Ein Vertrauens- und Reputationsmodell für Multi-Agenten Systeme

Inaugural-Dissertation
zur Erlangung der Würde eines
Doktors der Wirtschaftswissenschaften
der Wirtschaftswissenschaftlichen Fakultät
der Albert-Ludwigs-Universität Freiburg im Breisgau

vorgelegt von
Boris Padovan
aus Rheinfelden (Baden)

2000

Erstgutachter: Prof. Dr. Günter Müller
Zweitgutachter: Prof. Dr. Franz Schober
Dekan: Prof. Dr. Heinz Rehkugler
Tag des Promotionsbeschlusses: 13.12.2000
Die Gutachter wurden durch die Fakultät bestimmt.

Vorwort

Die vorliegende Arbeit ist an der Abteilung Telematik des Instituts für Informatik und Gesellschaft der Albert-Ludwigs-Universität Freiburg im Rahmen des Projektes Avalanche entstanden. Unter Leitung von Professor Dr. Günter Müller und gemeinsam mit meinen Kollegen Dr. Detlef Schoder, Dr. Torsten Eymann und Stefan Sackmann wurden in Avalanche die Möglichkeiten des Einsatzes von Softwareagenten auf elektronischen Märkten erforscht. Eine weitere bisher in dieser Gruppe entstandene Dissertation ist die Arbeit des Projektgruppenleiters Dr. Torsten Eymann, „AVALANCHE - Ein agentenbasierter dezentraler Koordinationsmechanismus für elektronische Märkte“.

Besonders danken möchte ich meinem Doktorvater Professor Dr. Günter Müller, der mich gut bei meiner Forschungsarbeit geführt und mir gleichzeitig sehr große akademische Freiheiten gewährt hat. Er war stets ein kompetenter Ansprechpartner, von dem ich über das akademische Arbeiten hinaus sehr viel gelernt habe.

Auch meinen Kollegen Dr. Detlef Schoder, Dr. Torsten Eymann und Stefan Sackmann bin zu großem Dank verpflichtet, da ihre wertvollen und kritischen Anmerkungen einen wesentlichen Beitrag zu dieser Arbeit geleistet haben. Danken möchte ich auch meinem Kollegen Alf Zugenmaier für die kritische Durchsicht des Textes.

Herrn Professor Dr. Franz Schober möchte ich für die Übernahme des Koreferats herzlichen danken.

Ganz besonders Danken möchte ich jedoch meiner Frau Simone, die mich in den Jahren der Promotion stets zuverlässig und kritisch begleitet hat und unseren Zwillingen Gregor und Tristan, die mit ihrer Geburt bis zum Tag nach Abgabe der Arbeit gewartet haben.

Boris Padovan

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Ökonomische Transaktionen über offene Netze, wie dem Internet, verändern den Handel zwischen Unternehmen und zwischen Unternehmen und privaten Endverbrauchern. Dieser Handel wird in zunehmendem Maße auf offenen und im gesamten Internet verteilten elektronischen Marktplätzen stattfinden, auf denen Anbieter und Nachfrager von Gütern zusammentreffen. Elektronische Marktplätze werden die Zukunft des Electronic Commerce mitgestalten. Ihre steigende Verbreitung erschwert es menschlichen Marktteilnehmern, Angebot, Nachfrage und Preise auf den einzelnen Marktplätzen parallel zu verfolgen. Dies macht es notwendig, daß sich Marktteilnehmer auf ihnen repräsentieren lassen. Eine Möglichkeit hierzu ist der Einsatz von Software-Agenten. Sie repräsentieren ihre menschlichen Nutzer auf elektronischen Marktplätzen und sollen in deren Auftrag Transaktionen durchführen.

Software-Agenten sind in den letzten Jahren aus einer Synthese von Konzepten der Wirtschaftsinformatik, der Künstlichen Intelligenz und der Wirtschaftswissenschaften entstanden. Sie unterstützen wirtschaftliche Transaktionen, z.B. im Rahmen elektronischer Märkte und des elektronischen Geschäftsverkehrs. Sie unterscheiden sich von traditionellen Software-Anwendungen vor allem durch ihre Autonomie. Software-Agenten laufen kontinuierlich und selbständig in einer definierten Umgebung zusammen mit anderen Software-Agenten und Prozessen. Diese Eigenschaften sind insbesondere in dynamischen Umgebungen mit einer Vielzahl von sich ständig ändernden Informationen und parallelen Prozessen von großem Nutzen, wie sie im elektronischen Geschäftsverkehr in offenen Netzen und auf elektronischen Marktplätzen zu finden sind.

Software-Agenten beobachten diese Vielfalt von Märkten nicht nur permanent im Auftrag ihres Absenders, sondern treiben auf ihnen auch aktiv Handel. Identifiziert der Software-Agent ein interessantes Kauf- oder Verkaufsangebot, so beginnt er umgehend Verhandlungen mit dem betreffenden Partneragenten. Entsprechen sich Angebot und Nachfrage, können die Software-Agenten selbständig eine Transaktion durchführen.

Bisher werden Software-Agenten ausschließlich in geschlossenen Systemen genutzt, so daß ein globaler Zugriff auf sie möglich ist. Daher kann von zentraler Stelle ein Gesamtziel für das System definiert und alle Software-Agenten eines Informationssystems auf die Erreichung des Gesamtziels ausgerichtet werden.

Der elektronische Handel der Zukunft wird jedoch in offenen, heterogenen und komplexen Systemen stattfinden. Dadurch entstehen Probleme, die in geschlossenen Systemen nicht gegeben sind. Daher müssen die Verhaltensweisen der Software-Agenten dahingehend untersucht werden, daß auch in offenen Systemen eine Koordination von ökonomischen Prozessen möglich ist. Die Arbeiten von Eymann[1] geben deutliche Hinweise darauf, daß diese Koordinationsleistung durch den Einsatz von autonom handelnden Software-Agenten erbracht werden kann. Dadurch sind Software-Agenten auch in Anwendungen des offenen Electronic Commerce prinzipiell einsetzbar.

1.1 Vertrauen und Reputation in offenen Agenten-Systemen

Durch die Möglichkeiten von Ad-hoc-Kooperationen im Electronic Commerce begegnen sich auf offenen, elektronischen Marktplätzen Marktteilnehmer, die vollkommen autark und nur auf ihr eigenes Wohl bedacht sind. Jeder Teilnehmer wird als Güter- oder Diensteanbieter bzw. -nachfrager autonom, kompetitiv, individualistisch und nicht notwendigerweise kooperativ seine Ziele verfolgen. In offenen Systemen wirken die Teilnehmer nicht mehr auf ein gemeinsames Gesamtziel hin, wie in Abbildung 1 dargestellt.

[1] Vgl. hierzu etwa: Eymann, T. (2000).

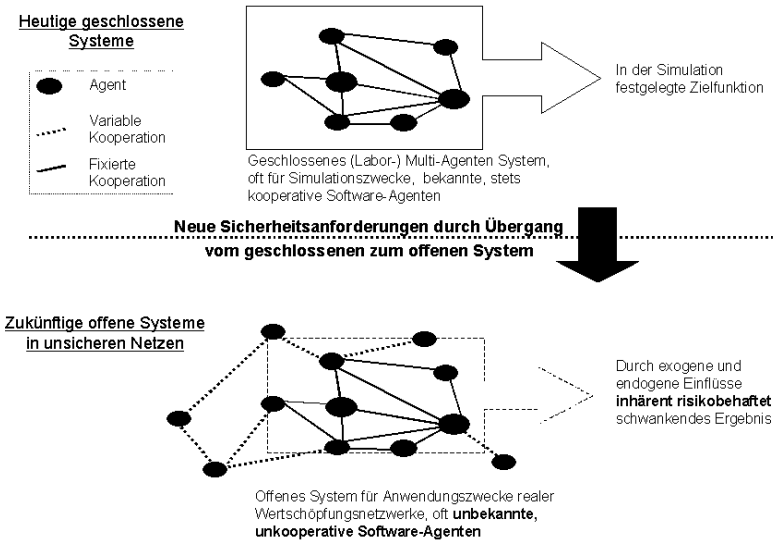


Abbildung 1: Neue Sicherheitsanforderungen für offene Systeme

Sie können sogar versuchen, durch betrügerisches, eigennütziges Verhalten Vorteile gegenüber ihren Mitbewerbern zu erhalten und schädigen damit nicht nur die rechtschaffenen, kooperativen Akteure, sondern eventuell auch den Betreiber eines elektronischen Marktplatzes, der dieses Verhalten zuläßt. Da sie in keinerlei hierarchischer oder weisungsberechtigter Beziehung zueinander stehen, kann keine direkte Verhaltensänderung durch Anweisung oder Sanktionierung initiiert werden.

Die empirische Relevanz dieser Problemstellung ist besonders aus den an der Abteilung Telematik des Instituts für Informatik und Gesellschaft durchgeführten Studien *Electronic Commerce Enquête I*[2] und *Electronic Commerce Enquête*

[2] Die Electronic Commerce Enquete I wurde im Herbst 1997 von Institut für Informatik und Gesellschaft, Abteilung Telematik zusammen mit der Computer Zeitung und Gemini Consulting durchgeführt. Es war eine umfangreiche Erhebung mit insgesamt 116.621 ausgesandten Fragebögen an Management und Entscheidungsträger im deutschsprachigen Raum. Gegenstand der Untersuchung ist der betriebswirtschaftliche Nutzen von Web-basiertem Electronic Commerce aus Unternehmenssicht. Erhoben wurden die Unternehmenseinschätzungen zu den Hürden, Auswirkungen sowie strategischen Implikationen des Einsatzes von Electronic Commerce. Insgesamt beteiligten sich über 900 Unternehmensvertreter an den 157 Fragen umfassenden Erhebungsbogen.
Vgl. Schoder, D. / Strauß, R. / Welchering, P. (1998).

II[3] ersichtlich. In diesen Studien wurden fehlende Instanzen, die über Sanktionsmöglichkeiten verfügen, als eines der größten Hemmnisse in der Durchführung von wirtschaftlichen Transaktionen im Internet identifiziert. Besonders bedeutend wurden in diesem Zusammenhang „regulatorische Defizite, beispielsweise für elektronisch signierte Verträge“[4] eingeschätzt. Aber auch Probleme asymmetrischer Informationen, die sich aus der unterschiedlichen Verteilung wettbewerbs- und kooperationsrelevanter Informationen zwischen den Partnern ergeben haben, führen zur Möglichkeit opportunistischen Verhaltens zu Lasten anderer Kooperationsteilnehmer. Dies stellte sich als größte Hürde für den ökonomischen Erfolg von Unternehmenskooperationen heraus. Durch Kooperationen mit neuen Partnern erhoffte man sich eine Realisierung von Größen- und Verbundvorteilen. Dem stand jedoch die „ex ante“ unbekannte Fähigkeit und Willen der „Neulinge“ gegenüber, zur Zielerreichung einer Unternehmenskooperation beizutragen und dafür egoistische Ziele zurückzustellen: Als bedeutende bzw. sehr bedeutende nicht-technische Probleme wurden die Einschätzung der Leistungsfähigkeit potentieller Kooperationspartner (62,6% aller Antwortenden) und die Einschätzung der Vertrauenswürdigkeit möglicher Kooperationspartner (60,6%) genannt.[5]

Diese Probleme gelten unabhängig davon, ob ein menschlicher Nutzer in einem offenen, elektronischen Marktsystem Kooperationen führt, oder ob er sich von einem Software-Agenten repräsentieren läßt, der nach seinen Maßgaben und in seinem Auftrage handelt.

Ziel der vorliegende Arbeit ist es, einen Vertrauens- und Reputationsmechanismus für agentenbasierte, offene, elektronische Märkte zu entwickeln, der den Software-Agenten einen Anreiz verschafft, sich kooperativ und nicht betrüge-

---

[3] Die Electronic Commerce Enquete II der Abteilung Telematik des Instituts für Informatik und Gesellschaft wurde zusammen mit dem Konradin Verlag durchgeführt. Es war die größte deutschsprachige, empirische Erhebung, die erstmals neben den nicht-technischen, organisatorischen Erfolgsfaktoren der Unternehmensvernetzung auch deren internetbezogenen Wettbewerbsfaktoren identifiziert. Es wurde ein Rücklauf von 738 Fragebögen erzielt, aus denen über 135.000 Datenpunkte gewonnen wurden. Vgl. Eggs, H. / Englert, J. (2000).

[4] Vgl. Schoder, D. / Müller, G. (1999).

[5] Vgl. Eggs, H. / Englert, J. (2000).

risch zu verhalten. Falls sich die Software-Agenten dennoch betrügerisch und nicht kooperativ verhalten, sollen sie auf eine Weise durch den Mechanismus sanktioniert werden, daß sie keine ökonomischen Vorteile gegenüber ehrlichen Partnern erzielen können. Um dies zu erreichen soll durch den Mechanismus ein *Vertrauensverhältnis* zwischen den einzelnen Software-Agenten aufgebaut werden. Vertrauen bietet in realen, menschlichen Gesellschaften eine Möglichkeit für den einzelnen, Interaktionen durchzuführen, trotz möglicherweise betrügerischen und unkooperativen Verhaltens anderer. Diese Funktion von Vertrauen soll auf Software-Agenten übertragen werden. Dazu stellt der Vertrauensmechanismus den Software-Agenten vor Durchführung einer Transaktion Reputationsinformationen über ihrer agentenbasierten Transaktionspartner zur Verfügung. So kann sein Kooperationsverhalten vorab eingeschätzt und das eigene Verhalten entsprechend angepaßt werden.

Dabei soll auch dargestellt werden, welche technischen Voraussetzungen gegeben sein müssen, damit ein Vertrauens- und Reputationsmechanismus die Interaktionen der Software-Agenten beeinflussen kann. Die Arbeit schließt mit der Evaluation des Vertrauens- und Reputationsmechanismus, die durch mehrere experimentelle Testreihen durchgeführt wurde.

Der in der vorliegenden Arbeit entwickelte Vertrauens- und Reputationsmechanismus wurde im Multi-Agenten System AVALANCHE implementiert. AVALANCHE ist ein Prototyp für einen agentenbasierten elektronischen Marktplatz und bildet eine wesentliche Grundlage der vorliegenden Arbeit. Es wurde in den Jahren 1996 bis 2000 an der Abteilung Telematik des Instituts für Informatik und Gesellschaft der Albert-Ludwigs-Universität Freiburg konzipiert und implementiert.[6]

[6] Vgl. AVALANCHE (2000).

1.2 Vorgehensweise und Aufbau der Arbeit

In Kapitel 2 werden die wesentlichen Eigenschaften von Software-Agenten herausgestellt. Außerdem wird eine Abgrenzung zu „herkömmlichen“ Software-Anwendungen gezogen. Insbesondere wird gezeigt, in welchen Abstufungen Software-Agenten mit ihrer Umwelt interagieren können. Ausgehend von einem einfachen, modellhaften und idealisierten Anwendungsszenario in Kapitel 2.2 wird in Kapitel 2.3 die Funktionsweise von AVALANCHE ausführlich dargestellt. Die einzelnen Komponenten des Systems und der Ablauf der Transaktionen sind in den Abschnitten 2.4 und 2.5 beschrieben. Anhand von Testläufen des geschlossenen Prototyps kann dessen Funktionsfähigkeit gezeigt werden (Kapitel 2.6). Bei einer Öffnung des Prototyps für fremde, agentenbasierte Marktteilnehmer besteht die Gefahr, daß diese durch betrügerisches Verhalten ökonomische Vorteile gegenüber ehrlichen Marktteilnehmern erzielen, was schließlich sogar zum Zusammenbruch des gesamten Systems führen kann (Kapitel 2.7). Am Ende des Kapitels wird daher die Motivation zur Entwicklung eines Mechanismus deutlich, der einen Anreiz schafft, Transaktionen vereinbarungsgemäß durchzuführen.

In Kapitel 2.10 werden in einem Exkurs Konflikte beschrieben, die beim Einsatz eines Vertrauens- und Reputationsmechanismus auftreten können. Im Mittelpunkt steht hierbei der gegenläufige Wunsch des Einzelnen nach Anonymität und der Anspruch Geschädigter auf Identifizierung betrügerischer Akteure.

Der vorgestellte Vertrauen- und Reputationsmechanismus beruht auf der Bildung von Vertrauen zwischen Software-Agenten unter Berücksichtigung ihrer Reputation. In Kapitel 3 wird deshalb in einem ersten Schritt der Begriff *Vertrauen* aus soziologischer Sicht erörtert. Dargestellt sind die wesentlichen Merkmale und Funktionen von Vertrauen. Dazu werden auch Kriterien erarbeitet, unter welchen Voraussetzungen die Funktionen von Vertrauen überhaupt genutzt werden können. In einem weiteren Schritt wird der Begriff der Reputation erörtert und das Verhältnis zwischen Vertrauen und Reputation diskutiert. Die Diskussion der beiden Begriffe wird in Kapitel 3.3 dahingehend fokussiert,

ob Software-Agenten überhaupt in der Lage sind, die Funktionen von Vertrauen und Reputation zu nutzen. Dazu wird in einem weiteren Schritt ausführlicher geklärt, wem eigentlich vertraut wird: dem Software-Agenten oder seinem menschlichen Nutzer.

In Kapitel 3.4 folgt eine Beschreibung und Diskussion verwandter Forschungsprojekte, von denen fünf ausführlicher behandelt werden. Anhand dieser Diskussion werden Aspekte abgeleitet, die für den Vertrauens- und Reputationsmechanismus in AVALANCHE relevant sind.

In Kapitel 4 wird der Reputationsmechanismus für AVALANCHE entwickelt. Seine Funktionsweise wird vor dem Hintergrund der unterschiedlichen Kenntnisse über die Transaktionspartner aufgezeigt (Kapitel 4.3). Dem folgt eine Evaluation des Mechanismus durch den Einsatz im Prototypen AVALANCHE. Zum Ende des Abschnitts werden die Ergebnisse ausführlicher diskutiert und interpretiert.

Die Arbeit schließt mit einer Bewertung des durch den Mechanismus geleisteten Beitrags für die Gestaltung offener Marktplätze und leitet daraus weitere Forschungsfragen ab.

2 Software-Agenten und das MAS AVALANCHE

Software-Agenten werden auf elektronischen Märkten in naher Zukunft eine zentrale Rolle spielen[7]. Ein zentrales Forschungsgebiet ist dabei die Entwicklung automatisierter Verhandlungskonzepte: auf elektronischen Marktplätzen werden Käufer- und Verkäufer-Agenten automatisiert und im Auftrag ihres Anwenders autonom Handel betreiben. Die Marktplätze selbst können dabei auf unterschiedliche Produkte spezialisiert sein, bspw. Rohstoffe, Informationsgüter, Dienstleistungen. Möglich ist auch der Handel mit immateriellen Gütern, z.B. Bandbreite[8]. Software-Agenten werden diese Vielfalt an Märkten im Auftrag ihres Absenders permanent beobachten und auf ihnen auch aktiv Handel betreiben: Identifiziert der Software-Agent ein interessantes Kauf- oder Verkaufsangebot, so beginnt er umgehend Verhandlungen mit dem betreffenden Partneragenten. Entsprechen sich Angebot und Nachfrage, werden die Software-Agenten selbständig eine Transaktion durchführen. Erste Beispiele hierzu sind der elektronische Marktplatz *Kasbah*,[9] der am *Massachusetts Institute of Technology* entwickelt wurde, und die *Information Economies* des Unternehmens IBM.[10]

Bevor das Multi-Agenten System AVALANCHE vorgestellt wird, werden in den folgenden Abschnitten die wesentlichen Eigenschaften von Software-Agenten und Multi-Agenten Systemen dargestellt.[11] Ausführlicher behandelt wird lediglich die Klassifizierung von Software-Agenten nach José Vidal und Edmund Durfee,[12] da deren Einordnung eine grundlegende Strukturierung der Diskussion von Vertrauens- und Reputationsmechanismen in Multi-Agenten Systemen bietet.

---

7 Vgl. Preist, C. (1998).
8 Vgl. Huberman, B. / Lukose, R. (1997).
9 Vgl. Chavez, A. / Maes, P. (1996).
10 Vgl. Kephart, J. / Hanson, J. / Levine, D. / Grosof, B./ Sairamesh, J. / Segal, R. / White, S. (1998).
11 Eine weiter Behandlung von Software-Agenten findet sich in: Eymann, T. (2000). S. 76-133.
12 Vidal, J. / Durfee, E. (1996).

2.1 Eigenschaften von Software-Agenten

Software-Agenten unterscheiden sich von herkömmlichen Rechneranwendungen durch ihr „zielgerichtetes, proaktives und selbst-startendes Verhalten.“[13] Dieses Verhalten ermöglicht es, daß Software-Agenten auch schlecht strukturierte Problemstellungen im Auftrag ihrer Nutzer lösen können.[14]

> „An agent is a system that tries to fulfil a set of goals in a complex, dynamic environment. An agent is situated in the environment: it can sense the environment through its sensors and act upon the environment using its actuators.“[15]

So können Software-Agenten in einer definierten Umgebung laufen, in der sich ständig eine Vielzahl von Informationen ändert. Eine weitergehende eindeutige Definition von Software-Agenten ist in der Literatur nicht vorhanden. Dies wird besonders in ihren verschiedenen wissenschaftlichen Einordnungen deutlich.[16] Lediglich in einem Punkt stimmen alle Definitionen überein:

- **Autonomes Handeln**

  Das definierende Merkmal von Software-Agenten, und somit ihr wesentlicher Unterschied zu herkömmlichen Softwareanwendungen, ist ihre *Autonomie*. Per Definition laufen Software-Agenten „continuously and autonomously in a particular environment, often inhabited by other agents and processes.“[17] Autonomie wird als zielgerichtetes, proaktives und selbst-startendes Verhalten begriffen.[18] So faßt Wooldridge die verschiedenen Definitionen pragmatisch zusammen:

---

[13] Vgl. Bradshaw, J. (1997).
[14] Vgl. Eymann, T. (2000). S. 23.
[15] Maes, P. (1994b). S. 136.
[16] Vgl. hierzu etwa Gilbert, D. / Aparicio, M. / Atkinson, B. / Brady, S. / Giccarino, J. / Grosof B. / O’Connor, P. / Osisek, D. / Pritko, S. / Spagna, R. / Wilson, L. (1995); Nwana, H. (1996), Franklin, S. / Graesser, A. (1996) oder Bradshaw, J. (1997).
[17] Shoham, Y. (1997).
[18] Vgl. Bradshaw, J. (1997). S. 8.

> „To summarise, agents are simply computer systems that are capable of autonomous action in some environment in order to meet their design objectives.“[19]

Wooldridge[20] sieht neben den Autonomie zwei weitere wesentliche Eigenschaften, die typisch für Software-Agenten sind.

- **Soziales Verhalten**

  Software-Agenten sind zu *Sozialverhalten* fähig. Dieses Verhalten geht über eine rein technische Kommunikation weit hinaus und ermöglicht Verhandlungen und Kooperationen mit anderen Software-Agenten oder menschlichen Akteuren. Dadurch können Software-Agenten eigene Problemstellungen lösen oder weitere Software-Agenten bei der Lösung von Problemstellungen unterstützen.

- **Pro- und Reaktivität**

  Durch ihre *Proaktivität* können Software-Agenten selbst initiativ werden und neue Handlungsmöglichkeiten identifizieren und verfolgen. Bei einer sich ändernden Umgebung können Software-Agenten durch ihre *Reaktivität* ihre Vorgehensweisen ändern, um den neuen Bedingungen Rechnung zu tragen. Dabei stehen Proaktivität und Reaktivität[21] in einer engen Wechselbeziehung. Jedoch muß sichergestellt werden, daß der Software-Agent lange genug auf sein Ziel fokussiert bleibt und nicht permanent zwischen verschiedenen Verfahren wechselt.

---

[19] Wooldrigde, M. (1999). S. 32.
[20] Vgl. Wooldrigde, M. (1999). S. 33ff.
[21] Vgl. Eymann, T. (2000). S. 78ff.

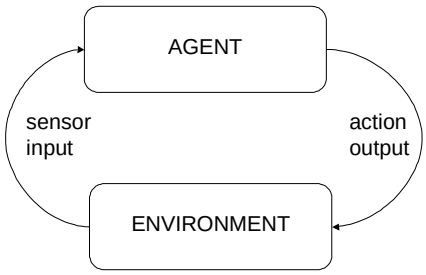


Abbildung 2: Rückkopplung zwischen Software-Agent und Umgebung[22]

Abbildung 2 verdeutlicht die Interaktion eines Software-Agenten mit seiner Umwelt. Seine *Reaktivität* ist im wesentlichen durch den *Sensor Input* möglich, mit dem er seine Umwelt erfaßt. Das *proaktive* Verhalten (*Action Output*) ist nicht nur eine Reaktion, sondern auch eine aktive Beeinflussung der Umwelt.

2.1.1 Das *k-level* Konzept nach Vidal und Durfee

Software-Agenten unterscheiden sich voneinander in ihren Fähigkeiten, ihr Verhalten auf andere Software-Agenten abzustimmen. Dieses Verhalten beruht auf ihrem internen Modell von ihrer Umwelt. Dieses Modell ist für die komplette Informationsverarbeitung, die Ziehung von Schlußfolgerungen, für Entscheidungsfindungen und für Lernvorgänge verantwortlich.[23] Vidal und Durfee ordnen Software-Agenten nach diesen Fähigkeiten in das *k-level Konzept* ein.[24] Sie unterscheiden zwischen drei Kategorien.

[22] Wooldridge, M. (1999). S. 29.
[23] Vgl. Eymann, T. (2000). S. 83.
[24] Vgl. Vidal, J. / Durfee, E. (1996).

- ***0-level* Agenten**

  Software-Agenten, deren Verhalten lediglich von ihren Inputs und den aus ihren Handlungen resultierenden Belohnungen bestimmt wird, werden von Vidal und Durfee als *0-level* Agenten bezeichnet. Sie verfügen nur ein äußerst einfaches Modell ihrer Umwelt, die sie als „Black-Box" auffassen. Beide charakterisieren sie als Software-Agenten, „[which] must learn everything they know from observations they make about the environment, and from any rewards they get."[25] Für *0-level* Agenten spielen eventuell weitere vorhandene Software-Agenten in ihrer Umgebung keine Rolle für ihre Entscheidungen.

- ***1-level* Agenten**

  1-level Agenten hingegen berücksichtigen die Existenz anderer Software-Agenten und beziehen diese in ihre Verhaltensstrategien mit ein. Sie verfügen jedoch über keinerlei Möglichkeit, die Entscheidungen der anderen Software-Agenten nachzuvollziehen oder deren strategischen Absichten zu ermitteln. „[They] can only model others by looking at their past behavior and trying to predict, from it, their future actions."[26] Software-Agenten der *1-level* Stufe verfügen somit über ein „Gedächtnis", mit dessen Hilfe sie Erfahrungen mit anderen Software-Agenten abspeichern können, ohne jedoch Rückschlüsse auf deren zukünftiges Verhalten ableiten zu können.

- ***2-level* Agenten**

  Software-Agenten die auf Grundlage des *1-level* Modells die Aktionen fremder Software-Agenten zu prognostizieren, werden von Vidal und Durfee als *2-level* Agenten bezeichnet. Jedoch sind diese Software-Agenten in ihrer Entscheidungsfindung und Implementation komplex und die Projektion fremder Handlungen führt oft zu einer unendlichen Rekursion.[27]

Zusammenfassend meinen Vidal und Durfee:

---

[25] Vidal, J. / Durfee, E. (1996). S. 6.
[26] Vgl. Vidal/Durfee (1996). S. 7.

„There are agents with no models (0-level agents) that are unaware of the existence of other agents in the world, agents with sub-intentional models (1-level agents) that keep models of others, and agents with intentional models (2-level agents) that have deeper models of others.“[28]

Agentenklassifizierung **Kriterien**	**0-level Agenten**	**1-level Agenten**	**2-level Agenten**
Agent verfügt über ein Modell seiner Umwelt	**Nein**	**Ja**	**Ja**
Agent berücksichtigt die Existenz weiterer Agenten	**Nein**	**Ja**	**Ja**
Agent modifiziert sein Verhalten auf Grund des Verhaltens weiterer Agenten	**Nein**	**Ja**	**Ja**
Agent berücksichtigt die Handlungsstrategie weiterer Agenten	**Nein**	**Nein**	**Ja**
Agent versucht durch seine Handlungen die Handlungen weiterer Agenten zu beeinflussen	**Nein**	**Nein**	**Ja**

Tabelle 1: Wesentliche Agenteneigenschaften zur Einordnung in das *k-level* Konzept.

2.1.2 Distributed Problem Solving und Multi-Agenten Systeme

Aus den vorhergehenden Ausführungen wird deutlich, daß Software-Agenten häufig gemeinsam mit weiteren Software-Agenten genutzt werden. Dabei lassen sich bei der gemeinsamen Nutzung von mehreren Software-Agenten zwei grundlegende Ansätze unterscheiden: *Distributed Problem Solving* und *Multi-Agenten Systeme*.

[27] Vgl. Eymann, T. (2000). S. 84.
[28] Vidal, J. / Durfee, E. (1996). S. 2.

Beim Distributed Problem Solving wird davon ausgegangen, daß ein bekanntes Problem durch viele einzelne Module gemeinsam und kooperativ gelöst werden kann.[29] Diese Module können durchaus autonom handelnde Software-Agenten sein. Zur Lösung eines Problems werden Software-Agenten von einer zentralen Instanz eingesetzt. Sie interagieren stets kooperativ mit ihren Partnern und tauschen Informationen untereinander aus, die für die gemeinsame Lösung des Problems relevant sind. Da die Strategie jedes einzelnen Software-Agenten bekannt ist, können Konflikte vorhergesehen und gelöst werden.[30]

Multi-Agenten Systeme hingegen verfolgen einen konträr zum Distributed Problem Solving stehenden Ansatz. Es gibt kein zentrales Problem, das bereits vorab bekannt ist und gemeinsam gelöst werden soll. Software-Agenten werden in Multi-Agenten Systemen eingesetzt, um Probleme, die erst im System entstehen, zu lösen. Dabei verfolgen die Software-Agenten kein gemeinsames Gesamtziel, sondern versuchen ausschließlich ihren eigenen Nutzen bzw. den Nutzen ihrer menschlichen Prinzipale zu optimieren.[31] Ihr Handeln ist daher nicht notwendigerweise kooperativ, sondern stets eigennützig. Die Software-Agenten sind in ihren Handlungen autonom und können grundsätzlich von einander verschieden sein. Jeder Software-Agent verfügt nur über unvollständige Informationen über seine Umwelt und von einem Informationsaustausch zwischen den konkurrierenden Software-Agenten kann nicht ausgegangen werden.

Die in der Literatur beschriebenen Implementationen von Multi-Agenten Systemen variieren zwischen der Nutzung von wenigen Software-Agenten, bspw. zehn bei einem *Agent-enhanced Workflow*[32] und von mehreren Tausend, wie von Kephart[33] in der *Information Economy* implementiert.

---

[29] Vgl. Eymann, T. (2000).
[30] Vgl. Malone, T. (1987).
[31] Vgl. Durfee, E. / Rosenschein, J. (1994). S. 3.
[32] Vgl. Jennings, N. / Sycara, K. / Wooldridge, M. (1998).
[33] Vgl. Kephart, J. / Hanson, J. / Greenwald, A. (2000).

2.2 Das MAS AVALANCHE

AVALANCHE ist ein Multi-Agenten System (MAS) und ein Prototyp für agentenbasierte, elektronische Marktplätze. Es ist im Rahmen des Projektes *Telematik und Organisationsstrukturen (TELOS)* an der Abteilung Telematik des Instituts für Informatik und Gesellschaft der Universität Freiburg entstanden.[34] Der Prototyp besteht aus mehreren (ca. 5-10) elektronischen Marktplätzen, auf denen der agentenbasierte Handel erfolgt. Der Aufbau und die Funktionsweise der elektronischen Marktplätze und der Software-Agenten soll im Folgenden anhand eines modellhaften Szenarios beschrieben werden.[35] Anhand des Szenarios wird aufgezeigt, welche Anforderungen das Multi-Agenten System AVALANCHE für ökonomische Transaktionen erfüllen kann.

2.2.1 Anwendungsszenario

Das als Ausgangspunkt verwendete Szenario bildet eine Wertschöpfungskette in der holzverarbeitenden Industrie modellhaft ab. Idealtypisch besteht eine Wertschöpfungskette aus mehreren verbundenen organisatorischen Einheiten,[36] die Inputgüter (Rohstoffe) beziehen und Outputgüter (Produkte) verkaufen. Diese Entitäten verarbeiten Rohstoffe mit den Produktionsfaktoren Arbeit und Wissen und erhöhen so den Wert der Produkte. Diese Produkte dienen wiederum als Inputfaktoren der nächsten Stufe der Wertschöpfungskette, bis das Endprodukt den Konsumenten erreicht.

Anhand einer modellhaften Wertschöpfungskette, der Produktion von Tischen, wird die Funktionsweise das MAS AVALANCHE dargestellt. Drei unterschiedliche Klassen von Handwerkern (*Holzfäller*, *Zimmermänner* und *Schreiner*) stel-

---

[34] Das Projekt TELOS wurde an der Abteilung Telematik des Insituts für Informatik und Gesellschaft der Albert-Ludwigs-Universität Freiburg vom 1997 bis 2000 durchgeführt. Untersucht wurden die Wechselwirkungen zwischen inner- und interbetrieblichen Organisationsstrukturen und dem Einsatz von Telematiksystemen in Unternehmungen. Vgl. Telos (1998).

[35] Eine weiter Beschreibung des Prototyps findet sich in: Eymann, T. / Padovan, B. / Schoder, D. (1998a); Eymann, T. / Padovan, B. / Schoder, D. (1998b); Eymann, T. / Padovan, B. / Schoder, D. (1998c); Eymann, T. / Padovan, B. / Schoder, D. (1998d) und Eymann, T. / Padovan, B. (1999).

[36] Vgl. Porter, M. / Millar, V. (1985).

len gemeinsam Tische her. Dazu führt jeder Handwerker eine bestimmte Tätigkeit durch:

- Holzfäller kaufen Bäume als Rohstoffe und produzieren aus ihnen Bretter.
- Zimmermänner kaufen diese Bretter und stellen aus ihnen Platten her.
- Schreiner kaufen die Platten der Zimmermänner und produzieren aus ihnen Tische.

2.2.2 Die MAS AVALANCHE Wertschöpfungskette

Der jeweiligen Produktion liegt eine einfache lineare Produktionsfunktion zu Grunde. Die erzeugten Produkte (Bretter, Platten, Tische) werden von den Handwerkern auf Marktplätzen gehandelt. Dabei existieren mehrere Holzfäller, Zimmermänner und Schreiner, die untereinander im Wettbewerb stehen.[37] Die Schnittstellen der Wertschöpfungskette zur Außenwelt bilden zwei besondere Entitäten: Produzenten und Konsumenten. Die Eingangspreise und -mengen, durch die Bäume zu Beginn der Wertschöpfungskette in das System eingeführt werden, sind durch die Produzenten determiniert. Diese verfolgen lediglich eine Absatzstrategie unter Annahme unendlicher Verfügbarkeit von nicht näher bezeichneten Rohstoffen. In gleicher Weise werden die fertigen Tische durch Konsumenten aufgekauft, die eine Einkaufsstrategie mit einer prinzipiell unendlichen Liquidität darstellt. Durch die Betrachtung dieses kleinen Ausschnitts, der aus einer komplexen Volkswirtschaft herausgelöst ist, kann von makroökonomischen Problemen, wie Inflationen, exogene Preisschocks, etc. abstrahiert werden.

[37] Vgl. Porter, M. / Millar, V. (1985).

