindung überhaupt abdecken muss und nicht ein dynamischer Plan sinnvoller wäre.

Literaturverzeichnis

[Abd00-ol] Abdecker, A.: Wissensrepräsentation mit Semantischen Netzen und Frames. Vorlesungsunterlagen SS 2000 des DFKI, Kaiserslautern, unter: http://www.dfki.uni-kl.de/~aabecker/Mosbach/mosbach2000NetzeFrames.pdf

[AHK96] Arens, Y.; Hsu, C.; Knoblock, C.C.: Query processing in the SIMS information mediator. In: Advanced Planning Technology. AAAI Press California, USA, 1996.

[Ala03] Alan, Y.: Konstruktion der Kowien-Ontologie. KOWIEN-Projektbericht 2/2003, Institut für Produktion und industrielles Informationsmanagement Universität Duisburg-Essen, Essen 2003.

[Alb03-ol] Alber, R.: Einführung in die künstliche Intelligenz: Institut für Statik und Dynamik der Luft- und Raumfahrtskonstruktion, Universität Stuttgart, 2003, unter: http://www.isd.uni-stuttgart.de/ .

[Alp03] Alparslan, A: Ontologiebasiertes Wissensmanagement - Computergestütztes Management von Wissen über Mitarbeiterkompetenzen. Wirtschaftsforum 10, Institut für Produktion und industrielles Informationsmanagement, Universität Duisburg-Essen, Essen 2003.

[AN95] Aamodt, A.; Nygard, M.: Different roles and mutual dependencies of data, information and knowledge. Data and Knowledge Engineering 16, 1995, S. 191 – 222.

[Ang03] Angele, J.: Einsatz von Ontologien zur intelligenten Verarbeitung von Informationen, Industrie Management 19, 2003, GITO-Verlag, S. 53-55.

[ANS00] American National Standards Institute, Inc. ANSI B56.5: Safety Standard for Guided Industrial Vehicles and Automated Functions of Manned Industrial Vehicles. 2000.

[AP94] Aamodt, A.; Plaza, E. Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Communications 7, 1994, S. 39-59.

[AS03] Annamalai, M.; Sterling, L.: Dealing with Mathematical Relations in Web-Ontologies. In: Autonomous Agents and Multi Agent Systems, Melbourne Australien, 2003.

[BA95] Branting, L.; Aha, D.: Stratified Case-Based Reasoning: Reusing hierarchical problem solving episodes. In: Proceeding of the Fourteenth International Joint Conference on Artificial Intelligence, Providence, RI, USA, S. 384-390.

[Bac96] Bachmann, B.: A Solution for the Semantic Unification Problem to Reuse Knowledge-Based Systems, Infix, Sankt Augustin, 1996.

[BAP+95] Borst, P.; Akkermans, J.; Pos, A; Top, J.: The PhysSys ontology for physical systems. In: Bredeweg, B., (Hrsg.): Working Papers of the Ninth International Workshop on Qualitative Reasoning QR'95, University of Amsterdam, 1995, S. 11-21.

[Ban01-ol] Banzhaf, W.: Introduction to Machine Learning, Vorlesungsunterlagen WS01/02, Universität Dortmund, 2001, unter: http://univis.uni-

erlangen.de/formbot/dsc_3Danew_2Flecture_view_26lvs_3Dtech_2FIMMD_2FIMMD2_2Fmaschl_26dir_3Dtech_2FIMMD_2FIMMD2_26ref_3Dlecture

[Bay99] Bayer, K.: Argument und Argumentation. Logische Grundlagen der Argumentationsanalyse, Westdeutscher Verlag, Opladen/Wiesbaden, 1999.

[BE05] Buckley, J.J.; Eslami, E.: An Introduction to Fuzzy Logic and Fuzzy Sets, Physica-Verlag, 2005.

[Bec03] Beckstein, C.: Suche. In: Görz, G.; Rollinger, C.R.; Schneeberger, J. (Hrsg.): Handbuch der Künstlichen Intelligenz, Oldenbourg Verlag, München, 2003.

[BEP+00] Backhaus, K.; Erichson, B.; Plinke, W.; Weiber, R.: Multivariate Analysemethoden: Eine anwendungsorientierte Einführung, 9.-Auflage, Springer-Verlag, Berlin, 2000.

[Ber96] Bergmann, R.: Effizientes Problemlösen durch flexible Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen. Infix, Sankt Augustin, 1996.

[Ber02] Bergmann, R.: Experience Management. Foundations, Development, Methodology and Internet-Based Applications. Springer Verlag, Berlin, Heidelberg, 2002.

[Ber03a-ol] Bergmann, K.: Das PECS Referenzmodell – Modellierung menschlichen Verhaltens und menschlicher Entscheidungen. Kognitive Architekturmodelle, Sommersemester 2003. unter: http://www.techfak.uni-bielefeld.de/ags/wbski/lehre/digiSA/S03/KogMod/pecs.pdf

[Ber03b-ol] Bernd, R: Physikalische Größen und Einheiten, Physikalisches Praktikum für Anfänger (Haupfach) – Teil 2, unter: http://www.ieap.uni-kiel.de/surface/ag-berndt/lehre/aprakt2/einheiten.pdf

[Ber04] Bernemann, A.: Der Einsatz von Splines in der Finanzwirtschaft. Diplomarbeit in Zusammenarbeit mit der WestLB-Düsseldorf, FHDW Paderborn, 2004.

[BF95] Barbuceanu, M.; Fox, M.S.: COOL: A language for describing coordination in multi agent systems. In: Proceedings of the First International Conference on Multi-Agent-Systems (ICMAS-95), Menlo Park, Californien, AAAI Press, S.17-24, 1995.

[BFM+04-ol] Brach, T.; Frank, F.; Müller, C.; von Deuster, C.; Tchetchelnitski, V.: Gradientenverfahren in der globalen Optimierung. Referat im Fach „Mathematik und Informatik B“ für den Studiengang „Molekulare Biotechnologie“ an der Ruprecht-Karls-Universität Heidelberg, Sommersemster 2002, unter http://www.iwr.uni-heidelberg.de/~Markus.Kirkilionis/ProjekteB/Gradientenverfahren.pdf

[BGG04] Bleymüller, J.; Gehlert, G.; Gülicher, H.: Statistik für Wirtschaftswissenschaftler, Vahlen-Verlag, München, 2004

[BK03] Beierle, C.; Kern-Isberner, G.: Methoden wissensbasierter Systeme. Grundlagen, Algorithmen, Anwendungen. Vieweg-Verlag, Braunschweig, Wiesbaden. 2003.

[Böh74] Böhmer, K.: Spline-Funktionen. B.G. Teubner-Verlag, Stuttgart, 1974.

[Bor97] Borst, W.N.: Constructing of Engineering Ontologies for Knowledge Sharing and Reuse, PhD thesis, University of Twente, September 1997.

[BP94] Baumgartner, P.; Payr, S.: Lernen mit Software, Österreichischer Studien Verlag, Innsbruck, 1994.

[Bra00] Bratko, I.: Prolog Programming for Artificial Intelligence, Addison-Wesley, 2000.

[BS02] Borgida, A.; Serafini, L.: Distributed description logics. In: Proceedings of the International Description Logics Workshop DL'2002, 2002.

[BSW91] von Bechtolsheim, M.; Schweichhart, K.; Winand, U.: Expertensystem-werkzeuge – Produkte, Aufbau, Auswahl. Vieweg-Verlag, Braunschweig, 1991.

[Bün05] Kleine Büning, H.: Grundlagen Wissensbasierter Systeme, Folienscript zur Vorlesung, SS 2005.

[Bus99] Busse, S.: Non-standard Logiken zur Spezifikation von verteilten Systemen, Ausarbeitung im Seminar "Modellierungs- und Spezifikationsmethoden für verteilte Systeme", TU Berlin, 1999.

[BWC+03-ol] Bödcher, Wagner, Carstens, Könemann : Formelsammlung der Mechanik, Fachhochschule Hannover, 2003. unter: http://www.stud.fh-hannover.de/~koeneman/MECHANIK.pdf

[Cal03] Callan, R.: Neuronale Netze im Klartext, Pearson Studium, 2003.

[Cas95] Castelfranchi, C.: Commitments: From individual intentions to groups and organizations. In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), Menlo Park, Kalifornien, AAAI Press, S. 41-48, 1995.

[CH67] Cover, T.M.; Hart, P.E.: Nearest Neighbor Pattern Classification. In: Proceedings of the IEEE Transactions on Information Theory, 1967.

[Cha00] Chalupsky, H.: OntoMorph: A translation system for symbolic knowledge. In: Proceedings of the seventh International Conference on Principles of Knowledge Representation and Reasoning, Breckenridge, Colorado, USA, 2000.

[CL95] Cohen, P.R.; Levesque, H.J.: Communication actions for artificial agents. In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), Menlo Park, Kalifornien, AAAI Press, S. 65-72, 1995.

[Cla96] Claus, T.: Objektorientierte Simulation und Genetische Algorithmen zur Produktionsplanung und –steuerung, Verlag Peter Lang Frankfurt/M. 1996.

[Cor01] Cormen, T.H.: Introduction to algorithm. 2nd edition. Cambridge, Mass. MIT Press, 2001.

[CP02] Craw, S; Preece, A.: Advances in Case-Based Reasoning, Springer-Verlag, Heidelberg, 2002.

[Cyc04-ol] Cycorp Inc.: The Cyc knowledge base, unter: http://www.cyc.com/ .

[Dav97] Davenport, T.: Information Ecology: Mastering the Knowledge and Information Management, Oxford University Press, 1997.

[Dec98] Decker, S.: On Domain-Specific Declarative Knowledge Representation and Database Languages. In: Borgida, A., Chaudri, V., Staudt, M. (Hrsg.): Proceedings of the 5th International Workshop on Knowledge Representation meets Databases (KRDB'98), 1998.

[DF02] Ding, Y.; Foo, S.: Ontology research and development, Part 2 – A review of ontology mapping and evolving. In: Journal of Information Science, 28(5), 2002, S. 375-388.

[DFH+00] Decker, S.; Fensel, D.; van Harmelen, F.; Horrocks, I.; Melnik, S.; Klein, M.; Broekstra, J.: Knowledge representation on the web. In: Proceedings of the 2000 International Workshop on Description Logics, Aachen, Deutschland, 2000.

[Dil03-ol] Dilger, W.: Einführung in die Künstliche Intelligenz, Vorlesung an der Technischen Universität Chemnitz, Sommersemester 2003, unter: http://www.tu-chemnitz.de/informatik/HomePages/KI/skripte.php

[DM95] Dörner, D.; van der Meer, E. (Hrsg.): Das Gedächtnis. Probleme - Trend - Perspektiven. Göttingen: Hogrefe 1995.

[DMD+02] Doan, A.; Madhaven, J.; Domingos, P.; Halevy, A.: Learning to Map between Ontologies on the Semantic Web. In: Proceedings of the WWW2002, Honolulu, Hawaii, USA, 2002.

[DP98] Davenport, T.; Prusak, L.: Working Knowledge: How organizations manage what they know. Havard Business School Press, 1998.

[DS04] Dangelmaier, W.; Scheideler, P.: Solving Route Planning Problems with Experiences. In: Proceedings of the International Conference of Applied Simuation and Modelling, Rhodos, Griechenland, 28.-30.Juni 2004.

[EH04-ol] Eschenbach, C.; Habel, C.: Vorlesung Raum, Zeit, Ereignisse, WS 2003/2004, unter: http://www.informatik.uni-hamburg.de/WSV/teaching/vorlesungen/rze-Unterlagen/RZE01-Einleitung.pdf

[EH04] Elpelt, B.; Hartung, J.: Grundkurs Statistik, Oldenbourg, München 2004.

[Eps97] Epstein, S.: Spatial Representation for Pragmatic Navigation, In: Hirtle, S.; Frank, A. (Hrsg.): Spatial Information Theory – A theoretical basis for GIS, International Conference COSIT´97, Laural Highlands PA. Lecture Notes in Computer Science 1329, Springer Verlag, Berlin, 1997, S. 373-388.

[Fau96] Faulhaber, S.: Einsatz und Entwicklung von computerunterstützten Lernprogrammen in der medizinischen Aus- und Weiterbildung, Studienarbeit am Lehrstuhl für Künstliche Intelligenz und Angewandte Informatik der Universität Würzburg, 1996.

[Fer99] Ferber, J.: Multi-Agent-Systems. An Introduction to distributed artificial intelligence, Addison-Wesley, Harlow, 1999.

[Fer03] Ferber, R.: Information Retrieval - Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web, dpunkt.verlag, Heidelberg, 2003.

[Fel98] C. Fellbaum: WordNet – An electronic lexical database. MIT Press, Cambridge, Massachusetts and London, England, 1998.

[FFR97] Fikes, R.; Farquhar, A.; Rice, J.: Tools for assembling modular ontologies in Ontolingua. In: Proceedings of AAAI 97, 1997, S. 436-441.

[FGK+04] Frank, U.; Giese, H.; Klein, F.; Oberschelp, O.; Schmidt, S.; Vöcking, H.; Witting, K.: Selbstoptimierende Systeme des Maschinenbaus – Definitionen und Konzepte, HNI-Verlagsschriftenreihe, Band 155, Paderborn, 2004.