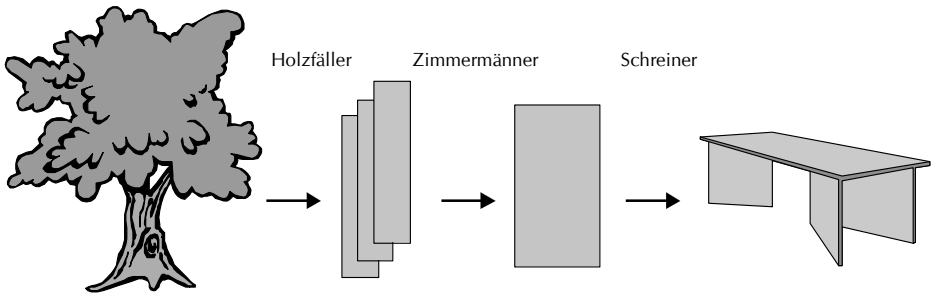


Abbildung 3: Eine modellhafte Wertschöpfungskette.

Die Software-Agenten von AVALANCHE repräsentieren die einzelnen Handwerker. Sie verfolgen ihre Ziele durch autonomes, deliberatives Handeln und setzen dabei adaptive Strategien ein.

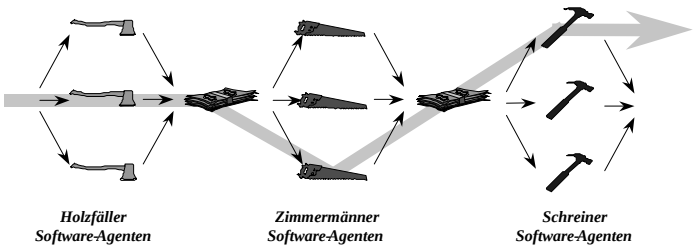


Abbildung 4: Software-Agenten repräsentieren die Handwerker

2.3 Architektur und Komponenten des MAS AVALANCHE

Die Implementation des Multi-Agenten-Systems wird in der Programmiersprache JAVA (Version 1.3) durchgeführt. Neben den Diensten von JAVA selbst wird eine Object Request Broker (ORB)-Klassenbibliothek namens VOYAGER® eingesetzt,[38] welche die Erstellung mobiler Agenten unterstützt. Die in AVALANCHE eingesetzten Software-Agenten sind damit vollständig voneinander unabhängige Java-Prozesse.

[38] Vgl. ObjectSpace (1999).

2.3.1 Marktplätze

Lokationen bilden die Marktplätze und sind Java Virtual Machines (JVM), die als Serverprozesse auf einem vorgegebenen Rechner in einem TCP/IP-basierten lokalen Netz laufen und durch eine IP-Port-Adresse eindeutig identifizierbar sind. Sie werden zu Beginn jedes Experiments gestartet und nach Beendigung des Experiments wieder heruntergefahren. Es ist möglich, mehrere Java Virtual Machines (JVM) gleichzeitig auf einem Computer laufen zu lassen. Jede JVM ist dann durch die Angabe der IP-Adresse des Computers (z.B. 132.230.16.32) und eine Portadresse (z.B. 7777) eindeutig ansprechbar (z.B. 132.230.16.32:7777). Dadurch kann ein ganzes Netzwerk von Marktplätzen aufgesetzt werden, innerhalb dessen sich die mobilen Agenten von Lokation zu Lokation bewegen können.

Auf jeder Lokation gibt es eine einzige Instanz der Klasse *AvLocationAgent*. Jede Klasse *AvLocationAgent* stellt für die Software-Agenten ein Verzeichnisliste der am Marktplatz anwesenden Software-Agenten und bestimmte Hilfsmethoden zur Unterstützung der Interaktion bereit. Um sich miteinander zu vernetzen, stellt jede *AvLocationAgent* die Methoden *link()* und *unlink()* bereit, über die ein dynamisches Netzwerk von Lokationen aufgebaut wird. Der Verzeichnisdienst des *AvLocationAgent* wird von den *AvTradeAgent*s benutzt, um sich auf dem Marktplatz an- und abzumelden (*register()* und *deregister()*). Dadurch wissen andere Software-Agenten von deren Anwesenheit und können ihnen zur Initiierung einer Verhandlung Nachrichten senden.

Die Marktplätze stellen für die Software-Agenten lediglich eine Umgebung bereit in der sie interagieren können, d.h. sie liefern ein Verzeichnis, in dem die Software-Agenten ermitteln können, welche weiteren Software-Agenten sich auf dem Marktplatz befinden. Der Marktplatz als solcher beeinflußt jedoch niemals das Handeln einzelner Software-Agenten und deren Interaktion.

2.3.2 Software-Agenten im MAS AVALANCHE

Das Kernstück von AVALANCHE sind die autonom handelnden Software-Agenten, die aus der Klasse *AvTradeAgent* gebildet sind. In dieser Klasse werden die Handelsprotokolle und die Kommunikation zwischen den Software-Agenten definiert. Jeder Software-Agent ist durch eine eindeutige Kennziffer identifizierbar. Er kann mit anderen Software-Agenten direkt und bilateral über Nachrichten kommunizieren, ebenso wie mit dem *AvLocationAgent* desjenigen Marktplatzes, auf dem sich der Agent zum jeweiligen Zeitpunkt befindet. Durch die Implementation als jeweils separate „Threads“ werden die Software-Agenten parallel ausgeführt, die Reihenfolge ihres Aufrufs ergibt sich durch die Implementation der JVM und deren Scheduler und wird weder von den Software-Agenten selbst noch von den elektronischen Marktplätzen beeinflußt.

Neben den beschriebenen Bausteinen, *AvTradeAgent* und *AvLocationAgent*, besteht noch die Klasse *AvInfoServer*. Sie steuert das Führen von Protokolldateien, die Bildschirmanzeige und den Start bzw. das Ende der Testläufe von AVALANCHE. Somit übernimmt sie lediglich technische Funktionen und wird im vorliegendem Text nicht weiter beschrieben.

Zusammenfassend sind in Abbildung 5 die einzelnen Komponenten von AVALANCHE und ihr Zusammenwirken abgebildet.

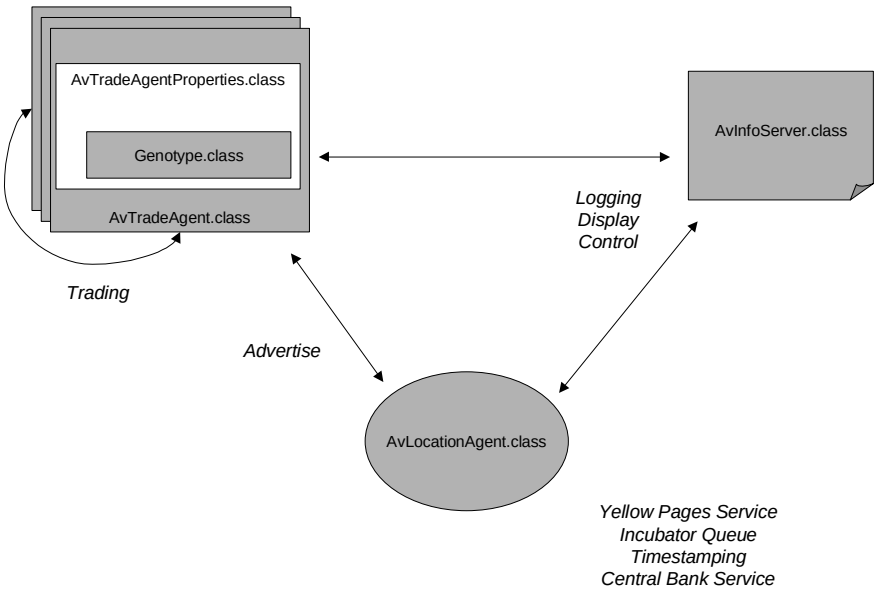


Abbildung 5: Architektur von AVALANCHE

2.4 Verhalten der AVALANCHE Software-Agenten

2.4.1 Produktionsfunktion

Jeder AVALANCHE Software-Agent verfügt über eine „virtuelle“ Produktionsfunktion, die auf der realen Produktionsfunktion des von ihm repräsentierten Unternehmers beruht. In der augenblicklichen Realisierung sind die Produktionsfunktionen linear und fest in ihrem Input-Outputverhältnis. Diese Restriktion kann jedoch aufgegeben werden, so daß die Produktionsfunktionen auf einer Reihe unterschiedlicher In- und Outputfaktoren beruhen kann, um bspw. Kuppelproduktionen zu beschreiben.

Nach dem Einkauf von Inputfaktoren warten die Software-Agenten einen festlegbaren Zeitabschnitt, um den laufenden Produktionsprozeß ihrer menschlichen Nutzer abzuwarten. Dieser Zeitabschnitt ist in der augenblicklichen Implementation für alle Software-Agenten identisch.

2.4.2 Kommunikation

Die AVALANCHE Software-Agenten kommunizieren ohne fremde Moderation und bilateral durch das Übersenden von Nachrichten. Das genutzte Verhandlungsprotokoll beruht zu großen Teilen auf der FIPA *Agent Communication Language*.[39] Implementiert wurden zusätzlich das *Iterated Contract Net Protocol*[40] sowie Elemente aus den Arbeiten von Erikkson[41] und Sandholm,[42] um so das stark eigennützige Verhalten der Software-Agenten herauszustellen. Vergleichbare Ansätze zu einer vollautomatischen Verhandlungsstrategie existieren außerdem im Bereich des agentenbasierten Electronic Commerce[43] und der marktorientierten Programmierung.[44]

2.4.3 Verhandlungsstrategie

Die Verhandlungsstrategie des einzelnen Software-Agenten basiert auf acht Parametern, die sein jeweiliges Verhalten bestimmen. Sie sind aus den Eigenschaften abgeleitet, die nach Pruitt menschliches Handeln determinieren.[45] Die Parameter haben Werte, die jeweils bei der Initialisierung der Software-Agenten festgelegt werden.

- Der Parameter *acquisitiveness* stellt die Wahrscheinlichkeit dar, daß ein Software-Agent seinen Angebots- oder Nachfragepreis ändert.

---

39 Vgl. FIPA (1998).
40 Vgl. Smith, R. (1980).
41 Vgl. Eriksson, J. / Finne, N. / Janson, S. (1996).
42 Vgl. Sandholm, T. (1996).
43 Vgl. Chavez A. / Maes, P. (1996) und Preist, C. (1998).
44 Vgl. Wellman, M. (1996).
45 Vgl. Pruitt, D. (1981).

- Durch den Parameter *delta price change* wird festgelegt, in welchen Schritten die Verhandlungspartner ihre Preise einander angleichen. Bei einem Wert von *del_change=0.25* für Nachfrager und Anbieter sind, falls keiner die Verhandlungen abbricht, mindestens zwei Verhandlungsrunden notwendig, um zu einer Einigung zu kommen.
- Beim Neubeginn einer Verhandlung wird ein Verkäufer wieder mit hohen Preisen, ein Einkäufer mit niedrigen Preisen beginnen. Diese erstmalige Preissetzung wird durch den Parameter *pre-negotiation delta price change* bestimmt. Er beeinflußt zusätzlich die neuen Preise basierend auf denen der letzten erfolgslosen Transaktion.
- Der Parameter *satisfaction* entscheidet, ob ein Software-Agent Verhandlungen fortführen wird. Je mehr Verhandlungsschritte durchgeführt wurden, desto größer ist die Wahrscheinlichkeit, daß der Software-Agent die Verhandlung abbrechen und kein weiteres Angebot ausgetauscht wird.
- Die Wahrscheinlichkeit mit der ein Software-Agent den Marktplatz wechselt wird durch den Parameter *impatience* festgelegt.
- Informationen über Marktpreise aus früheren Verhandlungen werden im Parameter *memory* berücksichtigt. Das Gewicht ihrer Berücksichtigung erfolgt im Parameter
- *Price memory weight.* Es handelt sich hierbei um einen gleitend gewichteten Durchschnitt.
- Der Parameter *reputation* steuert das Kooperationsverhalten des Software-Agenten. Er wird in Kapitel 4 ausführlich behandelt.

Jede Aktion der Software-Agenten wird durch eine stochastische Probe durch „Würfeln“ gegen diese acht Parameter bestimmt. Das konkrete Verhalten eines Software-Agenten ist daher nicht vorhersehbar, sondern ergibt sich aus der Kombination seiner Parameter und aus einem Zufallselement.

Die Software-Agenten berücksichtigen als Teil ihrer eigenen Strategie die Existenz weiterer Software-Agenten. Jedoch verfügen sie über keinerlei Möglichkeit, die Entscheidungen der anderen Software-Agenten nachzuvollziehen oder deren strategische Absichten zu ermitteln. Sie verfügen somit lediglich über ein „Gedächtnis“, in dem sie Erfahrungen mit anderen Software-Agenten abspeichern können und entsprechen damit Software-Agenten der Kategorie *1-level* nach Vidal und Durfee.[46]

Eine wichtige Variable ist das Eigenkapital, das ein Rechen- und Wertaufbewahrungsmittel bei Kauf und Verkauf von Gütern ist und somit als Indikator des relativen Erfolges des einzelnen Software-Agenten dient. In regelmäßigen Zeitabständen werden fixe Geldbeträge aus dem Kapitelbestand der Software-Agenten abgezogen. Sie stellen die „Lebenshaltungskosten“ ihrer Nutzer dar. Bei Inaktivität des Software-Agenten kann so sein Kapitalbestand auf Null sinken. In diesem Fall terminiert der Software-Agenten. Ein Software-Agent, der schneller oder besser handelt als ein anderer, wird einen relativ höheren Kapitalbestand erwirtschaften, was wiederum Rückschlüsse auf die Wettbewerbsfähigkeit seiner Parameterkombination zuläßt.

2.4.4 Mobilität

Die Fähigkeit der Software-Agenten, sich zwischen verschiedenen elektronischen Marktplätzen zu bewegen, ist notwendiger Bestandteil des -Szenarios. Elektronische Marktplätze können über das ganze Internet verteilt sein. Daher ist es sinnvoll, daß die Software-Agenten zwischen den Marktplätzen wandern können, um bspw. Arbitragen abzuschöpfen. Die Nutzung von Java bei der Implementation des Prototyps erlaubt es auch, daß Marktplätze und Software-Agenten nicht nur auf herkömmlichen Personalcomputern, sondern auch bspw. auf Personal Digital Assistants (PDA), Mobiltelefonen oder sonstigen Systemen plattform-übergreifend laufen können.

[46] Vgl. Kapitel 2.1.1.

Falls ein Software-Agent auf einem elektronischen Marktplatz keine für ihn befriedigende Situation vorfindet, wird er ihn verlassen und zu einem benachbarten, ebenfalls offenen Marktplatz migrieren. Dort kann der Software-Agent wiederum auf fremde Software-Agenten treffen, deren kooperatives, ehrliches Verhalten nicht ex ante angenommen werden kann.

2.4.5 Aktionswahl der Software-Agenten

In Abbildung 6 ist dargestellt, wie sich die AVALANCHE Software-Agenten grundsätzlich verhalten. Der Software-Agent prüft als erstes, ob er über Produkte verfügt, die er verkaufen kann. Falls dies der Fall ist, wird er versuchen diese Produkte zu verkaufen, indem er einen entsprechenden Eintrag in dem Verzeichnis des aktuellen Marktplatzes setzt. Danach werden aus seinem Kapitalbestand Lebenshaltungskosten abgezogen. Falls der Software-Agent augenblicklich über keine Endprodukte verfügt, wird er versuchen, diese schnellstmöglich aus seinen Vorprodukten zu produzieren. Im Anschluß wird er wiederum seine Lebenshaltungskosten abziehen. Falls der Software-Agent weder über End- noch über Vorprodukte verfügt, wird er versuchen, diese einzukaufen. Dazu erstellt er eine Liste aller Anbieter von Vorprodukten und ordnet sie nach den Angebotspreisen. Anhand dieser Liste werden Verhandlungen mit den einzelnen Anbietern gestartet. Die Verhandlungen sind im folgenden Abschnitt (2.5) genauer beschrieben. Bei einer erfolgreichen Verhandlung wird der Software-Agent die Vorprodukte einkaufen und seine Lebenshaltungskosten von seinem Kapital abziehen. Sollte es mit keinem Anbieter zu einer Einigung gekommen sein, wird der Software-Agent den Marktplatz verlassen und zu einem fremden Marktplatz migrieren. Auch in diesem Fall werden die Lebenshaltungskosten abgezogen.

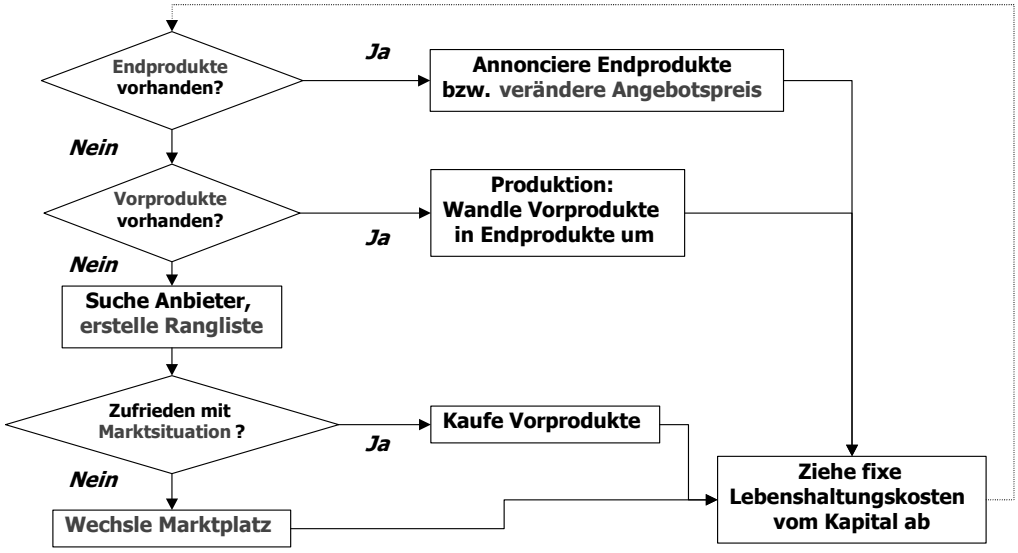


Abbildung 6: Hauptschleife des AVALANCHE Software-Agenten[47]

2.5 Ablauf einer Transaktion

Ökonomische Transaktionen auf elektronischen, wie auch auf herkömmlichen Märkten können in drei Phasen unterteilt werden: Anbahnungs-, Verhandlungs- und Abwicklungsphase.[48] In der Anbahnungsphase wird eine Transaktion initiiert und der Software-Agent muß potentielle Transaktionspartner identifizieren. Dem folgt die Vereinbarungsphase, in der Preise, Anzahl, Art und Beschaffenheit der Güter, Lieferbedingungen etc. ausgehandelt werden. Die Transaktion wird mit der Abwicklungsphase abgeschlossen, bei der Ware gegen Geld getauscht wird.

[47] Eymann, T. / Padovan, B. (2000). S. 19.
[48] Müller, G. / Kohl, U. / Schoder, D. (1997). S. 299ff.

Im folgendem wird der Ablauf einer Transaktion zwischen zwei Software-Agenten am Beispiel in einer Verkaufsverhandlung dargestellt. Die beiden Software-Agenten *A* (Holzfäller) und *B* (Zimmermann) befinden sich auf einem ansonsten als leer angenommenen Markt. A hat ein Brett für € 24 zu verkaufen, *B* möchte ein Brett für € 20 kaufen.

1. Anbahnungsphase

Der Software-Agent *A* schickt als erste Aktion eine Nachricht an den Verzeichnisdienst, in der sein Angebot (Brett) und seine Identität (ID) (Alice), jedoch kein Preis eingetragen sind. Der Verzeichnisdienst listet ihre ID dann in seiner Rubrik Brett/Verkäufe auf. Software-Agent *B* schickt eine Anfrage als Nachricht an den Verzeichnisdienst, welche Einträge in der Rubrik Brett/Verkäufe vorhanden sind. Der Verzeichnisdienst übergibt ihm einen Vektor mit allen IDs in dieser Rubrik, so daß *B* durch Abfrage von Preisen und Mengen aller Software-Agenten eine Übersicht des aktuellen Marktgeschehens als „snapshot“ erhält. *B* reiht die Angebote dann nach den Preisen und befriedigt anhand dieser Liste seine Nachfrage. Da die Software-Agenten gleichzeitig agieren, ist es prinzipiell möglich, daß ein Nachfrageüberhang im Augenblick der Kaufentscheidungen entsteht, so daß der Software-Agent auf schlechtere Angebote ausweichen oder die Prozedur wiederholen muß.

2. Verhandlungsphase

Im Beispielfall hat der Anbietervektor nur einen Eintrag, die ID von Software-Agenten *A*. *B* schickt eine Nachricht „call for proposal“ (cfp) an Alice, in der er *A* um ein Gebot bittet. *A* antwortet mit einer Nachricht („propose-offer“), in der er € 24 als Stückpreis für ein Brett fordert. *B* antwortet zuerst mit einer Nachricht, in der er € 20 vorschlägt. Über eine Abfolge von „propose-offer“-Nachrichten, bei denen Preiskonzessionen eingegangen werden (können), nähern sich beide Software-Agenten einem Kompromißpreis an. Welcher Software-Agent zu welchem Zeitpunkt eine unilaterale Konzession eingeht, wird durch die Strategieparameter bestimmt. In diesem Fall erfolgt

anhand des Parameters *acquisitiveness* eine Prüfung, ob der Angebotspreis bestehen bleibt oder verändert wird. Schließlich wird z.B. der Software-Agent *B* dem Software-Agenten *A* eine Nachricht zurücksenden, in der er sein Gebot akzeptiert („accept-offer“).

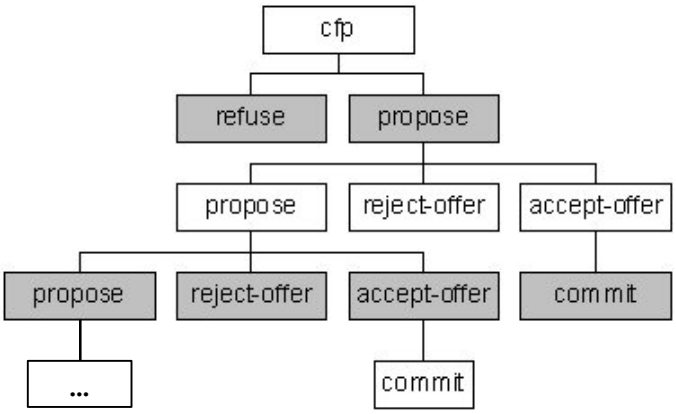


Abbildung 7:
Verhandlungen zwischen Software-Agenten in AVALANCHE

Die Verhandlung kann als eine Abfolge von Angeboten und Gegenangeboten gesehen werden. Der Ablauf der Verhandlung ist angelehnt an *FIPA's iterated contract-net-protocol*.[49] In Abbildung 7 werden die Abläufe während der Verhandlungsphase zusammenfassend dargestellt, wobei die unterschiedlichen Graustufen sich auf verschiedene Software-Agenten beziehen.

3. Durchführungsphase

Software-Agent *A* bucht die Produkteinheiten von seinem Lagerbestand ab und den Kaufpreis auf sein Geldkonto und schickt eine Bestätigungsmitteilung („commit“). Nach deren Erhalt führt auch Software-Agent *B* die entsprechenden Waren- und Geldbuchungen durch. Zu einem späteren Zeitpunkt, d.h. nachdem *B* (bzw. sein Besitzer) den Rohstoff verarbeitet hat, wird er in einem nächsten Schritt dann seine ID und sein Produktangebot (Platte) an den Verzeichnisdienst schicken, um nun einem Schreiner Software-Agenten sein Gut zu verkaufen.

[49] Vgl. FIPA (1998).

2.6 Ergebnisse in einem geschlossenen System

Um den Prototyp zu prüfen, wurden mehrere Testreihen durchgeführt. Jede Testreihe bestand aus mehreren, mindestens fünf Testläufen, die stets unter gleichen Startbedingungen zu vergleichbaren Ergebnissen führten. Im Folgenden ist je Testreihe das Ergebnis eines typischen Testlaufs dargestellt.

I. Testreihe

Die erste Testreihe wurde mit jeweils 50 Software-Agenten jeden Typs durchgeführt. Exemplarisch ist in Abbildung 8 die Entwicklung der im System entstandenen Güterpreise im Zeitablauf dargestellt. Der zeitliche Ablauf eines Experiments ist auf der Abszisse in Millisekunden abgetragen, auf der Ordinate ist eine Wertskala angezeigt, die die Güterpreise in Geldeinheiten mißt. Jeder Punkt in der Grafik markiert eine abgeschlossene Transaktion. Der Typ des verkauften Gutes wird durch unterschiedliche Punktformen dargestellt. Die Legende zeigt unterschiedliche Symbole, die von oben nach unten die Wertschöpfungskette anhand der Güter zeigt (wood, board, panel, table).

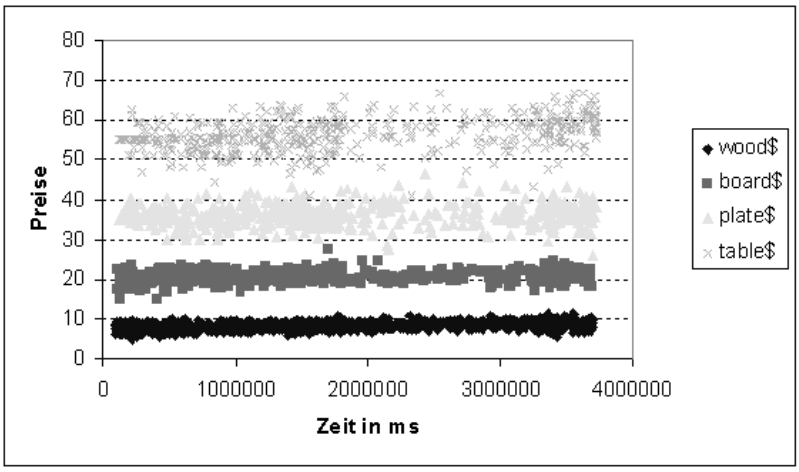


Abbildung 8: Preisentwicklung der Güter

In einer Stunde fanden 7163 Transaktionen statt. Deutlich sichtbar ist, daß sich konstante Güterpreise auf dem Marktplatz ergeben haben und ein stabiles Marktsystem entstanden ist.

II. Testreihe

In einer zweiten Testreihe (Abbildung 9) startete der Prototyp mit zufällig festgelegten Startpreisen der Güter.[50]

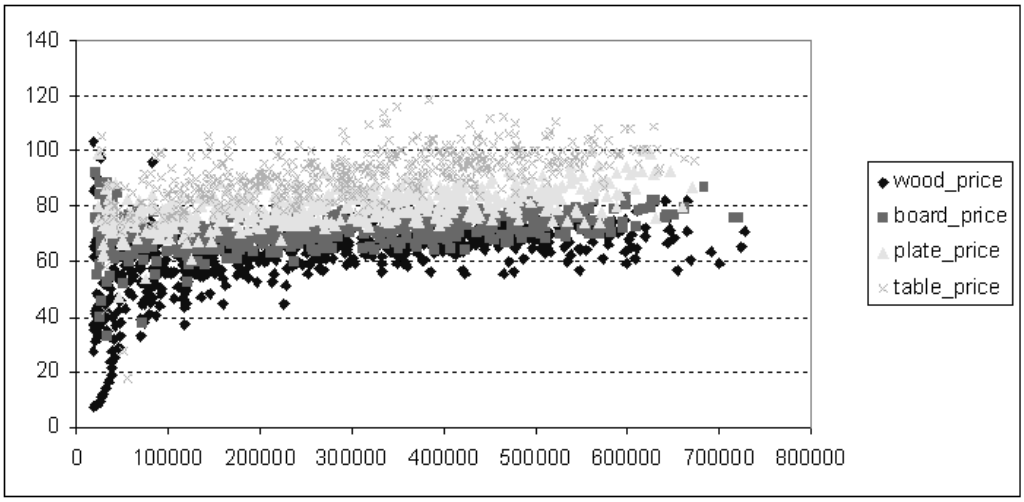


Abbildung 9:
Preisentwicklung der gehandelten Güter bei zufälliger Ausgangssituation

Nach einer „Einschwingphase“ haben sich auf dem Marktplatz konstante Güterpreise ergeben und es ist ein stabiler Markt entstanden. Es fanden insgesamt 1970 Transaktionen statt.

III. Testreihe

In der dritten Testreihe wurde die wirtschaftliche Entwicklung der einzelnen Software-Agenten anhand ihrer Bargeldbestände ermittelt, wobei alle Parameter bei den Einstellungen der ersten Testreihe belassen wurden. Die Anzahl der Software-Agenten je Typ wurde jedoch auf drei beschränkt, um den Erfolg des einzelnen Software-Agenten besser im Diagramm darstellen zu können. Stellvertretend für alle Software-Agenten sind in Abbildung 10 die Bargeldbestände der Zimmermann-Agenten (Carpenter) abgebildet.

---

[50] Vgl. Eymann, T. (2000). S. 189ff.

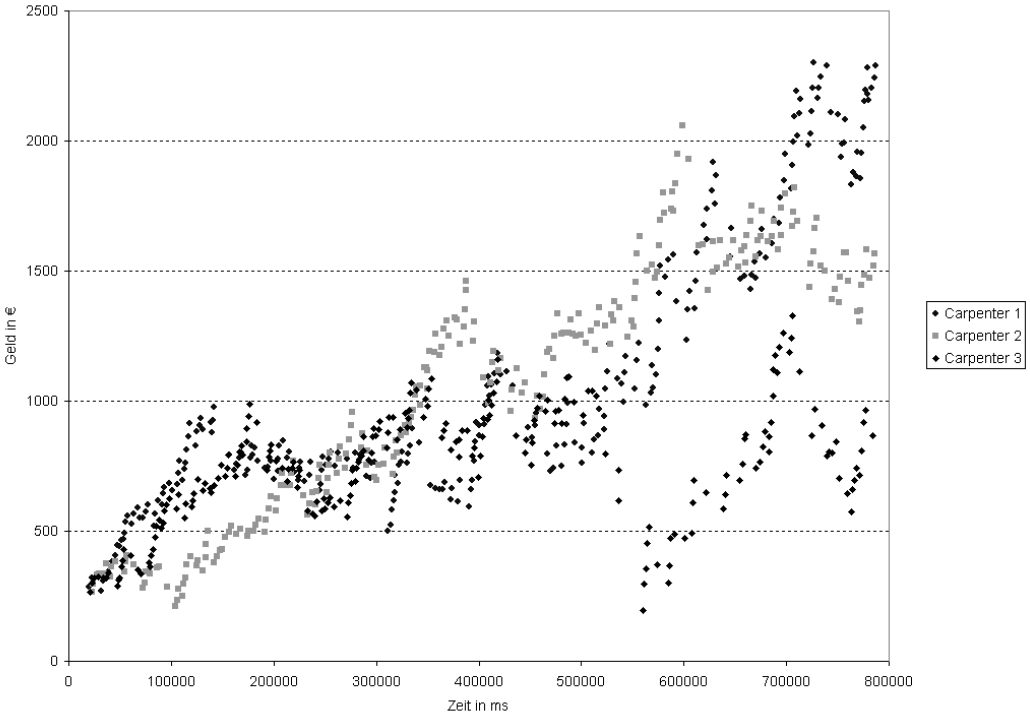


Abbildung 10:
Kooperatives Verhalten aller beteiligten Software-Agenten

Es wird deutlich, daß alle Software-Agenten ihren Kapitalbestand bei stabilem Preisniveau erhöhen konnten, also ökonomisch erfolgreich gehandelt haben. Insgesamt wurden durch die drei Software-Agenten 729 Transaktionen durchgeführt, davon 256 von Zimmermann_1, 243 von Zimmermann_2 und 220 von Zimmermann_3.

Die Ergebnisse der ersten drei Testreihen kamen unter der Voraussetzung zustande, daß sich alle Software-Agenten stets kooperativ verhalten, d.h. Geld und Waren zu den vereinbarten Werten bzw. Mengen austauschen. Betrügerisches Verhalten wird als unkooperativer Akt aufgefaßt.[51] Ob sich ein Software-Agent kooperativ verhält, wird in AVALANCHE durch den unveränderlichen Parameter *reputation*, d.h. seine individuelle Reputationskonstante gesteuert. Dieser Wert liegt zwischen 0 und 1, ist für andere Software-Agenten nicht einsehbar. Er de-

[51] Eine ausführlichere Beschreibung erfolgt in Kapitel 0.

terminiert die statistische Wahrscheinlichkeit, mit der sich dieser Software-Agent bei einer Transaktion kooperativ verhalten wird. In diesen drei Testreihen wurden alle Software-Agenten mit einer Reputationskonstante mit dem Wert 1 gestartet. Dies entspricht der Situation eines geschlossenen Multi-Agenten-Systems, in dem das kooperative Verhalten von Software-Agenten erzwungen werden kann, damit z.B. ein vorgegebenes Gesamtziel erreicht wird.

2.7 Ergebnisse in einem offenen System

Elektronische Marktplätze sind jedoch nicht geschlossen, sondern offen, so daß Software-Agenten vieler einander unbekannter menschlicher Marktteilnehmer miteinander interagieren. In diesem Fall wirken die agentenbasierten Marktteilnehmer nicht mehr auf ein gemeinsames Gesamtziel hin, sondern können sogar versuchen, durch betrügerisches, eigennütziges Verhalten Vorteile gegenüber ihren Mitbewerbern zu erhalten. Sie schädigen damit die rechtschaffenen, kooperativen Agenten, aber auch einen Marktplatzbetreiber, der dieses Verhalten zuläßt.

IV. Testreihe

In einer vierten Testreihe wurde ein Szenario eines betrügerischen Verhaltens durch einen in ein offenes System eintretenden Agenten nachgebildet. Die Setzung der Parameter entsprach der aus der zweiten Testreihe. Einer der drei Zimmermann-Agenten wurde als Betrüger mit einer Reputationskonstante = 0 modelliert, er lieferte daher nach Abschluß einer Transaktion entweder die Ware nicht an die Schreiner aus oder bezahlte die Holzfäller nicht. Die beiden anderen Zimmermann-Agenten hingegen betrogen nie, verhielten sich also stets kooperativ (Reputationskonstant = 1).

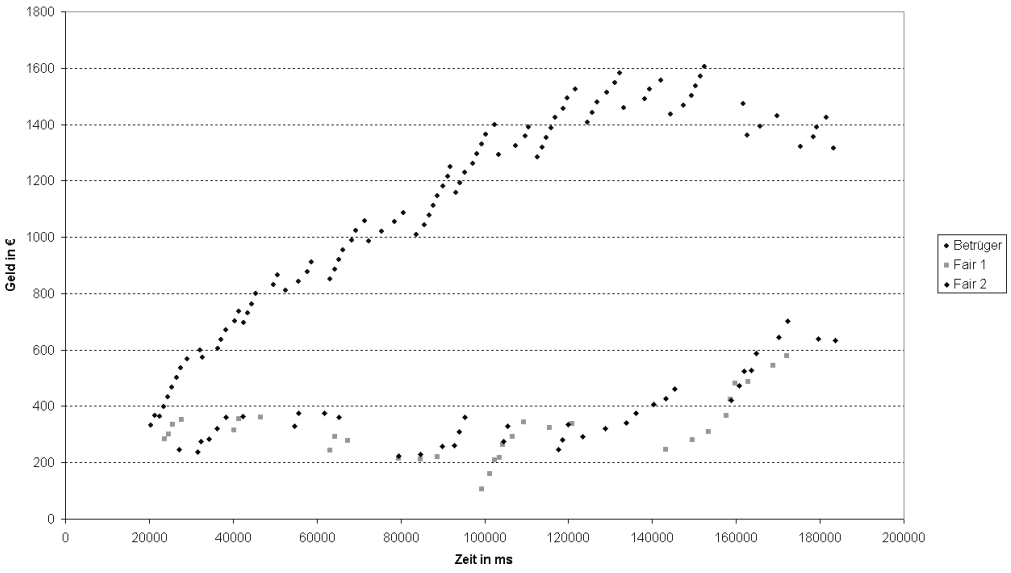


Abbildung 11: Unkooperatives Verhalten eines agentenbasierten Transaktionspartners

In Abbildung 11 sind die sich aus dieser Testreihe ergebenden Muster exemplarisch dargestellt. Aus mikroökonomischer Sicht konnte der betrügende Software-Agent seinen Bargeldbestand erheblich erweitern, während die Bargeldbestände der sich ehrlich verhaltenden Software-Agenten stagnierten. Der betrügerische Software-Agent konnte außerdem den Großteil der durchgeführten Transaktionen auf sich ziehen (insgesamt 159 Transaktionen, davon 90 durch den unkooperativen und 31 bzw. 38 durch die kooperativen Software-Agenten).

Makroökonomisch betrachtet führte die Teilnahme des betrügerischen Software-Agenten zum Zusammenbruch des Systems. Dieser lag im wirtschaftlichen Niedergang seiner vertikalen Transaktionspartner (Holzfäller- und Schreiner-Agenten) begründet, die keine Gewinne aus den Transaktionen realisieren und den Betrüger weder vor einer Transaktion erkennen noch nachträglich sanktionieren konnten.

2.8 Anforderungen an offene Systeme

Der Vergleich der dritten und vierten Testreihe zeigt deutlich, daß bei einer Öffnung des Systems das Risiko des betrügerischen Verhaltens einzelner agentenbasierter Marktteilnehmer zu einem Versagen des Systems führen kann. In offenen Netzen existieren kaum Instanzen, die Sanktionen gegen betrügerische Marktteilnehmer durchsetzen können. Derartige Instanzen sind aber offensichtlich notwendig, um einen Marktplatz in offenen Systemen überhaupt zu ermöglichen.[52] Folgende Anforderung für offene, agentenbasierte Marktplätze kann aus den Testläufen abgeleitet werden:

Es muß ein Mechanismus existieren, der betrügerische Software-Agenten sanktioniert oder sogar vom Handel ausschließt und so einen Anreiz zu kooperativem Verhalten schafft. Dazu müssen die betrügerischen Handlungen dem entsprechenden Software-Agenten zugeordnet und dieser identifiziert werden können.

Um dies zu erreichen, sind in einem ersten Schritt drei Anforderungen zu erfüllen:

- **Authentifizierung**

  Die Authentifizierung der Software-Agenten muß gewährleistet sein. Dazu müssen ihre Besitzer identifizierbar sein, um so bspw. die Nutzung von Pseudonymen zu verhindern, da Pseudonyme sich in kurzer Zeit ändern lassen und dieselbe Person unter mehreren Pseudonymen gleichzeitig auftreten könnte. Bei Auktionen führt dies bspw. zu einem *Phantom-Bieter-Problem*.

- **Zurechenbarkeit**

  Transaktionen müssen den beteiligten Software-Agenten zugerechnet werden können. Software-Agenten dürfen nicht unter Angabe einer fremden Identität eine Nachricht versenden oder eine Transaktion durchführen können.

[52] Vgl. Rannenberg, R. / Pfitzmann, A. / Müller, G. (1999).

- **Privatheit**

  Die Privatheit der Kommunikation muß gewährleistet sein. Durch das Abhören fremder Nachrichteninhalte können Software-Agenten unberechtigterweise Informationen erhalten, die ihnen Vorteile gegenüber anderen Software-Agenten verschaffen können. In offenen und unsicheren Netzen, wie dem Internet, existieren eine Vielzahl von Angriffsmöglichkeiten auf Kommunikationsinhalte, unabhängig von der Tatsache, daß bei einer vertraulichen Kommunikation die Partner vorab authentifiziert sein müssen.

2.9 Sicherheit im MAS AVALANCHE

Authentifizierung, Zurechenbarkeit und Privatheit können mit Hilfe von informatischen Verfahren zur Sicherheit gelöst werden. Dazu wird in AVALANCHE ein asymmetrisches Public-Key System genutzt. Bei einem Public-Key System besitzt jeder Nutzer zwei Schlüssel, einen privaten und einen öffentlichen. Beide Schlüssel werden gemeinsam generiert, da das System nur bei einer gemeinsamen Nutzung der Schlüssel funktioniert. Der öffentliche Schlüssel wird vom Nutzer veröffentlicht, während der private Schlüssel nur dem Nutzer bekannt ist.

- **Digitale Signatur**

  Durch die Nutzung seines privaten Schlüssels kann ein Software-Agent seine Nachrichten mit seiner Signatur ergänzen. Diese Signatur kann nur von ihm erzeugt werden. Jeder andere Software-Agent kann jedoch mit dem öffentlichen Schlüssel des Nachrichtenabsenders die Echtheit der Signatur überprüfen. Durch die Nutzung eines Prüfsummenverfahrens kann darüber hinaus überprüft werden, ob die Nachricht manipuliert wurde.

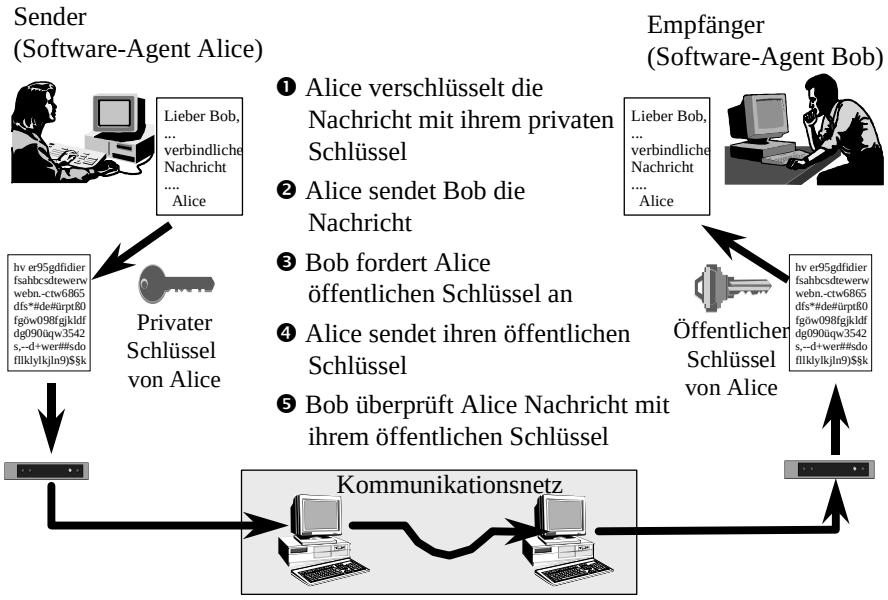


Abbildung 12: Funktionsweise der digitalen Signatur

- **Verschlüsselung**

  Nachrichten können auch verschlüsselt werden, damit Unbefugte keine Kenntnis über ihren Inhalt erhalten können. Dazu verschlüsselt der Absender einer Nachricht ihren Inhalt mit dem öffentlichen Schlüssel des Empfängers. Nur der Empfänger kann nun mit seinem privaten Schlüssel die Nachricht entschlüsseln und ihren Inhalt einsehen. Es ist auch möglich, beide Verfahren zu kombinieren, so daß eine Nachricht digital vom Absender signiert und ihr Inhalt verschlüsselt wurde.[53]

[53] Die Funktionsweise eines Public-Key Systems ist bspw. in: Müller, G. / Pfitzmann, A. (Hrsg.) (1997) beschrieben.

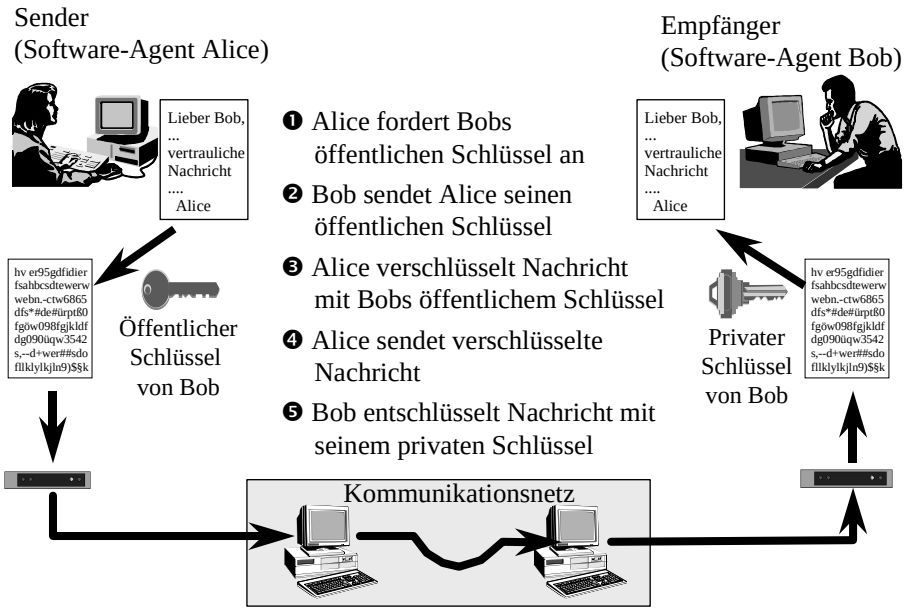


Abbildung 13: Funktionsweise asymmetrischer Verschlüsselung

In AVALANCHE wird der öffentliche Schlüssel[54] jeweils gemeinsam mit der ID des Software-Agenten und der Liste seiner angebotenen, bzw. nachgefragten Güter im Verzeichnis des aktuellen Marktplatzes veröffentlicht. Der private Schlüssel wird im mobilen Code des Software-Agenten gehalten.[55,56]

In AVALANCHE signieren die Software-Agenten digital sämtliche Nachrichten, die sie an andere Software-Agenten oder an die Marktplätze senden. So kann ein Wechsel von Identitäten oder die Nutzung falscher Identitäten verhindert werden.[57] Auf diese Weise wird auch die Zurechenbarkeit von Transaktionen sichergestellt. Nachrichten zwischen den Software-Agenten, bspw. bei Preisver-

[54] Die Verwendung von Zertifikaten beim Einsatz eines Public-Key Systems wird in dieser Arbeit nicht behandelt.

[55] Es wird unterstellt, daß es für einen Software-Agenten möglich ist, einen privaten Schlüssel in seinem Code tamper-resistent zu halten.

[56] Zur Implementation des Public-Key Systems werden die in JDK 1.3 mitgelieferten Klassenbibliotheken genutzt. Die Erzeugung der öffentlichen und privaten Schlüssel erfolgt über einen externen Dienst.

[57] Probleme die bei der Schlüsselzertifizierung und –verteilung entstehen können, werden in diesem Zusammenhang nicht erörtert.

handlungen, werden zusätzlich verschlüsselt, so daß der Inhalt der Kommunikation zwischen beiden vertraulich bleibt.

Unkooperatives Verhalten kann jedoch durch Sicherheit nicht verhindert werden, selbst wenn alle Software-Agenten identifiziert werden können und ihre Kommunikation privat und vertraulich erfolgt. Im menschenbasierten Electronic Commerce stellt die fehlende Kenntnis über die Vertrauenswürdigkeit und die Kompetenz eines unbekannten Transaktionspartners die größte Hürde für neue Internet-basierte Geschäftskooperationen dar[58] und dies trotz der Verfügbarkeit anerkannter Verschlüsselungs- und Signaturverfahren.[59]

[58] Vgl. Eggs, H. / Englert, J. (2000). S. 28f.

[59] Für eine weitere Behandlung dieses Themengebietes eigenen sich Müller, G. / Rannenberg, K. (1999).

2.10 Zurechenbarkeit und Vertraulichkeit

Bei der Kommunikation über offene Netze haben die Nutzer, unabhängig davon ob es sich bei ihnen um einen Menschen oder einen Software-Agenten handelt, ein Interesse daran, daß ihre Kommunikation vertraulich bleibt. Diese Vertraulichkeit kann dadurch erreicht werden, daß die Kommunikationsinhalte durch fremde Instanzen nicht eingesehen werden können und daß die Kommunikationspartner gegenseitig und gegenüber den Systembetreibern oder einer sonstigen zentralen Instanz anonym bleiben.

Eine Sanktionierung betrügerischer Software-Agenten kann jedoch nur erfolgen, wenn die Zurechenbarkeit einer betrügerischen Handlung möglich ist, d.h. wenn die betreffenden Software-Agenten authentifiziert und ihre Handlungen ihnen beweisbar zugeordnet werden können.

Diese beiden Anforderungen sind Schutzziele einer mehrseitig sicheren Kommunikation.[60] Die gleichzeitige Verwirklichung beider Anforderungen ist nicht möglich. Deswegen muß zwischen den Beteiligten ausgehandelt werden, welche Schutzziele verfolgt werden sollen. Diese Aushandlung wird in dieser Arbeit nicht weiter untersucht.

[60] Vgl. Müller, G. / Kohl, U. / Schoder, D. (1997). S. 309f.

3 Vertrauen und Reputation in Multi-Agenten Systemen

Vertrauen ist ein Begriff, den jeder Mensch kennt und häufig nutzt. Vertrauen, bzw. Mißtrauen bestimmt zwischenmenschliche Handlungen. Eng verbunden mit dem Begriff Vertrauen ist der Begriff der *Reputation*. Im folgendem Kapitel soll dargestellt werden, in welchem Zusammenhang die beiden Begriffe zueinander stehen und welche Funktionen Vertrauen und Reputation generell erfüllen.[61]

In einem ersten Schritt werden Funktion und Aufbau von Vertrauen in zwischenmenschlichen Beziehungen aus soziologischer Sicht untersucht, wobei aus der Fülle der Erklärungsansätze über Funktion und Aufbau von Vertrauen drei näher erörtert werden. Ihre wesentlichen gemeinsamen Punkte werden zu einer Definition zusammengefaßt, die für die weitere Nutzung des Begriffs Vertrauen als Grundlage dienen soll.

Dem folgt eine nähere Betrachtung des Begriffs Reputation, der – ebenso wie Vertrauen – Bestandteil des tagtäglichen Sprachgebrauchs ist. Um sich der Bedeutung des Ausdrucks Reputation klar zu werden, wird auch hier die soziologische Sicht erläutert.

Anschließend wird geklärt, ob die Funktionen von Vertrauen und Reputation auch in einem System von Software-Agenten genutzt werden können.

Das Kapitel schließt mit einer Beschreibung von Forschungsprojekten, in denen eine Vertrauensermittlung quantitativ erfolgt. Fünf Ansätze werden ausführlicher behandelt. Anhand von vier Kriterien, die in diesem Abschnitt entwickelt werden, werden Aspekte abgeleitet, die für den Vertrauens- und Reputationsmechanismus in AVALANCHE relevant sind.

---

[61] Eine weitergehende Erörterung von *Vertrauen* und *Reputation* bei Software-Agenten findet sich in: Westermayer, T. (2000).

3.1 Deskriptive Einordnung von Vertrauen

Allgemeingültige Definitionen von Vertrauen existieren nicht. Dies liegt u.a. daran, daß dem Begriff Vertrauen in der Soziologie keine zentrale Bedeutung zukommt. Barbara Misztal führt hierzu aus:

> „The concept of trust entered sociological theory by way of philosophical and political writings, never having been a central focus of sociological theory. ... [It] was seldom explicitly questioned or studied".[62]

Dementsprechend entstanden verschiedene, sich teilweise widersprechende Theorien über die Entstehung und die Funktionen von Vertrauen. Im Folgenden sollen die wichtigsten in der Literatur zu findenden Beiträge zum Thema Vertrauen kurz skizziert werden.

3.1.1 Vertrauen als Mittel zur Reduktion sozialer Komplexität

Niklas Luhmann sieht Vertrauen als Mechanismus zur Reduktion sozialer Komplexität. Er unterscheidet dabei zwischen Vertrauen und Vertrautheit.

> „Vertrautheit ist Voraussetzung für Vertrauen, [...] das heißt für jede Art des Sichengagierens in eine bestimmte Einstellung zur Zukunft."[63]

Das vorhandene Hintergrundwissen über die Umwelt bildet die Vertrautheit und ermöglicht so ein „relativ sicheres Erwarten".[64] Erst diese Vertrautheit ermöglicht Vertrauen. Vertrauen unterscheidet sich von der bloßen Hoffnung dadurch, daß eine kritische Entscheidungssituation vorhanden sein muß, bei der, um eine Entscheidung treffen zu können, vertraut werden muß. Nur wenn das Risiko des nicht-wohlwollenden Verhaltens des Interaktionspartners existiert, ist Vertrauen möglich:[65]

---

[62] Misztal, B. (1996). S. 1.
[63] Luhmann, N. (1989). S. 19.
[64] Luhmann, N. (1989). S. 19.
[65] Vgl. Luhmann, N. (1989).

> „Vertrauen bezieht sich also stets auf eine kritische Alternative, in der der Schaden bei Vertrauensbruch größer sein kann als der Vorteil, der aus dem Vertrauenserweis gezogen wird.“[66]

In Situationen, in denen keine Informationen über die Transaktionspartner vorhanden sind, verhindert Vertrauen die Abschätzung aller möglichen Folgen einer geplanten Interaktion. Nur bei einer zumindest teilweisen Auslassung der Abschätzung aller eventuellen Folgen einer Interaktion ist eine Transaktion überhaupt durchführbar. Vertrauen reduziert somit die Komplexität einer Entscheidungsfindung. Mißtrauen ist nicht nur das Gegenteil von Vertrauen, sondern erfüllt die gleichen Funktionen wie Vertrauen, da Mißtrauen ebenfalls äquivalent soziale Komplexität reduziert. Jedoch läßt sich Vertrauen wesentlich schneller in Mißtrauen verwandeln, als umgekehrt.

Vertrauen bildet sich in einem langsamen Prozeß. Dieser Prozeß besteht aus einer wiederholten, wechselseitigen und riskanten Vorausleistung eines der beiden Transaktionspartner. Der Vertrauende tritt dabei in eine riskante Vorleistung. Dieses Element wiederholt sich wechselseitig mit jeweils wechselndem Einsatz. Dabei kann eine Störung schnell zu einem Verlust des Vertrauens führen.

Luhmann sieht sowohl Vertrauen als auch Mißtrauen als Notwendigkeit, um Handlungspotentiale in einem sozialen System zu steigern.

> „Ohne Vertrauen sind nur sehr einfache [...] Kooperationen möglich.“[67]

[66] Luhmann, N. (1989). S. 24.
[67] Luhmann, N. (1989). S. 98.

3.1.2 Vertrauen als „Wette auf die Zukunft“

Piotr Sztompka untersucht die Rolle und die Entstehung von Vertrauen in Gesellschaften.[68] Er verfolgt die These, daß Vertrauen ein Merkmal moderner Gesellschaften ist. In älteren Gesellschaften, in denen alles klar geregelt war, bspw. durch Normen oder religiöse Vorschriften, wurde kein Vertrauen benötigt. Erst als Risiken und Unsicherheiten auftraten, bspw. durch Marktbeziehungen, wurde Vertrauen zu einem notwendigen Bestandteil einer Gesellschaft. Vertrauen reduziert Unsicherheiten und ermöglicht den Umgang mit Risiken. Für Sztompka ist Vertrauen „a bet about the future contingent actions of others.“[69] Damit ist für ihn Vertrauen immer eine Eigenschaft in Bezug auf andere Menschen und deren Handlungen. Einem Vulkan, so Sztompka, kann man nicht Vertrauen. Gesellschaftlichen Institutionen zu vertrauen, bedeutet stets, den Menschen dahinter zu vertrauen.[70] Vertrauen wird für ihn folgendermaßen motiviert:[71]

- *Anticipatory Trust* ist das Vertrauen, das dadurch motiviert ist, daß die Handlungen einer fremden Person dem eigenen Interesse zugänglich sein wird.
- *Responsive Trust* beschreibt das Vertrauen, das durch die vermutete Reaktion der Gegenseite motiviert ist, etwa wenn es darum geht einer Person ein wertvolles Objekt zu überlassen.
- Als *Evocative Trust* beschreibt Sztompka das Vertrauen, das dadurch motiviert ist, das erwartet wird, daß einem selbst Vertrauen entgegengebracht wird.

Diese unterschiedlichen Formen von Vertrauen sind unterschiedlich stark ausgeprägt. Von besonderer Bedeutung ist für Sztompka der Zusammenhang zwi-

---

[68] Seine Motivation liegt in der Untersuchung des Übergangs von sozialistischen zu pluralistischen Gesellschaften in Osteuropa.
[69] Sztompka, P. (1999). S. 25.
[70] Vgl. Sztompka, P. (1999). S. 21, S. 41ff.
[71] Vgl. Sztompka, P. (1999). S. 27ff.

schen Risiko und Vertrauen. Je höher das Risiko ist, desto höher ist auch das Vertrauen. Das Risiko bezieht sich stets auf den Schaden, der durch einen Vertrauensbruch entstehen kann. Der mögliche Schaden kann durch das Vorhandensein einer Versicherung minimiert werden. Somit ermöglicht eine Versicherung eine Reduktion von Vertrauen. Des Weiteren ist die Stärke des Vertrauens abhängig von der Zeitdauer der Vertrauensbeziehung und von der Möglichkeit, das Vertrauen zu entziehen oder nicht.

Jede Handlung, die mit Vertrauen verbunden ist, beinhaltet das Risiko, die oben angesprochene „Wette auf die Zukunft“ zu verlieren. Es müssen demnach, so argumentiert auch Luhmann, Indizien gesucht werden, warum eine solche „Wette“ überhaupt erst eingegangen werden soll. Diese Indizien stehen für die Vertrauenswürdigkeit eines Interaktionspartners. Sie entstehen entweder aus der Beziehung der beiden Partner, aus dem situativen Kontext oder sind psychologisch motiviert.[72] Sztompka geht über den *Rational-Choice-Ansatz*[73] deutlich hinaus, da dieser lediglich einer Kosten-Nutzen Analyse folgt und keine psychologischen und situativen Ansätze berücksichtigt. Für Sztompka sind derartige Ansätze die Bewertung der aktuellen Handlung, das Auftreten (bspw. Kleidung), die zukünftige Verfügbarkeit des Interaktionspartners sowie ein psychologisch motivierter *Trusting Impulse*.[74] Sztompkas Betrachtung läßt sich dahingehend zusammenfassen, daß Menschen anderen Menschen oder Institutionen dann vertrauen, wenn sie eine Vielzahl situativer, psychologischer und historischer Hinweise über das Verhalten der zu vertrauenden Partei haben.

---

[72] Vgl. Good, D. (1988).

[73] Der *Rational-Choice-Ansatz* geht davon aus, daß Menschen zum Erreichen von Vorhaben zielorientiert vorgehen. Sie verfügen über klare Präferenzen und Nutzeneinschätzungen. Bei Entscheidungen verhalten sich die Menschen absolut rational, entsprechend ihrer Präferenzen, Nutzeneinschätzungen und Opportunitätskosten. Dabei haben sie stets zum Ziel, ihren Nutzen zu maximieren. Alle soziologische Phänomene, wie soziale Strukturen, kollektive Entscheidungen oder kollektives Verhalten, sind letztendlich Ergebnisse dieses absolut rationalen Verhalten aller Menschen. Vgl. Turner, J. (1991). S. 354.

[74] Vgl. Sztompka, P. (1999). S. 69ff.

3.1.3 Vertrauen und kooperatives Handeln

Diego Gambetta[75] greift in *„Trust. Making and Breaking Cooperative Relations*“ den Zusammenhang zwischen Vertrauen und kooperativen Handeln auf. Er folgt dabei der Leitfrage, ob *Kooperation* ohne Vertrauen möglich ist. Seine Sichtweise richtet sich auf einzelne Akteure und basiert u.a. auf spieltheoretischen Ansätzen. Für Gambetta ist Kooperation, wenn zwei Partner sich auf Regeln für ihre Interaktion verständigen. Diese können implizit, bspw. durch Gewohnheitsrechte, oder explizit, bspw. durch die Festlegung von Regeln, vereinbart werden. Sie sollen im Verlauf der Interaktion von beiden Partnern eingehalten werden.[76] Das Gegenteil von Kooperation ist nach Gambetta die *Konkurrenz*. Jedoch läßt sich eine Gesellschaft weder auf reine Kooperation, noch auf reine Konkurrenz ausrichten, da Konkurrenz nur dann bestehen kann, wenn darauf vertraut werden kann, daß die Akteure bestimmten Regeln folgen, in diesem Sinne also Kooperation vorhanden ist. Eine solche Kooperation muß dabei nicht notwendigerweise positiv sein, auch kriminelle Machenschaften können in einer Kooperation durchgeführt werden.

Gambetta sieht Vertrauen als

> „a particular level of subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action [...] and in a context in which it affects his own action.“[77]

Die Frage nach der Vertrauenswürdigkeit eines Akteurs stellt sich nur dann, wenn eine mögliche Interaktion mit ihm erfolgversprechend erscheint und daher eine Kooperation eingegangen werden sollte. Vertrauen ist nur dann notwendig, wenn zumindest ein Akteur die Möglichkeit hat, das Vertrauen zu brechen und trotzdem eine Kooperation zustande kommen soll. Gambetta geht davon aus,

---

[75] Vgl. Gambetta, D. (1988a).
[76] Vgl. Gambetta, D. (1988b). S. 213.
[77] Vgl. Gambetta, D. (1988a). S. 217. Gambetta nutzt den Begriff *agent*. Dieser Begriff ist nicht mit dem Begriff des Software-Agenten zu verwechseln. Ein *agent* nach Gambetta ist als *Akteur* zu verstehen.

daß eine hauptsächlich auf Vertrauen basierende Interaktion sehr viel effizienter ist als eine, die auf Zwang beruht.

3.1.4 Weitere Definitionen von Vertrauen

Neben den drei dargestellten Beschreibungen von Vertrauen existieren zahlreiche weitere, die hier beispielhaft aufgeführt werden sollen. So spezifiziert Morton Deutsch Vertrauen wie folgt:

> „(a) the individual is confronted with an ambiguous path, a path that can lead to an event perceived to be beneficial (*Va+*) or to an event perceived to be harmful (*Va-*);
>
> (b) he perceives that the occurrence of *Va+* or *Va-* is contingent on the behavior of another person; and
>
> (c) he perceives the strength of *Va-* to be greater than the strength of *Va+*. If he chooses to make an ambiguous path with such properties, I shall say he makes a trusting choice; if he chooses not to take the path, he makes a distrustful choice.“ [78]

Auch Michael Koller[79] sieht Vertrauen als die Erwartung, daß ein Interaktionspartner wohlwollendes Verhalten zeigen wird, obwohl dieser auch die Möglichkeit hat, andere, nicht-wohlwollende Verhaltensweisen zu wählen. Situationen, in denen Vertrauen notwendig ist, beinhalten stets Risiken.[80] Von einem Transaktionspartner dem man vertraut, erwartet man wohlwollendes Verhalten, da man ihn positiv bewertet, wobei die Zuverlässigkeit des Partners aus früheren Transaktionen die Grundlage für das Vertrauen bildet.

---

[78] Vgl. Deutsch, M. (1973a).
[79] Vgl. Koller, M. (1990). S. 10.
[80] Vgl. Koller, M. (1997). S. 13 ff.

Eine vertrauenswürdige Beziehung zwischen Menschen ist für Robert L. Swith

> „choosing to take an ambiguous path that can lead to a benificial event or a harmful event depending on the behaviour of the other person – where the harmful event is more punishing than the beneficial event is rewarding.“[81]

Partha Dasgupta bringt die Beschreibung von Vertrauen auf den Punkt:

> „One trusts when one has much to lose and little to gain.“ [82]

3.1.5 Voraussetzungen für Vertrauen

Zu klären ist nun, welche Voraussetzungen gegeben sein müssen, damit Vertrauen bei Interaktionen entstehen und genutzt werden kann. Drei Voraussetzungen lassen sich identifizieren:

1. Entscheidungsautonomie

Zumindest einer der beiden an einer Interaktion beteiligter Akteure muß die Möglichkeit haben, Entscheidungen zu treffen und entsprechend zu handeln. So ist es bei einem PKW sinnlos darüber nachzudenken, ob das Gaspedal Vertrauen zum Motor hat, da es nicht in der Lage ist, eine Entscheidung zu treffen. Vertrauen ist daher ein Merkmal sozialer Beziehungen,[83] die eine gewisse Entscheidungsautonomie zumindest eines Akteurs voraussetzt.

2. Unsicherheit über den Ausgang einer Interaktion

Eine weitere Voraussetzung ist, daß Vertrauen stets mit einer Interaktion verbunden sein muß, die in der Zukunft liegt und über deren Ausgang Unsicherheit besteht, also, nach Sztompka, eine „Wette auf die zukünftigen Handlungen anderer“.[84] Wenn der Ausgang einer Handlung vorab vollkommen klar ist, so ist kein Vertrauen notwendig: Es muß nicht darauf vertraut werden, daß ein Apfel vom Baum nach unten fällt. Andererseits ist bei ei-

---

[81] Swith, R. (1967).
[82] Vgl. Dasgupta, P. (1990). S. 304.
[83] Vgl. Preisendörfer, P. (1995). S. 264.

nem absolut zufälligem Ergebnis ebenfalls kein Vertrauen notwendig: So ist es zwecklos bspw. auf den Ausgang der Ziehung der Lottozahlen zu vertrauen. Die „Wette“ auf die gezogenen Lottozahlen ist hier als „blinde“ Vorleistung zu sehen. Vertrauen spielt demnach nur dann eine Rolle, wenn die möglichen Folgen einer Interaktion vorab realistisch eingeschätzt werden können. Dazu muß ein *mittlerer Kenntnisstand* der Akteure über den möglichen Ausgang der Transaktion vorhanden sein, d.h. die Akteure kennen in einem gewissen Rahmen die möglichen Ergebnisse einer Interaktion. Vertrauen in eine Umgebung, über die vollständige Informationen vorhanden sind, ist genauso sinnlos, wie in eine, in der keinerlei Informationen vorhanden sind. Der mittlere Kenntnisstand führt dazu, daß Vertrauen auch enttäuscht werden kann.[85] Dies ist jedoch eine notwendige Voraussetzung für das Vorhandensein von Vertrauen.[86]

3. Sich wiederholende Beziehungen

Vertrauen kann nur entstehen, wenn es sich um eine sich wiederholende Beziehung zwischen den Interakteuren handelt. Akteure, die nur einmal aufeinander treffen, sich komplett unbekannt sind und nur eine Interaktion miteinander durchführen, können kein Vertrauen aufbauen.

Eine wichtige Rolle spielt hierbei, daß Vertrauen auch von Gruppen auf einzelne Personen übertragen werden kann. So ist bspw. die Zertifizierung eines Akteurs durch eine bekannte Gruppe eine oft genutzte Möglichkeit, auch bei einem einmaligen Aufeinandertreffen Vertrauen zu schaffen.

3.1.6 Die Verwendung von Vertrauen im Vertrauens- und Reputationsmechanismus

Allen Definitionen von Vertrauen ist gemeinsam, daß bei einer Interaktion, bei der Vertrauen im Spiel ist, stets das Risiko der Nichterfüllung der positiven Erwartung vorhanden ist. Auch die Elemente „positive Bewertung“ des Interakti-

[84] Vgl. Sztompka, P. (1999). S. 41ff.
[85] Vgl. Gambetta, D. (1998b). S. 218-219.
[86] Vgl. Luhmann, N. (1989). S. 45.

onspartners und „positive Erwartungshaltung“ dem anderen gegenüber sind Bestandteil zahlreicher weiterer Definitionsvorschläge für Vertrauen.[87] Während Niklas Luhmann Vertrauen auf Risiko bezieht und dessen Hauptfunktion in der Reduktion der Komplexität einer Interaktion sieht, betrachtet Sztompka Vertrauen als eine „Wette in die Zukunft“. Gambetta verbindet Vertrauen mit Kooperation. Für ihn ist Kooperation zwar auch ohne Vertrauen möglich, mit Vertrauen wird diese jedoch erheblich effizienter. Andere Autoren sehen Vertrauen dann als vorhanden an, wenn eine Interaktion trotz des Risikos der Nichterfüllung einer Vereinbarung durchgeführt wird.

Für die vorliegende Arbeit ergibt sich zusammenfassend folgende Definition von Vertrauen:

> Die Interaktion, für die Vertrauen notwendig ist, liegt in der Zukunft. Zumindest ein Interaktionspartner verfügt nur über unvollständige Informationen über das Verhalten seines Partners, so daß für ihn das Risiko der Nichterfüllung der Vereinbarung besteht. Bei Eintritt des Risikofalls wäre sein Schaden größer als der bei einer korrekt durchgeführten Transaktion erzielte Nutzen. Trotzdem führt er aufgrund einer positiven Bewertung seines Partners eine Transaktion durch. Bei fehlendem Vertrauen würde eine Abschätzung aller möglichen Folgen der Transaktion dazu führen, daß die Transaktion nicht durchgeführt werden könnte.

Vertrauen erfüllt so die Funktion, daß Handlungen durchgeführt werden können, obwohl über ihren Ausgang Ungewißheit besteht. Eine Folgenabschätzung der Handlung findet nur noch in eingeschränktem Maße statt, da ihre Komplexität durch das Vorhandensein von Vertrauen reduziert wird.[88]

---

[87] Vgl. Petermann, F. (1996).

[88] Die Elemente „Risiko“ und „positive Bewertung“ sind Bestandteil zahlreicher Definitionsvorschläge für Vertrauen. Vgl. Petermann, F. (1996).

3.2 Reputation als Maß erwarteten kooperativen Verhaltens

Eng verbunden mit dem Begriff des Vertrauens ist der Begriff der *Reputation*. Reputation umschreibt den Ruf oder das Ansehen einer Person. Bei der Entstehung des Rufs und Ansehens einer Person spielt die zeitliche Komponente eine große Rolle. So meint Koller,[89] daß die erfahrene Zuverlässigkeit eines Partners aus früheren Transaktionen einen Hinweis auf ihre Vertrauenswürdigkeit liefert. Diese Hinweise entsprechen der Reputation, in der alle Informationen, die direkt oder indirekt über das Handeln gesammelt wurden, enthalten sind.

Auch für Sztompka verweist die Reputation auf die Vertrauenswürdigkeit eines Interaktionspartners. Er sieht Reputation als „Record of past Deeds."[90] Dies bedeutet, daß einer Person oder einer Institution, die schon über einen längeren Zeitpunkt hinweg bekannt ist, Vertrauen entgegengebracht wird. Die Reputation dieser Person oder Institution entsteht, wenn bereits früher Kontakte zu ihr bestanden und ihre Handlungen ausgewertet werden können. Wenn keine Kontakte zu dieser Person oder Institution bestanden, so ist es möglich, über Dritte, die Erfahrungen mit ihr gemacht haben, Informationen über deren Reputation zu erhalten. Diese Informationen liefern bspw. auch Zeitungsartikel, Medallien, Zeugnisse, Zertifikate, Orden, etc., da sie Hinweise auf das frühere Verhalten der Person oder Institution geben.

Für Gambetta ist Reputation der Ruf der Vertrauenswürdigkeit eines Interakteurs.[91] Diese wird wie folgt gebildet:

> „trust begins with keeping oneself open to evidence, acting as if one trusted, at least until more stable beliefs can be established on the basis of further information."[92]

Daraus läßt sich ableiten, daß die Bildung von Reputation in sehr kleinen Schritten erfolgt, während ihr Verlust sehr schnell vonstatten gehen kann.

---

89 Vgl. Koller, M. (1990).
90 Sztompka, P. (1999). S. 71.
91 Vgl. Gambetta, D.(1988a).
92 Vgl. Gambetta, D. (1988b). S. 234.

Eine Aufzeichnung des früheren Verhaltens ist nicht notwendig, „Words of mouth is the most common mode in which such records are kept.“[93] Auch läßt sich, so Misztal,[94] Reputation von einer Organisation mit Reputation auf eine dazugehörige Person übertragen.

Zusammenfassend läßt sich Reputation als das vertrauensbildende Ansehen eines Akteurs bezeichnen, wobei die Zuverlässigkeit des Partners aus früheren Transaktionen die Grundlage für Vertrauen bildet.[95] Dies entspricht seiner Reputation. Sie ermöglicht eine Einschätzung des Verhaltens eines Akteurs und bietet einem potentiellen Transaktionspartner einen *mittleren Kenntnisstand*.[96]

3.3 Vertrauen und Reputation von Software-Agenten

3.3.1 Voraussetzungen für Vertrauen bei Software-Agenten

In Kapitel 3.1.5 wurden drei Kriterien als Voraussetzung für die Entstehung und Nutzung von Vertrauen identifiziert.