[FIP01] FIPA Ontology Service Specification, Dokumentennummer XC00086D, Foundation for Intelligent Physical Agents, 2001.

[Fip02] FIPA ACL Message Structure Specification, Dokumentennummer SC00061G, 2002.

[FMF+94] Finn, T.; McKay, D.; Fritzson, R.; McEntire, R.: KQML: An information and knowledge exchange protocol. In: Fuchi, K.; Yokoi, T. (Hrsg.): Knowledge Building and Knowledge Sharing. Ohmsha und IOS Press, 1994.

[FNP+99] Fowler, J.; Nodine, M.; Perry, B.; Bargmeyer, B.: Agent-based semantic interoperability in Infosleuth. Sigmod Records, 28, 1999.

[FSW+95] Fayyad, U.M.; Smyth, P.; Weir, N.; Djorgovski, S.: Automated analysis and exploration of image databases: Results, progress and challenges. Journal of Intelligent Information Systems , 4, 1995, S. 1-19.

[Gät04] Gäthke, S.: Immer pünktlich dank Charles Darwin, Financial Times Deutschland, Hamburg, 10.12.2004.

[Gau02] Gausemeier, J. Von der Mechanik zur Selbstoptimierung, 20th CAD-FEM Users Meeting, Friedrichshafen, Deutschland, 2002.

[GEF+99] Grosso, E.; Eriksson, H.; Ferguson, R. W.; Tu, S. W.; Musen, M. M.: Knowledge modeling at the millenium – the design and evolution of Protege-2000. In: Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management, Banff, Canada, 1999.

[Gen95] Genesereth, M.R.: Knowledge Interchange Format. Specification. Vorläufige Version eingereicht bei der American National Standard Kommission. 1995.

[GF92] Genesereth, M.R.; Fikes, R.E.: Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[GFH90] Gottlob, G.; Frühwirth, T.; Horn, W.: Expertensysteme, Springer Verlag, Wien/New York, 1990.

[GGM+03] Gangemi, A.; Guarino, N.; Masolo, C.; Oltramari, A.: Sweetening WordNet with Dolce, AI-Magazine 24(3), 2003, S. 13-24.

[GHR+03] Gu, D,; Hu, H.; Reynolds, J.; Tsang, E.: GA-based learning in behaviour based robotics. In: Proccedings of IEEE International Symposium on Conceptual Intelligence in Robotics and automation, Kobe, Japan, 16.-20. Juli, 2003.

[GJ02-ol] Gordon, J.; Jorgensen, L.: Knowledge Representation. Applied Knowledge Research Institute, 2002, unter: http://www.akri.org/museum/rep.htm

[GKK04] Gerdes, I.; Klawonn, K.; Kruse, R.: Evolutionäre Algorithmen, Vieweg Wiesbaden, 2004.

[GKM+04] Gawrilow, E.; Köhler, E.; Möhring, R.H.; Stenzel, B.: Conflict-free Real-time AGV Routing, Technische Universität Berlin, Institut für Mathematik, Technical Report 026-2004.

[GKS89] Gopal, S.; Klatzky, R.; Smith T.: Navigator: A Psychologically based Model of environmental Learning Through Navigation, Journal of Environmental Psychology, (9), 1989, S. 309-331.

[GL93] Glover, F; Laguna, M.: Tabu search. In: Reeves, C. (Hrsg.): Modern Heuristic Techniques for Computational Problems, Blackwell Scientific Publishing, Oxford, England, 1993.

[GL00] Ghosh, S.; Lee, T.: Intelligent Transportation Systems: New Principles and Architectures, CRC Press, Boca Raton, Florida, 2000.

[GO94] Gruber, T.R.; Olsen, G.R.: An Ontology for engineering mathematics. In: Doyle, J.; Torasso, P.; Sandewall, E. (Hrsg.): Proceedings of International Conference on Principles of Knowledge Representation and Reasoning, Mai, Bonn, Deutschland, 1994

[Gom98] Gomez, F.: Linking WordNet Verb Classes to Semantic Interpretation. In: Proceedings of the COLING-ACL Workshop on the Usage of WordNet on NLP Systems. Universite de Montreal, Quebec, Canada, 1998 S. 58-64.

[GR94] Goldman, C. ; Rosenheim, J. : Emergent coordination through the use of cooperative state-changing rules. In: Proceedings of the Twelfth National Conference on Artificial Intelligence, Philadelphia, PA, Morgan Kaufmann, S.408-413, 1994.

[Gru93] Gruber, T.: A translation approach to portable ontology specifications. In: Knowledge Acquisition, 5. Jg. (1993), Nr. 2, S. 199-220.

[GRS03] Görz, G.; Rollinger, C.-R.; Schneeberger, J. (Hrsg.): Handbuch der Künstlichen Intelligenz, Oldenbourg, München, 2003.

[Gua98] Guarino, N.: Some Ontological Principals for designing Upper-Level Lexical Resources. In: Proceedings of the First International Conference on Lexical Resources and Evaluation. May 28-30, Granada, Spanien, 1998.

[GW99] Ganter, B.; Wille, R.: Formal Concept Analysis: Mathematical Foundations. SpringerVerlag, München, 1999.

[GW00] Guarino, N.; Welty, C.: Identity, unity and individuality: Towards a formal toolkit for ontological analysis. In: Proceedings of ECAI-2000: Workshop on New Results in Planning, Scheduling, and Design. Amsterdam, Niederlande, 2000

[Har94] Hars, A.: Referenzmodelle – Grundlagen effizienter Datenmodellierung, Gabler, Wiesbaden, 1994.

[Häu03-ol] Häusler, K.G.: UiW – Unterricht im Web. Lexikon der Physikalischen Größen, HMTC – Halbmikrotechnik Chemie GmbH, 2003, unter: http://www.muenster.org/uiw/fach/physik/lexikon/allg/phygr.htm

[Hel96] Helbig, H.: Künstliche Intelligenz und automatisierte Wissensverarbeitung. 2. Auflage,Verlag Technik GmbH, Berlin, 1996.

[Hel01] Helbig, H.: Die semantische Struktur natürlicher Sprache - Wissensrepräsentation mit MultiNet. Springer, Heidelberg, 2001.

[Hen00 -ol] Hennauer, S.; Prolog – Programmieren mit Logik; unter http://www.stud.tu-muenchen.de/~sven.hennauer/studium/prolog/prolog_ausarbeitung.pdf;

[Her01] Herzog, C.: Zentralübung zu Einführung in die Informatik I, Technische Universität München, Wintersemester 00/01.

[HF97] Hamp, B.; Feldweg, H.: Germanet – a lexical-semantic net for German. In: Proceedings of the ACL Workshop on Automatic Information Extraction and

Building of Lexical Semantic Resources for NLP Applications, Madrid, Spanien, 1997.

[HHB00] Huber, F.; Herrmann, A.; Beckmann, S.C.: Affective influences on information processing style: A means-end analysis, ANZMAC 2000, Visionary Marketing for the 21st Century: Facing the Challenge, Gold Coast, Australien, 28.11. – 01.12.2000.

[HM00] Hendler, J.; McGuinness, D.L.: The DRAPA Agent Markup Language. IEEE Intelligent Systems 16(6), 2000, S. 67-73.

[HMO04] Hestermeyer, T.; Münch, E.; Oberschelp, O: Sollbahnplanung für schienengebundene Fahrzeuge. In: VDI-Tagung: Berechnung und Simulation im Fahrzeugbau, Würzburg, 2004.

[HO03] Hestermeyer, T.; Oberschelp, O.: Selbstoptimierende Fahrzeugregelung Verhaltensbasierte Adaption. 1. Paderborner Workshop Intelligente Mechatronische Systeme, Paderborn, 2003.

[HOG04] Hestermeyer, T; Oberschelp, O.; Giese, H.: Structured information processing for self-optimizing mechatronic systems. In: Araujo, H.; Viera, A.; Braz, J.; Encarnacao, B.; Cavalho, B. (Hrsg.): Proceedings of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal, IEEE Computer Science Press, 2004.

[Hol86] Hollier, R.H. (Hrsg.): Automated Guided Vehicles. Springer-Verlag, Heidelberg, 1986.

[HSH03] ten Hagen, S.; van Someren, M.; Hollink, V.: Exploration/Exploitation in Adaptive Recommender Systems. In: Proceedings of the European Symposium on Intelligent Technologies, Hybrid Systems and their Implementations in Smart Adaptive Systems, Finland, 2003.

[Hub00] Huber, H.-P.: Statistik und Datenauswertung – Lineare Regression, Department Chemie der Universität Basel, 2000.

[Hwa99] Hwang, C.H.: Incompletely and imprecisely speaking: Using dynamic ontologies for representing and retrieving information. Technical Microelectronics and Computer Technology Corperation (MCC), 1999.

[Hyp00-ol] Hyperwave: Einführung in die künstliche Intelligenz, unter http://www.iicm.edu/greif/node5.html.

[IFC03-ol] Institut für Chemie: SI-Einheiten. Freie Universität Berlin, 2003. unter: http://www.chemie.fu-berlin.de/chemistry/general/si.html.

[JM96] Jaanineh, G; Maijohann, M.: Fuzzy-Logic und Fuzzy-Control, Vogel Fachbuch, München, 1996.

[Kah01] Kahane, L.H.: Regression basics, Thousand Oaks, Californien, USA, 2001.

[Kel03] Kelly, J.: Logik im Klartext, Pearson Studium, München, 2003.

[Kis03] Kistner, K.-P.: Optimierungsmethoden. Physica-Verlag, Heidelberg, 2003.

[Kle01] Klein, M.: Combining and relating ontologies: An analysis of problems and solutions. In: Gomez-Perez, A.; Gruninger, M.; Stuckenschmidt, H.; Uschold, M. (Hrsg.): Workshop on Ontologies and Information Sharing, IJCAI'01, Seattle, USA 2001.

[KLW95] Kifer, J.-U.; Lausen, G.; Wu, J.: Logical Foundations of Object-Oriented and Frame Based Languages. Journal of the ACM, 42(4), 1995, S. 741-843.

[KMS+03] Kalinichenko, L; Missikoff, M; Schiapelli, F.; Skvortsov, N.: Ontological Modeling. In: Proceedings of the 5th Russian conference on Digital Libraries, RCDL2003, St.-Petersburg, Russland, 2003.

[Kno00] Knott, G.D.: Interpolating Cubic Splines, Birkhäuser-Verlag, Berlin, 2000.

[Kol93] Kolodner. J.: Case-Based Reasoning. Morgan Kaufmann Publisher, San Francisco, 1993.

[Kös00] Köster, A.: Modellbildung und Simulation unter Verwendung wissensbasierter Klassen, Technischer Bericht Nr. 2000-04, Universität der Bundeswehr München, Fakultät für Informatik, Dezember 2000.

[Kre03] Krengel, U.: Einführung in die Wahrscheinlichkeitstheorie und Satistik, Vieweg-Verlag, Wiesbaden, 2003.

[KTT01] Koller, F.; Thies, P.; Traphoener, R.: INVITE – Interaktive Mensch Technik Interaktion für die vernetzte Informationswelt der Zukunft. Workshop „Domänenmodellierung und Ontologien“, INIGraphicsNet, Darmstadt, 2001.

[Kui78] Kuipers, B.: Modeling Spatial Knowledge, Cognitive Science (2), 1978, S.129-153.

[Kur89] Kurbel, K.: Entwicklung und Einsatz von Expertensystemen – Eine anwendungsorientierte Einführung in wissensbasierte Systeme. Springer-Verlag, Berlin, 1989.

[KWZ98] Krahl, D.; Windheuser, U.; Zick, K.F.: Data Mining. Addison-Wesley, Bonn, 1998.

[LC01] Lämmel, U.; Cleve, J.: Lehr- und Übungsbuch Künstliche Intelligenz, Fachbuchverlag Leipzig im Carl Hanser Verlag, München, Wien 2001.

[Lee89] Lee, K.: Automatic speech recognition: The development of the Sphinx system. Kluwer Academic Publisher, Boston, 1989.

[Leh98 -ol] Lehner, C.; Prolog und Linguistik; unter: http://www.uni-hildesheim.de/~chlehn/p_ueb/probuch.pdf

[Lev03] Levitin, A.: The Design & Analysis of Algorithms. Addison Wesley, 2003

[LHL01] Lückel, J., Hestermeyer, T.; Liu-Henke, X.: Generalization of the Cascade Principle in View of a Structured Form of Mechatronic Systems. IEEE/ASMA International Conference on Advanced Intelligent Mechatronics (AIM 2001), Villa Olmo; Como, Italien, 2001.

[LKS00] Lückel, J.; Koch, T.; Schmitz, J.: Mechatronik als integrative Basis für innovative Produkte. VDI-Bericht 1533, Mechatronik – Mechanisch/ Elektrische Antriebstechnik, VDI-Verlag, Düsseldorf, 2000, S. 1-26.

[LS95] Lux, A; Steiner, D.: Understanding cooperation: an agent’s perspective. In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), Menlo Park, Kalifornien, IAAA Press, S. 261-268, 1995.

[LS99] Lassila, O.; Swick, R.R.: Resource Description Framework (RDF), Model and Syntax Specification. W3C Recommendation, Februar 1999.