(1) Die Möglichkeit Entscheidungen zu treffen,

(2) einen mittleren Kenntnisstand über den Transaktionspartner und die Folgen einer mit ihm durchgeführten Transaktion und

(3) eine wiederkehrende Beziehung.

Anhand der wesentlichen Eigenschaften von Software-Agenten, die in Kapitel 2.1 beschrieben wurden, soll nun geklärt werden, ob sie diese drei Voraussetzungen erfüllen können.

---

[93] Dasgupta, P. (1988). S. 66.
[94] Vgl. Misztal, B. (1996).
[95] Vgl. Koller, M. (1990). S. 14.
[96] Vgl. Kapitel 3.1.5.

(1) Software-Agenten laufen „continuously and autonomously in a particular environment...“[97] und „[they] are capable of autonomous action in some environment in order to meet their design objectives.“[98] Das wesentliche Merkmal von Software-Agenten ist also ihre Fähigkeit, autonom Entscheidungen zu treffen.

(2) Software-Agenten reagieren in unterschiedlichem Maße auf ihre Umwelt und auf weitere Software-Agenten. Ihre Reaktion beruht auf ihrem internen Modell der Umwelt, wie bereits in Kapitel 2.1.1 beschrieben (*k-level* Konzept). Software-Agenten, die dem *1-level* oder dem *2-level* zugeordnet werden, können aufgrund ihres internen Modells einen mittleren Kenntnisstand berücksichtigen. Da Software-Agenten des *0-levels* die Existenz weiterer Software-Agenten nicht in ihr Handeln einbeziehen können, sind sie auch nicht in der Lage, eine mittlere Kenntnis über sie zu erlangen und diese zu verarbeiten. Die AVALANCHE Software-Agenten sind dem *1-level* zuzuordnen.

(3) Die Möglichkeit der wiederkehrenden Beziehungen ist ebenfalls eine Voraussetzung, um Vertrauen zwischen Software-Agenten aufzubauen. Software-Agenten, die Teil eines Multi-Agenten Systems sind, verfügen über die Möglichkeit, mit einem Transaktionspartner wiederholt zu interagieren. Bei einem großen Multi-Agenten System mit sehr vielen Software-Agenten existiert zumindest die Wahrscheinlichkeit, daß Software-Agenten wieder auf Partner treffen, mit denen bereits zuvor eine Interaktion durchgeführt wurde. Daher ist die Möglichkeit wiederkehrender Beziehungen beim Einsatz von Software-Agenten in Multi-Agenten Systemen ebenfalls gegeben.

[97] Shoham, Y. (1997).
[98] Wooldrigde, M. (1999). S. 32.

Zusammenfassend läßt sich feststellen, daß Software-Agenten in der Lage sind, die Funktionen von Vertrauen zu nutzen wenn sie in einem Multi-Agenten System interagieren und ihr internes Modell ihrer Umwelt mindestens dem von Vidal und Durfee beschriebenen *1-level* Konzept zugeordnet werden kann.

Wie Vertrauen beruht auch die Nutzung von Reputation auf dem *mittleren Kenntnisstand* über die Transaktionspartner, mit denen *wiederkehrende Beziehungen* möglich sind. Da die oben beschriebenen Software-Agenten in der Lage sind zu vertrauen, verfügen sie auch über die Möglichkeiten, eine eigene Reputation aufzubauen und fremde Reputationen einzuschätzen.

3.3.2 Vertrauen in den Nutzer oder in den Software-Agenten?

Vertrauen ist ein Merkmal zwischenmenschlicher Beziehungen.[99] Software-Agenten repräsentieren ihren menschlichen Nutzer in elektronischen Systemen. Falls in einem Multi-Agenten System jeweils ein menschlicher Nutzer durch einen Software-Agenten repräsentiert wird, sollten die Reputations- und Vertrauensbeziehungen der Software-Agenten untereinander denen ihrer menschlichen Prinzipale entsprechen.

In AVALANCHE kann sich jedoch ein Nutzer durch mehrere Software-Agenten auf verschiedenen elektronischen Marktplätzen parallel repräsentieren lassen. Da das Verhalten des einzelnen Software-Agenten grundsätzlich nicht vorab berechnet werden kann, weil seine Verhandlungsstrategie stochastisch determiniert ist,[100] ist es demnach prinzipiell möglich, daß verschiedene Software-Agenten eines Nutzers, obwohl sie von ihm mit identischen Parametersetzungen gestartet wurden, in vergleichbaren Situationen unterschiedlich handeln.[101] Dadurch können die Software-Agenten eines Nutzers in unterschiedlichen Situationen unterschiedliche Reputationen genießen. Dies wäre nur dann zu vermeiden, wenn die Reputation nicht dem Software-Agenten, sondern seinem Prinzipal zugeordnet würde. Alle seine Software-Agenten würden in diesem Fall eine

[99] Vgl. Sztompka, P. (1999).
[100] Vgl. Kapitel 2.4.

identische Reputation genießen. Eine korrekte Einschätzung der Verhaltensweise des einzelnen Software-Agenten wäre dann aber nicht mehr möglich.

In AVALANCHE ist daher festgelegt, daß Reputation und Vertrauen nicht dem Prinzipal, sondern nur dem jeweiligen Software-Agenten zugeordnet werden. Bei einer ausreichend großen Zahl von Software-Agenten, die einem Prinzipal gehören, entsprechen ihre durchschnittlichen Reputations- und Vertrauenswerte denen ihres Prinzipals.

Diese Festlegung beinhaltet jedoch ein weiteres Problem. Ein Nutzer könnte sich durch mehrere Software-Agenten mit unterschiedlicher Parametersetzung repräsentieren lassen. Er könnte bspw. zwei ehrlich handelnde und einen betrügerisch handelnden Software-Agenten auf einen Marktplatz senden; in diesem Fall würden die drei Zimmermann-Agenten des Testlaufs in Abbildung 11 (Kapitel 2.7) einem Nutzer gehören. Es wäre denkbar, daß sich der Nutzer dadurch ökonomisch besser stellen würde, als wenn er drei ehrlich handelnde Software-Agenten entsandt hätte, wie in Abbildung 10 (Kapitel 2.6) dargestellt. Der Nutzer könnte mit dieser Strategie den Aufbau von Vertrauens- und Reputationsbeziehungen zwischen den Software-Agenten zu seinen Gunsten beeinflussen. Dieses Verhalten des Nutzer wird in AVALANCHE aus Modellgründen ebenfalls ausgeschlossen.

[101] Vgl. Kapitel 2.4.

3.4 Quantitative Modellierung von Vertrauen

Vertrauen zwischen Software-Agenten ist nur in offenen Multi-Agenten Systemen von Bedeutung. Da aus der Literatur kaum offene Multi-Agenten Systeme bekannt sind, wird die Bedeutung von Vertrauen und Reputation in diesem Zusammenhang ebenfalls kaum behandelt. Jedoch gibt es im Bereich der Soziologie Forschungsarbeiten, die durch die Nutzung von Multi-Agenten Systemen die Bildung von Vertrauen untersuchen. Multi-Agenten Systeme werden aber in diesem Zusammenhang nicht wie bei AVALANCHE als Prototyp, sondern zur Simulation von Gesellschaften eingesetzt. Trotz dieses unterschiedlichen Ansatzes, können diese Arbeit von großem Interesse sein, da auch in ihnen die Bildung von Vertrauen formalisiert wird. Diese Ansätze sollen im folgenden dargestellt werden. Auf folgende Fragen sollen Antworten gefunden werden:

1. Wird ein System geschaffen, das betrügerisches Verhalten unterbindet?
2. Sind die Software-Agenten in der Lage, die Funktionen von Vertrauen zu nutzen? In Kapitel 3.3.1 wurden hierzu die Aspekte des *mittleren Kenntnisstandes* und der *wiederkehrenden Beziehungen* angesprochen.[102]
3. Ist eine technische Implementierbarkeit des Systems möglich?
4. Ist die Formalisierung auch auf ein offenes Multi-Agenten System übertragbar?

Da einige der vorgestellten Modelle spieltheoretisch motiviert sind, soll an dieser Stelle kurz das Spiel des *Gefangenendilemmas* erklärt werden. Es wurde insbesondere durch Robert Axelrod[103] in Computersimulationen zur Evaluation von politischen Verhandlungsstrategien eingeführt.

[102] Es sei nochmals darauf hingewiesen, daß Software-Agenten per Definition in der Lage sind, autonom Entscheidungen zu treffen.
[103] Vgl. Axelrod, R. (1984).

3.4.1 Das Gefangenendilemma

Das *Gefangenendilemma* beruht auf der Annahme, daß zwei Verbrecher, *A* und *B*, von der Polizei – nach heftiger Gegenwehr – verhaftet wurden.[104] Die Tat läßt sich jedoch mangels Beweisen keinem von beiden nachweisen. Beide Gefangenen werden getrennt verhört. Die Polizisten machen ihnen jeweils folgenden Vorschlag: Gesteht (kooperiert) einer, wird er freigelassen und der andere erhält die Höchststrafe von 10 Jahren Gefängnis (Nutzung einer Kronzeugenregelung). Falls beide gestehen, erhalten sie jeweils 5 Jahre Gefängnis. Falls beide die Tat leugnen (Defektieren), erhalten sie jeweils eine Strafe von einem Jahr wegen Widerstandes gegen die Staatsgewalt bei ihrer Verhaftung. Die möglichen Strafmaße sind in Tabelle 2 dargestellt.

		Gefangener B	
		Kooperieren	**Defektieren**
Gefangener A	**Kooperieren**	**5, 5**	**0, 10**
	Defektieren	**10, 0**	**1, 1**

Tabelle 2: Gefangenendilemma

Das Dilemma liegt nun darin, daß es zwar aus Sicht des einzelnen Gefangenen am besten wäre, zu gestehen, aus gemeinsamer Sicht beider Gefangenen jedoch besser wäre, nicht zu gestehen. Dies soll aus Sicht des Gefangenen *A* erläutert werden: Gestehen *A* und *B*, dann erhält *A* eine Strafe von 5 Jahren. Gesteht *A*, aber *B* nicht, dann wird *A* freigelassen. Gesteht *A* nicht, aber *B* gesteht, dann muß A für 10 Jahre ins Gefängnis. *A* stellt sich durch sein Geständnis immer besser, da er keine Garantie hat, daß *B* nicht gesteht. Gestehen ist daher in diesem Fall die dominante Strategie. Das gleiche gilt auch für *B*. Wenn beide Gefangenen gestehen, wird ein Nash-Gleichgewicht erreicht. Würden sich jedoch

[104] Vgl. Weise, P. / Brandes, W. / Eger, T. / Kraft, M. (1991). S. 85ff.

beide Gefangenen vertrauen und die Tat leugnen, wären sie insgesamt bessergestellt.[105]

Wird die Entscheidungssituation nicht nur einmal, sondern beliebig oft durchlaufen, so entsteht das Grundmuster des *Iterated Prisoners' Dilemma* (IPD). In diesem Fall zeigten die Forschungsergebnisse von Axelrod[106] eine erfolgreiche Evolution von langfristig kooperierendem Verhalten anstatt eine kurzsichtige Defektion.[107]

[105] Vgl. Varian, H. (1991). S. 448f.
[106] Vgl. Axelrod, R. (1984).
[107] Die Auszahlungsmatrix nach Axelrod ist in Tabelle 5 auf Seite 67 dargestellt.

3.4.2 Marsh „Formalising Trust as a Computational Concept“

Stephen Marsh formalisiert in seinen Arbeiten die Entstehung von Vertrauen bei Software-Agenten. Er ist einer der ersten durch die Literatur bekannten Wissenschaftler, der sich mit diesem Thema auseinandersetzt. Die Formalisierung von Vertrauen führt er jedoch sehr allgemeingültig durch, was bereits in seiner Beschreibung des Begriffs *Trust* deutlich wird.

> „We call our formal description Trust in order to differentiate it from wider definitions.“[108]

Seine hier dargestellte mathematische Formalisierung ist im wesentlichen seiner Arbeit „Formalising Trust as a Computational Concept“[109] entnommen.

3.4.2.1 Vertrauen eines Agenten

Ein Agent *x* hat ein Vertrauen $T_x(y,\alpha)$ in Agenten *y* in einer Situation α.[110]

$$T_x(y,\alpha) = U_x(\alpha) \cdot I_x(\alpha) \cdot \hat{T}_x(y)$$

Agent *x* erwartet für diese Situation den Nutzen $U_x(\alpha)$. Dieser Nutzen liegt per Definition im Bereich:

$$-1 \geq U_x(\alpha) \geq 1.$$

Die Bedeutung der Situation für den Agenten *x* wird durch $I_x(\alpha)$ beschrieben, die einen Wert zwischen 0 und 1 haben kann. Der Wert $\hat{T}_x(y)$ stellt das bisherige Vertrauen von Agent *x* in Agenten *y* dar, er kann im Intervall $-1 \geq \hat{T}_x(y) > 1$ liegen. Den Wert von $\hat{T}_x(y) = 1$ schließt Marsh aus, da dies für ihn „blindem“ Vertrauen entsprechen würde, was er, mit Hinweis auf die Arbeiten Luhmanns,[111] für nicht sinnvoll erachtet.

---

[108] Vgl. Jones, S. / Marsh, S. (1997).
[109] Marsh, S. (1994a).
[110] Marsh, S. (1994a). S. 62.
[111] Vgl. Luhmann, N. (1989).

Weiter geht Marsh von drei verschiedenen Ausgangssituationen aus, die von ihm als unterschiedliche *Dispositionen* bezeichnet werden. Die Unterteilung erfolgt danach, ob der Software-Agent *optimistisch*, *realistisch* oder *pessimistisch* aus seinen Erfahrungen das Vertrauen in einen Partner einschätzt. Ein *optimistisch* einschätzender Software-Agent würde stets den möglichen Maximalwert aller bisherigen Begegnungen in vergleichbaren Situationen nehmen, ein *pessimistischer* würde stets den kleinsten Wert nutzen. *Realistische* oder *pragmatisch* handelnde Software-Agenten hingegen würden, so Marsh, den Mittelwert aus ähnlichen Begegnungen nutzen.

3.4.2.2 Die Vertrauensschwelle

Ein Agent wird, nach Marsh, nur dann eine Transaktion durchführen, wenn der situative Vertrauenswert als ausreichend hoch angesehen wird. Dazu muß der situative Vertrauenswert den Schwellenwert $Cooperation\_Treshold_x(\alpha)$ übersteigen. Marsh definiert diesen Schwellenwert als

$$Cooperation\_Treshold_x(\alpha) = \frac{Perceived\_Risk_x(\alpha)}{Perceived\_Competence_x(y,\alpha) + \hat{T}_x(y)} \cdot I_x(\alpha)$$

$Preceived\_Risk_x(\alpha)$ beschreibt hierbei das von einem Agenten x wahrgenommene Risiko in der Situation α, während

$Preceived\_Competence_x(y,\alpha)$ die Kompetenz darstellt, die Agent x dem Agenten y in der Situation α zutraut.

Marsh diskutiert diese beiden Variablen anhand von drei unterschiedlichen Kenntnisgraden des jeweiligen Agenten.[112]

Beim $Preceived\_Risk_x(\alpha)$ kann der Agent entweder *kein*, ein nur *teilweises* oder ein *komplettes* Wissen über die Situation α haben. Eine Erklärung, wie sich Zahlenwerte für diese drei unterschiedlichen Ausgangssituationen rechnerisch ablei-

[112] Vgl. Marsh, S. (1994a). S. 71ff.

ten lassen, gibt Marsh jedoch nicht. Er verwendet stets festgesetzte Werte für das $Preceived\_Risk_x(\alpha)$.

Auch bei der $Preceived\_Competence_x(y,\alpha)$ unterscheidet Marsh drei Fälle. Diese richten sich danach, wie gut der Agent seinen Partner bereits kennt.

(1) Bei einem unbekannten Agenten schätzt der Agent dessen Kompetenz mit einem allgemeinen *basic trust* ein:

$$Preceived\_Competence_x(y,\alpha) = T_x I_x(\alpha).$$

Marsh erweitert dies in seinen späteren Arbeiten und führt zusätzlich zum *basic trust* eine situative Komponente hinzu:

Vertraue (y) wenn:

$$U_x(\alpha) \cdot I_x(\alpha) \cdot T_x > \frac{Perceived\_Risk_x(\alpha)}{T_x + T_x / I_x(\alpha)}.$$ [113]

(2) Sollten sich die beiden Agenten bereits aus früheren Transaktionen bekannt sein, so kann Agent *x* auf alle Erfahrungen, die er bereits mit Agent *y* in den verschiedensten, nicht miteinander vergleichbaren Situationen *B* gemacht hat, zurückgreifen

$$Perceived\_Competence_x(y,\alpha) = \frac{1}{|B|} \sum_{\beta \in B} (Experienced\_Competence_x(y,\beta)^{t'}) \cdot \hat{T}_x(y)$$

(3) Der Agent *x* kann nur diejenigen Erfahrungen mit Agent *y* nutzen, der er in vergleichbaren Situationen *A* gemacht hat

$$Perceived\_Competence_x(y,\alpha) = \frac{1}{|A|} \sum_{\alpha \in A} (Experienced\_Competence_x(y,\alpha)^{t'})$$

[113] Vgl. Jones, S. / Marsh, S. (1997).

3.4.2.3 Auswahl des Interaktionspartners

Zur Auswahl eines Kooperationspartners schlägt Marsh vor, daß entweder der im Allgemeinen vertrauenswürdigste oder der in dieser speziellen Situation vertrauenswürdigste Agent gewählt werden sollte. Auch weitere Möglichkeiten werden von Marsh ansatzweise diskutiert, so daß bspw. der Agent mit dem niedrigsten *Cooperation_Treshold* ausgewählt wird oder der Agent mit der größten Distanz zwischen *Cooperation_Treshold* und dem situativen Vertrauen. Auch eine Kombination aus beiden Werten wäre möglich.

3.4.2.4 Anpassung des Vertrauens

Bei Marsh wird das Vertrauen nach einer abgeschlossenen Kooperation angepaßt, was sowohl das Vertrauen in den Kooperationspartner $T_x(y)$ als auch das Vertrauen in die Situation $T_x(\alpha)$ beinhaltet. Ausgehend vom *Gefangenendilemma* führt Marsh die in Tabelle 3 beschriebenen Veränderungen des Vertrauenswertes durch (aus Perspektive des Agenten *x*, T^{t+1} entspricht den neuen Werten).

	***Y* kooperiert**	***Y* defektiert**
***X* kooperiert**		
Anpassung des situativen Vertrauens	$T_x^{t+1} = T_x^t \cdot 1{,}01$	$T_x^{t+1} = T_x^t \cdot 0{,}99$
Anpassung des Vertrauens in *Y*	$T_x(y)^{t+1} = T_x(y)^t \cdot 1{,}10$	$T_x(y)^{t+1} = T_x(y)^t \cdot 0{,}90$
***X* defektiert**		
Anpassung des situativen Vertrauens	$T_x^{t+1} = T_x^t \cdot 1{,}05$	$T_x^{t+1} = T_x^t \cdot 0{,}95$
Anpassung des Vertrauens in *Y*	$T_x(y)^{t+1} = T_x(y)^t \cdot 1{,}01$	$T_x(y)^{t+1} = T_x(y)^t \cdot 0{,}90$

Tabelle 3: Anpassung des Vertrauens von Agent *X* in Marshs Modell

Die Auswahl der jeweiligen Zahlenwerte wird von Marsh nicht weitergehend diskutiert. Da die jeweiligen Vertrauenswerte auf das Intervall $-1 \geq T_x(y) > 1$ begrenzt wurden, müßte eigentlich eine absolute Grenze bei -1 und <1 festgelegt werden. Die häufig gestellte Annahme in der Soziologie, daß Vertrauen schnel-

ler verloren geht als es aufgebaut werden kann, ist bei der Anpassung der Vertauenswerte von Marsh nicht durchgängig berücksichtigt worden.

3.4.2.5 Bewertung

Marshs Vertrauensmodell soll nun dahingehend bewertet werden, ob es betrügerisches Verhalten der einzelnen Agenten unterbinden kann. In Tabelle 4 sind die möglichen Ergebnisse der Vertrauensanpassung dargestellt.

$T_x(y,\alpha)$	$U(\alpha)<0$	$U(\alpha)=0$	$U(\alpha)>0$
$\hat{T}_x(y)<0$	**positiv**	**0**	**negativ**
$\hat{T}_x(y)=0$	**0**	**0**	**0**
$\hat{T}_x(y)>0$	**negativ**	**0**	**positiv**

Tabelle 4: Mögliche Werte der Vertrauensfunktion nach Marsh[114]

Zwei mögliche Werte sind besonders auffällig: So kann das situative Vertrauen $T_x(y,\alpha)$ auch dann einen positiven Wert annehmen, wenn zwei der drei Faktoren der Gleichung negativ sind. Würden bspw., wie in Tabelle 4, der Nutzen einer Situation negativ eingeschätzt $U_x(\alpha)<0$ und einem Agenten negatives Vertrauen entgegengebracht $\hat{T}_x(y)<0$, so wäre das situative Vertrauen trotzdem positiv. Marsh hat diesen Mangel erkannt und argumentiert, daß dieses Verhalten „machiavellistisch“ sei, da es Situationen geben könnte, in denen man mit Agenten interagiert, denen man nicht vertraut, und dies in Situationen, von denen man sowieso keinen Nutzen erwartet. Diese Begründung führt jedoch zu einer grundlegenden Beschränkung der Nutzbarkeit seines Modells auf diejenigen Agenten, die sich machiavellistisch verhalten. Ein weiterer Kritikpunkt an Marshs Arbeit liegt darin, daß bei einer Indifferenz des Vertrauens ($T_x(y) = 0$) oder des Nutzens ($U_x(\alpha) = 0$) das situative Vertrauen den Wert 0 annimmt, der nicht zu interpretieren ist. Marsh versucht zwar verschiedene Erklärungen für

[114] Aus: Westermann, T. (2000). S. 30.

den Wert 0 zu geben,[115] jedoch liegen diese, auch von Marsh zumindest teilweise akzeptierten Mängel, in der mathematischen Formalisierung begründet. Die Ergebnisse der mathematischen Formeln werden dadurch in Grenzfällen nicht mehr interpretierbar, was das ganze Modell in Frage stellt.[116] Aufgrund dieser beiden Kritikpunkten ist es fraglich, ob das System tatsächlich Anreize für einen Agenten schafft, zu kooperieren.

Ein internes Modell der Software-Agenten wird von Marsh nicht explizit, sondern nur implizit behandelt. Die Software-Agenten verfügen über die Möglichkeit, Entscheidungen zu treffen. Dies wird bspw. bei der Nutzung der Vertrauensschwelle deutlich. Die Möglichkeit wiederkehrender Beziehungen ist ebenfalls eine implizite Grundannahme im Modell. Die Tatsache, daß die Software-Agenten die Bedeutung und den Nutzen einer Situation in ihren Entscheidungen berücksichtigen, läßt zudem auf einen *mittleren Kenntnisstand* schließen.

Marshs Konzept ist recht einfach zu implementieren. In der Literatur sind diverse Implementationen beschrieben, die jedoch ebenfalls zu den nicht plausiblen Ergebnissen (s.o.) geführt haben.

Unbekannte Software-Agenten werden von Marsh mit einem *basic trust* bewertet, der in seinen späteren Arbeiten um eine situative Komponente erweitert wird. Daher ist sein Modell auch grundsätzlich auf offene Multi-Agenten Systeme übertragbar.

Die Arbeiten von Stephen Marsh haben in der Wissenschaft den Grundstein für den Diskurs über Vertrauen bei Software-Agenten gelegt. Sein Modell ist die erste bekannte Formalisierung von Vertrauen in diesem Zusammenhang. Es ist auch auf offenen Systeme übertragbar. Jedoch sind die oben beschriebenen mathematischen Effekte von erheblicher Problematik, so daß das Modell ohne tiefgreifende Modifikationen kaum zu einem allgemeingültigen Aufbau der Funktionen von Vertrauen zwischen Software-Agenten führen kann.

---

[115] Vgl. Marsh, S. (1994a). S. 56f.

[116] Eine weitere Diskussion des Modells von Marsh findet sich auch in: Schillo, M. (1999). S. 26ff. und in: Westermayer, T. (2000). S. 26ff.

3.4.3 Schillo: „TrustNet“

Das Vertrauensmodell von Michael Schillo[117] beruht auf den Arbeiten von Cristiano Castelfranchi und Rino Falcone.[118] Schillo möchte mit seinem System nicht nur eine Formalisierung und Implementierung von Vertrauen ermöglichen, sondern es auch ermöglichen, daß Agenten untereinander Wissen über andere Agenten in Form von Zeugenaussagen weiterreichen. Sein Ziel ist die Beantwortung der Frage, ob Agent *A* einem Agenten *C* vertrauen kann, obwohl er *C* nicht kennt, er aber einem Agenten *B* bereits vertraut, der wiederum *C* kennt und vertraut.

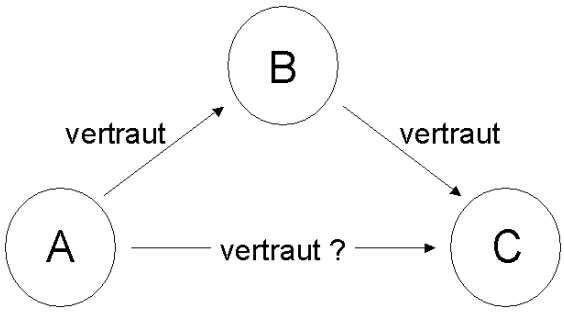


Abbildung 14: Vertrauen in unbekannte Dritte[119]

Schillo geht davon aus, daß grundsätzlich kein kooperatives Verhalten der Agenten untereinander angenommen werden kann. Daher muß bei fremden Agenten eigennütziges Verhalten unterstellt und ihre Ehrlichkeit erst ermittelt werden.

---

[117] Die Beschreibung des Vertrauensmodells basiert auf: Schillo, M. (1999) und: Schillo, M. / Funk, P. / Rovatsos, M. (1999).

[118] Die Arbeiten von C. Castelfranchi und R. Falcone werden im Rahmen der vorliegenden Arbeit nicht weiter behandelt. Sie sind in Castelfranchi, C. / de Rosis, F. / Falcone, R. (1997); Castelfranchi, C. / Falcone, R. (1998); Castelfranchi, C. / Falcone, R. (1999) und Castelfranchi, C. / Tan, Y. / Falcone, R. (1999) ausführlich beschrieben.

[119] Nach Schillo, M. (1999). S. 2.

3.4.3.1 Modifiziertes *Gefangenendilemma*

Das System wird anhand eines von Schillo modifizierten *Iterierten Gefangendilemmas*[120] modelliert.[121] Die Agenten haben vorab keinerlei Informationen über das Sozialverhalten ihrer Partner. Sie können jedoch durch Erfahrungen und Beobachtungen lernen. Dazu modelliert Schillo egoistisches und altruistisches Verhalten einzelner Agenten. Jeder Agent wird sich im Spiel mit einer festgelegten Wahrscheinlichkeit egoistisch, d.h. defektiv, oder altruistisch verhalten, d.h. stets mit seinem Spielpartner kooperieren. Permanente Defektion eines Agenten kann zu seinem Spielausschluß in der nächsten Runde führen. Zu Beginn der Experimente erhält jeder Agent 20 Geldeinheiten.

Die einzelnen Runden des modifizierten *Iterierten Gefangendilemmas* laufen in fünf Schritten ab:

1. Die Agenten müssen eine Teilnahmegebühr entrichten. Diese beträgt eine Geldeinheit.
2. Die Agenten finden sich in Paaren für Verhandlungen zusammen. Dabei können die Agenten über ihre Intentionen (geplantes egoistisches oder altruistisches Verhalten) Angaben machen. Diese müssen allerdings nicht wahr sein.
3. Das Spiel wird gespielt. Hierbei führen die beteiligten Agenten simultan ihre Spielzüge durch.
4. Nach Ende des Spiels werden die Ergebnisse veröffentlicht. Der einzelne Agent erfährt dabei jedoch lediglich die Ergebnisse seines Spiels und die der sich aus der Liste (s.u.) ergebenden unmittelbaren Nachbarschaft.
5. Die Preise werden in Form von Geldeinheiten ausbezahlt.

[120] Vgl. Axelrod, R. (1984).
[121] Vgl. Schillo, M. / Funk, P. / Rovatsos, M. (1999). S. 4ff.

In Abbildung 15 ist der vollständige Ablauf aller Phasen des Spiels in Form eines Ablaufdiagramms dargestellt.

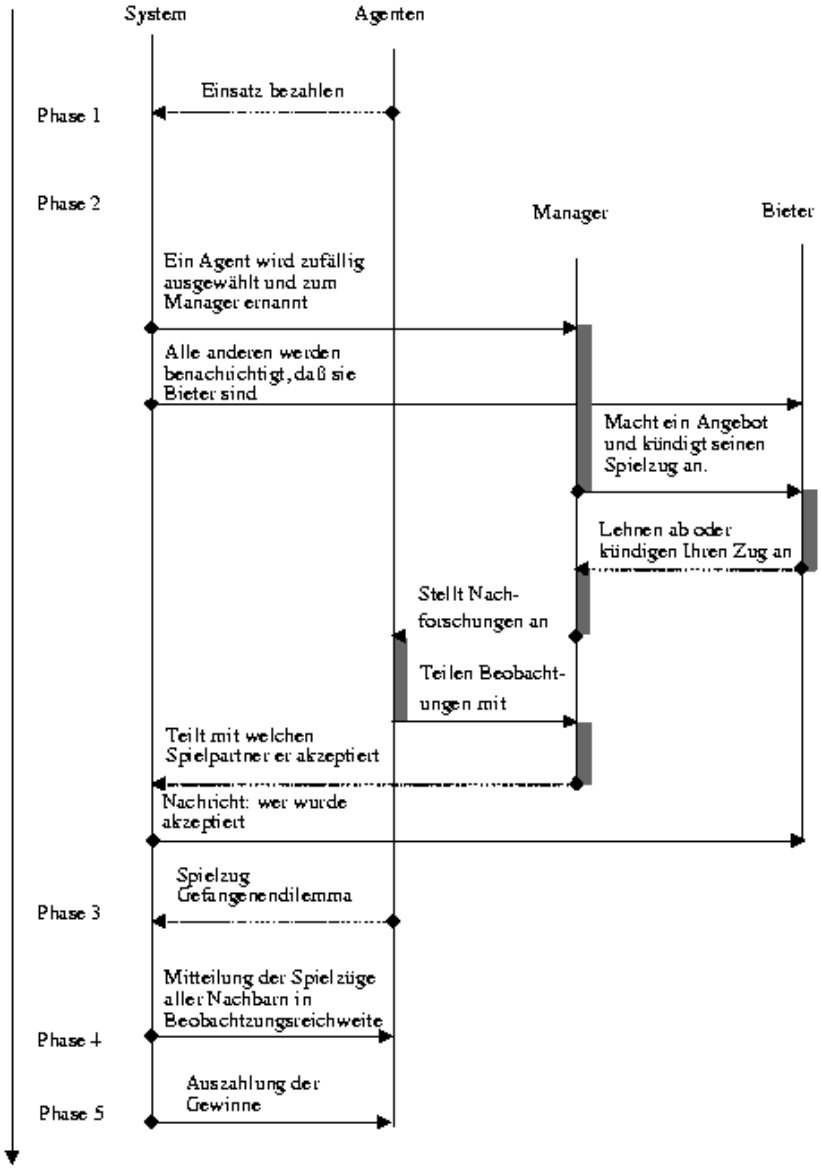


Abbildung 15: Ablauf des modifizierten *Iterierten Gefangenendilemmas*[122]

[122] Schillo, M. (1999). S. 64.

Von besondere Bedeutung ist der zweite Schritt des Spiels. Durch die Nutzung eines contract-net Protokolls[123] haben die Agenten jeweils gleiche Chancen, Partner zu finden. In jeder Runde wird durch zufälliges Sortieren eine Liste erzeugt, in der alle Agenten nacheinander aufgeführt sind. Sie garantiert, daß die Agenten jeweils mit gleicher Wahrscheinlichkeit ihre Partner tauschen. Der in der Liste als erstes genannte Agent ist der Spielführer des aktuellen Spiels. Er teilt allen Agenten mit, daß er ein *Iteriertes Gefangenendilemma* spielen wird. Alle Agenten antworten ihm und teilen ihm mit, ob sie mitspielen wollen und was ihre Intention wäre (altruistisches oder egoistisches Verhalten). Der Spielführer sucht dann seinen Spielpartner *S* aus. Da er nur über beschränkte Informationen über *S* verfügt, hat er die Möglichkeit, sich über ihn bei einem weiteren anderen Agenten *Q* zu erkundigen. Der Inhalt der Anfrage ist jedoch sehr stark eingeschränkt. Er besteht lediglich aus dem Namen des Agenten *S*. *Q* antwortet dem Spielführer, indem er eine Liste mit seinen eigenen Kooperationserfahrungen über *S* zurückschickt. Die Informationen dieser Liste müssen jedoch nicht unbedingt wahr sein. Der Spielführer überprüft daher die erhaltene Liste anhand der Erfahrungen, die er selbst bereits mit *Q* gemacht hat. Nachdem der Spielführer so seinen Partner für die Spielrunde bestimmt hat, suchen sich die anderen Agenten, entsprechend ihres Listenplatzes ihre Partner. Dabei handeln sie so wie der Spielführer. Der Spielführer kann nun also selbst nach Erfahrungslisten gefragt werden.

Nachdem sich die Spielpaare gefunden haben, wird das Spiel gespielt. Die Agenten müssen sich dabei jedoch nicht so verhalten, wie sie dies ursprünglich gegenüber ihren Spielpartnern intendiert haben. Die Ergebnisse des Spiels werden im vierten Schritt veröffentlicht. Agenten werden so zusätzlich über das Verhalten benachbarter Spielpaare in Kenntnis gesetzt. Mit diesen Ergebnissen erneuern die Agenten ihre gespeicherten Einschätzungen über andere Agenten. Das Spiel endet mit der Auszahlung der Preise im fünften Schritt. Die Auszahlung erfolgt nach der in Tabelle 5 dargestellten Auszahlungsmatrix.

[123] Vgl. Smith, R. (1980).

		Spieler B	
		Kooperieren	**Defektieren**
Spieler A	**Kooperieren**	**3, 3**	**0, 5**
	Defektieren	**5, 0**	**1, 1**

Tabelle 5: Auszahlungsmatrix nach Schillo[124]

Software-Agenten, die über kein Geld mehr verfügen, werden von den weiteren Runden ausgeschlossen.

3.4.3.2 Altruismus und Ehrlichkeit

Die Einschätzung der Vertrauenswürdigkeit fremder Agenten erfolgt, nach Schillo, durch die Einschätzung ihrer Ehrlichkeit und ihres Altruismus. Ehrlich ist ein Agent dann, wenn er sich in Schritt 3 so verhalten hat, wie in Schritt 2 angekündigt (*ehrliche_Interaktion*).

Die Ehrlichkeit *E* eines Agenten *Q* ist wie folgt definiert:

$$E(Q) = \frac{\sum ehrlicher\_Interaktionen}{\sum Interaktionen}$$

A(Q) hingegen beschreibt die Wahrscheinlichkeit mit der sich Agent *Q* altruistisch verhalten wird, also stets kooperiert.

$$A(Q) = \frac{\sum altruistischer\_Interaktionen}{\sum Interaktionen}$$

[124] Schillos Auszahlungsmatrix beruht auf der von: Axelrodt, R. (1984).

Durch mehrmaliges Spielen des *Iterierten Gefangenendilemmas* mit jeweils wechselnden Partnern ergibt sich so ein Vertrauensnetz (TrustNet).