[LZ89] Leiser, D; Zilberschatz, A: The Traveller – A Computational Model of Spatial Network Learning. Environment and Behaviour 21(4), 1989, S. 435-463.

[MAA+03] Mentzas, G.; Apostolou, D.; Abecker, A; Young, R.: Knowledge Asset Management - Beyond the Process-centred and Product-centred Approaches, Springer, Heidelberg, 2003.

[Mae03] Maedche, A.: Ontology Learning for the semantic web. Kluwer Academic Publisher, Dordrecht. 2. Auflage 2003.

[Mai03] A. Maier: Integration with Ontologies, Conference Paper WM2003, Luzern, April 2003.

[Mat97] Mates, B.: Elementare Logik: Prädikatenlogik der ersten Stufe. Vandenh. u. R., Göttingen, 1997.

[MBD+02] Madhaven, J.; Bernstein, P.A.; Domingos, P.; Halevy, A.Y.: Representing and Reasoning about Mappings between Domain Models. In: Proceedings of the AAAI Eighteenth National Conference on Artificial Intelligence, Edmonton, Canada, 2002.

[MBF+93] Miller, G.A.; Beckwith, B.; Fellbaum, C.; Gross, D.; Miller K.: Introduction to WordNet: An on-line lexical Database, revised Paper 1993. International Journal of Lexicography, Vol. 3 (1990), No. 4, S. 235-244.

[MCD99-ol] Mittelmann, A.; Csajtai, K.; Derntl, P.; Mayrhofer, D.; Pilsl, V.; Putz, P.; Schwab, M.: Wissensmanagement: Grundlagen - Modelle - Instrumente. Interner Arbeitsbericht der CC WPM Arbeitsgruppe Wissensmanagement. Linz 1999, unter: http://www.artm-friends.at/am/publications/km_ab1999.pdf

[MD84] McDermott, D.; Davis E.: Planning routes through uncertain territory. Artificial Intelligence, (22), 1984, S. 107-156.

[Mic00] Michalewicz, Z.: How to solve it – modern heuristics. Berlin, Springer-Verlag, 2000.

[MIK+96] Mena, E.; Illarramendi, A.; Kashyap, V.; Sheth A.P.: Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In: Proceedings of the First IFCIS International Conference on Cooperative Informations Systems, Brüssel, Belgien, 1996.

[Mil95] Miller, G.A.: WordNet: A lexical database for English. In: Communication of the ACM, Vol. 38 (1995), No. 11, S. 39-41.

[Min75] Minsky, M.: A Framwork for Representing Knowledge. The Psychology of Computer Vision, McGraw-Hill, Boston, 1975.

[Mit97] Mitchell, T.M.: Machine Learning, The McGraw-Hill, Boston, 1997.

[MRS82] McCalla, G.; Reid, L.; Schneider, P.: Plan Creation, Plan Execution and knowledge acquisition in a dynamic microworld. International Journal of Man-Machine Studies, (16), 1982, S. 89-112.

[MS01a] McGinty, L; Smyth, B.: Collaborative case-based reasoning: Applications in personalised route planning. In: Proceedings of the Fourth International Conference on Case-Based Reasoning, Berlin, 2001.

[MS01b] McGinty, L.; Smyth, B.: Collaborative CBR for Real-World Route Planning. In: Proceedings of the International Conference on Artificial Intelligence (IC-AI'01), Las Vegas, Nevada, 2001.

[MS01c] McGinty, L.; Smyth, B.: Identifying the Best Routes Travelled by Collaborative CBR, In: Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science (AICS-01), Maynooth, Ireland, 2001.

[Mur98] Murthy, K.S.: Automatic Construction of Decision Trees from Data – A Multi-disciplinary survey. Data Mining and Knowledge Discovery, 2(4), 1998, S.345-389

[NBP02] Neue Bahntechnik Paderborn. Eine Forschungsinitiative zur Steigerung der Attraktivität des Systems Bahn. Informationsbroschüre des Bahntechnik-Konsortiums, 2002.

[NBP03-ol] Neue Bahntechnik Paderborn, 2003, unter: http://www.neue-bahntechnik.de/index.php

[NFM00] Noy, N.F.; Fergerson, R.W.; Musen, M.A.: The knowledge model of Protege-2000: Combining interoperability and flexibility. 2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, Frankreich, 2000.

[Nis97] Nissen, V.: Einführung in evolutionäre Algorithmen. Vieweg, Braunschweig, 1997.

[NM99] Noy, N.F.; Musen, M.A.: SMART: Automated support for ontology merging and alignment, In: Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW), Banff, Alberta, Canada, 1999.

[NM00] Noy, N.F.; Musen, M.A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Proceedings of the 17th National Confrence on Artificial Intelligence, Austin, Texas, USA, 2000.

[NP02] Niles, I.; Pease, A. : IEEE Standard Upper Ontology (draft proposal). IEEE Working Group P1600.1, Teknowledge Corp. 2002.

[NT97] Nonaka, I.; Takeuchi, H.: Die Organisation des Wissens. Campus Verlag, Frankfurt am Main, 1997.

[OHK+02] Oberschelp, O.; Hestermeyer, T.; Kleinjohann, B.; Kleinjohann, L.: Design of Self-Optimizing Agent-Based Controllers. CFP Workshop 2002 – Agent Based Simulation 3. Universität Passau, 2002.

[ONe91] O´Neill, M.: A Biologically Based Model of spatial cognition and wayfinding. Journal of environmental Psychology (11), 1991, S. 299-320.

[Ort02-ol] Ortmann, Matthias: Die Anwendung von Evolutionsstrategien zur adaptiven echtzeitfähigen Trajektorienplanung von Manipulatorarmsystemen. Unter: http://www-brs.ub.ruhr-unibochum.de/netahtml/HSS/Diss/OrtmannMatthias/diss.pdf.

[Pag91] Page, B.: Diskrete Simulation. Eine Einführung mit Modula-2, Springer-Verlag, Heidelberg, 1991.

[Pfl01] Pflüglmayer, M.: Informations- und Kommunikationstechnologien zur Qualitätsverbesserung im Krankenhaus – Computerbasierte Terminologien als semantische Basis für medizinische Informationssysteme. Dissertation, Institut für Wirtschaftsinformatik, Universität Linz, 2001

[PGM99] Pinto, S.; Gomez-Perez, H.; Martins, J.P.: Some issues on ontology integration. In: Proceedings of IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons learned and Future Trends, Stockholm, Schweden, 1999.

[PHG+01] Preece, A.; Hui, K.; Gray, A.; Marti, P.; Bench-Capon, T.; Cui, Z.; Jones, D.: Kraft: an agent architecture for knowledge fusion. International Journal of cooperative information systems, 10(1/2), 2001, S. 171-196.

[PLL96] Prasad, M.V.; Lesser, V.; Lander, S.: Retrieval and reasoning in distributed case bases. Journal of Visual Communication and Image Representation, Special Issue on Digital Libraries, 7(1), 1996, S. 74-87.

[Pom89] Pomerleau, D.A.: ALVINN: An autonomous land vehicle in a neuronal network. Technical report CMU-CS89-107. Pittsburgh, PA: Carnegie Mellon University. 1989.

[PP96] Prasad, M.V.; Plaza, E.: Corporate Memories as Distributed Case Libraries. In: Proceedings of the Corporate Memory and Enterprise Modelling Track in the Tenth Knowledge Acquisition Workshop. Ban, Kanada, 1996.

[PRR03] Probst, G.J.B., Raub, S.; Romhardt, K.: Wissen managen, Gabler-Verlag, 2003.

[Pro03-ol] Profactor: Nutzen und Grenzen von Simulation, 2003, unter: http://www.profactor.at/SimWeb/NutzenGrenzen/NutzenGrenzen1.htm

[PSS03] Puppe, F.; Stoyan, H.; Studer, R.: Knowledge Engineering, In: Görz, G.; Rollinger, C.R.; Schneeberger, J. (Hrsg.): Handbuch der künstlichen Intelligenz, Oldenbourg, München, 2003.

[Pup91] Puppe, F.: Einführung in Expertensysteme. Springer-Verlag, Berlin, 1991.

[QD99] Quigley, E.J.; Debons, A.: Interrogative Theory of Information and Knowledge. In: Proceedings of SIGCPR'99, ACM Press, New Orleans, L.A., 1999, S. 4-10.

[Rau01] M. Raubal: Agent-based Simulation of Human Wayfinding: A Perceptual Model for Unfamiliar Buildings. Ph.D. Thesis, Vienna University of Technology, Wien. 2001

[Reu04] Reusch, B.: Darstellung, Verarbeitung und Erwerb von Wissen. Vorlesungsunterlagen zum WS 2003/2004 der Universität Dortmund.

[Ric95] Richter, D.: Ergebnisse methodischer Untersuchungen zur Korrektur des systematischen Messfehlers des Hellmann-Niederschlagsmessers. Berichte des Deutschen Wetterdienstes Nr. 194, Offenbach am Main, 1995.

[Ric03] Richter, M.M.: Fallbasiertes Schließen. In: Görz, C.; Rollinger, C.R.; Schneeberger, J. (Hrsg.): Handbuch der Künstlichen Intelligenz, Oldenbourg Verlag, München, 2003.

[Ril02] Rill, G.: Skript: Fahrzeugdynamik. 2002.

[RN03] Russell, S.; Norvig, P.: Artificial Intelligence. A modern Approach. Prentice Hall, New Jersey, 2003.

[RS89] Riesbeck, C.K.; Schank, R.C.: Inside Case-Based Reasoning. Lawrence Erlbaum Associates, 1989.

[RWT01 -ol] RWTH Aachen 2001, unter: http://www-i2.informatik.rwthaachen.de/lufgi2/programmierung/vorlesungsfolien.html

[Sac04] Sachs, L.: Angewandte Statistik, Springer-Verlag, Berlin, 2004.

[SB98] Sutton, R. S.; Barto, A. G.: Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.

[SC96] Smyth, B.; Cunningham, P.: The Utility Problem Analysed: A Case-Based Reasoning Perspective. In: Smith, I.; Falting, B. (Hrsg.): Proceedings of the American Associations of Artificial Intelligence Symposium on Agents with Adjustable Autonomy, 1996, S. 392-399.

[Sch99] Schekelmann, A.: Materialflusssteuerung auf Basis des Wissens mehrerer Experten, Dissertation, Universität Paderborn, 1999. HNI-Verlagsschriftenreihe, Band 47.

[Sch00-ol] Schneider, B.: KI in der betrieblichen Anwendung. Vorlesungsunterlagen Universität Münster, Lehrstuhl für Wirtschaftsinformatik und Interorganisationssysteme, 2000, unter: http://www.wi.uni-muenster.de/wi/studieren/wbs/ws01-02/index.cfm

[Sch01 -ol] Schmitt, D.: Zur Entstehung der Programmiersprache Prolog; unter http://www.dietmar-schmitt.de/essays/SGI/Prolog/inhalt.html

[Sch03] Schimming, R.: Grundlagen der Analysis: Logik, Mengenlehre, Arithmetik; Institut für Mathematik und Informatik Uni Greifswald; Vorlesungsunterlagen SS2003.

[Sch03a-ol] Schiele, B.: Maschinen Lernen II. Vorlesungsunterlagen der ETH Zürich, 2003, unter: http://www.vision.ethz.ch/ml/slides/2003-05-22.pdf .

[Sch03b-ol] Schmidt, B.: Künstliche Intelligenz. Sommercamp 2003, unter: http://www.or.uni-passau.de/schwarzes_brett/sc2003/KI/ .

[Sch04] Schmaltz, R.: Semantic Web Technologien für das Wissensmanagement, Arbeitsbereicht Nr. 1/2004 Georg-August-Universität Göttingen, Institut für Wirtschaftsinformatik, 2004.

[Sch04-o1] Scheiffert, N.: Spline in Theorie und Praxis, 01.12.2004, unter: http://www.mathematik.de/spudema/spudema_beitraege/beitraege/scheiffert/

[SFB04] Sonderforschungsbereich 614, Selbstoptimierende Systeme des Maschinenbaus, Antrag auf Finanzierung der 2. Förderperiode, Universität Paderborn, 2004.

[SHF96] Schöneburg, E.; Heinzmann, F.; Feddersen, S.: Genetische Algorithmen und Evolutionsstrategien. Addison-Wesley, Bonn, 1996

[Sho81] Shortliffe, E.H.: Consultation Systems for Physicians: The Role of Artificial Intelligence Techniques. In: Webber, B.L.; Nilsson, N.J. (Hrsg.): Readings in Artificial Intelligence. Tioga Publishing Company, Palo Alto, California, 1981, S. 323-333.

[Sie92] Siefkes, D.: Formalisieren und Beweisen, 2. Auflage, Vieweg, Braunschweig Wiesbaden, 1992

[SL95] Sandholm, T.; Lesser, V.: Issues in automated negotiation and electronic commerce: Extending the contract net framework, In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), Menlo Park, Kalifornien, AAAI Press, 1995 S. 328-335.

[SM01] Stumme, G.; Maedche, A.: FCA-MERGE: Bottom-up Merging of Ontologies. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, Washington, USA, 2001, S. 225-230.