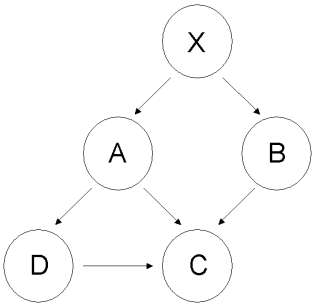


Abbildung 16: TrustNet nach Schillo[125]

In Abbildung 16 kennzeichnen die Kreise die jeweiligen Agenten und die Pfeile ihre Beobachtungen. Aus Sicht des Agenten *X* zeigt die Abbildung sein Wissen über die Agenten *A, B, C* und *D,* wobei er nur *A* und *B* direkt kennt. Über *C* und *D* hat er nur indirektes Wissen. Über *D* hat ausschließlich *A* berichtet, über *C A* und *B.* Außerdem liegt *X* eine Mitteilung von *A* vor, mit dem Inhalt, was *D* von *C* hält.

Die in den Pfeilen enthaltenen Beobachtungen der Agenten bestehen aus der Nummer der Spielrunde sowie den beiden Informationen, ob sich ein Agent altruistisch verhalten hat und ob seinem angekündigten Verhalten gefolgt ist. Beide Informationen sind binär und können lediglich die Werte *ja* oder *nein* annehmen. Jeder Agent verfügt außerdem über die Beobachtungen der anderen verbundenen Agenten. Problematisch ist die Zusammenführung von Aussagen verschiedener Agenten über einen bestimmten Agenten, wie etwa in Abbildung 16 die Aussagen von *A* und *B* über *C*. Durch einen rekursiven Algorithmus, der auf wahrscheinlichkeitstheoretischen Ansätzen beruht, versucht Schillo in diesem Fall eine integrierte Abschätzung von Ehrlichkeit und Altruismus zu ermitteln.[126]

[125] Schillo, M. (1999). S. 73.
[126] Vgl. Schillo, M. (1999). S. 77.

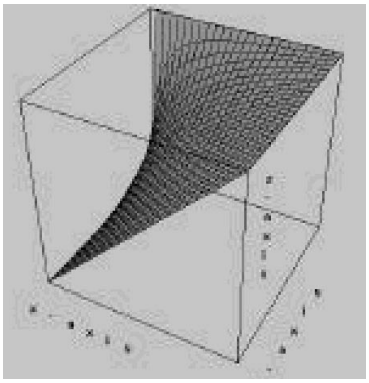
3.4.3.3 Berechnung von Vertrauen

Beruhend auf dem so erzeugten Netzwerk von Altruismus und Ehrlichkeit läßt sich die Vertrauenswürdigkeit[127] eines Agenten aus der Sicht eines anderen Agenten berechnen. Hat bspw. Agent *X* eine Kooperationszusage von Agent *Q* im zweiten Schritt des Spiels erhalten, so muß *X* aus seinen Informationen über die Ehrlichkeit $E_x(Q)$ und den Altruismus $A_x(Q)$ von *Q* sein Vertrauen $V_x(Q)$ berechnen:

$$V_x(Q) = \frac{A_x(Q)}{A_x(Q) + (1 - A_x(Q)) \cdot (1 - E_x(Q))}.$$

In Abbildung 17 ist dargestellt, wie unterschiedliche Werte von Altruismus und Egoismus zu unterschiedlichen Vertrauenswerten $V_X(Q)$ führen. Abgetragen sind an der x-Achse der Altruismus *A(Q)*, an der y-Achse die Ehrlichkeit E(Q) und an der z-Achse das daraus resultierende Vertrauen *V(Q)*.

[127] Schillo unterscheidet explizit zwischen der Vertrauenswürdigkeit und dem Vertrauen. Damit folgt er Deutsch (Vgl. Deutsch, M. (1973b)) in dessen Sichtweise von Vertrauen.



$$V_X(Q): x \rightarrow A(Q); y \rightarrow E(Q); z \rightarrow V(Q)$$

Abbildung 17: Berechnung von Vertrauenswerten[128]

Deutlich wird, daß bei einer hohen Einschätzung der Ehrlichkeit, die Einschätzung des Altruismus zur Vertrauensermittlung kaum noch ins Gewicht fällt.

3.4.3.4 Bewertung

Schillo arbeitet bei der Ermittlung der Vertrauenswürdigkeit ausschließlich mit Wahrscheinlichkeiten. Eine Kombination bzw. Multiplikation „schlechter" Werte ergibt daher, im Gegensatz zu Marsh, einen positiven Vertrauenswert. Dies wird auch darin deutlich, daß Schillo wesentlich mehr Wert auf die mathematische Herleitung legt als Marsh. Schillos Modell ist besonders deshalb interessant, weil es Zeugenaussagen berücksichtigt und somit an Realitätsnähe gewinnt.

Die Agenten verfügen über mittlere Kenntnisse bezüglich ihrer Spielpartner und dem möglichen Ausgang eines Spiels. Durch die zufällige Erzeugung von Spielpaaren besteht auch die Möglichkeit wiederkehrender Beziehungen.

Schillo untersucht jedoch nicht, ob sein System auch mit einer sehr großen Anzahl, bspw. 10.000 bis 100.000, von Agenten implementierbar ist. Die Speicherkapazität seiner Agenten wird von ihm mit $O(n^2 r)$ angeben, wobei n die Anzahl der einem Agenten bekannten Agenten ist und r die Anzahl der gespielten Run-

[128] Westermayer, T. (2000). S. 37.

den. Damit ist eine sehr große Speicherkapazität bzw. Rechenkapazität notwendig.

Des weiteren bietet Schillos System keine Ansätze zur Vertrauensermittlung von komplett unbekannten Agenten. Es wird stets vorausgesetzt, daß sich die Agenten zumindest indirekt untereinander kennen.

Insgesamt ist *TrustNet* ein sehr vielversprechendes Modell zur Berechnung von Vertrauen. Durch die Berücksichtigung der Interagentenkommunikation bietet es einen erheblichen Mehrwert gegenüber dem System von Stephen Marsh. Jedoch ist das System stets geschlossen. Ferner ist es lediglich im *Iterierten Gefangenendilemma* evaluiert worden und nicht in einem weiteren, realistischeren Anwendungsszenario. Daher muß geprüft werden, ob es auch für offene, große elektronische Marktplätze einsetzbar ist.

3.4.4 Yu und Singh: „Reputation Management in Electronic Communities“

Bin Yu und Munindar P. Singh haben ein agentenbasiertes soziales Reputationsmanagement entwickelt.[129] Das System läßt sich für elektronische Gemeinschaften im allgemeinen und für Multi-Agenten Systeme im speziellen einsetzen. Es nutzt einen Mechanismus, der Nachbarn von Software-Agenten über deren kooperatives bzw. unkooperatives Verhalten informiert.

> „If an agent *A* encounters a bad partner *B* during some exchange, *A* will penalize *B* by decreasing its rating of *B* by β and informing its neighbors.“[130]

3.4.4.1 „Tratsch“ und Zeugen

Bei der Weitergabe von Informationen über das Kooperationsverhalten eines Agenten unterscheiden Yu und Singh zwei Fälle. Zum einem kann die Weitergabe als *gossip*[131] erfolgen und zwar inkrementell von Nachbarschaft zu Nachbarschaft. Zum anderen kann die Weitergabe auch in einer *referral-chain* erfolgen. Die *referral-chains* sind eine Verkettung von Vertrauensbeziehungen. *A* vertraut *B* und *B* vertraut *C*, daher vertraut *A* auch *C*.[132] Die beiden Formen unterscheiden sich in der Vertrauenswürdigkeit der weitergegebenen Informationen sowie in der Vertrauenswürdigkeit der Informanten. Daher ist *Gossip* intuitiv weniger zu trauen als den Nachrichten eines vertrauenswürdigen Zeugen. Jedoch ist selbst bei der Nutzung von *referral-chains* nicht gewährleistet, daß alle Zeugen tatsächlich wahre Informationen weiterreichen.

In beiden Fällen werden die weitergeleiteten Informationen von den Software-Agenten zur Berechnung der Vertrauenswürdigkeit eines Transaktionspartners genutzt. [133] Der Wert des Vertrauens *T*, den ein Software-Agent *i* einem Soft-

---

[129] Vgl. Yu, B. / Singh. M. (2000).
[130] Vgl. Yu, B. / Singh. M. (2000). S. 6.
[131] Gossip [engl.]: Tratsch.
[132] Referral Chains sind in Kapitel 3.4.7.1 ausführlicher behandelt.
[133] Yu und Singh setzen Reputation mit Vertrauenswürdigkeit gleich.

ware-Agenten *j* zum Zeitpunkt *t* entgegenbringt, kann im Intervall [-1;1] liegen. Dieses Vertrauen ist definiert als $T_i(j)^t$,[134] wobei positive Werte ein Vertrauensverhältnis darstellen. Falls Software-Agent *j* kooperiert hat, erhöht sich das Vertrauen um den Zunahmefaktor α ($\alpha > 0$). Falls nicht kooperiert wurde, sinkt das Vertrauen um den Abnahmefaktor β ($\beta < 0$). Zusätzlich gilt $|\alpha| < |\beta|$, was zeigt, daß Vertrauen wesentlich schneller zerstört als aufgebaut werden kann. Der Vertrauenswert ändert sich nach jeder Interaktion.

Yu und Singh unterscheiden drei Gründe, weshalb sich der Wert der Vertrauenswürdigkeit ändern kann. Die Veränderung kann

- aufgrund einer eigenen Transaktion vorgenommen werden,
- aufgrund der Aussage eines vertrauenswürdigen Zeugen,
- aufgrund des Tratsches über einen Agenten.

3.4.4.2 Veränderung des Vertrauens nach einer eigenen Transaktion

In einem ersten Szenario haben zwei Software-Agenten direkt miteinander kooperiert. Bei der Ermittlung der neuen Vertrauenswerte unterscheiden Yu und Singh sechs Fälle.

1. Falls *j* bereits vertraut wurde ($T_i(j)^t > O$) und *j* wieder kooperiert hat, ergibt sich der neue Vertrauenswert aus der Gleichung

$$T_i(j)^{t+1} = T_i(j)^t + \alpha(1 - T_i(j)^t).$$

2. Falls *j* vor der Kooperation nicht vertraut wurde ($T_i(j)^t < O$) und *j* kooperiert hat, ergibt sich der neue Vertrauenswert aus der Gleichung

$$T_i(j)^{t+1} = \frac{T_i(j)^t + \alpha}{1 - \min(|T_i(j)^t|, |\alpha|)}.$$

[134] Die Notation ^{t} beschreibt den Zeitindex und keinen Exponenten.

3. Falls zu *j* kein Verhältnis vor der Kooperation bestand ($T_i(j)^t = O$) und *j* kooperiert hat, entspricht der neue Vertrauenswert dem Zunahmefaktor α

$$T_i(j)^{t+1} = \alpha .$$

4. Falls *j* vor der Transaktion vertraut wurde, *j* aber nicht kooperiert hat, ergibt sich der neue Vertrauenswert aus der Gleichung

$$T_i(j)^{t+1} = \frac{T_i(j)^t - \beta}{1 - \min(|T_i(j)^t|, |\beta|)} .$$

5. Falls *j* vor der Transaktion nicht vertraut wurde und *j* erneut nicht kooperiert hat, ergibt sich der neue Vertrauenswert aus der Gleichung

$$T_i(j)^{t+t} = T_i(j)^t + \beta(1 - T_i(j)^t) .$$

6. Falls zu *j* kein Verhältnis vor der Kooperation bestand und j nicht kooperiert hat, entspricht der neue Vertrauenswert dem Abnahmefaktor β.

$$T_i(j)^{t+1} = \beta .$$

3.4.4.3 Veränderung des Vertrauens aufgrund der Aussage eines vertrauenswürdigen Zeugen

In einem zweiten Szenario wird ein Software-Agent *n* auf Grund von Zeugenaussagen bewertet. Insgesamt liegen über *n L* verschiedene Zeugenaussagen *E* vor. Aus *E* wird eine Teilmenge *V* gebildet, die nur die Aussagen derjenigen Zeugen beinhaltet, denen vertraut wird. Falls mehrere Aussagen eines einzigen Zeugen über den Software-Agenten *n* vorliegen, wird nur die beste Aussage berücksichtigt. Der Wert $\hat{E}$ ist der Mittelwert aller vertrauenswürdigen Zeugenaussagen in *V*. Bei der Ermittlung der Vertrauenswürdigkeit werden drei Fälle unterschieden.

1. *n* wurde bereits vor der Transaktion vertraut ($T_i(n)^t > 0$) und auch die vertrauenswürdigen Zeugen vertrauen *n*; $\hat{E} > 0$. Der neue Vertrauenswert ergibt sich wie folgt:

$$T_i(n)^{t+1} = T_i(n)^t + \hat{E}\,(1 - T_i(n)^t)\,.$$

2. Falls *n* bereits vor der Transaktion mißtraut wurde, die vertrauenswürdigen Zeugen *n* aber vertraut haben ($\hat{E} > 0$) oder umgekehrt, dann ergibt sich der neue Vertrauenswert durch die Gleichung

$$T_i(n)^{t+1} = T_i(n)^t + \frac{\hat{E}}{(1 - \min(|T_i(n)^t|, |\hat{E}|)}\,.$$

3. Falls *n* bereits vor der Transaktion mißtraut wurde und die vertrauenswürdigen Zeugen *n* ebenfalls mißtrauen, dann ergibt sich der neue Vertrauenswert durch die Gleichung

$$T_i(n)^{t+1} = T_i(n)^t + \hat{E}\,(1 + T_i(n)^t)\,.$$

3.4.4.4 Veränderungen des Vertrauens aufgrund von Gerüchten

Die dritte Möglichkeit, den Vertrauenswert eines Software-Agenten zu ermitteln ist die Berücksichtigung des „Tratsches“, der über ihn existiert. Software-Agent *i* berücksichtigt bei seiner Vertrauensermittlung die Gerüchte $T_i(n)$ über den Software-Agenten *n*, die er von *k* gehört hat. Auch hier lassen sich wieder drei Ausgangssituationen unterscheiden.

1. Falls *i* *n* bereits vertraut hat ($T_i(n)^t > 0$) und *i* ebenfalls *k* vertraut ($T_i(k)^t > 0$) ergibt sich der neue Vertrauenswert nach neuem Erhalt eines Gerüchts durch

$$T_i(n)^{t+1} = T_i(n)^t + T_i(k)^t \cdot T_k(n) \cdot (1 - T_i(n)^t)\,.$$

2. Falls *i* weder *n* noch *k* vertraut ($T_i(n)^t < 0$; $T_i(k)^t < 0$) ergibt sich der neue Vertrauenswert nach neuem Erhalt eines Gerüchts durch

$$T_i(n)^{t+1} = T_i(n)^t + T_i(k)^t \cdot T_k(n) \cdot (1 + T_i(n)^t)\,.$$

3. Falls *i n* mißtraut, jedoch *k* vertraut oder falls *i n* vertraut, jedoch *k* mißtraut ergibt sich der neue Vertrauenswert nach neuem Erhalt eines Gerüchts durch

$$T_i(n)^{t+1} = \frac{T_i(n)^t + T_i(k)^t \cdot T_k(n)}{1 - \min(|T_i(n)^t|, |T_i(k)^t \cdot T_k(n)|)}.$$

3.4.4.5 Bewertung

Yu und Singh zeigen durch eine Reihe umfangreicher Experimente, daß ihr System zu einem raschen Ausschluß nicht kooperativer Software-Agenten führt. Kooperative Software-Agenten hingegen können ihren Vertauenswert nahezu linear erhöhen. Von besonderer Bedeutung ist die Setzung der Werte für den Zunahmefaktor α und für den Abnahmefaktor β ($|\alpha| < |\beta|$).

Die Agenten berücksichtigen bei ihren Handlungen stets die Existenz weiterer Agenten. Sie wären zwar in der Lage, Entscheidungen zu treffen, machen dies jedoch nicht, da das System lediglich dazu dient, einen menschlichen Nutzer bei der Pflege seiner Vertrauensbeziehungen zu unterstützen. Diese Vertrauensbeziehungen sind wiederkehrend.

Das System ist bereits des öfteren implementiert worden. Über die Anzahl der teilnehmenden Agenten ist jedoch in der Literatur nichts bekannt.

Der Fall, daß ein Agent erstmalig bewertet wird, wird von Yu und Singh ebenfalls berücksichtigt. Daher ist ihr System auch für ein offenes Multi-Agenten System übertragbar.

3.4.5 Zacharia: „Sporas“

Das Reputationsinformationssystem *Sporas* entstand am *Massachusetts Institute of Technology* (*MIT*).[135] Es soll lose verbundenen Agentenpopulationen Reputationsinformationen zur Verfügung stellen. Der Entwickler von *Sporas*, Giorgos Zacharia, sieht Reputation als „... the amount of trust inspired by a particular person in a specific setting or domain of interest.“[136] Grundlage des Systems ist eine einzige zentrale Instanz, die Informationen über das Kooperationsverhalten aller Software-Agenten sammelt und aus diesen Information eine Berechnung der Reputationswerte jedes Software-Agenten durchführt. Die dazu notwendigen Informationen über das Kooperationsverhalten erhält die Instanz von den an der Transaktion beteiligten Software-Agenten, die sich gegenseitig nach Ablauf einer Transaktion bewerten. Die Reputationswerte werden jedem Software-Agenten auf Anfrage mitgeteilt und gelten daher für das ganze System. Die Werte liegen zwischen 0 und 3000, wobei 0 für den schlechst möglichen und 3000 für den bestmöglichen Reputationswert steht. Ziel ist es, daß sich der jeweils ermittelte Reputationswert der tatsächlichen Reputation des betreffenden Software-Agenten annähert. Es wird daher implizit angenommen, daß jeder Software-Agent über eine objektive Reputation verfügt.

3.4.5.1 Systemgrundsätze

Zacharias System beruht auf drei Grundsätzen.

- Software-Agenten können niemals eine schlechtere Reputation erreichen, als diejenigen, die neu ins System dazukommen und erstmalig in der Instanz erfaßt werden. Damit soll verhindert werden, daß ein Software-Agent mit einer schlechten Reputation einen Anreiz erhält, seine Identität zu ändern, um den höheren Reputationswert eines neuen Software-Agenten zu erhalten.

---

[135] Vgl. Zacharia, G. (1999).
[136] Zacharia, G. (1999). S.1. Er bezieht sich mit dieser Definition auf Marsh. Vgl. Marsh, S. (1994a).

- Bei einer mehrfachen gegenseitigen Bewertungen wird jeweils nur die letzte Bewertung berücksichtigt. Dies schließt aus, daß Software-Agenten durch mehrfache, gegenseitige positive Bewertungen ihre Reputationswerte künstlich in die Höhe treiben können.
- Die Reputationswerte von Software-Agenten mit einer guten Reputation werden bei einer neuen Bewertung schwächer verändert, als bei Software-Agenten mit einer schlechten Reputation.

3.4.5.2 Ermittlung der neuen Reputationswerte

Der Reputationswert wird nach jeder Transaktion neu berechnet:[137]

$$R_{t+1} = \frac{1}{\theta} \sum_{1}^{t} \Phi(R_i) \cdot R_{i+1}^{other} \cdot (W_{i+1} - \frac{R_t}{D})$$

$$\Phi(R) = 1 - \frac{1}{1 + e^{\frac{-(R-D)}{\sigma}}} .$$

Die neue Reputation R_{t+1} ergibt sich aus der Bewertung der Transaktion W_{i+1} und der Reputation des Software-Agenten R_{i+1}^{other}, der die Bewertung durchgeführt hat. W_{i+1} kann einen Wert des Intervalls [0,1;1] annehmen. Die Formel beinhaltet zusätzlich einen Gedächtnisfaktor θ ($\theta > 0$), der die Größe des „Reputationsgedächtnisses“ steuert und eine Dämpfungsfunktion Φ, deren Steigung durch den Faktor σ determiniert ist. Die Dämpfungsfunktion beeinflußt die Stärke der neuesten Bewertung. Je kleiner σ ist, desto steiler ist die Dämpfungsfunktion. Die Variable *D* steht für den maximal erreichbaren Reputationswert, der im System auf 3000 festgelegt ist.

[137] Zacharia, G. (1999).

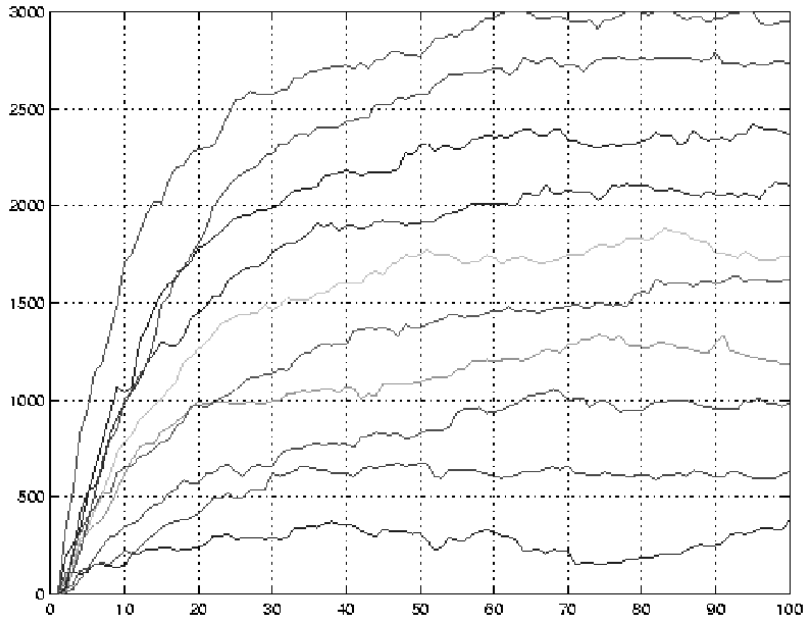


Abbildung 18: Anpassung der Reputationswerte bei Sporas[138]

In Abbildung 18 ist die Anpassung der Reputationswerte von zehn unterschiedlichen Software-Agenten abgebildet. Pro Software-Agent wurden 100 Bewertungen abgegeben (x-Achse). Auf der y-Achse sind die entstandenen Reputationswerte abgetragen. Der Gedächtnisfaktor θ betrug $\theta = 10$. Es wird deutlich sichtbar, daß sich die jeweiligen Reputationseinschätzungen näherungssweise einem bestimmten Wert anpassen.

3.4.5.3 Bewertung

Sporas ist ein System, das die Einschätzung der Vertrauenswürdigkeit von Software-Agenten ermöglicht. Es berücksichtigt zudem die Vertrauenswürdigkeit der berichtenden Zeugen. Gegenüber den bereits vorgestellten Systemen bietet es den Mehrwert, daß Software-Agenten nicht gegenseitig ihre Bewertungen in die Höhe treiben können und daß Software-Agenten keinen Anreiz haben, ihre Identitäten zu wechseln. Allerdings führt dieses System ausschließlich zu einer Ermittlung der Reputation. Eine weiterführende Sanktionierung betrügerischer Software-Agenten erfolgt nicht.

[138] Zacharia, G. (1999). S. 3.

Zacharia unterstellt außerdem implizit, daß die Software-Agenten über eine objektive, nicht veränderliche Reputation verfügen. Es ist daher zu hinterfragen, weshalb im System der Gedächtnisfaktor θ existiert. Da sich die tatsächliche Reputation der zu bewertenden Software-Agenten nicht verändert, müssen um diesen Reputationswert so genau wie möglich zu ermittlen, alte Erfahrungen genauso stark berücksichtigt werden, wie neue Erfahrungen: Durch eine ausreichend große Anzahl einzelner Erfahrungen könnte über ein arithmetisches Mittel die tatsächliche Reputation des betreffenden Software-Agenten am besten eingeschätzt werden.

Ebenfalls zu klären wäre, ob die Software-Agenten in der Lage sind, die Funktionen von Vertrauen zu nutzen. Da Zacharia die Möglichkeit berücksichtigt, daß Software-Agenten durch gewollte, gegenseitige Bewertung ihre Reputation verbessern könnten, unterstellt er, daß die Software-Agenten über ein internes Modell ihrer Umwelt in Form eines *2-levels* verfügen. Eine mögliche Entscheidungsfindung der Software-Agenten wird von Zacharia jedoch nicht angesprochen. Die Aspekte des *mittleren Kenntnisstandes* und die Möglichkeit *wiederkehrender Beziehungen* sind im System hingegen gegeben.

Als schwierig dürfte sich die technische Implementierung des Systems darstellen. Da das System jeweils nur die letzten Werte einer gegenseitigen Bewertung berücksichtigt, muß, trotz des Gedächtnisfaktors, die komplette „Entstehungsgeschichte“ jeder einzelnen Bewertung gespeichert werden. Nur so ist es möglich, daß bei einer neuen Bewertung die ältere entfernt und der Wert neu berechnet werden kann. Dadurch erhält das System eine sehr große Speicheranforderung.

Sporas ist ein System für lose verbundene Agentenpopulationen. Der Eintritt unbekannter Software-Agenten wird im System berücksichtigt, so daß es den Anforderungen für offenen Multi-Agenten Systeme entspricht.

3.4.6 Zacharia: „Histos“

Während *Sporas* für lose zusammenhängende Agentenpopulationen gedacht war, bezieht sich das System *Histos* auf Populationen, in denen die Teilnehmer sehr häufig miteinander interagieren. Das System beruht daher auf der Idee, daß sich mit der Zeit Netze von wechselseitigen Beziehungen ergeben. In diesem Zusammenhang stellt die gegenseitige Bewertung einer Transaktion eine Beziehung dar. Die Ansammlung der Beziehungen bildet ein Netz, wobei die Netzknoten für die Software-Agenten stehen und die Pfeile zwischen den Knoten für die einzelnen Bewertungen.

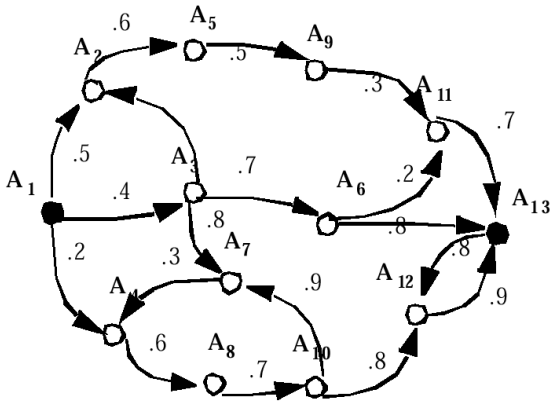


Abbildung 19: Beziehungsnetz in Histos[139]

Um die Reputation eines Software-Agenten A_1 (in Abbildung 19 der äußerst linke schwarze Knoten) aus Sicht des Software-Agenten A_{13} zu berechnen (A_{13} ist der äußerst rechte Knoten), ermittelt *Histos* anhand des Netzes, ob es eine direkte Beziehung in Form eines direkten Pfades zwischen A_{13} und A_1 gibt. Falls dies der Fall wäre, hätte der Pfad die Länge 1. A_1 würde dann unmittelbar von A_{13} bewertet. In diesem Fall würde *Histos* diese Bewertung als Reputationswert nutzen. Falls es aber, wie in Abbildung 19, keinen direkten Pfad zwischen A_1

[139] Zacharia, G / Moukas, A. / Maes, P. (1999). S. 5.

und A_{13} gibt, wird eine Suche durchgeführt, die alle Pfade zwischen A_1 und A_{13} ermittelt. Die maximale Länge eines Pfades ist jedoch auf N Verkettungen beschränkt. Falls zwischen zwei Software-Agenten, die beide einer Verkettung angehören, mehrfach Transaktionen durchgeführt wurden, werden jeweils nur die Ergebnisse der letzten Transaktion berücksichtigt. Die Anzahl der Pfade beträgt θ. Die Berechnung der Reputationswerte erfolgt nun in mehreren Schritten. Als erstes werden die Reputationswerte der Software-Agenten berechnet, die nur einen Schritt von A_1 entfernt sind. Anschließend werden dann rekursiv die Werte ihrer jeweils unmittelbaren Nachbarn ermittelt bis hin zu den direkt an A_{13} grenzenden Nachbarn. Der bekannte Reputationswert von A_{13} wird dann als Rechengrundlage genutzt. Die maximale Anzahl der Beziehungen beträgt somit $N - 1$. Die Berechnung des Reputationswertes erfolgt anhand der folgenden Formel:[140]

$$R_{t+1} = \frac{1}{\theta^t} \sum_{t-\theta^r}^{t} \frac{\Phi(R_{i+1}) \cdot (R_{i+1}^{other} W_{i+1})}{\sum_{t-\theta^r}^{t} R_{i+1}}$$

$$\theta^t = \min(\theta, m); m = \deg(A_L)$$

Die Anzahl der Pfade von A_{13} zu A_1 beträgt m. Falls es einen direkten Pfad zwischen A_{13} und A_1 gibt, wird R_{t+1} als Reputationswert genutzt. Falls die Länge der Pfade größer N ist, wird der *Sopras*-Mechanismus zur Reputationsermittlung eingesetzt, da man in diesem Fall nicht mehr von einer eng zusammenhängenden Agentenpopulation ausgegangen werden kann.

3.4.6.1 Bewertung

Histos berücksichtigt jeweils nur den neusten Pfad bei der Reputationsermittlung, so daß auch hier jeweils nur der neueste Reputationswert genutzt wird. Die Software-Agenten sind aber, wie in *Sporas*, in der Lage, die Verhaltensstrategie fremder Software-Agenten bei ihren Handlungen zu berücksichtigen (*2-level*

[140] Zacharia, G. (1999). S. 5.

Agenten). Dadurch könnte sich ein Software-Agent wie folgt verhalten:[141] Ein bereits bekannter Software-Agent *B* führt in einer ersten Runde mit jedem anderen Software-Agenten eine ehrliche Transaktion durch. Er wird dann mit einem guten Reputationswert versehen. In einer zweiten Runde führt er erneut mit allen Partnern eine Transaktion durch. Jedoch betrügt er jetzt alle Partner bis auf den Software-Agenten *C*. Dadurch wird er nun von fast allen Transaktionspartnern schlecht bewertet. Da von jedem Software-Agenten ein Pfad < *N* zu *B* gegeben ist, erfolgt die Abfrage der Bewertung in *Histos*. Falls *B* eine dritte Transaktion mit *C* durchführen will, wird der Reputationswert seiner zweiten Transaktion genutzt, da es einen direkten Pfad zwischen beiden gibt. Dieser Wert entspricht aber nicht dem tatsächlichen durchschnittlichen Verhalten von *B. C* erhält also durch *Histos* einen falschen Reputationswert. Dies wurde auch von Mike Winter[142] nachgewiesen. Winter empfiehlt daher, den zentralen Reputationsmechanismus *Sporas* anstelle von *Histos* zu nutzen.

Die Software-Agenten in *Histos* entsprechen denen aus *Sporas*. Dementsprechend gelten die Anmerkungen aus Kapitel 3.4.5.3 auch für *Histos*. Auch die Implementierbarkeit dürfte sich als problematisch erweisen, da wiederum die vollständige „Entstehungsgeschichte“ einer Bewertung gespeichert sein muß.

Histos ist vom Ansatz her ausschließlich für eng verbundene Populationen gedacht, während *Sporas* bei lose verbundenen Populationen genutzt werden soll. Daher ist *Histos* nur in Kombination mit *Sporas* auf offene Multi-Agenten Systeme übertragbar.

[141] Vgl. Westermann, T. (2000). S. 50.
[142] Vgl. Winter, M. (1999).

3.4.7 Weitere Reputationsmechanismen

Im Folgenden werden drei weitere Reputationsmechanismen beschrieben, wobei nur die beiden ersten (Kapitel 3.4.7.1 und 3.4.7.2) agentenbasiert sind. Als Ergänzung dieser Forschungsarbeiten wird in Kapitel 3.4.7.3 der Reputationsmechanismus des online Auktionshauses Ebay beschrieben, da dieser tatsächlich für ad-hoc Kooperationen in einem offenen System eingesetzt wird.

3.4.7.1 „Trust Relationships“ von Abdul-Rahman und Halles

Das Modell von Alfarez Abdul-Rahman und Stephan Halles bezieht sich direkt auf Multi-Agenten Systeme im Internet.[143] Vertrauen soll im Modell als Grundlage für ad-hoc Transaktionen dienen. Jeder Software-Agent führt eine Datenbank mit sich, in der sein Netzwerk an *Trust Relationships* gespeichert ist. *Trust Relationship* bedeutet hier die Beziehung zwischen zwei Software-Agenten. Abdul-Rahman und Halles unterscheiden zwischen zwei Formen von *Trust Relationship*. In einem Fall handelt es sich um eine direkte Vertrauensbeziehung zwischen zwei Software-Agenten (*Direct Trust Relationship*): Agent *A* vertraut Agenten *B.* Im anderen Fall vertraut Agent *A* vertraut den Empfehlungen, die Agent *B* über die Vertrauenswürdigkeit eines Agenten *C* ausspricht (*Recommender Trust Relationsship*). Im Gegensatz zu den bisher vorgestellten Arbeiten nutzen Abdul-Rahman und Halles keine stetigen Werte für die Vertrauenswürdigkeit, die sie als *Trust Values* bezeichnen, sondern diskrete, um die qualitativen Unterschiede von Vertrauen zu verdeutlichen.[144] Die möglichen Vertrauenswerte sind in Tabelle 6 dargestellt. Sie sind jedoch nie allgemein gültig, sondern beziehen sich stets auf eine Situation (*Trust Categorie*), in der bspw. ein Agent *A* einem Agenten *B* vertraut, wenn es um den Handel von Tischen geht. Dementsprechend liegen die Einträge in der Datenbank als Tripel vor, bestehend aus der ID des Agenten, der *Trust Category* und dem *Trust Value*.

---

[143] Vgl. Abdul-Rahman, A. / Halles, S. (1997).
[144] Vgl. Abdul-Rahman, A. / Halles, S. (1997). S. 53.

Wert	Bedeutung	Beschreibung
-1	**Distrust**	**Absolut vertrauensunwürdig**
0	**Ignorance**	**Keine Bewertung über die Entität möglich**
1	**Minimal**	**Niedrigstmögliches Vertrauen**
2	**Average**	**Vertrauenswürdig. Die meisten Entitäten verfügen über diesen Wert**
3	**Good**	**Vertrauenswürdiger als die meisten Entitäten**
4	**Complete**	**Völliges Vertrauen zu dieser Entität**

Tabelle 6: Direkter Vertrauenswert nach Abdul-Rahman / Halles[145]

Wert	Bedeutung	Beschreibung
-1	**Distrust**	**Absolut vertrauensunwürdig**
0	**Ignorance**	**Keine Bewertung über den Software-Agenten möglich**
1	**„Nähe“ der Beurteilung des Empfehlenden zur eigenen Beurteilung von Vetrauenswürdigkeiten**	
2		
3		
4		

Tabelle 7: Bedeutung der *Recommender Trust* Werte nach Abdul-Rahman / Halles[146]

Abdul-Rahman und Halles liefern jedoch keine Hinweise, wie die Software-Agenten ihre Bewertungen erhalten, da der Schwerpunkt ihrer Arbeit in ihrem *Recommendation Protocol* liegt, das Empfehlungen, Anfragen für Empfehlungen, und Aktualisierungen von Vertrauenswerten innerhalb des Multi-Agenten Systems wie folgt festlegt: Eine Empfehlungsanfrage wird solange von Software-Agent zu Software-Agent weitergereicht bis sie ein Software-Agent erhält,

[145] Abdul-Rahman, A. / Halles, S. (1997). S. 53.
[146] Abdul-Rahman, A. / Halles, S. (1997). S. 53.

dem der ursprünglich anfragende und der zuletzt weiterreichende Software-Agent vertraut.[147] Der anfragende Software-Agent berechnet die Vertrauenswürdigkeit der Empfehlung anhand folgender Formel:

$$tv_r(T) = \frac{tv(R_1)}{4} \cdot \frac{tv(R_2)}{4} \cdot \ldots \cdot \frac{tv(R_n)}{4} \cdot rtv(T)$$

Die verschiedenen Empfehlungen der beteiligten Software-Agenten (*Recommender Trust Value*) gehen als $tv(R_x)$ in die Gleichung ein, wobei die Empfehlung des letzten Software-Agenten *rtv(T)* am stärksten gewichtet wird. Falls ein Software-Agent auf diese Weise mehr als eine Empfehlung über die Vertrauenswürdigkeit eines weiteren Software-Agenten erhält, nutzt er das arithmetische Mittel aller Empfehlungswerte.