[SM02] Mirtl, M.; Schentz, H.: Strukturen und Funktionen zur Abbildung interdisziplinärer Langzeitprojekte im Bereich von Ökosystem-Monitoring und -Forschung: Der Weg zum Hauptmenü von MORIS. In: Pillmann,W.; Tochtermann, K. (Hrsg.): Environmental Communication in the Information Society. International Society for Environmental Protection, Vienna, 2002, S. 106-117.

[Smi80] Smith, R.G.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers, C-29(12), S.1104-1113, 1980.

[Sow97] Sowa, J.F.: Electronic communication in the onto-std mailing list, 4th of December, 1997.

[SS97] van der Spek, B.R.; Spijkervet, A.L.: Knowledge Management: Dealing intelligently with knowledge, CIBIT, 1997.

[SS02] Studer, R., Stojanovic, N.: Wissensmanagement. Vorlesungsunterlagen zum SS 2002. unter: http://www.aifb.uni-karlsruhe.de/Lehrangebot/Sommer2002/Wissensmanagement/script.html#skript

[SS03a] Scheideler, P.; Schmidt, A.: Entwicklung einer Modellwelt für ein dezentrales, intelligentes, mechatronisches System. In: Gausmeier, J; Lückel, J.; Wallascheck, J. (Hrsg.): Intelligente mechatronische Systeme, 1. Paderborner Workshop Intelligente Mechatronische Systeme, 20.-21. März 2003 Paderborn, HNI-Verlagsschriftenreihe, Band 122, Paderborn, 2003.

[SS03b] Schmidt, T.; Schlender, D.: Untersuchung zum saisonalen Reifenwechsel unter Berücksichtigung technischer und klimatischer Aspekte. Projektbericht, Bergische Universität Wuppertal, Sicherheitstechnik (Fachgebiet Verkehrssicherheitstechnik), 2003.

[SSK+04] Schmidt, A.; Scheideler, P.; Koch, M.; Kleinjohann, B.; Gambuzza, A.; Münch, E.; Oberschelp, O.; Hestermeyer, T.: Neuro-Fuzzy Approaches for Self-Optimizing Concepts and Structures of Mechatronic Systems. In: Proceedings of International Conference on Computing, Communications and Control Technologies (CCCT 2004). August 14.-17., 2004, Austin, Texas, USA.

[Ste02] Steyer, R.: Wahrscheinlichkeit und Regression, Springer, Berlin, 2002.

[Ste04-ol] Stenzel, B.: Optimierungsverfahren zur dynamischen Routenführung in Verkehrs- und Transportnetzen; Technische Universität Berlin; 2004; unter: http://soft-pc3.zib.de/create_website?button=view&dc.identifier=03MONJB1&fkz=03MONJB1&pagetype=homepage&

[Ste03-ol] Stein, B.: Softwaretechnik für wissensintensive Aufgaben, Vorlesungsscript Sommersemester 2003, unter: http://wwwcs.upb.de/cs/ag-klbue/de/courses/ss04/confdiag04/

[Sto95] Stolz, W.: Starthilfe Physik: Ein Leitfaden für Studienanfänger der Naturwissenschaften, des Ingenieurwesens und der Medizin. Teubner-Verlag, Stuttgart, 1995.

[Sto00] Stone, P.: Layered Learning in Multiagent Systems. MIT-Press, Cambridge, 2000.

[Stu03] Stuckenschmidt, H.: Ontology-Based Information Sharing in weakly structured environments. Dissertation, Universität Amsterdam. 2003.

[SU00] Szczerbicka, H.; Uthmann, T.: Modellierung, Simulation und Künstliche Intelligenz, Frontiers in Simulation, SCS European Publishing House, 2000.

[Suh02] Suhl, L.: Grundlagen von Optimierungssystemen. Vorlesungsscript der Universität Paderborn, Fakultät Wirtschaftswissenschaften, 2002.

[SW69] Shannon, C.E.; Weaver, W.: The Mathematical Theory of Communication, Chapman and Hall, 4.Auflage, September 1969, S. 20-21,

[Tam01-ol] Tamma, V.A.M.: An Ontology Model supporting multiple Ontologies for Knowledge Sharing. Dissertation, The University of Liverpool, 2001, unter: http://www.csc.liv.ac.uk/~valli/Publ.html

[Tuw05-ol] Technische Universität Wien: B-Spline Kurven, 02.01.2005, unter: http://www.ads.tuwien.ac.at/docs/lva/mmgdv/k1___014.htm

[TW02] Tuschik, H.P.; Wolter, H.: Mathematische Logik, kurzgefasst. Spektrum Akademischer Verlag, Heidelberg, 2002.

[UG96] Uschold, M.; Grunninger, M: Ontologies: Principals, methods and applications. Knowledge Sharing and Review, 11(2)2, 1996, S. 93-155.

[Ung02] Unger, H.: Modellierung des Verhaltens autonomer Verkehrsteilnehmer in einer variablen städtischen Umgebung. Dissertation der Technischen Universität Berlin, 2002.

[Uni04 -ol] Universität Zürich, Institut für Computerlinguistik; unter: http://www.ifi.unizh.ch/cl/Glossar/Hornklausel.html

[Unt01-ol] Unterstein, K.: Ontologiebasierte Wissensextraktion, Vorlesung Wintersemester 01/02, Universität Dortmund 2001. unter: http://www-ai.cs.uni-dortmund.de/LEHRE/PG/PG402/seminar/OWE_2001.pdf

[VDI02] VDI Ausschuss A 127, VDI Richtlinie 2206: Entwicklungsmethodik für mechatronische Systeme, Stand: 06. August. 2002.

[Vis03-ol] Visek-Projekt: Virtuelles Software Engineering Kompetenzzentrum. 2003. unter: http://www.visek.de/?2409

[VJB+97] Visser, P.R.S.; Jones, D.M.; Bench-Capon, T.J.M.; Shave, M.J.R.: An Analysis of Ontology Mismatches: Heterogenity versus Interoperability. AAA/ Spring Symposium on Ontological Engineering, California, 1997.

[Vog04-o1] Vogt, T.: Approximation – Interpolation. Aktualisierungsdatum: 15.12.2004, http://www.hausarbeiten.de/faecher/hausarbeit/ma5/23491.html

[VT99] Visser, P.R.S.; Tamma, V.A.M.: An experience with ontology clustering for information integration. In: Proceedings of the IJCAI-99 Workshop on Intelligent Information Integration, Stockholm, Schweden, 1999.

[WDA99] Wolf, T.; Decker, S.; Abdecker, A.: Unterstützung des Wissensmanagements durch Informations- und Kommunikationstechnologie. 4. Internationale

Tagung Wirtschaftsinformatik 1999. Electronic Business Engineering. Saarbrücken, 03. - 05. März 1999..

[WE00] Witten, I. H.; Eibe, F.: Data Mining. Hanser Fachbuch, 2000

[Web94] Weber, G.: Fallbasiertes Lernen und Analogien. Beltz Verlags-Union, Weinheim, 1994.

[Wen05-ol] Wendland, M: Prinzipien der Analyse und Gestaltung von Mensch-Maschine-Systemen 2005, unter: http://www.mirko-wendland.de/skripte/aundo/aundo08.html

[Wes96] Weß, C.: Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik. INFIX-Verlag, Sankt-Augustin, 1996.

[Wie04-ol] Wiedemann, U.: Philosophenlexikon.de; unter: http://www.philosophen-lexikon.de/frege.htm

[Wii93] Wiig, K.M.: Knowledge Management Foundations: Thinking about thinking – How people and organizations create, represent and use knowledge, Vol. 1, Schema Press, Arlington, Texas, 1993.

[Wik04 -ol] Wikipedia; unter http://de.wikipedia.org/wiki/Hauptseite

[Win03-ol] Windischbauer, G.: Physikalische Basisgrößen und abgeleitete Größen. Institut für Medizinische Physik und Biostatistik. Veterinärmedizinische Universität Wien, 2003. unter: http://i115srv.vu-wien.ac.at/physik/ws95/w9521dir/w9521110.htm.

[WMJ03] Wrobel, S.; Morik, K; Joachims, T.: Maschinelles Lernen und Data Mining. In: Görz, G.; Rollinger, C.-R.; Schneeberger, J. (Hrsg.): Handbuch der Künstlichen Intelligenz, Oldenbourg, München, 2003

[Woo02] Wooldridge, M.: An Introduction to Multi-Agentsystems. John Wiley & Sons LTD, West Sussex, England, 2002.

[WVV+01] Wache, H.; Vögele, T.; Visser, U.; Stuckenschmidt, H.; Schuster, G.; Neumann, H.; Hübner, S.: Ontology-based integration of information - a survey of existing approaches. In: Stuckenschmidt, H. (Hrsg.): IJCAI-01 Workshop: Ontologies and Information Sharing, Seattle, WA, 2001, S. 106-117.

[Yan02] Yang, S.Y.: Personal Computer-Do It Yourself, Department of Electronic Engineering, National Taiwan University of Science and Technology, 2002

[YKN+03] Yeh, I.; Karp, P.D.; Noy, N.F.; Altman, R.B.: Knowledge Acquisition, Consistency Checking and Concurrency Control in Gene Ontology, Bioinformatics 19, 2003, S. 241-248.

[Zel02] Zelewski, S.: Wissensmanagement mit Ontologien – eine einführende Darstellung. Arbeitsbericht Nr. 15, Institut für Produktion und Industrielles Informationsmanagement, Uni Essen, Essen 2002.

Anhang

A Die Basisontologie

A.1 Klassenstruktur

Die unterschiedliche Anordnung in der Spalte *Klassen* symbolisiert die Klassenhierarchie. Slots, die in einer höheren Klassenhierarchie definiert wurden, werden aufgrund ihrer Vererbung in den unteren Klassen nicht mehr genannt.

Oberklasse	**Slots**
Entität der Basisontologie	Dokumentation, Bezeichnung

Klassen	**Slots**
1 Körperliche Entität	Hat_Merkmale, Hat-Ort, Hat_Zustand, Hat_Verhalten
2 System	Hat_Richtung, Hat_Ziel, Teil_von, Besteht_aus, DirektHinter, DirektVor, LinksVon, RechtsVon, Unter, Vor, Hinter, Auf, Verbunden_mit, Ähnlich_zu, Identisch_zu
3 Technisches System	
4 Technisches Fahrzeugsystem	
5 Bauteil	
6 Dämpfer	Dämpfungskonstante
6 Rad	
6 Feder	Federkonstante
6 Sensor	
6 Regler	
5 Mechatronisches Funktionsmodul	
6 Antriebs- und Bremsmodul	
6 Feder- und Neigemodul	
6 Fahrwerk	
6 Aufbau	
6 Spurführungsmodul	
5 Autonomes Mechatronisches System	Transportgut, Fährt_auf
6 Schienenfahrzeug	
6 Straßenfahrzeug	
6 Wasserfahrzeug	
6 Luftfahrzeug	
5 Vernetztes Mechatronisches System	
6 Konvoi	
4 Technisches Streckensystem	
5 Fahrbahnsystem	Anfangspunkt, Endpunkt
6 Streckenabschnitt	
6 Schiene	
6 Verbindung	Startbahnhof, Zielbahnhof
5 Knotenpunktsystem	
6 Bahnhof	Gleise
6 Weiche	
6 Depot	Depotplätze
5 Streckenelementsystem	
6 Schranke	
6 Ampel	
3 Organisches System	

4 Mensch	Auftrag, Alter
5 Privater Benutzer	
6 Fahrgast	
5 Geschäftlicher Benutzer	
6 Unternehmer	
4 Pflanze	
4 Tier	
2 Material	
3 Metalle	
4 Eisen	
4 Aluminium	
4 Kupfer	
4 Blei	
4 Chrom	
3 Nicht-Metalle	
4 Glas	
4 Porzellan	
4 Graphit	
4 Keramik	
3 (nicht-biologische) Naturstoffe	
4 Holz	
4 Kohle	
4 Gummi	
4 Sand	
4 Kies	
1 Abstrakte Entität	
2 Einheit	Abkürzung, Referenzeinheit, Beziehung
3 Physikalische Einheit	
4 SI-Einheit	
5 Physikalische Basiseinheit	
6 Meter	
6 Sekunde	
6 Kilogramm	
6 Ampere	
6 Kelvin	
6 Candela	
6 Mol	
5 Physikalische Funktionseinheit	
6 Hertz	
6 Newton	
6 Pascal	
6 Joule	
6 Watt	
6 Coulomb	
6 Volt	
6 Ohm	
6 Farad	
6 Tesla	
6 Henry	
6 Gray	
6 Bequerel	
5 Zusätzliche SI-Einheit	
6 Quadratmeter	
6 Kubikmeter	
6 Grad	
6 Kilometer/Stunde	
6 Newtonmeter	
6 Prozent	
3 Erweiterte Physikalische Einheit	

4 Längeneinheit	
5 Inch	
5 Fuß	
5 Meile	
5 Zoll	
5 Yard	
5 Millimeter	
5 Seemeile	
5 Zentimeter	
5 Kilometer	
4 Masseeinheit	
5 Tonne	
5 Gramm	
5 Milligramm	
5 Pfund	
4 Stromstärkeneinheit	
5 Kiloampere	
5 Milliampere	
4 Temperatureinheit	
5 Celsius	
5 Fahrenheit	
4 Zeiteinheit	
5 Minute	
5 Stunde	
5 Tag	
5 Woche	
5 Jahr	
4 Lichtstärkeneinheit	
5 Kilo-Candela	
4 Stoffmengeneinheit	
5 Kilo-Mol	
3 Monetäre Einheit	
4 Euro	
4 US-Dollar	
4 Pfund	
2 Größe	Wert, Formelzeichen, Formel, Parameter, Einheit, Gleich, Größer, GrößerGleich, Kleiner, KleinerGleich
3 Monetäre Größe	
4 Einnahme	
4 Ausgabe	
5 Streckengebühr	
5 Fahrtpreis	
5 Transferzahlungen	
3 Logistische Größe	
4 Logistische Fahrzeuggröße	
5 Pünktlichkeit	
5 Ruhezeit	
5 Fahrtzeit	
5 Lieferservice	
4 Logistische Verbindungsgröße	
5 Auslastung	
5 Kapazität	
5 Durchsatz	
3 Physikalische Größe	
4 Physikalische Basisgröße	
5 Länge	
5 Masse	
5 Zeit	

5 Stromstärlke	
5 Temperatur	
5 Stoffmenge	
5 Lichstärke	
4 Physikalische Funktionsgröße	
5 Mechanische Größe	
6 Drehimpuls	
6 Dichte	
6 Fläche	
6 Durchmesser	
6 Impuls	
6 Viskosität	
6 Trägheitsmoment	
6 Bruchfestigkeit	
6 Winkel	
6 Tiefe	
6 Steigung	
6 Krümmung	
6 Breite	
5 Elektrische Größe	
6 Leitwert	
6 Stromdichte	
6 Feldstärke	
5 Magnetische Größe	
6 Fluss	
6 Induktion	
6 Feldstärke	
5 Thermodynamische Größe	
6 Wärmemenge	
6 Wärmekapazität	
5 Optische Größe	
6 Lichstärke	
6 Lichstrom	
6 Leuchtdichte	
5 Atomare Größe	
6 Grad	
6 Aktivität	
5 Chemische Größe	
6 Molare Masse	
6 Stoffmenge	
1 Örtliche Entität	
2 Raum	
3 Ort	
4 Stadt	
4 Nation	
4 Provinz	
4 Kontinent	
4 Insel	
3 Stellung	
4 vertikal	
4 horizontal	
3 Graph	
4 x-Wert	
4 y-Wert	
4 z-Wert	
2 Richtung	
3 Bewegungsrichtung	
4 links	
4 rechts	

4 vor	
4 zurück	
3 Himmelsrichtung	
4 Nord	
4 Süd	
4 West	
4 Ost	
1 Situative Entität	
2 Zustand	Hat_Merkmal
3 Visueller Zustand	
4 porös	
4 matt	
4 gebraucht	
4 beschädigt	
4 neu	
3 Physikalischer Zustand	
3 Schaltzustand	
4 Ein	
4 Aus	
3 Aggregatzustand	
4 gasförmig	
4 flüssig	
4 fest	
2 Aktivität	Startzeitpunkt, Endzeitpunkt, Intensität, Methode, Wird_ausgeführt_durch, Kosten, Vor_, Nach_, Direkt_Vor, Direkt_Nach, Startet_zusammen_mit, Endet_ zusammen_mit, Überlappt_mit, Läuft_Innerhalb, Gleich
3 Kognitives Aktivität	
4 Planen	
4 Selektieren	
4 Suchen	
4 Explorieren	
4 Lernen	
4 Vergleichen	
4 Klassifizieren	
4 Voraussagen	
3 Soziales Aktivität	
4 Führen	
4 Kommunizieren	
4 Kooperieren	
4 Bekämpfen	
4 Vortäuschen	
4 Ablehnen	
4 Annehmen	
4 Kaufen	
4 Kontrollieren	
3 Motorisches Aktivität	
4 Bremsen	
4 Trennen	
4 Verbinden	
4 Transportieren	
4 Beschleunigen	
4 Neigen	
4 Reparieren	
2 Verhalten	Bezeichnung, Einfluss, Wirkungsmöglichkeit, Energieabgabe, Ausgangszustand, Endzustand, Wird_ausgeführt_durch,

	Startzeitpunkt, Endzeitpunkt, Dauer, Kosten, Aktion
3 Kognitive Verhaltensweisen	
4 Suchverhalten	
4 Explorationsverhalten	
4 Lernverhalten	
3 Soziale Verhaltensweisen	
4 Kommunikationsverhalten	
4 Kooperationsverhalten	
3 Motorische Verhaltensweisen	
4 Bremsverhalten	
4 Beschleunigungsverhalten	
4 Fahrverhalten	
3 Physikalisches Verhalten	
4 Spannungsverhalten	
4 Dehnungsverhalten	
4 Korrosionsverhalten	

Tabelle 23: Definition der Klassen

A.2 Die Slots

Name	Typ	Kardinalität	Ausprägung (Klasse =)
Abkürzung	String	Single	
Ähnlich_zu	Instanz	Multiple	Körperliche Entität
Aktivität	Instanz	Multiple	Aktivität
Alter	Instanz	Single	Zeiteinheit
Anfangspunkt	Instanz	Single	Fahrbahnsystem
Auf	Instanz	Single	System
Auftrag	Instanz	Multiple	Verbindung
Besteht_aus	Instanz	Multiple	Material
Bezeichnung	String	Single	
Beziehung	Float	Single	
Dämpfungskonstante	Float	Single	
Dauer	Instanz	Single	Zeiteinheit
Depotplätze	Integer	Single	
Direkt_Nach	Instanz	Single	Aktivität
Direkt_Vor	Instanz	Single	Aktivität
DirektHinter	Instanz	Single	System
DirektVor	Instanz	Single	System
Dokumentation	String	Single	
Einfluss	Instanz	Multiple	System
Einheit	String	Single	Einheit
Endet_zusammen_mit	Instanz	Multiple	Aktivität
Endpunkt	Instanz	Single	Fahrbahnsystem
Endzeitpunkt	Instanz	Single	Zeiteinheit
Energieabgabe	Instanz	Single	Physikalische Größe
Fährt_auf	Instanz	Single	Schiene
Federkonstante	Float	Single	
Formel	Float (Prozedur)	Single	
Formelzeichen	String	Single	
Gebühr	Instanz	Single	Monetäre Einheit
Gleich	Instanz	Multiple	Größe
Gleise	Integer	Single	
Größer	Instanz	Multiple	Größe
GrößerGleich	Instanz	Multiple	Größe
Hat_Merkmale	Instanz	Multiple	Abstrakte Entität
Hat-Ort	Instanz	Multiple	Örtliche Entität

Hat_Richtung	Instanz	Single	Richtung
Hat_Verhalten	Instanz	Multiple	Verhalten
Hat_Ziel	Instanz	Multiple	Entität
Hat_Zustand	Instanz	Multiple	Zustand
Hinter	Instanz	Multiple	System
Identisch_zu	Instanz	Multiple	Körperliche Entität
Intensität	Instanz	Multiple	Physikalische Größe
Kleiner	Instanz	Multiple	Größe
KleinerGleich	Instanz	Multiple	Größe
Kosten	Instanz	Single	Monetäre Einheit
Läuft_innerhalb	Instanz	Multiple	Aktivität
LinksVon	Instanz	Multiple	System
Nach_	Instanz	Multiple	Aktivität
Methode	Instanz	Multiple	
Parameter	Instanz	Multiple	Größe
RechtsVon	Instanz	Multiple	System
Referenzeinheit	Klasse	Single	Einheit
Richtung	Instanz	Single	Attribut
Startet_zusammen_mit	Instanz	Multiple	Aktivität
Startbahnhof	Instanz	Single	Bahnhof
Startzeitpunkt	Instanz	Single	Zeiteinheit
Teil_von	Instanz	Multiple	System
Transportgut	Instanz	Multiple	System
Überlappt_mit	Instanz	Multiple	Aktivität
Ursprung	Instanz	Single	System
Uhrzeit	Instanz	Single	Zeiteinheit
Unter	Instanz	Multiple	System
Verbunden_mit	Instanz	Multiple	System
Vor	Instanz	Multiple	System
Vor_	Instanz	Multiple	Aktivität
Wert	Float	Single	
Wird_ausgeführt_durch	Instanz	Single	System
Wirkt_auf	Instanz	Multiple	Abstrakte Entität
Wirkungsmöglichkeiten	Instanz	Multiple	System
Zielbahnhof	Instanz	Single	Bahnhof

Tabelle 24: Definition der Slots

B Der SCOUT-Algorithmus

B.1 Die Variablen

Variable	Beschreibung
a_k	Head einer Regel der Ontologie A.
aufnahmeArt	Variable dient zur Differenzierung der Art der Aufnahme („AND“ oder „OR“). Sie beinhaltet somit die Information darüber, ob die Übersetzung (falls vorhanden) eindeutig ist, also eine „AND“-Übersetzung, oder bzgl. der Variablenausprägungen eine „ODER“-Übersetzung darstellt. Die Ausgangsbelegung ist für jede Regel von Ontologie C „AND“.
ausgabe	Zusammenführung der Heads aus *ausgabeAND* und *ausgabeOR* unter Beachtung der entsprechenden Verknüpfung ($\wedge$ oder $\vee$).
ausgabeAND[d]	Sammlung aller Heads, deren Regeln als eindeutig matchbar klassifiziert wurden. Sie kommen mit einem $\wedge$-Junktor in die *ausgabe*.
ausgabeOR[e]	Sammlung aller Heads deren Regeln, bedingt durch die Wertebereiche, nicht als eindeutig matchbar klassifiziert wurden. Sie kommen, sofern diese Zuordnung in dem Teilalgorithmus VALIDIERUNG bestätigt wurde, mit einem $\vee$-Junktor in die *ausgabe*.

ausprägung	Variable, welche die Wertangabe des zu detaillierenden Literals enthält
ausrGG	Variable zur Ermittlung der *aufnahmeArt*. Bildet die Differenz zwischen dem “Tendenz-Größer A“ ($tg^{A)}$ und “Tendenz-Größer C“ (tg^{C}).
ausrGK	Variable dient zur Ermittlung von Überschneidungen in den Wertebereichen. Bildet die Differenz zwischen dem “Tendenz-Größer A“ (tg^{A}) und “Tendenz-Kleiner C“ (tk^{C}).
ausrKG	Variable dient zur Ermittlung von Überschneidungen in den Wertebereichen. Bildet die Differenz zwischen dem “Tendenz-Kleiner A“ (tk^{A}) und “Tendenz-Größer C“ (tg^{C}).
ausrKK	Variable zur Ermittlung der *aufnahmeArt*. Bildet die Differenz zwischen dem “Tendenz-Kleiner A“ (tk^{A}) und “Tendenz-Kleiner C“ (tk^{C}).
c_i	Head einer Regel von der Ontologie c.
$basisOnt^{typ}_{id}$	Klasse des betrachteten Literals.
basisOntA	Variable beinhaltet bei einer Funktion die Basisontologie des ersten inneren Slots.
basisOntB	Variable beinhaltet bei einer Funktion die Basisontologie des zweiten inneren Slots.
$BodyGröße^{p}$	Variable zur Speicherung der Anzahl der übereinstimmenden Slots einer als matchbar klassifizierten Regel von Ontologie C. Damit kann nach Durchlauf aller Regeln von Ontologie C auf eine potentielle 1-1-Übersetzung geschlossen werden.
bodyIdentität	Booleanvariable enthält die Information über den erfolgreichen Slot-Vergleich. Die Ausgangsbelegung für jede Regel von Ontologie C ist „TRUE“. Nur wenn ein Slot aus einer Regel von Ontologie C nicht in der betrachteten Regel von Ontologie A enthalten ist, wird der Wert auf „FALSE“ gesetzt.
d	Index zur eindeutigen Identifikation eines Heads in *ausgabeAND*, bzw. der zugehörigen Slots in *prädikatmengeAND.*
depot	Zusammenfassung aller Slots einer Regel von Ontologie C welche zu einem Slot von Ontologie A äquivalent sind – auch bzgl. sich überschneidender Wertebereiche. Die Regel von Ontologie C wird nur dann als matchbar empfunden, wenn die gemeinsamen Slots zumindest Überschneidungen in den Wertebereichen besitzen und wenn bei Vorhandensein von mehr Slots in der Regel von Ontologie C mindestens auch alle Slots der Regel von Ontologie A existieren.
detaillierungsliteral	Zusammenfassung aller logisch - zusammengehörigen Literale einer Regel, so z.B. “Regen(Wetter,R), R>10, R<15“.
diffGG	Variable dient zur Ermittlung der *aufnahmeArt*. Es wird somit geprüft, ob das Intervall von Ontologie A das von Ontologie C umfasst. Bildet die Differenz zwischen dem “Wert-Größer A“ (wg^{A}) und “Wert-Größer C“ (wg^{C}).
diffGK	Variable dient zur Ermittlung von Überschneidungen zwischen den Wertebereichen. Bildet die Differenz zwischen dem “Wert-Größer A“ (wg^{A}) und “Wert-Kleiner C“ (wk^{C}).
diffKG	Variable dient zur Ermittlung von Überschneidungen zwischen den Wertebereichen. Bildet die Differenz zwischen dem “Wert-Kleiner A“ (wk^{A}) und “Wert-Gößer C“ (wg^{C}).
diffKK	Variable dient zur Ermittlung der aufnahmeArt. Es wird somit geprüft, ob das Intervall von Ontologie A das von Ontologie B umfasst. Bildet die Differenz zwischen “Wert-Kleiner A“ (wk^{A}) und “Wert-Kleiner C“ (wk^{C}).
e	Index zur eindeutigen Identifikation eines Heads in *ausgabeOR*, bzw. der zugehörigen Slots in *prädikatmengeOR*.
literal	Literal, welches zur Extraktion der Werte an die DETAILLIERUNG übergeben wird.
$HeadCollection^{p}$	Zusammenfassung aller Heads der als matchbar klassifizierten Regeln. Im Falle einer existierenden 1-1-Übersetzung werden die übrigen Heads aus der *ausgabeAND* und *ausgabeOR* entfernt.
i	Index zur eindeutigen Identifikation der Regeln von Ontologie C.
id	Variable weist allen zusammengehörigen Informationen einer Regel eine gemeinsame Nummerierung zu.
j	Index zur eindeutigen Identifikation einers Literals innerhalb einer Regel