Das Modell von Abdul-Rahman und Halles sieht Vertrauen nicht als einen stetigen Wert an, sondern kategorisiert es in verschiedene Stufen. Leider wird nicht erläutert, wie die Software-Agenten ihre *Trust Values* erhalten und wie die spezielle Art der Vertrauensberechnung motiviert ist. Trotzdem ist es aus den oben genannten Gründen ein interessantes Modell, da es erstmalig Vertrauen in Zusammenhang mit Multi-Agenten Systemen kategorisiert.

[147] Vgl. Westermann, T. (2000). S. 42.

3.4.7.2 „COORD“ von Rasmusson und Janson

Das Projekt *Coordination Methods for Open Distributed Systems (COORD)* entstand am *Swedish Institute of Computer Science* in Kist. Ziel der Forschungsarbeit ist die Entwicklung von Methoden, mit denen agentenbasierte, dezentrale Märkte modelliert werden können.[148] Da es sich bei den modellierten Märkten um offene Systeme handelt, spielen Sicherheitsaspekte für Rasmusson und Janson eine wesentliche Rolle. [149] Sie gehen davon aus, daß Sicherheitsaspekte den Erfolg von offenen Marktsystemen determinieren.[150]

Ihr System beruht auf einem dezentralen System mit verteilten Marktplätzen. Software-Agenten, die erstmalig auf einen Marktplatz kommen, werden von bereits am Markt bekannten und etablierten Software-Agenten betreut. Diese, so die Annahme von Rasmusson und Janson, quasi vertrauenswürdigen Software-Agenten nehmen somit die Stellung einer vertrauenswürdigen dritten Partei ein. Sie unterstützen die Transaktionen des neuen Software-Agenten, in dem sie bspw. treuhänderisch Güter unter der Garantie der erfolgten Gegenleistung übergeben.

Ferner nutzen sie *Reviewer Agents*, die aktiv eigene Erfahrungen oder aber auch indirekte, von Dritten gemachte Erfahrungen an andere Software-Agenten weiterreichen. Es ist durchaus möglich, daß es sich bei letztgenannten auch um Gerüchte handelt, die durch „Tratsch“ (Gossip) entstanden sind. Für Software-Agenten könnte somit einen Anreiz bestehen, sich durch „Tratschen“ Vorteile gegenüber anderen Software-Agenten zu verschaffen. Dies kann sogar so weit gehen, daß sich Software-Agenten verbünden, um sich gegenseitig anzupreisen, was von Rasmusson und Janson als *Advertising* bezeichnet wird.[151] *Advertising* kann im Modell auch gegen Bezahlung erfolgen.: Ein Software-Agent *A* bezahlt

---

[148] Vgl. Coord (2000).
[149] Vgl. Rasmusson, L. / Janson, S. (1996) und Rasmusson, L. / Rasmusson, A. / Janson, S. (1997).
[150] Bereits in ihren früheren Arbeiten haben sich die beiden Wissenschaftler mit der Rolle von Sicherheit in agentenbasierten, offenen Marktplätzen beschäftigt.
[151] Vgl. Rasmusson, L. (1996). S. 13f.

einen Software-Agenten *B* dafür, daß er ihn bei einem dritten Software-Agenten *C* anpreist.[152]

Rasmusson und Janson überprüfen ihr System in einem ersten Schritt durch eine Simulation von agentenbasierten Märkten. Dabei wird deutlich, daß sich die Stabilität der Märkte verringert, wenn sich die Software-Agenten bei ihren Transaktionen lediglich nach den Güterpreisen richten. Rasmusson und Janson nehmen an, daß die Berücksichtigung von Tratsch die Märkte stabilisieren könnte, was sie jedoch nicht beweisen.

In einem zweiten Schritt diskutieren Rasmusson und Janson die Integration eines weiteren Reputationsmechanismus. Dazu sollen die Handlungen der *Reviewer Agents* ausgeweitet werden. Sie sollen aktiv handeln und ein Reputationsmanagement betreiben. Dieses basiert auf anonymen Testkäufen bei den unbekannten Software-Agenten und einer abschließenden Bewertung ihres Kooperationsverhaltens. Eine Anonymität seitens der *Reviewer Agents* während der Testkäufe ist notwendig, damit die getesteten Software-Agenten nicht in ihrem Verhalten beeinflußt werden. Die getesteten Software-Agenten haben einen Anreiz, sich kooperativ zu verhalten, um eine gute Bewertung zu erzielen. Außerdem können sie die Bewertung der *Reviewer Agents* dadurch positiv beeinflussen, daß sie zu niedrigeren Preisen qualitativ hochwertige Waren verkaufen. Rasmusson und Janson sehen in ihrem Modell vor, daß die durch die Testkäufe gewonnenen Informationen anderen Software-Agenten zur Verfügung gestellt werden. *Reviewer Agents* können sogar mit diesen Informationen Handel betreiben. Jedoch ist nicht gewährleistet, daß es sich bei den *Reviewer Agents* tatsächlich um vertrauenswürdige dritte Parteien handelt.

Leider ist eine Implementation dieser Systemerweiterung nicht bekannt. Trotzdem ist die Arbeit von Rasmusson und Janson von Bedeutung, da sie davon ausgehen, daß zusätzliche Informationen über Marktteilnehmer, auch wenn es sich nur um Gerüchte handeln sollte, durchaus zu einer Stabilisierung von Märkten führen könnte. Außerdem entwickeln sie spezielle *Reviewer Agents*, deren

[152] Vgl. Rasmusson, L. (1996). S. 13 und S. 25.

Tätigkeit sowohl in der Prüfung des Kooperationsverhaltens unbekannter Software-Agenten als auch im Handel mit den so gewonnenen Informationen.

3.4.7.3 Exkurs: Ebay

Nach der Beschreibung der theoretischen Modelle und Systeme zur Vertrauensbildung in Multi-Agenten Systemen soll nun eine bereits existierende Anwendung behandelt werden. Sie wird zwar nicht von Software-Agenten, sondern von Menschen genutzt, jedoch läßt sich an ihr sehr gut darstellen, wie ein System zur Vertrauensbildung in offenen Systemen funktioniert.

Es handelt sich um das Bewertungssystem des online Auktionshauses Ebay, das hauptsächlich auf den Handel zwischen Konsumenten ausgerichtet ist.[153] Im virtuellen Auktionshaus treffen Transaktionspartner aufeinander, um ad-hoc Kooperationen durchzuführen.[154] Die Partner kennen sich jedoch häufig nicht. Der Ablauf einer Aktion verläuft wie folgt: Ein Verkäufer möchte einen Artikel, häufig gebraucht, per Auktion verkaufen. Dazu beschreibt er das Gut, eventuell fügt er auch ein digitales Photo bei und speichert diese Beschreibung in einem öffentlich zugänglichen Verzeichnis. Zusätzlich legt der Verkäufer einen Mindestpreis und die Länge der Auktionsdauer fest. Potentielle Käufer können sich im Verzeichnis über das Produkt informieren. Falls sie an einem Kauf interessiert sind, geben sie ein Angebot ab. Bei dem Auktionverfahren handelt es sich um eine *Englische Auktion*: Der Bieter, der das höchste Angebot in der vom Verkäufer festgelegten Zeitspanne abgegeben hat, erhält den Zuschlag.

Die Durchführung der Transaktion kann sich nun jedoch als problematisch erweisen, da das Auktionshaus lediglich die Anbahnungs- und Verhandlungsphase der Transaktion unterstützt, nicht aber die Durchführungsphase. Entweder muß der Käufer dem Verkäufer den vereinbarten Geldbetrag vorab überweisen, oder der Verkäufer schickt dem Käufer das Produkt per Post zu und erwartet die Bezahlung des vereinbarten Preises. Selbst wenn sich die Transaktionspartner auf eine Lieferung per Nachnahme geeinigt haben sollten, besteht für den Käufer

[153] Vgl. Ebay (2000a).
[154] Vgl. Ebay (2000b).

weiterhin das Risiko, daß das von ihm erstandene Gut nicht die Eigenschaften aufweist, die der Verkäufer zugesagt hat. Bei Transaktionen, die über das online Auktionshaus vermittelt wurden, geht also immer ein Transaktionspartner in eine riskante Vorleistung. Das Risiko besteht darin, daß auf eine Vorleistung nicht die vereinbarte Gegenleistung erfolgt.

Es erscheint daher sinnvoll, daß sich ein Nutzer über die Erfahrungen anderer Nutzer mit dem betreffenden Transaktionspartner informiert. Ebay bietet dazu seinen Nutzern ein *Bewertungsforum* an.[155] Es handelt sich hierbei um eine zentrale Datenbank, in der Informationen über das Kooperationsverhalten der einzelnen Ebay-Nutzer gespeichert sind. Das Kooperationsverhalten wird von den Transaktionspartnern erfaßt und bewertet. Die Bewertung erfolgt in den Schritten [-1]; [0]; [+1], wobei [-1] für eine negative, [0] für eine neutrale und [+1] für eine positive Bewertung steht. Somit entstehen über die einzelnen Nutzer Profile, die ihr bisheriges Kooperationsverhalten abbilden.

[155] Vgl. Ebay (2000c).

Abbildung 20: Bewertungsprofil eines Ebay Nutzers

In Abbildung 20 ist das Bewertungsprofil des Nutzer *telefonshop* dargestellt, der hier als Anbieter auftritt. Er hat bisher insgesamt 122 Transaktionen bei Ebay durchgeführt, dabei wurde er 117 mal positiv, 2 mal neutral und 3 mal negativ bewertet. Im Feld „Mitgliedskarte“ sind die Bewertungen der letzten sechs Monate zusammengefaßt.

Ebay erstellt zusätzlich ein sogenanntes *Gesamtprofil* über das Kooperationsverhalten des Nutzers, in dem Bewertungen aggregiert berücksichtigt sind. Hier fließen jedoch nicht alle Bewertungen ein, sondern pro Bewertendem jeweils nur eine. Der Nutzer *telefonshop* hat in Abbildung 20 ein *Gesamtprofil* von 108 Punkten. Das *Gesamtprofil* bildet sich grundsätzlich durch Addition aller positiven Bewertungen [+1], von denen alle negativen Bewertungen [-1] abgezogen werden. Falls der Nutzer *telefonshop* jedoch von einem anderen Nutzer mehrfach positiv oder negativ bewertet wurde, zählt jeweils nur eine Bewertung, falls *telefonshop* von einem Nutzer einmal positiv und einmal negativ bewertet wurde, werden beide Bewertungen berücksichtigt. Das *Gesamtprofil* von *telefon-*

shop wurde somit wie folgt ermittelt: 117 positive Bewertungen stammen von 111 unterschiedlichen Absendern. Daher wird nur der Wert 111 berücksichtigt. Davon werden 3 negative Bewertungen abgezogen, die auch von 3 unterschiedlichen Absendern abgegeben wurden. Neutrale Bewertungen [0] bleiben unberücksichtigt. Der Nutzer *telefonshop* erhält so ein *Gesamtprofil* von 108 Punkten.

Da das Bewertungssystem in Ebay auf Zeugenaussagen beruht, ist es hilfreich, Informationen über die Vertrauenswürdigkeit der betreffenden Zeugen zu erhalten. Dazu kann in Ebay die komplette „Entstehungsgeschichte“ eines *Gesamtprofils* eingesehen werden. In Abbildung 21 ist ein Teil der Entstehungsgeschichte des *Gesamtprofils* des Nutzers *telefonshop* dargestellt.

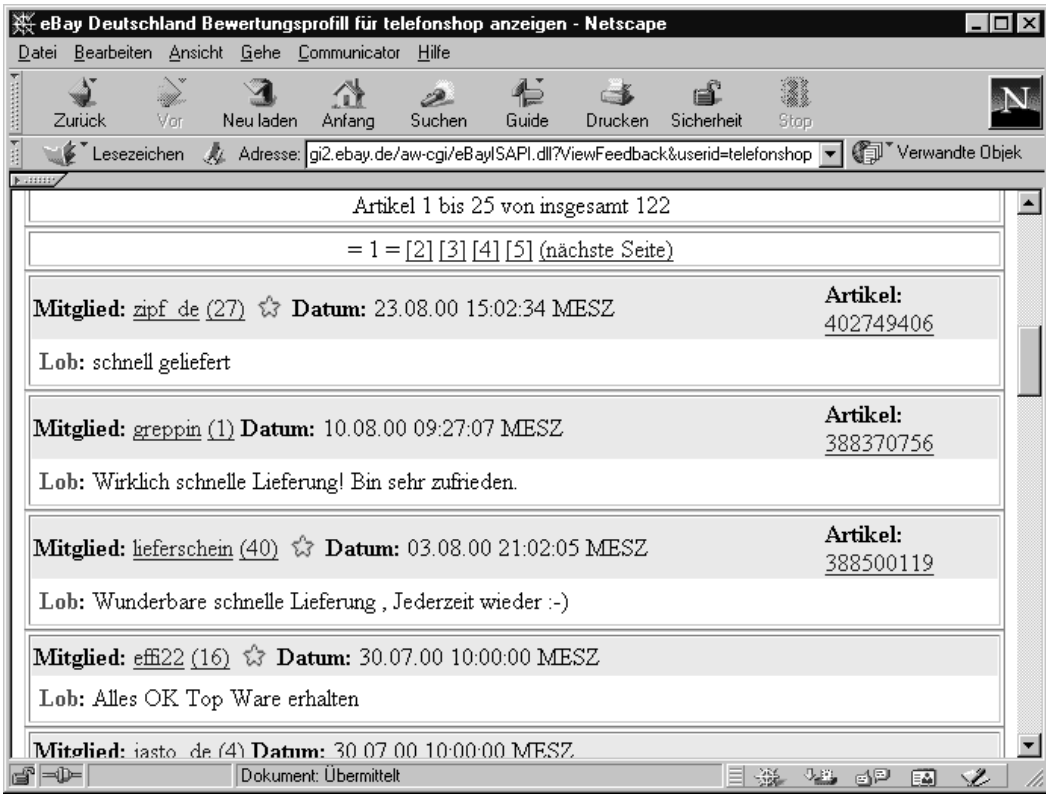


Abbildung 21: Entstehung eines *Gesamtprofils* in Ebay

Alle positiven Bewertungen [+1] sind als Eintrag „Lob“ in jeder Spalte dargestellt. Außerdem haben die Zeugen die Möglichkeit einige kurze schriftliche Anmerkungen über das erfahrene Kooperationsverhalten von *telefonshop* ab-

zugeben. Die Bewertung in der ersten Spalte wurde von dem Zeugen *zipf_de* abgegeben. Die Vertrauenswürdigkeit des Zeugen *zipf_de* ist ebenfalls berücksichtigt und durch den Wert 27 seines *Gesamtprofils* dargestellt. Wie für den Nutzer *telefonshop* in Abbildung 20 kann nun auch das Bewertungsprofil des Nutzers *zipf_de* eingesehen werden. Durch diese Verkettung von *Gesamtprofilen* entsteht ein Netzwerk von Bewertungen. Die so gewonnenen Informationen über das Kooperationsverhalten der einzelnen Ebay-Nutzer unterstützen die Einschätzung ihrer Vertrauenswürdigkeit durch einen nachfragenden potentiellen Transaktionspartner. Eine weitergehende, formelle Ermittlung eines Vertrauenswertes wird in Ebay, sieht man von dem *Gesamtprofil* einmal ab, nicht vorgenommen. Besonders sinnvoll wäre eine Ergänzung dahingehend, daß ein Wert „durchschnittliche Zufriedenheit“ ermittelt würde, der das Gesamtprofil in Verhältnis zu allen Transaktionen setzt: Der Nutzer *telefonshop* verfügt im oben genannten Beispiel über ein *Gesamtprofil* von 108 Punkten. Dieser absolute Wert ist jedoch ohne weitere Aussagekraft, da ein Nutzer *B*, der bspw. 324 Transaktionen durchgeführt hätte, von denen 216 mit [+1] und 108 mit [-1] bewertet wurden, ebenfalls ein *Gesamtprofil* von 108 Punkten erreichen würde. Der Wert des *Gesamtprofils* muß daher in Relation zur Anzahl aller Transaktionen eines einzelnen Nutzers gesetzt werden, um so eine verwertbare Aussage zu liefern.

Das Bewertungssystem von Ebay berücksichtigt eine Reihe von Empfehlungen, die in den bereits beschriebenen verwandten Forschungsprojekten gegeben wurden. So wird die Vertrauenswürdigkeit von Zeugen berücksichtigt, so wie es u.a. Schillo, Zacharia, Abdul-Rahman und Yu vorschlagen. Außerdem wird bei der Ermittlung des *Gesamtprofils* jeweils nur eine positive bzw. negative Wertung pro Nutzer berücksichtigt. Ebay folgt damit der Empfehlung von Zacharia, gegenseitiges, wiederholtes positives Bewerten zu unterbinden.[156] Die einzelnen Bewertungen können nur in ganzen Zahlenwerten abgegeben werden [-1; 0; +1], so wie es ansatzweise auch von Abdul-Rahman und Halles empfohlen wurde.

[156] Vgl. Zacharia, G. (1999). S. 1.

Ähnliche Bewertungssysteme gibt es heutzutage in zahlreichen E-Commerce Anwendungen.[157] Sie stellen einen erheblich Mehrwert dar, da sie Nutzern die Möglichkeit bieten, trotz einer vorhandenen unpersönlichen Atmosphäre, Informationen über das Verhalten ihrer Transaktionspartner zu erhalten.

3.4.8 Quantitative Modellierung von Vertrauen und AVALANCHE

In Tabelle 8 werden die einzelnen Forschungsansätze zusammenfassend betrachtet.

[157] Vgl. bspw. http://www.ricado.de, http://www.dooyoo.de oder http://www.andsold.de.

	Quantitative Modellierung von Vertrauen.	Mittlerer Kenntnisstand und wiederkehrende Beziehungen	Technische Implementierbarkeit	Berücksichtigung von Zeugenaussagen	Berücksichtigung von Tratsch	Anwendbar auf offenen Systeme	Sanktionierung unkooperativer Software-Agenten
Marsh	**JA** **Jedoch Ergebnisse, die das Modell in Frage stellen (siehe Kapitel 3.4.2.5)**	**JA**	**JA**	**NEIN**	**NEIN**	**JA** **Durch die Nutzung eines *basic trust*.**	**NEIN**
Schillo	**JA** **Nutzung von Wahrscheinlichkeiten.**	**JA**	**JA** **Bei einer sehr großen Anzahl von Software-Agenten jedoch nur mit sehr hoher Speicher- und Rechnerkapazität.**	**JA** **Software-Agenten erhalten Informationen über das Kooperationsverhalten benachbarter Software-Agenten.**	**NEIN**	**NEIN** **Dies wurde von Schillo auch nicht vorgesehen.** **Evaluation lediglich im Iterierten Gefangenendilemma.**	**NEIN**
Yu / Singh	**JA** **Zudem ist berücksichtigt, daß Vertrauen schneller verloren gehen, als aufgebaut werden kann.**	**JA**	**JA**	**JA**	**JA** **Tratsch kann sogar durch die Software-Agenten beauftragt werden.**	**JA**	**NEIN**
Zacharia: *Sporas*	**JA** **Umfangreiche Mathematische Formalisierung jedoch ohne Mehrwert.**	**JA**	**JA** **Jedoch muß die „Entstehungsgeschichte“ jeder einzelnen Bewertung komplett gespeichert werden.**	**JA**	**NEIN**	**JA**	**NEIN**
Zacharia: *Histos*	**JA** **Umfangreiche Mathematische Formalisierung jedoch ohne Mehrwert, außerdem kann das System umgangen werden (siehe Kapitel 3.4.6.1)**	**JA**	**JA** **Jedoch muß die „Entstehungsgeschichte“ jeder einzelnen Bewertung komplett gespeichert werden.**	**JA**	**NEIN**	**NEIN** **Deswegen ist von Zacharia eine gemeinsame Nutzung mit *Sporas* vorgesehen.**	**NEIN**

Tabelle 8: Quantitative Modellierung von Vertrauen

Es wird deutlich, daß sich alle lediglich auf die Ermittlung von Vertrauenswerten beschränken. Die Schaffung eines Mechanismus, der auf Grundlage dieser Vertrauenswerte betrügerische Software-Agenten sanktioniert ist in den Modellen nicht vorgesehen. Daher muß der Vertrauens- und Reputationsmechanismus in AVALANCHE über die besprochenen Ansätze hinausgehen und eine zusätzlich „Sanktionskomponente“ beinhalten.

Für die mathematischen Formalisierung von Vertrauen in AVALANCHE ist es sinnvoll, mit Wahrscheinlichkeiten zu arbeiten, um die fragwürdigen Ergebnisse zu verhindern, die in Marshs Modell entstanden sind.[158] Der Vertrauens- und Reputationsmechanismus sollte ebenfalls ein „Dazulernen“ der Software-Agenten berücksichtigen, so daß sie aufgrund ihrer wachsenden Erfahrungen das Kooperationsverhalten ihrer Transaktionspartner besser einschätzen können, so wie es von Schillo im *Iterierten Gefangendilemma* aufgezeigt wurde.[159] Dabei müssen die Software-Agenten auch auf fremde Erfahrungen zugreifen können, die sie in Form von Zeugenaussagen erhalten sollen, so wie es in den Forschungsansätzen von Schillo, Yu/Singh, Zacharia und Rassmuson/Janson vorgesehen und im Bewertungssystem von Ebay realisiert wurde. Die Entscheidung, ob und wie stark Zeugenaussagen berücksichtigt werden sollen, hängt auch davon ab, wie gut das bisherige Kooperationsverhalten des potentiellen Transaktionspartners bereits bekannt ist. Deswegen sollen die von Marsh[160] und Zacharia[161] beschriebenen Unterscheidungen nach dem „Bekanntheitsgrad“ der Transaktionspartner in den Vertrauens- und Reputationsmechanismus in AVALANCHE einfließen. Dabei ist sehr wichtig, daß auch hier die Vertrauenswürdigkeit der Zeugen beachtet wird.[162]

---

[158] Dadurch, daß Marshs Werte für Vertrauen in den Nutzer, für Vertrauen in eine Situation und für den Nutzen einer Situation aus dem Intervall [-1; 1] stammen, kann es bei einem negativen Vertrauen in den Nutzer und einem negativen Nutzen einer Situation zu positivem Vertrauen kommen. (Vgl. Kapitel 3.4.2.5.).

[159] Vgl. Kapitel 3.4.3.

[160] Vgl. Kapitel 3.4.2.

[161] Im Gegensatz zu Marsh unterscheidet Zacharia bei seinen beiden Ansätzen lediglich danach, ob die Agentenpopulationen eng oder lose miteinander verbunden sind. (Vgl. Kapitel 3.4.5 und 3.4.6.).

[162] Yu und Singh diskutieren diesen Aspekt ausführlich. (Vgl. Kapitel 3.4.4.).

Insbesondere Yu/Sing[163] sowie Zacharia[164] ziehen bei ihren Modellen die Möglichkeit in Betracht, daß die Software-Agenten durch verschiedene Handlungsstrategien Reputationswerte manipulieren können. Dies zeigt, daß ihre Modelle auf Software-Agenten basieren, die dem *2-level* nach Vidal und Durfee zugeordnet werden können. Die AVALANCHE-Agenten sind hingegen durch ihr einfaches internes Modell ihrer Umwelt zu einem derartigen Verhalten nicht in der Lage.[165] Daher muß eine Manipulation der Vertrauenswerte durch die AVALANCHE-Agenten nicht berücksichtigt werden.

Da das Kooperationsverhalten der Software-Agenten in AVALANCHE durch einen konstanten Parameter stochastisch determiniert ist, besteht kein Bedarf, ältere Erfahrungen über das Kooperationsverhalten nach einer gewissen Zeit zu vergessen, so wie es von Zacharia in *Sporas* und *Histos* vorgesehen ist.

[163] So sehen Yu und Singh vor, daß sich Software-Agenten gegenseitig dafür bezahlen könnten, daß sie „tratschen". (Vgl. Kapitel 3.4.4.1.).

[164] Zacharia unterstellt den Software-Agenten sowohl in *Histos* als auch in *Sporas*, daß sie in der Lage sind Vertrauenswerte durch häufiges Bewerten ungerechtfertigt zu beeinflussen. (Vgl. Kapitel 3.4.5 und 3.4.6.).

[165] Ihr internes Modell entspricht dem *1-level* nach Vidal / Durfee. (Vgl. Kapitel 2.4.).

4 Der Vertrauens- und Reputationsmechanismus im MAS AVALANCHE

Damit die Software-Agenten in AVALANCHE überhaupt ein unterschiedliches Kooperationsverhalten zeigen können, determiniert die Reputationskonstante *reputation* die Wahrscheinlichkeit, daß sich ein Software-Agent kooperativ verhält und Transaktionen korrekt abschließt.[166] Der Wert der Konstante liegt im Intervall [0; 1] Bei jeder Transaktion wird durch eine stochastische Probe gegen die Reputationskonstante ermittelt, ob der Agent kooperiert. Je nach Ausstattung der Agentenparameter in den Testläufen ist es damit möglich, betrügerische und ehrliche Software-Agenten zu modellieren und zu mischen.

4.1 Kooperationsverhalten in AVALANCHE als Gefangenendilemma

Vor dem Hintergrund des spieltheoretischen *Gefangenendilemmas* ist das unkooperative Verhalten des Software-Agenten als Defektieren anzusehen.[167] Die Implementation des *Gefangenendilemmas* in AVALANCHE stellt ein Defektieren des Käufers mit einer Nichtzahlung des vereinbarten Kaufpreises gleich. Analog dazu mündet ein Defektieren des Verkäufers in einer Nichtlieferung des Gutes. Dadurch ergibt sich folgende Ergebnismatrix:

[166] Vgl. Kapitel 2.4.3.
[167] Vgl. Kapitel 3.4.1.

		Käufer	
		Kooperieren	**Defektieren**
Ver-	**Kooperieren**	**Lieferung, Bezahlung**	**Lieferung, Nichtbezahlung**
käufer	**Defektieren**	**Nichtlieferung, Bezahlung**	**Nichtlieferung, Nichtbezahlung**

Tabelle 9: Ergebnismatrix bei AVALANCHE

Käufer und Verkäufer werden grundsätzlich nur dann eine Transaktion durchführen, wenn sich beide dadurch besserstellen. Also führt eine beiderseitige Kooperation zum höchsten Nutzen, der zudem Pareto-effizient[168] ist.

- **Einseitige Defektion**

  Bei einer einseitigen Defektion, stellt sich zwar der defektierende Software-Agent kurzfristig besser, langfristig aber kann der Schaden der Geschädigten so groß werden, daß ihr Kapitalbestand durch den Abzug der Lebenshaltungskosten aufgebraucht wird. Sie werden dann automatisch vom System entfernt. Im zugrunde gelegten Modellszenario kann dies zu einem Aussterben einer ganzen Handwerkerklasse, bspw. der Schreiner führen, was aus Makrosicht zu einem Systemzusammenbruch führen würde.

- **Beidseitige Defektion**

  Dies gilt auch bei einer beidseitigen Defektion, bei der zwar kein Transaktionspartner einen unmittelbaren Schaden erleidet, bei der aber auch keine Transaktion realisiert wird. Sollten alle Software-Agenten permanent defektieren, würde ihr Kapitalbestand ebenfalls auf Null sinken, was, wie oben beschrieben, zu einem Systemzusammenbruch führen würde.

[168] Pareto-Effizienz: „Ein Pareto-Optimum ist ein Zustand derart, daß sich der Nutzen irgendeines Individuums nur dann steigern läßt, wenn dadurch mindestens ein anderes Individuum schlechter gestellt wird.“ Weise, P. / Brandes, W. / Eger, T. / Kraft, M. (1991). S. 319.

- **Beidseitige Kooperation**

  Daher ist dauerhafte gegenseitige Kooperation notwendig, nicht nur um das System vor einem Zusammenbruch zu bewahren, sondern auch da nur sie langfristig zu einer erfolgreichen Entwicklung führt.[169] Da es sich um ein offenes System handelt, kann jedoch eine dauerhafte gegenseitige Kooperation nicht erzwungen werden. Deswegen muß durch den Vertrauens- und Reputationsmechanismus einen Anreiz zur dauerhaften Kooperation gegeben werden.

4.2 Reputation als „Record of past Deeds“

Reputation wird in AVALANCHE als ein Erwartungswert zukünftigen Kooperationsverhalten eines Software-Agenten gesehen und ergibt sich aus seinem bisherigen Kooperationsverhalten,[170] als „Record of past Deeds“.[171]

4.2.1 Definition von Reputation im MAS AVALANCHE

- ***Gute* Reputation**

  Hat sich ein Software-Agent in der Vergangenheit kooperativ verhalten, so erwartet seine Umwelt, daß er sich auch bei zukünftigen Transaktionen kooperativ verhalten wird, also daß er Vereinbarungen einhält, Geld für Ware bezahlt oder Ware liefert, nachdem er Geld erhalten hat. Dies wird in AVALANCHE als *gute* Reputation betrachtet

- ***Schlechte* Reputation**

  Eine *schlechte* Reputation ergibt sich, wenn das frühere Kooperationsverhalten eines Software-Agenten von seinen bisherigen Transaktionspartnern als unkooperativ gewertet wurde und daher auch bei zukünftigen Transaktionen unkooperatives Verhalten, bspw. die Nichteinhaltung von Vereinbarungen, erwartet wird.

---

[169] Vgl. Kapitel 3.4.1 und: Axelrod, R. (1984).
[170] Vgl. Marsh, S. (1994b).
[171] Sztompka, P. (1999). S. 71.

- ***Keine* Reputation**

  Falls keine Informationen über das bisherige Kooperationsverhalten eines Software-Agenten vorliegen, ist eine Reputationsermittlung unmöglich. Der Software-Agent verfügt dann über *keine* Reputation.

Dieses Bewertungsschema wird in einem Vertrauens- und Reputationsmechanismus umgesetzt. Software-Agenten können vor der Durchführung einer Transaktion Informationen über das frühere Kooperationsverhalten ihres agentenbasierten Transaktionspartners ermitteln und daraufhin ihre eigene Verhandlungsstrategie anpassen.

4.2.2 Der Reputationskoeffizient

Die Transaktionspartner des Software-Agenten versuchen, dessen ihnen unbekannte ***Reputationskonstante*** im Zeitablauf durch Erfahrungen aus Transaktionen zu ermitteln. Dazu führen sie über jeden Software-Agenten, mit dem bereits eine Transaktion durchgeführt wurde, einen ***Reputationskoeffizienten*** in dem ihre Erfahrungen über das bisherige Kooperationsverhalten des Transaktionspartners aggregiert werden. Der Wert des Koeffizienten wird nach jeder Transaktion mit dem betreffenden Software-Agenten modifiziert. Ziel ist eine möglichst genaue Prognose des zukünftigen Kooperationsverhaltens, im vorliegendem Modell also eine möglichst genaue Schätzung der jeweiligen Reputationskonstante.

Die formale Beschreibung des Reputationskoeffizienten ist wie folgt festgelegt.

R_Y^X steht für den Reputationskoeffizienten, wobei X die Identität des bewertenden Software-Agenten darstellt und Y die Identität des zu bewerteten Software-Agenten; dieser Wert ist die Einschätzung der Wahrscheinlichkeit des zukünftigen Kooperationsverhaltens von Y durch X und liegt zwischen

$0 \leq R_Y^X \leq 1$ wobei $R_Y^X = 1$ gute Reputation und

$R_Y^X = 0$ eine schlechte Reputation darstellt.

Ein einfaches Beispiel: Zwei Software-Agenten A_1 und A_2 möchten eine Transaktion durchführen. A_1 hat einen bestimmte Kooperationserwartung bezüglich A_2, abgebildet als Reputationskoeffizient $R_2^1 = \alpha$. Dieser Wert ist die von A_1 geschätzte Wahrscheinlichkeit, daß sich A_2 bei der nächsten Transaktion kooperativ verhalten wird. Ein dritter Software-Agent A_3 kann das Kooperationsverhalten von A_2 anders einschätzen und einen anderen Wert seines Reputationskoeffizienten für A_2 haben, es gilt dann: $R_2^3 \neq R_2^1$. Der Reputationskoeffizient bildet die Basis für einen Reputationsmechanismus, der zu der angestrebten Identifikation und Sanktionierung betrügerischer Software-Agenten eingesetzt wird.

4.3 Funktionsweise des Vertrauens- und Reputationsmechanismus

Bei der Funktionsweise des Mechanismus sind die Empfehlungen aus Kapitel 3.4.8 der beschriebenen verwandten Forschungsprojekte berücksichtigt. Auf die einzelnen Empfehlungen wird jeweils in den Fußnoten hingewiesen.

Die Funktion des Vertrauens- und Reputationsmechanismusses ist in fünf Phasen unterteilt:

1. **Erfassung**

   Nach der Durchführung einer Transaktion wird das jeweilige Kooperationsverhalten des Partners erfaßt. In AVALANCHE wird in dieser Phase festgestellt, ob der zu bewertende Software-Agent die Transaktion wie vereinbart durchgeführt hat.

2. **Bewertung**

   Das erfaßte Verhalten wird bewertet, d.h. mit einem Maßstab in Beziehung gesetzt. Sowohl die Erfassung als auch die Bewertung von Verhalten ist in der Realität nicht einfach. Im vorliegenden Modell gibt es nur den binären und objektiv meß- und bewertbaren Fall: Kooperation oder Defektion.

3. **Speicherung**

   Die erfaßten Daten werden gespeichert. Je nach Implementation kann diese Speicherung bei einem der Beteiligten der Transaktion, aber auch einer dritten Partei stattfinden.

4. **Abfrage**

   Vor der Durchführung einer neuen Transaktion werden die Bewertungen abgerufen und verarbeitet. In unserem Modell wird eine Entscheidungssituation über die Durchführung einer Verhandlung mit einem Software-Agenten durch diese Informationen beeinflußt.

5. **Anpassung**

   In dieser letzten Phase kann der Software-Agent aufgrund der erhaltenen Informationen die eigene Verhandlungsstrategie entsprechend modifizieren.

Im Gegensatz zu allen in Kapitel 3.4 beschriebenen verwandten Forschungsprojekten, wird beim Vertrauens- und Reputationsmechanismus in AVALANCHE die gewonnene Reputationsinformation in der fünften Phase genutzt, um die eigene Verhandlungsstrategie anzupassen.

Die fünf Phasen können von verschiedenen Aufgabenträgern durchgeführt werden. Um vertrauenswürdige Informationen über das Kooperationsverhalten eines Software-Agenten zu erhalten, ist daher auch die Vertrauenswürdigkeit der verschiedenen Aufgabenträger zu berücksichtigen. Zwei wesentliche Szenarien der Aufgabenverteilung lassen sich unterscheiden, zwischen denen es viele Mischformen geben kann. Bei der „dezentralen Verarbeitung“ werden sämtliche Phasen lokal durch den erfassenden Software-Agenten selbst durchgeführt. Bei der „zentralen Verarbeitung“ findet die Bewertung und Speicherung in einer zentralen Instanz statt.

Unterschiede gibt es auch in der Ermittlung der Reputation, wo sich drei Szenarien nach dem „Bekanntheitsgrad“ des zu bewertenden Software-Agenten *Y* unterscheiden lassen:

Fall A: Der zu bewertende Software-Agent *Y* ist dem nachfragenden Software-Agenten *X* bereits aus früheren Transaktionen direkt bekannt (siehe Kapitel 4.4).

Fall B: Der zu bewertende Software-Agent *Y* ist nur indirekt bekannt, d.h. es existiert eine dritte Instanz *Z*, die Reputationsinformationen über *Y* besitzt und diese an *X* weiterleitet (siehe Kapitel 4.5).

Fall C: Der zu bewertende Software-Agent *Y* ist *X* unbekannt. Es gibt auch keine Instanz, die über Information von *Y* verfügt (siehe Kapitel 4.6).