	von Ontologie C
k	Index zur eindeutigen Identifikation der Regeln von Ontologie A.
l	Index zur eindeutigen Identifikation eines Literals innerhalb einer Regel von Ontologie A.
ljInkrement	Anzahl, um welche l bzw. j weitergesetzt werden müssen, damit bei der nächsten Iteration ein noch nicht analysiertes Literals zur Verfügung steht.
$m_{k,l}$	Literal einer Regel
M^A	Menge der gesamten a_k´s mit dazugehörigen $m_{k,l}$´s
M^C	Menge der gesamten c_i´s mit dazugehörigen $m_{i,j}$´s
M^I	In dieser Menge werden die gesamten erfolgreichen Übersetzungen zusammengefasst und am Ende des Algorithmus als Ausgabe zurückgegeben.
operator	Extraktion des im *literal* befindlichen Vergleichsoperators (<, >, <=, >=, =)
p	Index zur Identifikation der *prädikatAnzB* in der *bodyGröße*, des Heads im *headCollection* und der zu der Regel von Ontologie C gehörigen Slots in *prädikatCollection*.
$prädikat^{typ}_{id}$	Slot des betrachteten Literals
prädikatA	Variable beinhaltet bei einem Literal, welches aus einer Funktion besteht, den ersten inneren Slot.
prädikatB	Variable beinhaltet bei einem Literal, welches aus einer Funktion besteht, den zweiten inneren Slot.
prädikatAlternativeA	Variable stellt bei einer Funktion, eine Verknüpfung der relevanten Informationen (äußerer Slot, erster innere Slot, Klasse vom ersten inneren Slot, zweiter innerer Slot, Klasse vom zweiten inneren Slot) dar. Eine Unterscheidung ist notwendig, da aufgrund unterschiedlicher Werte auch die innere Repräsentation mit berücksichtigt werden muss.
prädikatAlternativeB	Variable stellt bei einer Funktion, eine Verknüpfung der relevanten Informationen (äußerer Slot, zweiter innerer Slot, Klasse vom zweiten inneren Slot, erster innerer Slot, Klasse vom ersten inneren Slot) dar. Eine Unterscheidung ist notwendig, da aufgrund unterschiedlicher Werte auch die innere Repräsentation mit berücksichtigt werden muss.
prädikatAnzA	Variable zur Erfassung der Anzahl aller Slots einer Regel von Ontologie A.
prädikatAnzB	Variable zur Erfassung der Anzahl aller Slots einer Regel von Ontologie C, welche äquivalent sind bzgl. eines Slots und dessen Basisontologie der betrachteten Regel von Ontologie A.
$prädikatCollection^p$	Zusammenfassung aller Slots der als matchbar klassifizierten Regeln. Im Falle einer existierenden 1-1-Übersetzung werden die übrigen Slots aus der *prädikatmengeAND* und *prädikatmengeOR* entfernt.
$prädikatmengeAND^d$	Sammlung aller Slots, deren Regeln als eindeutig matchbar klassifiziert wurden. Die zusammengehörigen Slots einer Regel werden mit einem gemeinsamen Index gekennzeichnet.
$prädikatmengeOR^e$	Sammlung aller Slots, deren Regeln bedingt durch die Wertebereiche nicht als eindeutig matchbar klassifiziert wurden. Die zusammengehörigen Slots einer Regel werden mit einem gemeinsamen Index gekennzeichnet.
prädikatVereinigungA	Zusammenfassung aller Slots einer Regel von der Ontologie A. Eine Übersetzung darf nur dann erfolgen, wenn diese Slots (mindestens) vollständig von Ontologie C abgedeckt werden kann.
prädikatVereinigungB	Zusammenfassung aller Slots aus den Mengen *prädikatmengeAND* und *prädikatmengeOR*. Nur wenn diese die *prädikatVereinigungA* umfasst, ist eine erfolgreiche Übersetzung gefunden worden.
prüfmenge	Menge bildet eine disjunkte Vereinigung aller in der *prädikatmengeOR* vorhandenen Slots abzüglich derer, die bereits in der *prädikatmengeAND* vorhanden sind.
q	Index zur Identifikation der Slots in der *schnittmenge*.
r	Index zur eindeutigen Identifikation eines Literals innerhalb eines *detaillierungsliterals.*
restmenge	Menge besteht aus der *prüfmenge* abzüglich der aktuellen oder in

	früheren Iterationen betrachteten Slots der *prädikatmengeOR*.
sa	Index zur Identifikation der zu einer bestimmten Regel gehörenden Slots in der *prädikatmengeOR*.
sb	Index zur Identifikation der zu einer bestimmten Regel gehörenden Slots in der *prädikatmengeOR*.
schnittA	Detaillierungsliteral mit dem Prädikat *schnittmenge*^{q} aus der der zur *prädikatmengeOR*^{sa} gehörenden Regel.
schnittB	Detaillierungsliteral mit dem Prädikat *schnittmenge*^{q} aus der der zur *prädikatmengeOR*^{sb} gehörenden Regel.
schnittmenge	Menge besteht aus den gemeinsamen Slots der beiden aktuell betrachteten Mengen *prädikatmengeOR*.
tempA	Hilfsvariable zur Realisierung des Vergleichs von Literals, welche aus Funktionen bestehen. Abhängig davon wird der Wert von der Variablen entweder (bei *minus*) mit -1 multipliziert oder (bei *dividiert*) der Kehrwert genommen. Darüber hinaus fungiert die Variable bei der Vertauschung der Tendenzen.
tempB	Hilfsvariable zur Realisierung des Vergleichs von Literals, welche aus Funktionen bestehen. Abhängig davon wird der Wert von der Variablen entweder (bei *minus*) mit -1 multipliziert oder (bei *dividiert*) der Kehrwert genommen.
tg^{typ}_{id}	"Tendenz-Größer" – beschreibt den Vergleichsoperator, welcher zum minimalen Wert (*wg*) gehört, mittels einer Zahl. Folgende mögliche Zuordnungen existieren: "=" → 10, ">=" → 3, ">" → 2
tk^{typ}_{id}	"Tendenz-Kleiner" – beschreibt den Vergleichsoperator, welcher zum maximalen Wert (*wk*) gehört, mittels einer Zahl. Folgende mögliche Zuordnungen existieren: "=" → -10, "<=" → -4, "<" → -1
treffer	Booleanvariable beinhaltet die Information, ob die betrachteten Slots und Klassen identisch sind.
typ	Identifikation der betrachteten Ontologie (A bzw. C)
wertIdentität	Booleanvariable beinhaltet die Information, ob die betrachteten Wertebereiche Überschneidungen besitzen.
wg^{typ}_{id}	"Wert-Größer-(als)" – beinhaltet für jedes Werteintervall den minimalsten Wert. D.h., jeder für den Slot potentielle Wert ist mindestens – abhängig vom Vergleichsoperator – so groß wie dieser Wert.
wk^{typ}_{id}	"Wert-Kleiner-(als)" – beinhaltet für jedes Werteintervall den maximalsten Wert. D.h., jeder für den Slot potentielle Wert ist höchstens – abhängig vom Vergleichsoperator – so groß wie dieser Wert.

Tabelle 25: Die Variablen des SCOUT-Algorithmus

B.2 Der Pseudo-Code

Es sei:

a_k :- $m^A_{k,l}$ (mit k = 1…n und l = 1…m)
c_i :- $m^C_{i,j}$ (mit i=1…s und j=1…t)

$$M^A = \sum_{k=1}^{n} \sum_{l=1}^{m} [\, a_k :- m^A_{k,l} \,]$$

$$M^C = \sum_{i=1}^{s} \sum_{j=1}^{t} [\, c_i :- m^C_{i,j} \,]$$

Initialisierungen:

d = 1

```
e = 1
i = 1
id = 1
j = 1
ljInkrement = 0
k = 1
l = 1
p = 1
r = 1

1.   FOR  M^A DO
2.     WHILE a_k ≠ Ø
3.       prädikatAnzA = 0
4.       WHILE m_k,l ≠ Ø
5.           ljInkrement = 0
6.           detaillierungsliteral = m_k,l
7.           prädikatAnzA = prädikatAnzA + 1
8.           IF ( ∈ V_ergleichsoperator)  ∈  m_k,l+1
9.               detaillierungsliteral = detaillierungsliteral  ∧  m_k,l+1
10.              ljInkrement  = ljInkrement + 1
11.              IF ( ∈ V_ergleichsoperator) ∈ m_k,l+2
12.                  detaillierungsliteral = detaillierungsliteral  ∧  m_k,l+2
13.                  ljInkrement = ljInkrement + 1
14.          SEPARATION(A, id, detaillierungsliteral)
15.          prädikatVereinigungA = prädikatVereinigungA  ∧  prädikat^A_id
16.          id = id + 1
17.          l = l + ljInkrement + 1
18.      FOR  M^C DO
19.          WHILE c_i  ≠ Ø
20.              aufnahmeArt = AND
21.              prädikatAnzB = 0
22.              bodyIdentität = TRUE
23.              WHILE m_i,j  ≠ Ø
24.                  ljInkrement = 0
25.                  detaillierungsliteral = m_i,j
26.                  IF ( ∈ V_ergleichsoperator) ∈ m_i,j+1
27.                      detaillierungsliteral = detaillierungsliteral  ∧  m_i,j+1
28.                      ljInkrement = ljInkrement + 1
29.                      IF ( ∈ V_ergleichsoperator) ∈ m_i,j+2
30.                          detaillierungsliteral = detaillierungsliteral  ∧  m_i,j+2
31.                          ljInkrement = ljInkrement + 1
32.                  SEPARATION(C, 1, detaillierungsliteral)
33.                  id = 1
34.                  treffer = FALSE
35.                  WHILE (treffer = FALSE  ∧  prädikat^A_id ≠ Ø)
36.                      treffer = PRÄDIKATANALYSE(id)
37.                      id = id + 1
38.                  IF treffer = TRUE
39.                      id = id - 1
40.                      wertIdentität = INTERVALLANALYSE(id)
41.                      IF wertIdentität = TRUE
42.                          prädikatAnzB = prädikatAnzB + 1
43.                          IF aufnahmeArt ≠ OR
44.                              aufnahmeArt = INTERVALLVERHÄLTNIS(id)
45.                          depot= depot  ∧  prädikat^C_1
46.                          j = j + ljInkrement + 1
47.                      ELSE
48.                          depot = Ø
49.                          j = -1
50.                          prädikatAnzB = 0
51.                  ELSE
52.                      bodyIdentität = FALSE
53.                      j = j + ljInkrement + 1
54.              IF prädikatAnzB ≠ prädikatAnzA  ∧  bodyIdentität = FALSE
55.                  depot = Ø
56.              IF depot ≠ Ø
57.                  bodyGröße^P = prädikatAnzB
58.                  headCollection^P = c_i
```

```
59.                 prädikatCollection^P = depot
60.                 p = p + 1
61.                 IF aufnahmeArt = AND
62.                     ausgabeAND^d = c_i
63.                     prädikatmengeAND^d = depot
64.                     d = d + 1
65.                 ESLE
66.                     ausgabeOR^e = c_i
67.                     prädikatmengeOR^e = depot
68.                     e = e + 1
69.             i = i + 1
70.             j = 1
71.             depot = Ø
72.     VALIDIERUNG()
73.     prädikatVereinigungB = prädikatmengeAND ∧ prädikatmengeOR
74.     IF prädikatVereinigungA ⊆ prädikatVereinigungB
75.         ausgabe = ausgabeAND^1 ∧ … ∧ ausgabeAND^d ∧ (ausgabeOR^1 ∨ … ∨ ausgabeOR^e )
76.         M^T = M^T ∧ (a_k = ausgabe)
77.     prädikatVereinigungA = prädikatVereinigungB = … = Ø
78.     p = … = 1
79.     k = k + 1
80.  RETURN(M^T)


81.  SEPARATION(typ, id, detaillierungsliteral)
82.  wg^typ_id = -∞, tg^typ_id = 2, tk^typ_id = -1, wk^typ_id = ∞
83.  prädikat^typ_id = Slot
84.  basisOnt^typ_id = Basisontologie
85.  m_r = r-te Literal vom detaillierungsliteral
86.  IF m_r+1 = Ø
87.    DETAILLIERUNG(typ, id, m_r)
88.  ESLE
89.    DETAILLIERUNG(typ, id, m_r+1)
90.    IF m_r+2 ≠ Ø
91.      DETAILLIERUNG(typ, id, m_r+2)
92.   IF prädikat^typ_id ∈ Funktion
93.    prädikatA = 1.Slot in Funktion
94.    basisOntA = Basisontologie von dem 1.Slot
95.    prädikatB = 2.Slot in Funktion
96.    basisOntB = Basisontologie von dem 2.Slot
97.      IF typ = A
98.      prädikat^typ_id = prädikat^typ_id/ prädikatA/ basisOntA/ prädikatB/ basisOntB
99.    IF typ = C
100.     prädikatAlternativeA = prädikat^typ_id/ prädikatA/ basisOntA/ prädikatB/ basisOntB
101.     prädikatAlternativeB = prädikat^typ_id/ prädikatB/ basisOntB/ prädikatA/ basisOntA

102. DETAILLIERUNG(typ, id, literal)
103. operator = Vergleichsoperator aus literal
104. ausprägung = Wert aus literal
105. IF operator = "<="
106.   tk^typ_id = -4, wk^typ_id = ausprägung
107. IF operator = "<"
108.   tk^typ_id = -1, wk^typ_id = ausprägung
109. IF operator = "="
110.   tg^typ_id = 10, tk^typ_id = -10, wg^typ_id = wk^typ_id = ausprägung
111. IF operator = ">"
112.   tg^typ_id = 2, wg^typ_id = ausprägung
113. IF operator = ">="
114.   tg^typ_id = 3, wg^typ_id = ausprägung

115. PRÄDIKATANALYSE(id)
116. IF prädikat^C_1 ∉ Funktion
117.   IF (prädikat^A_id = prädikat^C_1 ∧ basisOnt^A_id = basisOnt^C_1)
118.     return TRUE
119.   ESLE
120.     return FALSE
121.  ELSE
122.   IF prädikat^A_id = prädikatAlternativeA
```

```
123.      prädikat^C_1 = prädikatAlternativeA
124.      return TRUE
125.    IF prädikat^A_id = prädikatAlternativeB
126.      prädikat^C_1 = prädikatAlternativeB
127.      IF prädikat^C_1 ∈ {minus, dividiert}
128.          VERTAUSCHUNG()
129.      return TRUE
130.    ELSE
131. return FALSE

132. VERTAUSCHUNG()
133. tempA = wg^C_1
134. tempB = wk^C_1
135. if prädikat^C_1 = minus
136.    wg^C_1 = tempB * (-1)
137.    wk^C_1 = tempA * (-1)
138. if prädikat^C_1 = dividiert
139.    wg^C_1 = (tempB)^-1
140.    wk^C_1 = (tempA)^-1
141. tempA = tg^C_1
142. tg^C_1 = tk^C_1
143. tk^C_1 = tempA
144. IF tg^C_1 = -4
145.    tg^C_1 = 3
146.  IF tg^C_1 = -1
147.    tg^C_1 = 2
148. ELSE
149.    tg^C_1 = 10
150. IF tk^C_1 = 2
151.    tk^C_1 = -1
152. IF tk^C_1 = 3
153.    tk^C_1 = -4
154. ELSE
155.    tk^C_1 = -10

156. INTERVALLANALYSE(id)
157. IF ((wg^A_id = wk^A_id) ∧ (wg^C_1 = wk^C_1))
158.    IF wg^A_id = wg^C_1
159.      return TRUE
160.    ELSE
161.      return FALSE
162. ELSE
163.    diffGK = wg^A_id – wk^C_1
164.    diffKG = wk^A_id – wg^C_1
165.    ausrGK = tg^A_id – tk^C_1
166.    ausrKG = tk^A_id – tg^C_1
167.    IF ((diffGK > 0 ∨ (diffGK = 0 ∧ (ausrGK ≠ 7 ∨ ausrGK <= 12))) ∨
          (diffKG < 0 ∨ (diffKG = 0 ∧ (ausrKG ≠ -7 ∨ ausrKG >= -12))))
168.      return FALSE
169.    ELSE
170.      return TRUE

171. INTERVALLVERHÄLTNIS(id)
172. IF wg^A_id = wk^A_id = wg^C_1 = wk^C_1
173.    return AND
174. ELSE
175.    diffGG = wg^A_id – wg^C_1
176.    diffKK = wk^A_id – wk^C_1
177.    ausrGG = tg^A_id – tg^C_1
178.    ausrKK = tk^A_id – tk^C_1
179.    IF((diffGG>0 ∨ (diffGG=0 ∧ (ausrGG=-1 ∨ ausrGG=0 ∨ ausrGG=7))) ∧
          (diffKK<0 ∨ (diffKK=0 ∧ (ausrKK=-6 ∨ ausrKK=0 ∨ ausrKK=3))))
180.      return AND
181.    ESLE
182.      return OR

183. VALIDIERUNG()
184. if prädikatAnzA ∈ bodyGröße
```

```
185.   p = 1
186.   while bodyGröße^P ≠ Ø
187.     if bodyGröße^P < prädikatAnzA
188.         (ausgabeAND ∨ ausgabeOR)\ headCollection^P
189.         (prädikatmengeAND ∨ prädikatmengeOR)\ prädikatCollection^P
190.     p = p + 1
191. prüfmenge = prädikatmengeOR^1 ∪ … ∪ prädikatmengeOR^e
192. prüfmenge = prüfmenge\ (prädikatmengeAND^1 ∪ … ∪ prädikatmengeAND^d)
193. sa = 1
194. WHILE prädikatmengeOR^sa ≠ Ø
195.   restmenge = prüfmenge \ prädikatmengeOR^sa
196.   sb = 1
197.   WHILE prädikatmengeOR^sb ≠ Ø
198.     IF prädikatmengeOR^sb ⊆ restmenge
199.         restmenge = restmenge \ prädikatmengeOR^sb
200.     ELSE
201.         If  prädikatmengeOR^sa ≠ prädikatmengeOR^sb
202.             schnittmenge = prädikatmengeOR^sa ∩ prädikatmengeOR^sb
203.             wertidentität = TRUE
204.             q = 1
205.             while schnittmenge^q ≠ Ø  ∧ wertidentität = TRUE
206.                 schnittA =detaillierungsliteral mit schnittmenge^q aus prädikatmengeOR^sa -Regel
207.                 schnittB =detaillierungsliteral mit schnittmenge^q aus prädikatmengeOR^sb -Regel
208.                 SEPARATION(A, 1, schnittA)
209.                 SEPARATION(B, 1, schnittB)
210.                 wertidentität = INTERVALLANALYSE(1)
211.                 q = q + 1
212.             if wertidentität = TRUE
213.                 prädikatmengeOR^sb = prädikatmengeOR^sb\ schnittmenge
214.                 if prädikatmengeOR^sb ⊆ restmenge
215.                     restmenge = restmenge\ prädikatmengeOR^sb
216.     sb = sb + 1
217.   IF restmenge ≠ Ø
218.     prädikatmengeOR = prädikatmengeOR \ prädikatmengeOR^sa
219.     ausgabeOR = ausgabeOR \ ausgabeOR^sa
220.   sa = sa  + 1
```

C Protégé-2000

Protégé-2000[44] ist ein frei verfügbares Wissensakquisitionstool, welches an der Universität Stanford entwickelt wurde. Es wurde zur Unterstützung bei der Entwicklung und dem Aufbau von wissensbasierten Systemen entwickelt.

Die grafische Benutzeroberfläche von Protégé-2000 besteht aus sich überlappenden Tabs, um ein komfortables Erstellen einer Wissensdatenbank zu ermöglichen. Das Design mittels Tabs erlaubt:

- die Modellierung einer Ontologie von Klassen, die ein spezielles Thema darstellt,
- das Füllen der Wissensdatenbank mit neuen Daten und
- die Ausführung von zusätzlichen Applikationen (Plug-ins).

Das Wissensmodell von Protégé-2000 basiert auf dem Frame-Konzept, das aus Klassen, Slots, Facetten und Axiomen besteht. Klassen sind Objekte aus dem zu modellierenden Weltausschnitt. Slots entsprechen den Attributen dieser Klassen. Facetten beschreiben

[44] http://protege.stanford.edu/

Eigenschaften von Slots. Axiome spezifizieren zusätzliche Einschränkungen (Constraints) [NFM00].

Mittlerweile benutzen mehr als 100 Projekte weltweit Protégé-2000. Dies kann auf mehrere Gründe zurückgeführt werden:

- Protégé-2000 ist ein leicht zu bedienendes konfigurierbares Tool.
- Es besitzt eine hohe Kompatibilität und Interoperabilität mit anderen Wissensrepräsentationstools.
- Protégé-2000 ist erweiterbar. Mehrere Plugin-Tabs stehen zur Verfügung unter http://protege.stanford.edu/plugins.html.

Eine der wichtigsten Merkmale für die Kompatibilität von Protégé-2000 mit anderen Wissensrepräsentationstools ist seine Fähigkeit, Ontologien und Wissensbasen im RDF-Format zu importieren und zu exportieren. RDF ist ein Wissensrepräsentationsstandard, der vom World-Wide Web Consortium (W3c; siehe auch www.w3.org) definiert wurde. Folgende Betrachtung galt als die Grundidee von RDF: Die bis dahin zu findenden web-basierten Dokumente waren maschinen*lesbar*, aber leider nicht maschinen*verständlich*. Anhand von RDF wird versucht, Dokumente so zu strukturieren, dass die darin enthaltene Semantik maschinell „verstanden“ werden kann. Damit kann die enorme Anzahl an Dokumenten und Daten, die im Internet vorhanden sind, in einer Weise beschrieben werden, die es nicht nur Menschen, sondern auch Maschinen ermöglicht, sie zu verarbeiten und zu verstehen.

C.1 Klassen und Instanzen

Die Klassen bilden in Protégé-2000 eine taxonomische Hierarchie, die als hierarchischer Klassenbaum dargestellt wird. Besitzt eine Klasse A eine Unterklasse B, so gilt für jede Instanz der Klasse B, dass sie zusätzlich eine Instanz der Klasse A ist. Protégé-2000 unterstützt dabei die Mehrfachvererbung, d.h. zwei Klassen können eine gemeinsame Unterklasse haben. Diese erbt die Eigenschaften beider Superklassen [NFM00]. Neben der hierarchischen Anordnung der Klassen können sie zudem horizontal über Slots verbunden werden.

C.2 Slots

Slots beschreiben die Eigenschaften der Klassen, wie z.B. den Namen eines Künstlers oder den Titel eines Songs [NFM00]. Bei den Slots kann es sich um relationale, deskriptive oder prozedurale Slots handeln. Die Bestimmung der Art der Slots erfolgt über den „Value Type“. Die verfügbaren Typen sind in Tabelle 26 aufgeführt.

Typ	Beschreibung	Beispiel	Art
Boolean	Logischer Wert	True, False	Deskriptiv
Klasse	Klasse der Wissensbasis	*Shuttle*	Relational
Float	Dezimalzahl	1.0, 1000, 0.1	Deskriptiv
Instanz	Instanz einer Klasse	*Siemens X2*	Relational
Integer	Ganze Zahl	1, 4, -5	Deskriptiv
String	Buchstaben	rot, hoch	Deskriptiv, Prozedural
Symbol	Liste an möglichen Werten	Hoch, tief und mittel	Deskriptiv

Tabelle 26: Die Value Types der Slots

C.3 Facetten

Facetten beschreiben Eigenschaften (Restriktionen) von Slots. Sie definieren Beschränkungen (engl. Constraints) auf der Zuordnung eines Slots zu einer bestimmten Klasse. In Protégé-2000 können folgende Constraints durch Facetten spezifiziert werden:

- Kardinalität eines Slots
- Minimaler und maximaler Wert eines numerischen Slots
- Inverse Slots: Die Werte der Instanz einer Klasse werden automatisch in eine andere Klasse übertragen.
- Template Values: Eintragungen eines Slots, die nicht geändert werden können
- Defaults: Eintragungen eines Slots, die geändert werden können.

D Resource Description Framework (RDF)

Ontologien als Träger von Wissen lassen sich mit Hilfe von Standards wie dem Resource Description Framework (RDF) in Verbindung mit dem Resource Description Framework Schema (RDFS) abbilden. Mittels RDF können beliebige Metadaten über beliebige Objekte erfasst werden. Metadaten sind Daten über (andere) Daten bzw. Informationen über (andere) Informationen. So sind Titel, Autor und Erstelldatum eines Dokuments dessen Metadaten. Meistens helfen Metadaten, Informationen zu finden. Während manche Daten für ein Programm Metadaten darstellen, können dieselben Daten aus anderer Sicht nur Daten an sich sein. Im Bereich der relationalen Datenbanken beispielsweise ist genau definiert, wo Metadaten enden und wo Daten beginnen: Der Name einer jeden Tabelle und der Typ einer jeden Spalte sind Metadaten der abgelegten Daten [Mae03].