Beide Software-Agenten *X* und *Y* bewerten jeweils parallel das Kooperationsverhalten ihres Transaktionspartners, d.h. *X* bewertet *Y*, *Y* bewertet parallel dazu *X*. Die Funktionsweise des Reputationsmechanismus wird im Folgenden aus Sicht des Software-Agenten *X* beschrieben: *X* plant mit dem Software-Agenten *Y* eine Transaktion durchzuführen und möchte dazu den Reputationsmechanismus nutzen.

4.4 Die Software-Agenten haben bereits miteinander kooperiert

In diesem Fall haben beide Software-Agenten bereits Erfahrungen über das bisherige Kooperationsverhalten ihres Transaktionspartners. *X* verfügt demnach bereits über eine Einschätzung der Reputation von *Y*. Im Modell wird angenommen, daß ausschließlich eigene Kooperationserfahrungen genutzt werden und diese nicht an Dritte zur dezentralen Verarbeitung weitergeleitet werden. Der Reputationsmechanismus wird nun anhand der fünf Phasen beschrieben:

1. **Erfassung**

   Das Kooperationsverhalten von *Y* wird durch *X* erfaßt.

2. **Bewertung**

   Das Kooperationsverhalten von *Y* wird durch *X* bewertet: Sollte Y die Transaktion kooperativ durchgeführt haben, so wird dies von *X* mit $r_j = 1$ bewertet, wobei *r* die Bewertung des Kooperationsverhaltens und *j* die Nummer der betreffenden Transaktion darstellt. Sollte sich *Y* unkooperativ verhalten haben, gilt $r_j = 0$.

3. **Speicherung**

   Da es keinen Austausch von Kooperationsinformationen mit anderen Agenten gibt, werden die Daten nur lokal durch *X* in aggregierter Form gespeichert. Dafür wird der Reputationskoeffizient R_Y^X berechnet:

   $$R_{Y\ n+1}^{X} = \frac{\sum_{j=1}^{n+1} r_j}{n+1}$$

   (*n* = Anzahl aller Transaktionen, die *X* mit *Y* bisher durchgeführt hat).

   Aus der Formel ist ersichtlich, daß ein arithmetisches Mittel zur Ermittlung der Reputationskonstante genutzt wird. Da die Software-Agenten über einen konstanten Reputationsparameter verfügen, findet keine Änderung ihres Kooperationsverhaltens statt.[172] Die genauste Einschätzung dieses Parameters kann somit nur durch häufige Transaktionen erfolgen.

4. **Abfrage**

   Um eine erneute Transaktion mit *Y* durchzuführen, ruft *X* den Reputationskoeffizienten des potentiellen Transaktionspartners in seiner eigenen, lokalen Datenbank ab.

[172] Dies ist auch bei Zacharia der Fall. Jedoch wurde in seine Modell eine „Gedächtnisfunktion“ eingeführt, die aber keinen Mehrwert bietet.

5. **Anpassung**

   Die Angebote potentieller Transaktionspartner werden entsprechend eines Erwartungswertes p_i^* eingeordnet, der sich aus dem tatsächlich geforderten Preis p_i und dem Reputationskoeffizienten R_Y^X des Anbieters errechnet:

$$p_i^* = p_i + \left(\left(1 - R_{Yi}^X\right) \cdot p_i\right) = p_i \cdot \left(2 - R_{Yi}^X\right).$$

   In der Ermittlung ist somit der Schaden durch das eventuell unkooperative Verhalten des Transaktionspartners berücksichtigt (Defektionsschaden), der genau dem vereinbarten Preis der zu handelnden Waren entspricht. Die Einordnung mehrerer Angebote soll anhand eines Beispiels verdeutlicht werden: Software-Agent *X* hat vier Angebote von vier unterschiedlichen Software-Agenten (3, 5, 12 und 16) erhalten. Diese werden von *X* wie folgt sortiert:

Angebot *i*	**Software-Agent Y_i**	**Angebotener Preis p_i**	R_{Yi}^X	$2 - R_{Yi}^X$	p_i^*	**Bewertete Angebote**
1	**12**	**47**	$R_{12}^X = 0.63$	**1.37**	**64.39**	**2**
2	**3**	**52**	$R_3^X = 0.65$	**1.35**	**70.20**	**3**
3	**16**	**54**	$R_{16}^X = 0.85$	**1.15**	**62.10**	**1**
4	**5**	**56**	$R_5^X = 0.44$	**1.56**	**87,36**	**4**

Tabelle 10: Modifikation der Angebotspreise

X wird die Reihenfolge seiner Verhandlungsversuche anhand der bewerteten Angebote durchführen, und nicht nach den originalen Angebotspreisen. Er startet seine Verhandlungen daher mit Software-Agent Y_{16}. Erst wenn diese Verhandlungen scheitern, wird *X* mit Y_{12} in Kontakt treten.

4.5 Die Software-Agenten haben noch nicht miteinander kooperiert

Falls beide Agenten einander unbekannt sind, verfügt *X* über keine Reputationsinformationen von *Y*. Es existiert jedoch eine Instanz *Z*, die Informationen über das frühere Kooperationsverhalten von *Y* von anderen Agenten zur Verfügung stellt. Wieder ausgehend von den fünf Phasen, wird der Reputationskoeffizient wie folgt ermittelt:

1. **Erfassung**

   Das aus einer Transaktion resultierende Kooperationsverhalten von *Y* muß erfaßt werden. Es kann entweder (wie hier) vom damaligen Transaktionspartner *T* oder von einem externen Beobachter *B* erfaßt werden. Beide sind nicht notwendigerweise vertrauenswürdig.

2. **Bewertung**

   Das Verhalten von *Y* muss entweder durch *T* oder *B* bewertet werden. Im Modell erfolgt die Bewertung durch *T*. Sie ist als $r_{Y\ n}^{T}$ definiert und stellt unkooperatives Verhalten von *Y* bei der Transaktion *n* als $r_{Y\ n}^{T} = 0$ und kooperatives Verhalten als $r_{Y\ n}^{T} = 1$ dar.

3. **Speicherung**

   Die Bewertung von *T* wird an eine zentrale Instanz *Z* weitergeleitet, die Bewertungen zentral bereithält und auf Anfrage verteilt. Wie bei *T* und *B* ist auch hier nicht prinzipiell gewährleistet, daß *Z* eine vertrauenswürdige Partei ist. *Z* könnte aus strategischen Gründen falsche Reputationsinformation über *Y* verteilen. In AVALANCHE wird *Z* jedoch als vertrauenswürdige dritte Partei betrachtet, die Bewertungen nicht manipuliert, sie stets weiterleitet und keinen Software-Agenten von ihrem Bezug ausschließt. Die Bewertungen müssen verarbeitet werden. Diese Verarbeitung kann in der zentralen Instanz *Z* oder im nachfragenden Software-Agenten *X* erfolgen. Im Prototyp wird die Verarbeitung der Bewertung durch *Z* vorgenommen. *Z* aggregiert die einzelnen Bewertungen zu einem Reputationskoeffizienten R_Y. Da *Z*

die Bewertungen nicht selbst erfaßt, sondern von *T* erhalten hat, muß auch dessen Vertrauenswürdigkeit R_T berücksichtigt werden.[173] Die Aggregation des Reputationskoeffizienten ergibt sich wie folgt.[174]

$$R_{Y\ t+1} = \frac{(R_{Y\ t} \cdot t) + (r_{Y\ n}^{T} \cdot R_T)}{t + R_T}$$

(*t* = Anzahl aller Transaktionen von *Y*, die *Z* bekannt sind)

4. **Abfrage**

   X ruft aus der von *Z* geführten zentralen Datenbank die neuesten Reputationsdaten über *Y* ab.

5. **Anpassung**

   Das Kooperationsverhalten muss modifiziert werden. In AVALANCHE wird die Modifikation von *X* durchgeführt, indem er die Angebotspreise mit den Reputationskoeffizienten der anbietenden Software-Agenten zu einem modifizierten Angebotspreis verknüpft und damit eine Verhandlungsreihenfolge bildet, wie bereits in Kapitel 4.4 beschrieben.

In Abbildung 22 sind zur Verdeutlichung der Funktionsweise des Vertrauens- und Reputationsmechanismus die Rollen der einzelnen Software-Agenten und Instanzen abgebildet.

[173] Hier folgt der Vertrauens- und Reputationsmechansimus in AVALANCHE den Empfehlungen von Yu und Singh. Vgl. Kapitel: 3.4.4.

[174] Hier sei nochmals auf die getroffene Annahme hingewiesen, dass die Reputationskonstante auch die Wahrscheinlichkeit bestimmt mit der *T* wahre Informationen weiterreicht.

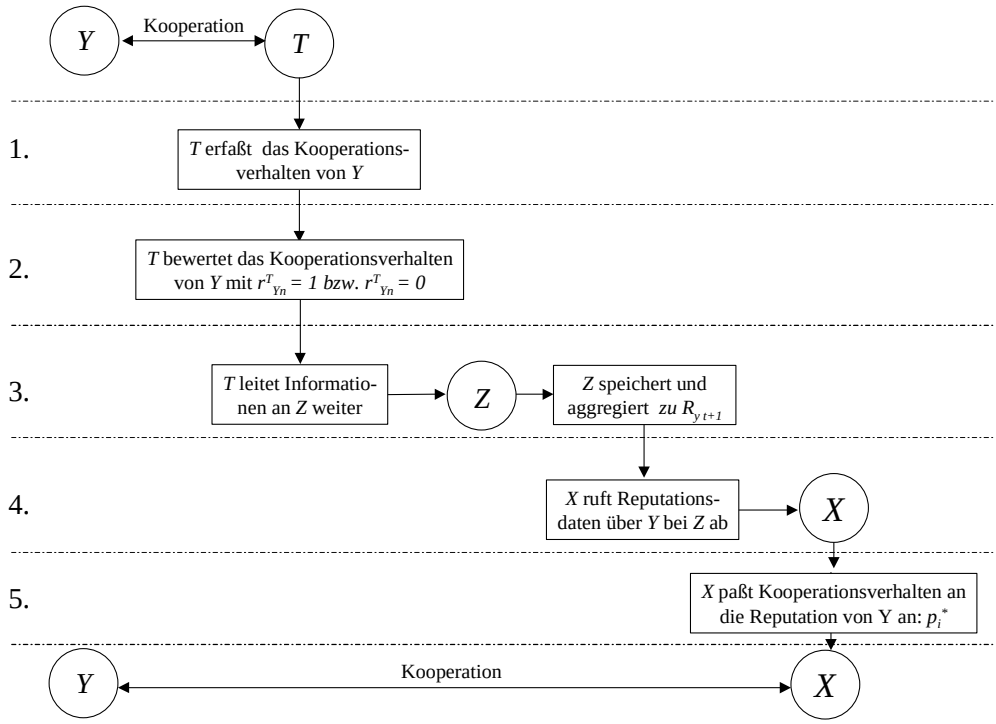


Abbildung 22: Modifikation des Reputationskoeffizienten

4.6 Die Software-Agenten sind einander und einer dritten Instanz unbekannt

In dem Fall daß die Software-Agenten einander und einer dritten Instanz unbekannt sind, läßt sich lediglich die letzte Phase des Modells anwenden. Anstelle der ersten vier Phasen kann Software-Agent *X* durch Festlegung durch den Nutzer unterschiedliche Werte des Reputationskoeffizienten von *Y* wählen. *X* kann

- den Wert seiner eigenen Reputationskonstanten nutzen;
- einen Mittelwert aller ihm bekannten Reputationskoeffizienten nutzen. Zugrunde gelegt werden können auch die Mittelwerte spezieller Agentenklassen (Schreiner, etc.), von Agentenpopulationen auf einem Marktplatz oder aller bisher bekannten Software-Agenten;

- einen festen Wert, der bei der Konfiguration des Software-Agenten festgelegt wurde (bspw. best-case, worst-case, 0.5, …), nutzen;
- einen zufälligen Wert nutzen;
- ...

Der so gewählte Reputationskoeffizient führt in Phase 5 zu einer Modifikation der Auswahl der Verhandlungspartner, wie bereits beschrieben.

4.7 Zusammenfassende Darstellung des Reputationsmechanismus

Zusammenfassend wird in Tabelle 11 dargestellt, welche Träger die einzelnen Funktionen des Reputationsmechanismus in AVALANCHE übernehmen.

Bekanntheitsgrad **Phasen**	**bekannter Partner**	**zentraler Instanz bekannt**	**unbekannter Partner**
Erfassung des Kooperations-verhaltens	**durch den nach-fragenden Agenten *X* (vertrauenswürdig)**	**Durch den damaligen Transaktionspartner *T* (nicht unbedingt vertrau-enswürdig)**	**Nicht möglich**
Bewertung des Kooperations-verhaltens	**durch den nach-fragenden Agenten *X* (vertrauenswürdig)**	**Durch den damaligen Transaktionspartner *T* (nicht unbedingt vertrau-enswürdig)**	**Nicht möglich**
Verteilung der Bewertung	**wird nicht unterstützt**	**Durch zentrale Instanz *Z* (vertrauenswürdig)**	**(willkürliche) Auswahl eines Reputations-koeffizienten**
Verarbeitung der Bewertung	**durch *X*:** $R_{Y\ n+1}^{X} = \frac{\sum_{j=1}^{n+1} r_j}{n+1}$	**durch *Z*:** $R_{Y\ t+1} = \frac{(R_{Y\ t} \cdot t) + (r_{Y\ n}^{T} \cdot R_T)}{t + R_T}$	**Nicht möglich**
Modifikation der Verhand-lungsstrategie	$p_i^* = p_i \cdot \left(2 - R_{Yi}^{X}\right)$	$p_i^* = p_i \cdot \left(2 - R_{Yi}^{X}\right)$	$p_i^* = p_i \cdot \left(2 - R_{Yi}^{X}\right)$

Tabelle 11: Funktionsweise des Vertrauens- und Reputationsmechanismus

5 Experimente und Evaluation des Vertrauens- und Reputationsmechanismus

Der Vertrauens- und Reputationsmechanismus wurde durch zwei weitere Testreihen, die wiederum aus mehreren Testläufen bestanden, überprüft. Die einzelnen Testreihen unterschieden sich durch das Kooperationsverhalten der Zimmerleute-Agenten. In der 5. Testreihe wurden, ausgehend von der 4. Testreihe eigene Reputationsinformationen zur Verhaltensanpassung genutzt (Dezentrale Verarbeitung). In der 6. Testreihe wurden im Gegensatz zur 5. Testreihe keine eigenen Reputationsinformationen, sondern die der zentralen Instanz genutzt (Zentrale Verarbeitung). In beiden Testreihen wurde bei vollkommen unbekannten Software-Agenten ein Reputationskoeffizient von 0,5 unterstellt. Die folgenden beiden Abbildungen zeigen jeweils zwei typische Ergebnisse eines Testlaufs.

5.1 Nutzung eigener Reputationsinformationen

In dieser Testreihe nutzen die Software-Agenten ausschließlich eigene Reputationsinformationen. Bei unbekannten agentenbasierten Transaktionspartnern wurde für die erstmalige Transaktion ein Reputationskoeffizient von 0,5 angenommen.

V. Testreihe

Abgetragen sind die Bargeldbestände (Y-Achse) im Zeitablauf (X-Achse), jeweils nach einer erfolgten Transaktion.

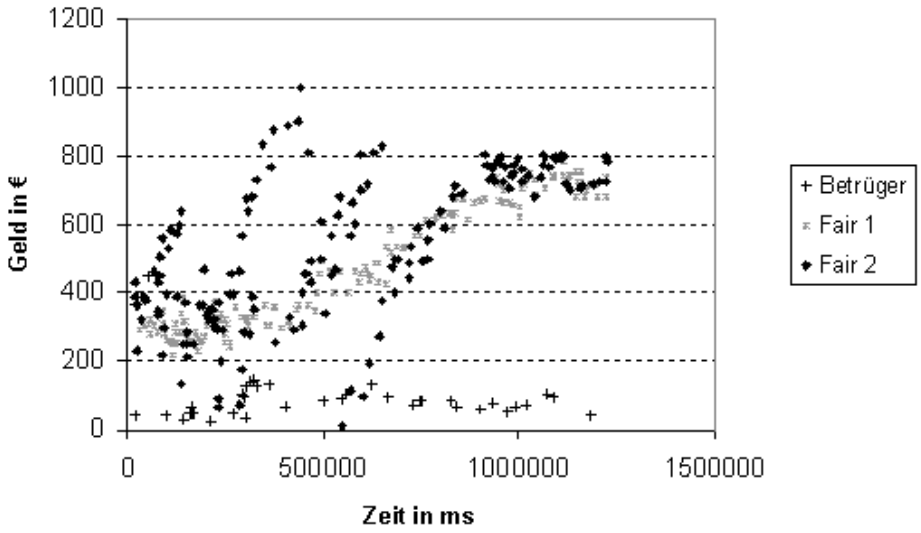


Abbildung 23: Entwicklung der Bargeldbestände bei der Nutzung eigener Reputationsinformationen (Ergebnis des Testlaufs 4 der V. Testreihe)

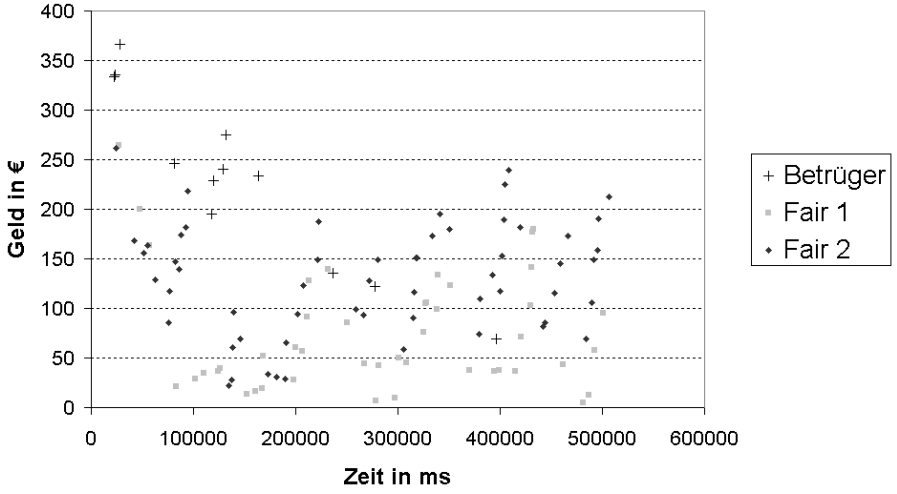


Abbildung 24: Entwicklung der Bargeldbestände bei der Nutzung eigener Reputationsinformationen (Ergebnis des Testlaufs 2 der V. Testreihe)

Aus Abbildung 23 und Abbildung 24 wird ersichtlich, daß der betrügende Software-Agent zu Beginn der Testreihe zum Teil erfolgreicher handeln konnte als seine Mitbewerber. Im Lauf der Zeit wurde er jedoch zunehmend vom Marktgeschehen ausgeschlossen und die beiden kooperativen

Software-Agenten konnten vermehrt Transaktionen durchführen. Dies ist durch die zunehmenden Erfahrungen der Transaktionspartner der Zimmermann-Agenten erklärbar. Nachdem diese Software-Agenten von unkooperativen Zimmermann-Agent betrogen wurden, änderten sie ihre Verhandlungsstrategie und griffen auf die Angebote der kooperativen Zimmermann-Agenten zurück.

5.2 Nutzung einer zentralen Reputationsinstanz

In dieser Testreihe wurde eine vertrauenswürdige Instanz genutzt, die einen Austausch von Reputationsinformationen zwischen den einzelnen Software-Agenten ermöglicht.

VI. Testreihe

Mehrere Testläufe zeigten, daß der unkooperativ handelnde Software-Agent seinen Kapitalbestand nicht ausweiten konnte. Des weiteren führte er wesentlich weniger Transaktionen als die kooperativen Software-Agenten durch. Er wurde, wie in der 4. Testreihe, de facto vom Marktgeschehen ausgegrenzt. Die kooperativ handelnden Software-Agenten konnten ökonomisch erfolgreich interagieren und ihre Kapitalbestände im Zeitablauf ausweiten.

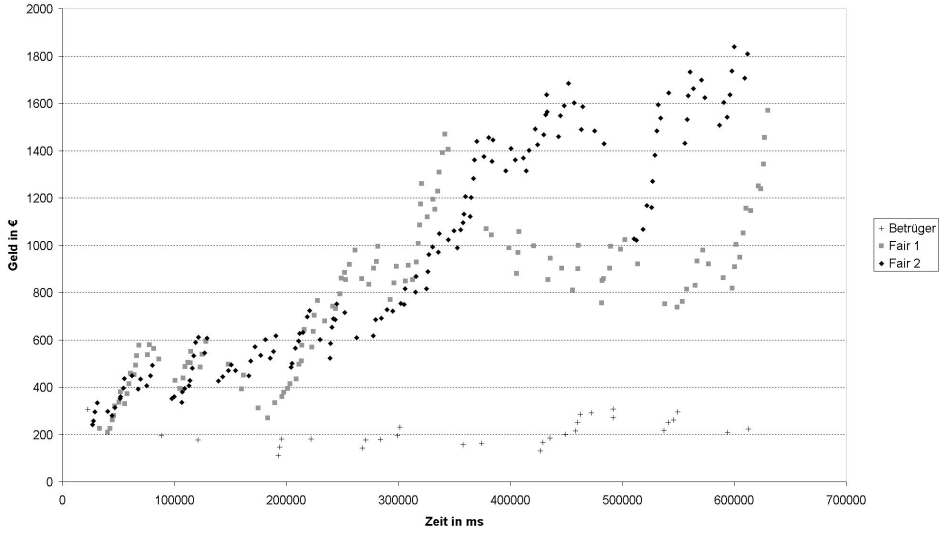


Abbildung 25: Entwicklung der Bargeldbestände bei Nutzung einer zentralen Reputationsinstanz (Ergebnis des Testlaufs 3 der VI. Testreihe)

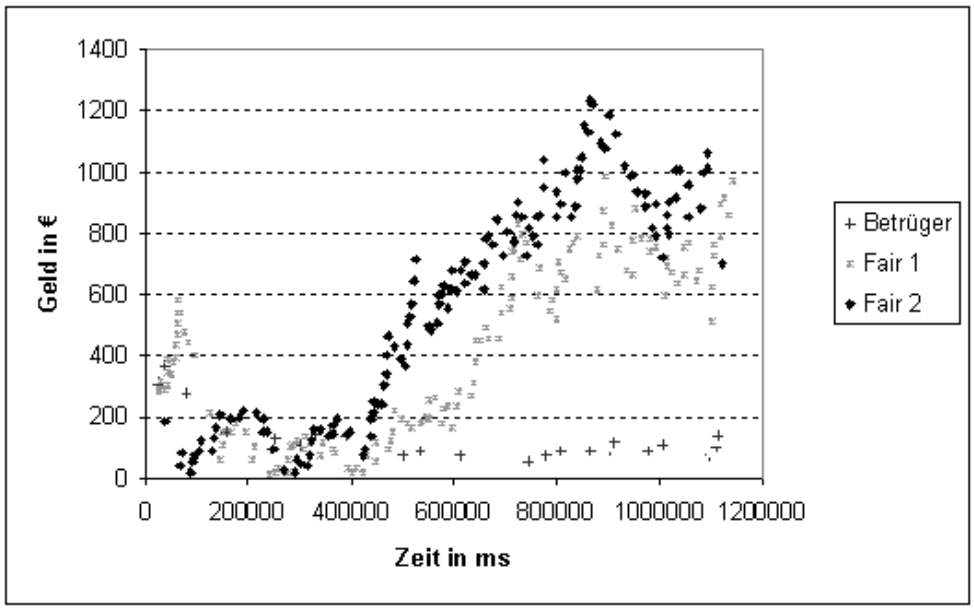


Abbildung 26: Entwicklung der Bargeldbestände bei Nutzung einer zentralen Reputationsinstanz (Ergebnis des Testlaufs 7 der VI. Testreihe)

5.3 Interpretation: Entstehung eines Vertrauensnetzes

Der Einsatz des Vertrauens- und Reputationsmechanismus führte zur Sanktionierung betrügerischer Software-Agenten. Diese fand durch Preisabschläge gegenüber betrügerischen Verkaufsagenten bzw. Preisaufschläge gegenüber betrügerischen Kaufagenten statt. Die Preisaufschläge lassen sich auch als Risikoprämien interpretieren. Durch die Berücksichtigung dieser Risikoprämien verloren die Kaufs- und Verkaufsangebote der betrügerischen Software-Agenten ihre Wettbewerbsfähigkeit. Daher führten die kooperativen Software-Agenten fast nur noch mit anderen kooperativen Software-Agenten Transaktionen durch und kaum noch mit betrügerischen. Somit wurden diese im Laufe der Zeit vom Handel ausgeschlossen. Es kam also zu einer impliziten Sanktionierung der betrügerischen Software-Agenten, obwohl explizit keine Sanktionsinstanz implementiert wurde.

Dies läßt sich wie folgt interpretieren: Die Sanktionierung erfolgt durch die rationalen Entscheidungen der ehrlichen Software-Agenten, Erwartungswerte zu berücksichtigen. Sie bilden untereinander ein Netzwerk des Vertrauens.[175] Zugangsvoraussetzung zum Eintritt in dieses Netzwerk ist ehrliches Verhalten, Betrüger werden ausgeschlossen. Dabei ist es unerheblich, in welcher Form die Software-Agenten Informationen über das Kooperationsverhalten ihrer Transaktionspartner erhalten. Bei der Nutzung einer zentralen Instanz, die vertrauenswürdig ist und auf deren Informationen alle Teilnehmer zugreifen können, bildet sich das Netz schneller, wie in Abschnitt 5.2 dargestellt, als wenn kein Informationsaustausch stattfindet. Wenn die Software-Agenten also nicht von den Erfahrungen anderer profitieren, muß jeder Software-Agent erst „seine eigenen Erfahrungen machen." Die Entstehung eines Vertrauensnetzwerks dauert dann zwar länger als im ersten Fall, aber letztendlich entsteht es trotzdem. In beiden Fällen lernen die Software-Agenten hinzu, entweder aus ihren eigenen Handlungen oder aus den Informationen, die sie über die Handlung anderer Software-Agenten von der zentralen Instanz erhalten.[176]

[175] Dies wurde auch von Schillo im *TrustNet* intendiert.

[176] Damit ist die Anforderung erfüllt, daß der Vertrauens- und Reputationsmechanismus ein „Hinzulernen" der Software-Agenten ermöglichen soll. (Vgl. Kapitel 3.4.8).

6 Schlußbetrachtung

Die Schaffung von Vertrauen zwischen Transaktionspartnern ist ein wesentlicher Faktor für den Erfolg von ökonomischen Transaktionen in offenen Systemen. Diese Vertrauensverhältnisse lassen sich jedoch nur aufbauen, wenn durch den Einsatz von kryptographischen Verfahren und digitalen Signaturen die Privatheit der Kommunikation und die Zurechenbarkeit der Transaktionen auf die Transaktionspartner gewährleistet ist.

Diese beiden Voraussetzungen reichen jedoch nicht aus, um betrügerisches Verhalten einzelner Transaktionspartner zu verhindern. Durch fehlende zentrale Sanktionsinstanzen in offenen Systemen bestehen kaum Möglichkeiten, betrügerisches Verhalten zu bestrafen. Daher ist ein dezentraler Mechanismus notwendig, der zu einer Sanktionierung betrügerischer Marktteilnehmer führt.

In der vorliegenden Arbeit wurde ein derartiger Mechanismus prototypisch für ein Multi-Agenten System konzipiert. Dabei wurde in einem ersten Schritt seine Notwendigkeit durch die Simulation betrügerischen Verhaltens einzelner Software-Agenten motiviert. Da der Mechanismus auf Vertrauen und Reputation beruht, wurden diese Begriffe aus soziologischer Sicht betrachtet und für beide Begriffe zusammenfassende Definitionen erarbeitet, die der Arbeit zugrunde gelegt wuden. Aus soziologischen Simulationen, für die ebenfalls Multi-Agenten Systeme genutzt wurden, konnten Lehren für die Formalisierung von Vertrauen im prototypischen Vertrauens- und Reputationsmechanismus gezogen werden.

Der konzipierte Vertrauens- und Reputationsmechanismus wurde im Multi-Agenten System AVALANCHE implementiert. Dabei war es nützlich, daß in AVALANCHE ausschließlich Software-Agenten genutzt wurden, die lediglich über ein sehr einfaches Modell ihrer Umwelt verfügen. Somit konnte bspw. eine Manipulation der Ergebnisse des Mechanismus durch die AVALANCHE-Agenten ausgeschlossen werden.

Dieser Mechanismus beruht auf fünf Funktionen und berücksichtigt sowohl den Bekanntheitsgrad der potentiellen agentenbasierten Transaktionspartner als auch eine zentrale und dezentrale Verarbeitung der Daten.

Zum Schluß der Arbeit wurde der Vertrauens- und Reputationsmechanismus durch mehrere Testläufe in AVALANCHE evaluiert. Diese Evaluation zeigte, daß eine inhärente Sanktionierung der betrügerischen Software-Agenten erfolgt war. Sie konnten aus ihrem Verhalten keine langfristigen ökonomischen Vorteile erzielen und erhielten so einen Anreiz, sich kooperativ zu verhalten.

Dieses Ergebnis ist in dreifacher Hinsicht ermutigend. Zum einem bestätigt es die allgemeinen spieltheoretischen Ergebnisse des *Iterated Prisoners Dilemma* nach Axelrod, der dargestellt hat, daß langfristige Kooperation zu besseren Ergebnissen führt als kurzfristige Defektion.

Zum anderen verfügt der Prototyp AVALANCHE nun über einen Mechanismus, der Sicherheit in einem nicht-technischen Bereich ermöglicht, was in bisherigen Multi-Agenten Systemen in dieser Form noch nicht gegeben ist.

Der Marktplatzprototyp AVALANCHE und die auf ihm interagierenden Software-Agenten sind aus Modellgründen in ihrer Komplexität noch sehr stark reduziert. Daher müßte nun in einem nächsten Schritt der Vertrauens- und Reputationsmechanismus dahingehend erweitert werden, daß er auch die Interaktionen komplexerer Software-Agenten unterstützt.

Außerdem wird deutlich, daß das Fehlen zentraler Sanktionierungsmechanismen im Internet durch dezentrale Mechanismen, zumindest teilweise, ausgeglichen werden kann. Dies bedeutet aber auch, daß im dezentralen und offenen Internet Sicherheit und Vertrauen nicht mehr von zentralen, in der realen Welt oft staatlichen oder gesellschaftlichen Instanzen geschaffen wird. Die Entstehung von Sicherheit und Vertrauen muß von den einzelnen Nutzern ausgehen.

Der in der vorliegenden Arbeit entwickelte Vertrauens- und Reputationsmechanismus leistet einen Beitrag dazu, daß ad-hoc Kooperationen zwischen unbekannten Transaktionspartnern sicherer werden und somit die häufigen Bedenken gegenüber Handel im Internet weiter schwinden.

7 Literatur

Abdul-Rahman, A. / Halles, S. (1997): *A Distributed Trust Model.* In: *Proceedings of the Workshop on New Security Paradigms.* ACM 1997. S. 48-60. 1997.

Avalanche (2000): *Agent-Based Value Chain Experiment - Agenten-basierte Koordination von Wertschöpfungsketten.* http://www.iig.uni-freiburg.de/telematik/forschung/projekte/e_sicherheit/avalanche/index.htm

Axelrod, R. (1984): *The Evolution of Cooperation.* New York: Basic Books, 1984.

Bachmann, R (1998): *Kooperation, Vertrauen und Macht in Systemen Verteilter Künstlicher Intelligenz. Eine Vorstudie zum Verhältnis von soziologischer Theorie und technischer Modellierung.* In Malsch, T. (Hrsg.): *Sozionik.* Berlin: edition sigma 1998. S. 197-234.

Beer, M. / d'Inverno, M. / Luck, M. / Jennings, N. / Preist, C. / Schroeder, M. (1998): *Negotiation in Multi-Agent Systems*. Workshop of the UK Special Interest Group on Multi-Agent Systems (UKMAS'98) 1998.

Bradshaw, J. (Ed.) (1997): *Software Agents*. Menlo Park: MIT Press, 1997.

Castelfranchi, C. / Tan, Y. / Falcone, R. / Firozabadi, B. (Eds.) (1999). In. *Proceedings of the Workshop „Deception, Fraud and Trust in Agent Societies“* . Autonomous Agents Conference, 1999.

Castelfranchi, C. / de Rosis, F. / Falcone, R. (1997): *Social Attitudes and Personalities in Agents.* In: Dautenhahn, K. (Hrsg.) (1997): *Socially Intelligent Agents.* Paper from the 1997 AAAI Fall Symposium. November 8-10. Camebrigde, Mass. Technical Report FS-97-02. 1997.

Castelfranchi, C. / Falcone, R. (1998): *Principles of Trust for MAS: Cognitive Anatomy, Social Importance, and Quantification.* In: *Proceedings of the International Conference on Multi Agent Systems (ICMAS'98).* Paris, France. July 4 - 7, 1998.

Castelfranchi, C. / Falcone, R. (1999): *The Dynamics of Trust: From Beliefs to Action.* In: Castelfranchi, C. / Tan, Y. / Falcone, R. (Eds.) (1999): *Proceedings of the Workshop of Deception, Fraud and Trust in Agent Societies at the Autonomous Agents Conference 1999.* Seattle 1999. National Research Council. Rome, Itlay. 1999.

Castelfranchi, C. / Tan, Y. / Falcone, R. (Eds.) (1999): *Deception, Fraud and Trust in Agent Societies.* In: *Proceedings of the Workshop at the Autonomous Agents Conference 1999.* Seattle 1999. National Research Council. Rome, Italy. 1999.

Chavez A. / Maes, P. (1996): *Kasbah - An Agent Marketplace for Buying and Selling Goods.* In: *Proceedings of the First Conference on Practical Applications of Agent Mechanisms.* London 1996.

Coord (2000): *Coordination Methods for Open Distributed Systems.* Swedish Institute of Computer Science. 2000. http://www.sics.se/isl/coord/

Dasgupta, P. (1988): *Trust as a Commodity*. In: Gambetta, D. (1988a): *Trust. Making and Breaking Cooperative Relations*. New York / Oxford. Basil Blackwell. 1988. S. 49-72.

Deutsch, M. (1973a): *The Resolution of Conflict.* Yale University Press; New Haven and London, 1973.

Deutsch, M. (1973b): *Trust and Suspicion: Theoretical Notes and Experimental Studies of Trust and Suspicion.* In: *The Resolution of Conflict. Constructive and Destructive Processes.* New Haven and London: Yale University Press 1973. S. 143-214.

Durfee, E. / Rosenschein, J. (1994): *Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples.* In: *Proceedings of the Thirteenth International Distributed Artificial Intelligence Workshop.* S. 94-104, July 1994.

Ebay (2000a): eBay Deutschland - Der weltweite Online-Marktplatz. 2000. http://www.ebay.de/.

Ebay (2000b): eBay Deutschland. Hilfe-Übersicht. 2000. http://pages.ebay.de/help/index.html.

Ebay (2000c): Ebay Bewertungs-Forum. 2000. http://pages.ebay.de/services/forum/feedback.html

Eggs, H. / Englert, J. (2000): *Electronic Commerce Enquête II - Business-to-Business Electronic Commerce*. Empirische Studie zum Business-to-Business Electronic Commerce im deutschsprachigen Raum, Executive Research Report. Stuttgart: Konradin-Verlag, 2000.

Epstein, J. / Axtell, R. (1996): *Growing Artificial Societies: Social Science from the Bottom Up.* MIT Press/Brookings, Cambridge, Mass. 1996.

Eriksson, J. / Finne, N. / Janson, S. (1996): *MarketSpace '96 - An Open Agent-Based Market Infrastructure.* In: Janson, S. (Hrsg.): *UPMAIL Technical Report*. No. 147.

Eymann, T. (2000): *Avalanche – Ein agentenbasierter dezentraler Koordinationsmechanismus für elektronische Märkte.* Dissertation. Freiburg im Breisgau. 2000.