Damit die Daten für eine automatische Verarbeitung durch verschiedene Systeme nicht nur maschinenlesbar, sondern auch maschinenverständlich sind, muss das semantische Datenmodell als Ontologie in einem RDF-Schema festgehalten werden. Damit ein Austausch dieser Daten zwischen verschiedenen Systemen problemlos funktionieren kann, kann RDF in dem systemneutralen Format Extensible Markup Language (XML) notiert werden.

D.1 RDF-Datenmodell und -Syntax

RDF selbst besteht aus den vier grundlegenden Komponenten *Ressourcen*, *Eigenschaften*, *Literale* und *Aussagen* [W3c99]:

- **Ressourcen:** Eine *Ressource* kann, ähnlich wie in einem ERM[45], ein materielles oder immaterielles Objekt sein, welches in RDF eindeutig über einen sogenannten URI (Unified Resource Identifier) identifizierbar ist[46]. Somit kann z.B. auf Webseiten, Personen, Unternehmen und auch (Wissens-)Objekte durch natürliche oder künstliche Schlüssel verwiesen werden.
- **Eigenschaften** (Properties): Eine *Property* ist eine Eigenschaft, ein Attribut oder eine Relation, die dazu benutzt wird, eine Ressource zu beschreiben. Die Bedeutung eines Propertys wird durch die Definition der zulässigen Werte, der Typen der Ressourcen, die es beschreiben kann, und die Definition der Beziehung dieses Propertys zu anderen Properties festgelegt.
- **Literale** (Literals): *Literale* können atomare Werte wie Unicode-Zeichenketten beinhalten. Eigenschaften können durch Literale ausgedrückt werden.
- **Aussagen** (Statements): In einem *Statement* wird der Property einer Ressource ein Wert zugewiesen, wobei dieser Wert auch wieder eine Ressource sein kann. Somit stellen die Statements als Tripel Subjekt-Prädikat-Objekt die Grundidee des RDF dar.

Die Aussage

"Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila"

hat die folgenden Teile [LS99]:

- Subjekt (Ressource): http://www.w3.org/Home/Lassila
- Prädikat (Eigenschaft): Creator
- Objekt (Wert, Literal): "Ora Lassila"

Damit lässt sich diese Aussage auf eindeutige Weise in das RDF-Datenmodell überführen.

In der folgenden grafischen Darstellung wird ein Oval für die Ressource, ein Pfeil für die Eigenschaft und ein Rechteck für das Objekt verwendet.

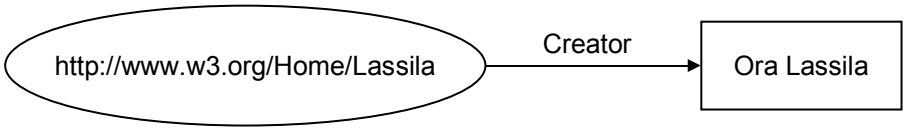


Abbildung 94: Einfache Aussage in RDF [LS99]

[45] Abkürzung für Entity Relationship Modell
[46] Informationen zu qualifizierenden URIs unter: http://www.ietf.org/rfc/rfc2396.txt

Wesentlich dabei ist, dass der Pfeil immer vom Subjekt auf das Objekt gerichtet ist. Diese einfache Diagrammdarstellung kann auch als "<Subjekt> hat <Prädikat> <Objekt>" gelesen werden. Also in diesem Fall: "http://www.w3.org/Home/Lassila has creator Ora Lassila". In der RDF/XML-Syntax würde diese Aussage in Form eines vollständigen XML-Dokuments wie folgt repräsentiert:

```
(1) <?xml version="1.0"?>
(2) <rdf:RDF
(3)  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
(4)  xmlns:s="http://description.org/schema/">
(5)  <rdf:Description about="http://www.w3.org/Home/Lassila">
(6)    <s:Creator>Ora Lassila</s:Creator>
(7)  </rdf:Description>
(8) </rdf:RDF>
```

In den Zeilen 3 und 4 werden zwei Namespaces deklariert: der RDF-Namespace wird dem Präfix `rdf` und ein Schema dem Präfix `s` zugewiesen. Ein Namespace dient in XML zur Unterscheidung verschiedener Vokabulare. Hier wird also zum Beispiel über die URL `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, die ein Dokument mit der Beschreibung des RDF-Vokabulars identifiziert, eine eindeutige Beschreibung für den Namespace `rdf` hergestellt und festgelegt, welche syntaktischen Elemente dieser Namespace enthält. Auf diese Weise werden mögliche Überschneidungen von verschiedenen XML-basierten Vokabularen verhindert, da jedes syntaktische Element durch seinen zugeordneten Namespace und die damit verbundene eindeutige URL ebenfalls eindeutig wird. In Zeile 5 beginnt die eigentliche Beschreibung mit dem Abschnitt `Description`. Über das Attribut `about` wird auf die Ressource "http://www.w3.org/Home/Lassila" Bezug genommen und damit das Objekt der Aussage festgelegt. Zeile 6 enthält die Eigenschaft (Prädikat) und ihren Wert (Objekt). Bei der Eigenschaft wird über das Namespace-Präfix `s` auf das Schema verwiesen. Wie oben bereits erwähnt, wird dann mittels dieses Schemas die Charakteristik der Eigenschaft festgelegt. Wesentlich ist, dass auf diese Weise für jede verwendete Eigenschaft ein Schema angegeben werden muss.

In den folgenden Beispielen wird zur Übersichtlichkeit die Namespace-Deklaration weggelassen. Das obige Beispiel würde dann so lauten:

```
<rdf:Description about="http://www.w3.org/Home/Lassila">
  <s:Creator>Ora Lassila</s:Creator>
</rdf:Description>
```

Weiterhin ist die Verwendung einer abgekürzten Syntax möglich, die eine kompaktere Form erlaubt [Las99]. Das Beispiel würde dann wie folgt dargestellt:

```
<rdf:Description about="http://www.w3.org/Home/Lassila" s:Creator="Ora Lassila" />
```

D.2 Das RDF-Schema

RDF ist ein Datenmodell zur Repräsentation von Metadaten, die Ressourcen beschreiben. Es werden also Aussagen auf Instanzenebene gemacht. Da im RDF selber aber keine Möglichkeit besteht, Strukturinformationen der generischen Klassen zu beschreiben, also zum Beispiel zu sagen welche Klassen von Ressourcen es gibt, benötigt man für diese Aufgabe zusätzlich zum RDF die Schema-Spezifikationssprache RDF-Schema. RDFS erlaubt die Definition von Schemata, die Begriffe für spezielle Anwendungsgebiete in maschinenverständlicher Form festlegen. Dadurch kann die Semantik von Klassen, Eigenschaften und Werten formal definiert werden. Hierfür stehen u.a. die folgenden Konstrukte zur Verfügung:

- *rdfs:Class* beschreibt Klassen von ähnlichen Objekten.
- *rdfs:subClassOf* ist eine vordefinierte Eigenschaft, welche eine Klasse als Unterklasse einer anderen Klasse definiert.
- *rdfs:Resource* ist die Superklasse in jedem Schema. Jede definierte Klasse ist rdfs:subClassOf rdfs:Resource.
- *rdf:type* ist eine Eigenschaft, die eine Ressource als Instanz einer bestimmten Klasse definiert. Der Wert dieser Eigenschaft definiert diese Klasse.
- *rdfs:Property* stellt die allgemeinste Eigenschaft dar. Jede definierte Eigenschaft ist rdfs:subPropertyOf rdfs:Property.
- *rdfs:subPropertyOf* beschreibt die Beziehungen zwischen einer allgemeineren und einer speziellen Eigenschaft.
- Constrains: In RDFS können Eigenschaften dadurch näher spezifiziert werden, indem der Anwendungsbereich dieser Eigenschaft auf bestimmte Klassen beschränkt wird (*rdfs:domain*) oder der Wertebereich definiert wird (*rdfs:range*) [LS99].

E Prolog

Bei Prolog (PROgramming in LOGic) handelt es sich um eine deklarative Programmiersprache. Auf dem Gebiet der Künstlichen Intelligenz (KI) ist sie neben LISP[47] die bedeutendste. Entwickelt wurde Prolog 1972 in Marseille von der „Groupe d'Intelligence Artificielle de Luminy“ um den französischen Informatiker Alain Colmerauer [Leh98-ol]. Ein erster kommerzieller Prolog-Interpreter kam allerdings erst Anfang der 80er Jahre auf den Markt. 1995 entstand schließlich eine ISO-Standardisierung von Prolog. Ursprünglich wurde diese Programmiersprache zur Verarbeitung natürlicher Sprachen entwickelt [Hen00-ol].

[47] LISP (LISt Programming) wurde in den 50er Jahren von John McCarthy entwickelt und ist die zweitälteste noch in Verwendung stehende Programmiersprache.Zielsetzung bei der Entwicklung der Sprache war es, ein System zur Symbolmanipulation zur Verfügung zu haben. LISP stellt eine funktionale Programmiersprache dar und verwendet als Datenstruktur Listen. Allerdings stellt diese Programmiersprache direkt keine Methoden zur Verfügung, um Fakten logisch zu verknüpfen und so auf neue Fakten zu schließen. Die für ein Wissensbasiertes System notwendigen Komponenten, die Wissensbasis und die Inferenzmaschine, sind somit erst aus LISP-Konstrukten zu erstellen [Hyp00 -ol].

Prolog kann auch als „Maschinensprache eines Logik-Prozessors“ bezeichnet werden, da sie auf den mathematischen Grundlagen der Prädikatenlogik beruht. Ein Prolog-Programm ist eine Sammlung von so genannten Hornklauseln [Wik04 -ol].

Die logische Programmierung unterscheidet sich sehr stark von anderen Programmierstilen. Der größte Unterschied liegt darin, dass in Prolog im Gegensatz zu den herkömmlichen prozeduralen Programmiersprachen wie Fortran, Pascal oder C der Lösungsweg nicht mehr algorithmisch angegeben wird, sondern nur die Bedingungen, die die Lösung des Problems erfüllen sollen. Das Ziel ist also eine deklarative Programmierung, bei der eine gegebene Spezifikation direkt in ein Programm umgewandelt werden kann.

Bei Prolog handelt es sich um eine deskriptive Programmiersprache, d.h. die Lösung eines Problems wird durch Regeln, Fakten und Anfragen erreicht. Prozedurale Programmiersprachen sind hingegen imperativ. Die Lösung eines Problems muss in Form von Befehlsfolgen ausprogrammiert werden [Hyp00-ol].

In Prolog gelten folgende Begriffe bzw. Definitionen:

Unter einem *Literal* wird allgemein ein positives oder negatives Prädikat mit einer bestimmten Stelligkeit verstanden, z.B. die Temperatur(Wetter, 20). Erfolgt nun eine disjunkte Verknüpfung von Literalen ($L_1 \vee \ldots \vee L_n$) so wird dies als *Klausel* bezeichnet [Sch01-ol].

Schließlich existiert noch der Spezialfall einer Klausel, die *Hornklausel*. Sie besitzt höchstens ein positives Literal ($\neg P_1, ..., \neg P_{n-1}, P_n$). Dargestellt werden kann eine solche Klausel auch als Implikation (P_n -> $P_1 \wedge ... \wedge P_{n-1}$), oder in Prolog ($P_n$:- $P_1 \ldots P_{n-1}$). Im Wesentlichen werden drei Arten von Hornklauseln unterschieden. Eine Hornklausel bestehend aus genau einem positiven Literal heißt in Prolog Faktum, eine Klausel bestehend aus höchstens einem positivem Literal und beliebig vielen negativen Literale ist eine Regel, und eine Hornklausel ohne positives Literal ist eine Anfrage [Uni04-ol].

Eine Prolog-Regel besteht aus einem Kopf (Head) und einem Rumpf (Body) und wird als eine logische Implikation angesehen, die entgegen der Konvention in der Logik von rechts nach links gelesen wird. So wird beispielsweise die Implikation „wenn a und b, dann c“ geschrieben als „$a \wedge b \rightarrow c$“ in Prolog zu „c :- a, b.“, wobei das Zeichen ’:-’ dem umgedrehten Implikationspfeil ’ $\leftarrow$ ’entspricht und der Punkt die Prologregel beendet. Der Kopf der Regel besteht bei diesem Beispiel aus ’c’, der Rumpf der Klausel aus ’a’ und ’b’ [Sch01-ol].

Zusätzlich zu den Regeln gibt es in Prolog die Fakten. Ein Faktum repräsentiert eine Aussage und besteht aus einem Literal, das mit einem Punkt beendet wird. Fakten sind beispielsweise ’Temperatur(Wetter, 20).’ oder ’Bahnhof(Paderborn).’ [Sch01-ol].

Schließlich existieren in Prolog-Programmen noch Anfragen. Hierbei handelt es sich um ein Literal oder mehrere durch Kommata getrennte Literale. Eingeleitet wird ein solcher Ausdruck durch ’?-’. Beim Stellen einer Anfrage wird die Regelbasis eines Prolog-Programms durchlaufen und, wenn möglich, ein Ergebnis deduziert. Die übliche Antwort eines Prolog-Programms auf eine Anfrage ist ein einfaches ’yes’ oder ’no’. Darüber hinaus lassen sich die Variablenbelegungen mit ausgeben, wodurch auch ein Erstellen von sehr komplexen Anfragen möglich wird [Sch01-ol].