Eymann, T. / Padovan, B. / Schoder, D. (1998a): *Artificial Coordination - Simulating Organizational Change with Artificial Life Agents.* In: *Proceedings of the IFAC Symposium on Computation in Economics, Finance, and Engineering: Economic Systems (CEFES'98).* Cambridge, UK, June 29-July 1, 1998. http://www.iig.uni-freiburg.de/~eymann/ research/cefes98.pdf

Eymann, T. / Padovan, B. / Schoder, D. (1998b): *The Living Value Chain - Coordinating Business Processes with Artificial Life Agents.* In: *Proceedings of the Third International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAM'98).* March 23-25, London 1998. The Practical Application Company: Blackpool, UK, 1998, S. 111-122.

Eymann, T. / Padovan, B. / Schoder, D. (1998c): *Simulating a Value Chain with Artificial Life Agents.* In: *Proceedings of the 3rd International Conference on Multi-Agent Systems, (ICMAS'98).* Paris. July 2-8, 1998.

Eymann, T. / Padovan, B. / Schoder, D. (1998d): *Avalanche - An Agent Based Value Chain Coordination Experiment.* In: *Proceedings of the Workshop on Artificial Societies and Computational Markets (ASCMA'98) at Autonomous Agents '98.* Minneapolis, USA. May 9th-14th, 1998. S. 48-53.

Eymann, T. / Padovan, B (1999): *Eine Multi-Agenten-Simulation zur ökonomischen Analyse der dezentralen Koordination von Wertschöpfungsketten.* In: Scheer, A.-W., Nüttgens, M. (Hrsg.): *Electronic Business Engineering, Konferenzband zur 4. Internationalen Tagung Wirtschaftsinformatik.* Physica Verlag, Heidelberg 1999, S. 625-641.

Eymann, T. / Padovan, B. (2000): *Avalanche. Agent-Based Value Chain Coordination Experiment. Einführung und Programmdokumentation.* White Paper. IIG-Telematik. http://www.iig.uni-freiburg.de/telematik/forschung/projekte/e_sicherheit/avalanche/downloads/dokumentation.pdf.

Eymann, T. / Padovan, B. / Schoder, D. (2000): *The Catallaxy as a new Paradigm for the Design of Information Systems.* In: *Proceeding of The World Computer Congress 2000 of the International Federation for Information Processing (IFIP).* Beijing, China, August 21-25, 2000.

FIPA (1998): *Foundation for Intelligent Physical Agents.* FIPA 98 Specification, Part 10, Version 1.0, Agent Security Management, Genf, 1998.

Foner, L. (1993): *What's An Agent, Anyway? A Sociological Case Study. Manuskript.* Cambridge: MIT Media Lab 1993. http://foner.www.media.mit.edu/people/foner/Julia/.

Foner, L. (1999): *Political Artifacts and Personal Privacy: The Yenta Multi-Agent Distributed Matchmaking System.* Dissertation 1999. Cambridge, USA. http://foner.www.media.mit.edu/ people/foner/PhD-Thesis /Dissertation/.

Franklin, S. / Graesser, A. (1996): *Is It an Agent or Just a Program? A Taxonomy for Autonomous Agents.* In: *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages.* New York: Springer 1996.

Gambetta, D. (1988a): *Trust. Making and Breaking Cooperative Relations*. New York / Oxford. Basil Blackwell. 1988.

Gambetta, D. (1988b): *Can We Trust Trust*. In: Gambetta, D. (1988a): *Trust. Making and Breaking Cooperative Relations*. New York / Oxford. Basil Blackwell. 1988. S. 213-237.

Gilbert, D. / Aparicio, M. / Atkinson, B. / Brady, S. / Giccarino, J. / Grosof B. / O'Connor, P. / Osisek, D. / Pritko, S. / Spagna, R. / Wilson, L. (1995): *IBM Intelligent Agent Strategy.* UNICOM Seminar on Agent Software. IBM Corporation 1995.

Good, D. (1988): *Individuals, Interpersonal Relations and Trust.* In: Gambetta, D. (Ed.) (1998): T*rust. Making and Breaking Cooperative Relations.* New York / Oxford. Basil Blackwell 1988. S. 31-48.

Holland, J. / Miller, J. (1991): *Artificial Adaptive Agents in Economic Theory.* In: *American Economic Review, Papers and Proceedings.* Vol. 81, No. 2, 1991, S. 365-370.

Huberman, B. / Lukose, R. (1997): *Social Dilemmas and Internet Congestion.* In: *Science*. Vol. 277, S. 535-537. 1997.

Jennings, N. / Sycara, K. / Wooldridge, M. (1998): *A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems.* Journal, No. 1, 1998, S. 275– 306, Kluwer Academic Publishers. ftp://ftp.elec.qmw.ac.uk/pub/isag/distributed-ai/publications/ roadmap.pdf.

Jennings, N. / Faratin P. / Johnson, M. / O'Brien, P. / Wiegand, M. (1996): *Using Intelligent Agents to Manage Business Processes.* 1996.

Jones, S. / Marsh, S. (1997): *Human-Computer-Human Interaction: Trust in CSCW.* In: *SIGCHI Bulletin.* Vol. 29, No 3, July 1997. http://www.cwi.nl/~steven/sigchi/bulletin/1997.3/jones.html.

Junge, K (1998): *Vertrauen und die Grundlagen der Sozialtheorie. Ein Kommentar zu James S. Coleman.* In: Müller, H. / Schmid, M. (Hrsg.): *Norm, Herrschaft und Vertrauen: Beiträge zu James S. Colemans Grundlagen der Sozialtheorie.* Opladen: Westdeutscher Verlag 1998, S.26-63.

Kephart, J. / Hanson, J. / Levine, D. / Grosof, B./ Sairamesh, J. / Segal, R. / White, S. (1998): *Dynamics of an information-filtering economy.* In: Klusch, M., Weiß, G. (Ed.): *Cooperative Information Agents II.* LNAI No. 1435. Heidelberg: Springer, 1998. http://www.ibm.com/iac/papers/cia98/cia98_public.html.

Kephart, J. / Hanson, J. / Greenwald, A. (2000): *Dynamic Pricing by Software Agents. Computer Networks.* 2000 (to appear). http://www.research.ibm.com/infoecon/paps/ html/rudin/rudin.html.

Koller, M. (1990): *Sozialpsychologie des Vertrauens: Ein Überblick über theoretische Ansätze*. Bielefelder Arbeiten zur Sozialpsychologie. 1990.

Koller, M. (1997): *Psychologie interpersonalen Vertrauens. Eine Einführung in theoretische Ansätze*. In: Schweer, M. (Hrsg.): *Interpersonales Vertrauen*. Opladen 1997.

Lane, D. (1993).: *Artificial worlds and economics (Part II).* In: *Journal of Evolutionary Economics.* p.177-197. Springer, 1993.

Luhmann, N. (1989): *Vertrauen. Ein Mechanismus zur Reduktion sozialer Komplexität.* Enke, Stuttgart, 1989.

Maes, P. (1994a): *Agents that reduce work and information overload.* Communications of the ACM, Vol. 37 (1994), No. 7, S. 30-40.

Maes, P. (1994b): *Modelling Adaptive Autonomous Agents.* In: Langton, C. (Ed.): *Artificial Life Journal.* Vol. 1, No. 1 & 2. S. 135-162. Camebrigde, MIT Press, 1994.

Malone, T. (1987): *Modelling Coordination in Organizations and Markets.* In: *Management Science 33*. S. 1317-1332.

Malone, T. / Crowston, K. (1990): *Toward an Interdisciplinary Theory of Coordination, Technical Report 120.* Center for Coordination Science, MIT, Cambridge MA, 1990.

Marsh, S (1994a): *Formalising Trust as a Computational Concept.* PhD-Thesis, Dept. of Computing Science and Mathematics, University of Stirling 1994. http://ai.iit.nrc.ca/~steve/Publications.html.

Marsh, S. (1994b): *Optimism and pessimism in trust.* White-Paper und Tagungsbeitrag. 1994. http://ai.iit.nrc.ca/~steve/Publications.html.

Marsh, S. (1994c): *Trust in Distributed Artificial Intelligence.* In: Castelfranchi and Werner (Eds).: *Artificial Social Societies.* Springer Verlag, Lecture Notes in AI 830, 1994. S. 94-112.

Misztal, B. (1996): *Trust in Modern Societies.* Cambridge: Polity Press, 1996.

Moukas, A. / Zacharia, G. / Maess, P. (1999): *Amalthaea and Histos: Multi-Agent Systems for WWW Sites and Reputation Recommendations.* In: Klusch, M. (Ed.): *Intelligent information agents: agent-based information discovery and management on the Internet.* Springer, Heidelberg, 1999.

Müller, G. / Kohl, U. / Schoder, D. (1997): *Unternehmenskommunikation: Telematiksysteme für vernetzte Unternehmen.* Bonn, 1997.

Müller, G. / Pfitzmann, A. (Hrsg.) (1997): *Mehrseitige Sicherheit in der Kommunikationstechnik: Verfahren, Komponenten, Integration.* Band 1, Addison-Wesley-Longman: München u.a., 1997.

Müller, G. / Rannenberg, K. (Eds.) (1999): *Multilateral Security in Communications.* Addison Wesley Longman, München 1999.

Müller, J. (1996): *Diversifikation und Reputation. Transferprozesse und Wettbewerbswirkungen.* Wiesbaden: Dt. Univ.-Verl. 1996.

Nwana, H. (1996): *Software Agents: An Overview.* In: *Knowledge Engineering Review.* 11(3), p. 24-29, 1996.

ObjectSpace (1999): *How Voyager simultaneously supports CORBA, RMI, and DCOM.* White Paper, http://www.objectspace.com/products/ documentation/VgerUnivOrb.pdf, 1999.

Padovan, B. / Müller, G. / Eymann, T. (1999): *Growing Intelligent Information Systems from the Bottom Up - The Role of Autonomous Agents and Multi-Agent Systems.* Invited paper. In: *Proceedings of the Conference on Intelligent and Information Systems.* 22.-24. September 1999, Varazdin, Kroatien. Invited lecture to the Conference on Intelligent and Information Systems, 22.-24. September 1999, Varazdin, Kroatien.

Padovan, B. (1999): *Potentiale, Hürden und Entwicklungen des Electronic Commerce.* In: *Electronic Commerce – Herausforderungen und Chancen für Baden-Württemberg.* Akademie für Technikfolgenabschätzung, Stuttgart 2000. ISBN 3-932013-97-2.

Padovan, B. / Eymann, T. / Sackmann, S. (2000a): *AVALANCHE - A prototypical agent based secure electronic commerce platform.* In: *Technical Papers of the Fourth International Conference on AUTONOMOUS AGENTS (Agents 2000).* Barcelona, Spain, June 3 - June 7, 2000.

Padovan, B. / Sackmann, S. / Eymann, T. (2000b): *Secure Electronic Marketplaces based on the Multi Agent System AVALANCHE.* In: *Proceedings of the 11th International Conference on Information and Intelligent Systems (IIS 2000).* September 20th - 22nd, Varazdin, Croatia. 2000.

Padovan, B. / Sackmann, S. / Eymann, T. / Pippow, I. (2000c): *A Prototype for an Agent-based Secure Electronic Marketplace including Reputation Tracking Mechanisms.* Accepted Paper to the Thirty-Fourth Hawaii International Conference on System Sciences (HICSS-34). January 3-6, 2001. Maui (USA)

Porter, M. / Millar, V. (1985): *How information gives you competitive advantage.* In: *Harvard Business Review.* July-August , S.149-160. 1985.

Preisendörfer, P. (1995): *Vertrauen als soziologische Kategorie. Möglichkeiten und Grenzen einer entscheidungstheoretischen Fundierung des Vertrauenskonzeptes.* In: *Zeitschrift für Soziologie.* 4 (24). 4. August 1995. S. 263-272. 1995.

Preist, C. (1998): *Economic Agents for Automated Trading.* HP Technical Reports HPL-98-77. Hewlett Packard Laboratories, Bristol, 1998.

Pruitt, D. (1981): *Negotiation Behavior.* New York: Academic Press, 1981.

Rannenberg, R. / Pfitzmann, A. / Müller, G. (1999): *IT Security and Multilateral Security.* In: Müller, G. / Rannenberg, K. (Eds.) (1999): *Multilateral Security in Communications.* Addison Wesley Longman, München 1999. S. 21-29.

Rasmusson, L. (1996): *Socially Controlled Global Agent Systems.* Royal Institute of Technology, Dept. Of Computer and System Science. Stockholm. 1996. http://www.sics.se.

Rasmusson, L. / Janson, S. (1996): *Simulated Social Control for Secure Internet Commerce.* In: *New Security Paradigms '96.* ACM Press. 1996. http://www.sics.se/~lra/nsp96/nsp96.html

Rasmusson, L. / Rasmusson, A. / Janson, S. (1997): *Using Agents to Secure the Internet Marketplace. Reactive Security and Social Control.* 1997. http://www.sics.se/isl/coord/.

Raub, W. / Weesie, J. (1991): *Reputation and Efficiency in Social Interactions. An Example of Network Effects.* In: Esser, H. / Troitzsch, K. (Hrsg.): *Modellierung sozialer Prozesse.* Bonn: Informationszentrum Sozialwissenschaften 1991, S. 735-772.

Reagle, J. (1996): *Trust in Electronic Markets. The Convergence of Cryptographers and Economists.* In: First Monday. http://www.firstmonday.dk/issues/issue2/markets/. 1996.

Sandholm, T. (1996): *Negotiation among self-interested computationally limited agents.* Dissertation, Amherst 1996.

Swith, R. (1967): *The establishement of the trust relationsship*. Journal of Conflict Resolution. 11(3). S. 335-344. 1967.

Ripperger, T. (1998): *Ökonomik des Vertrauens: Analyse eines Organisationsprinzips.* Tübingen: Mohr Siebeck 1998.

Schillo, M. (1999): *Vertrauen und Betrug in Multi-Agenten-Systemen. Erweiterung des Vertrauensmodells von Castelfranchi und Falcone um eine Kommunikationskomponente.* Universität der Saarlandes. 1999.

Schillo, M. / Funk, P. / Rovatsos. M. (1999): *Who can you Trust: Dealing with Deception.* In: Castelfranchi, C. / Tan, Y. / Falcone, R. / Firozabadi, B. (Eds.) (1999). *Proceedings of the Workshop „Deception, Fraud and Trust in Agent Societies" of the Autonomous Agents Conference.* Seattle, 1999. http://www.virtosphere.de/data/publications/workshops/1999/Agents99_Schillo.ps.

Schmid, B. / Lindemann, M. (1998): *Elements of a Reference Model for Electronic Markets.* In: *Proceedings of the 31st Annual Hawaii International Conference on System Sciences (HICSS).* IV; January 1998.

Schoder, D. / Hummel, T. / Müller, G. (1997): *Interdisziplinäre Modelle für Entwurf und Einsatz telematischer Systeme.* In: Jarke, M. (Hrsg.): *Informatik'97.* Heidelberg: Springer, 1997.

Schoder, D. / Müller, G. (1999): *Potentiale und Hürden des Electronic Commerce. Eine Momentaufnahme.* In: *Informatik Spektrum.* Band 22, Heft 4, August 1999, S. 252-260.

Schoder, D. / Strauß, R. / Welchering, P. (1998): *Electronic Commerce Enquete 1997/98, Empirische Studie zum betriebswirtschaftlichen Nutzen von Electronic Commerce für Unternehmen im deutschsprachigen Raum.* Executive Research Report. Stuttgart: Konradin-Verlag, 1998.

Shoham, Y. (1997): *An Overview of Agent-oriented Programming.* In: Bradshaw, J.M. (Ed.): *Software Agents.* Menlo Park, CA: AAAI Press, 1997, S. 271-290.

Smith, R. (1980): *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver.* In: *IEEE Transactions on Computers.* Vol. 29, 1980, p. 1104-1113.

Sztompka, P. (1999): *Trust. A Socological Theory.* Cambridge: University Press, 1999.

Telos (1998): *TELOS: Aufgabenstellung und Motivation.* http://www.iig.uni-freiburg.de/telematik/forschung/ab_projekte/e_sicherheit/telos/index.html

Turner, J. (1991): *The Structure of Sociological Theory. Homewood, Ill*. The Dorsey series in sociology. Dorsey, 1974. – XII. 1991.

Varian, H. (1991): *Grundzüge der Mikroökonomik.* Oldenbourg, München. 1991.

Vidal, J. / Durfee, E. (1996): *Building Agent Models in Economic Societies of Agents.* AAAI-96 Workshop on Agent Modeling, Portland, Oregon, July 1996. ftp:// ftp.eecs.umich.edu/people/durfee/aaai96ws-am-vd.ps.Z.

Weise, P. / Brandes, W. / Eger, T. / Kraft, M. (1991): *Neue Mikroökonomie.* Heidelberg. Physica-Verlag. 1991.

Wellman, M. (1996): *Market-Oriented Programming: Some Early Lessons.* In: Clearwater (Ed.): *Market-Based Control: A Paradigm for Distributed Resource Allocation.* World Scientific. 1996. ftp://ftp.eecs.umich.edu/people/wellman/mbc95.ps.Z

Westermayer, T. (2000): *Trust in Bots: Zur Rolle von Vertrauen und Reputation bei Multiagenten-Systemen.* IIG-Telematik 2000, White Paper.

White, J. (1997): *Mobile Agents.* White Paper. http:// www. genmagic.com/ agents/ Whitepaper/ whitepaper.html. 1997.

Winter, M. (1999): *The Role of Trust and Security Mechanisms in an Agent-Based Peer Help System.* In: Castelfranchi, C. / Tan, Y. / Falcone, R. (Eds.) (1999): *Deception, Fraud and Trust in Agent Societies.* Seattle 1999. S. 139-148.

Wooldridge, M. (1999): *Intelligent Agents.* In Weiss, G. (Ed.): *Multiagent Systems.* MIT Press, 1999. http://www.csc.liv.ac.uk/~mjw/pubs/mas99.ps.gz.

Yu, B. /Singh, M. (2000): *A Social Mechanism of Reputation Management in Electronic Communities.* In: *Proceedings of the 4th International Workshop on Cooperative Information Agents (CIA).* Berlin: Springer 2000. To appear.

Zacharia, G. (1999): *Trust management through reputation mechanisms.* In: Castelfranchi, C. / Tan, Y. / Falcone, R. (Eds.) (1999): *Deception, Fraud and Trust in Agent Societies.* Seattle 1999. S. 163-167.

Zacharia, G. / Moukas, A. / Maes, P. (1999): *Collaborative Reputation Mechanisms in Electronic Marketplaces.* In: *Proceedings of the 32nd Hawaii International Conference on System Sciences*. 1999.

8 Anhang

AVALANCHE: Technische Dokumentation

Verzeichnis der Initialisierungen der Testreihen

AVALANCHE

Agent-Based Value Chain Coordination Experiment

Einführung und Programmdokumentation

Version 0.9

30.10.2000

Autoren:

Dipl.-Volkswirt Boris Padovan
Dipl.-Wirtschaftsinformatiker Torsten Eymann
Institut für Informatik und Gesellschaft, Abt. Telematik
Albert-Ludwigs-Universität Freiburg, 79085 Freiburg

Projekt URL:

http://www.iig.uni-freiburg.de/telematik/forschung/projekte/e_sicherheit/avalanche/index.htm

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Tabellen

1 Übersicht

1.1 Einführung und Motivation

Das Agent-Based Value Chain Experiment (AVALANCHE) ist ein interdisziplinäres Projekt zur marktähnlichen Koordination einer Wertschöpfungskette durch ein MAS (Multi-Agenten System). Es wird seit 1997 an der Universität Freiburg entwickelt. Hierbei wurde ein lauffähiges technisches MAS erstellt. Das Projekt AVALANCHE wurde in mehreren, auch internationalen Publikationen vorgestellt.

Die modellhafte Wertschöpfungskette besteht aus kleinen Unternehmen, die auf verschiedenen vernetzten Marktplätzen miteinander handeln. Die einzelnen Unternehmen werden durch Software-Agenten repräsentiert, die als wichtigen Parameter eine individuelle Bereitwilligkeit zur Kooperation mit anderen Software-Agenten besitzen, was für den Erfolg und das Überleben des Software-Agenten von entscheidender Bedeutung ist. Da es sich um ein offenes System handelt, können während der Laufzeit der Simulation neue Software-Agenten in Marktplätze eintreten oder diese verlassen. Die Anzahl der Software-Agenten ist technisch nicht limitiert, bisherige Läufe wurden mit durchschnittlich 250 Software-Agenten durchgeführt. Bisher wurden organisationstheoretische Fragestellungen behandelt, z.B. inwieweit die Variation von Parametern wie Transaktionskosten, Konnektivität oder Markttransparenz (z.B. durch den Einsatz von Informationstechnologie) zu unterschiedlichen organisatorischen Strukturen führt.

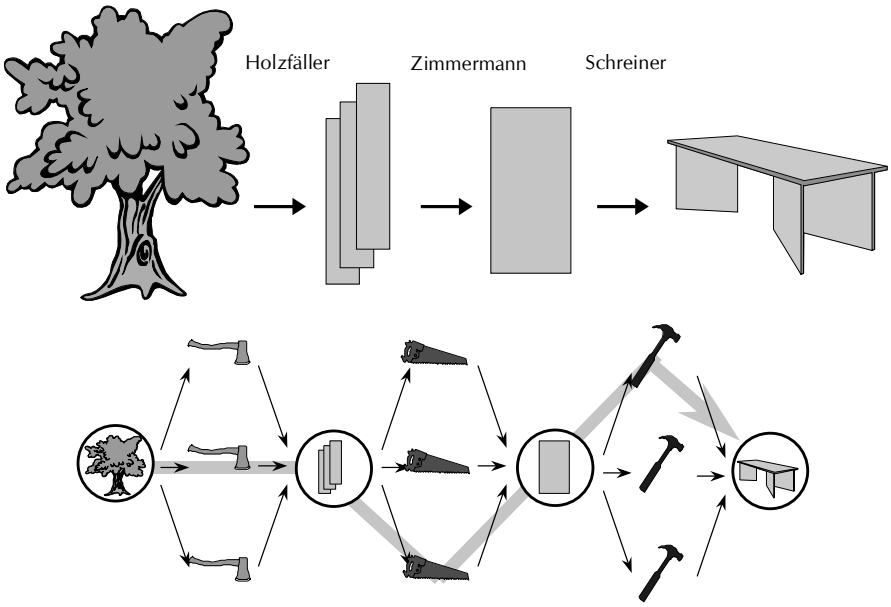


Abbildung 1: Die Wertschöpfungskette von AVALANCHE

Die grundlegenden technischen Komponenten von AVALANCHE sind in der Programmiersprache Java realisiert. Um die Gesamtkomplexität des Projektes gering zu halten, wurde bereits bei der Konzeption darauf geachtet, wo möglich standardisierte Konzepte einzusetzen. So werden die Software-Agenten mittels einer existierender Klassenbibliothek für mobile Software-Agenten auf CORBA-Basis implementiert. Lokationen/Marktplätze sind Java Virtual Machines, die als Serverprozesse auf einem vorgegebenen Rechner in einem TCP/IP-basierten lokalen Netz laufen, und durch eine IP-Port-Adresse identifizierbar sind. Die Lokationen stellen für die Software-Agenten eine „Sandbox"-Umgebung zur Interaktion bereit, ohne jedoch in die ökonomische Koordination einzugreifen. Die Software-Agenten kommunizieren bilateral mittels Message Passing. Die Kommunikationssyntax und -semantik, wie auch das Verhandlungsprotokoll implementiert die „Agent Communication Language" und die darin enthaltenen Protokolle für Auktionen. Die Software-Agenten

laufen gleichzeitig und parallel, um Auswirkungen von sequentiellen Scheduling-Algorithmen auf die Interpretationsmöglichkeiten zu vermeiden.

1.2 Änderungen von vorangegangenen Versionen

Version 0.9:	Erweiterung um den Vertrauens- und Reputationsmechanismus
Version 0.8:	Ersetzen der unterschiedlichen Namensverwaltungen in Infoserver, Location und TradeAgents durch Voyager's Federated Directory Services, damit fallen auch einige Protokolle weg
Version 0.7:	Einführen des –pull Parameters, mit dem vor Kauf und Fertigung geprüft wird, ob eine Nachfrage vorhanden ist (Nachfragegesteuerte Supply Chain).
Version 0.7:	Organisieren der Avalanche-Quellcodes in Packages und Ausgabe in einem .jar –File
Version 0.7:	Ersetzen der Variable „ontology“ in den FIPA-Protokollen durch das semantisch treffendere „protocoL“
Version 0.6:	Erweiterung der Parameter –g und –var von 5 auf 6 Gene (plus p_reputation bzw. var_reputation)
Version 0.6:	Zusammenführung von filename.prices, filename.profit, filename.cash und filename.trades zu einem einzigen Transaktionslog filename.xls
Version 0.5:	Erste Version

Tabelle 1: Versionen von AVALANCHE

2 Installation der Simulationsumgebung

AVALANCHE wurde bisher auf Windows NT entwickelt und unter Windows 95/98/NT erprobt. Erfahrungen im Einsatz auf anderen Betriebssystemen (e.g. Linux, Solaris) liegen bisher nicht vor. Die Autoren würden sich über derartige Informationen sehr freuen.

2.1 Verzeichnisstruktur

Für Windows hat sich folgende relative Verzeichnisstruktur als praktisch erwiesen:

```
C:\   (die Festplatte, auf der Java, Voyager und AVALANCHE
      installiert sind)
p\          (ein Verzeichnisbaum für Programmierung)
      jdk\        (ein versionsunabhängiges Verzeichnis für
                  Java)
            voy\            (ein versionsunabhängiges Ver-
                            zeichnis für Voyager)
            avalanche\      (ein Verzeichnis für die Sour-
                            cedateien von AVALANCHE)
            classes\        (ein Verzeichnis für die Klas-
                            sendateien)
            Docs\           (die von Javadoc erstellte
                            HTML-Dokumentation)
```

2.2 Installation von Java

AVALANCHE basiert zum gegenwärtigen Zeitpunkt auf Java 1.2, auf dessen Basis Collections und ein Statuspanel in Swing realisiert wurden. Frühere Versionen basierten auf Java 1.1.x, es wurde jedoch nicht überprüft, ob die gegenwärtige Version auf Java 1.1.7 mit den Swing-Bibliotheken läuft. Nach der Installation von Java sollte überprüft werden, ob die PATH und CLASSPATH - Variablen korrekt gesetzt sind.

2.3 Installation von Voyager

Im gegenwärtigen Zustand läuft AVALANCHE mit dem frei erhältlichen Voyager ORB (http://www.objectspace.com/products/vgrOverview.htm). Bei der Entwicklung wurde die Version 3.0 zugrunde gelegt. Die per Download erhältliche Version 3.1.1 ist einsetzbar, gibt aber Fehlermeldungen in Bezug auf fehlerhafte Serialisierung aus, deren Auswirkungen auf die Simulation bisher noch nicht untersucht wurden. Eine Setup-Datei mit der Version 3.0 ist daher bis zur nächsten Anpassung auf der AVALANCHE -Webseite erhältlich. Nach der Installation muss die CLASSPATH-Datei einen expliziten Link auf die Datei voyager.jar (im Verzeichnis voy\lib\) enthalten, wobei der Link auf das Verzeichnis alleine nicht ausreicht. Nach Installation von Java und Voyager sollten daher in den Variabeln PATH und CLASSPATH unter anderem folgende Einträge zu finden sein.

```
Path = .;c:\p\jdk\bin;c:\p\voy\bin
ClassPath = .;c:\p\jdk\lib;c:\p\voy\lib\voyager.jar
```

2.4 Installation der AVALANCHE-Klassendateien

AVALANCHE wird als Zip-Datei mit .java und .class-Dateien geliefert Die Zip-Datei sollte im Verzeichnis avalanche\ entpackt werden. Die mitgelieferten (mit CodeWarrior Pro5 und Java 1.2.2 unter Windows NT compilierten .class-Dateien) werden im Unterverzeichnis avalanche\classes\ automatisch entpackt. Da der Voyager-Server die Klassen zur Laufzeit finden muss, sollte im CLASSPATH ein Eintrag „.“ d.h. ein Link auf das zum Zeitpunkt des Programmaufrufs aktuelle Verzeichnis vorhanden sein.

Bei einer Recompilierung der Klassendateien, z.B. nachdem der Sourcecode geändert wurde, sollten die Voyager-Interfacedateien (z.B. IavAgent, IavTradeAgent,

IAvFactor) vor Aufruf des Compilers mit dem Voyager-eigenen Tool *igen* neu erstellt werden, z.B. durch Aufruf von „igen AvAgent“. Aus der Quelldatei AvAgent.java wird dadurch eine neue Version von IAvAgent.java generiert, einer Interfacedatei, welche für die remote-Aufrufe von Voyager unverzichtbar ist. Die Batchdatei *mi.bat* im Verzeichnis avalanche\ enthält Aufrufe von igen für alle relevanten Klassen.

2.5 Installation optionaler Tools

2.5.1 Cygwin32

Nach Installation des GNU-ähnlichen Tools Cygwin32 (http://sourceware.cygnus.com/cygwin/ bzw. ftp://ftp.franken.de/pub/win32/ develop/gnuwin32/cygwin/) lassen sich diverse Unix-Befehle zur Bearbeitung und Anzeige der Logdateien nutzen. Mit „tail -f *filename*.xls“ z.B. lassen sich während des Simulationslaufs die in die Datei *filename*.xls geschriebenen Kaufvorgänge dynamisch anzeigen. „grep“ erlaubt eine einfache Zusammenfassung, wenn nur bestimmte Software-Agenten verfolgt werden sollen.

2.5.2 Microsoft Excel

Die zentrale Logdatei, die beim Lauf des Programms erstellt wird, enthält durch Tabulatoren getrennte ASCII-Daten für eine spätere Auswertung. Für den einfachen Import in Excel wird diese Logdatei bereits mit der Dateiendung *.xls* erzeugt. Durch einfaches Markieren der Zeitspalte und z.B. aller Spalten mit Faktorpreisen lässt sich sehr schnell eine übersichtliche Punktgrafik erzeugen und darstellen.

3 Aufruf des Programms

3.1 Starten des Voyager Servers

Lokationen/Marktplätze sind Java Virtual Machines, die als Serverprozesse auf einem vorgegebenen Rechner in einem TCP/IP-basierten lokalen Netz laufen, und durch eine IP-Port-Adresse identifizierbar sind. Die Lokationen stellen für die Software-Agenten eine „Sandbox"-Umgebung zur Interaktion bereit, ohne jedoch in die ökonomische Koordination einzugreifen.

Zum Starten des Voyager Servers muss zuerst ein separates (MS/DOS-) Kommandozeilenfenster geöffnet und per „cd" in das Verzeichnis mit den AVALANCHE - Klassendateien gewechselt werden. Danach wird an der Kommandozeile „voyager 8000" eingegeben. Dadurch wird auf dem lokalen Rechner auf dem IP-Port localhost:8000 ein Voyager ORB Server gestartet. Dieser wird im nächsten Schritt die Software-Agenten aufnehmen. Nach dem Starten des ORB darf das Fenster bis zum Ende des Simulationslaufes nicht geschlossen werden.

Anmerkungen:

Eine Installation des Voyager ORB als z.B. Windows NT - Systemdienst ist möglich, wurde bisher aber nicht ausprobiert.

Besteht zu Monitoringzwecken der Bedarf, auch den ORB in einer Java-Umgebung zu starten, kann dies über „java VoyagerServer 8000" durchgeführt werden. Dieses Programm installiert ebenfalls den ORB auf der Adresse localhost:8000. Im Simulationsablauf ergeben sich jedoch keine Änderungen.

3.2 Starten von AVALANCHE

In einem zweiten, separat zu öffnenden Kommandozeilenfenster kann jetzt die eigentliche Multi-Agenten Umgebung gestartet werden. Das zentrale Javaprogramm ist *G1.class* (quasi der Agentengenerator), der über die Angabe von Kommandozeilenparametern gesteuert wird. Das Programm wird generell gestartet in der Form:

```
java G1 -p1 parameter_1 -p2 parameter_2 ... -pn parame-
ter_n
```

Beispiel:

```
java G1 -f a1 -g 0.1 0.26 0.10 0.8 0.2 -var 0 0 0.2 0 0
-mt 50 -ct 65 -t 240 -n 3 -summary a.sum -noreproduction
-col 0.02
```

Diese Einträge können direkt auf der Kommandozeile angegeben werden. Es ist auch möglich, mehrere solcher Aufrufe, die nacheinander ausgeführt werden sollen, in einer Batchdatei untereinander zusammenzufassen, z.B. um den Wertebereich des Parameters p_acquisitiveness durchzuprüfen, wie im folgenden Beispiel gezeigt:

```
java G1 -f a1 -g 0.1 0.26 0.10 0.8 0.2 1 -var 0 0 0 0 0 0 -mt 50 -ct 65 -t
240 -n 3 -summary a.sum -noreproduction -col 0.02

java G1 -f a1 -g 0.2 0.26 0.10 0.8 0.2 1 -var 0 0 0 0 0 0 -mt 50 -ct 65 -t
240 -n 3 -summary a.sum -noreproduction -col 0.02

java G1 -f a1 -g 0.3 0.26 0.10 0.8 0.2 1 -var 0 0 0 0 0 0 -mt 50 -ct 65 -t
240 -n 3 -summary a.sum -noreproduction -col 0.02

java G1 -f a1 -g 0.4 0.26 0.10 0.8 0.2 1 -var 0 0 0 0 0 0 -mt 50 -ct 65 -t
240 -n 3 -summary a.sum -noreproduction -col 0.02

java G1 -f a1 -g 0.5 0.26 0.10 0.8 0.2 1 -var 0 0 0 0 0 0 -mt 50 -ct 65 -t
240 -n 3 -summary a.sum -noreproduction -col 0.02

java G1 -f a1 -g 0.6 0.26 0.10 0.8 0.2 1 -var 0 0 0 0 0 0 -mt 50 -ct 65 -t
240 -n 3 -summary a.sum -noreproduction -col 0.02

java G1 -f a1 -g 0.7 0.26 0.10 0.8 0.2 1 -var 0 0 0 0 0 0 -mt 50 -ct 65 -t
240 -n 3 -summary a.sum -noreproduction -col 0.02

java G1 -f a1 -g 0.8 0.26 0.10 0.8 0.2 1 -var 0 0 0 0 0 0 -mt 50 -ct 65 -t
240 -n 3 -summary a.sum -noreproduction -col 0.02
```

Abbildung 2: Batchdatei “acq.bat” als Beispiel für den Aufruf von AVALANCHE

4 Startparameter von AVALANCHE

4.1 Kurzübersicht

Argument	Parameter	Beschreibung
-f	Filename	Prefix für Ausgabedateien.

-inifile	Filename	Erstellt die Agentenpopulation anhand der Datei filename.ini und setzt die Agentenparameter entsprechend.
-t	time	Laufzeit der Simulation in Sekunden.
-n	AgentNumber	Anzahl der Software-Agenten
-g	p_acquisitiveness del_change del_jump p_satisfaction w_memory p_reputation	Mittlere Startwerte aller 6 Gene
-var	p_acquisitiveness del_change del_jump p_satisfaction w_memory p_reputation	Varianzen der Startwerte aller 6 Gene (Default: 0 0 0 0 0 0)
-dist	{uniform|normal}	Switch zwischen Gleich- oder Normalverteilung der bei –var angegebenen Varianzen. (Default: uniform)
-col	CostsOfLife	Lebenshaltungskosten

		werden als Prozentanteil vom Bargeld des Software-Agenten abgezogen. (Default: 0.01)
-ct	courterThreshold	Minimalanzahl der erhaltenen Plumages, bevor der Software-Agent mit der Auswahl eines Vermehrungspartners beginnt.
-mt	maturityThreshold	Minimalanzahl der erfolgreich durchgeführten Transaktionen, bevor der Software-Agent mit Vermehrung beginnen kann.
-pt	profitThreshold	Minimalwert der Profitabilität eines Handelspartners, bevor dessen Plumages in die eigene Liste aufgenommen werden (Default: eigener Profit == elitistische Auswahl)

-trace	tracelevel	Detaillierungsgrad der Ausgaben in der Logdatei (1 = nur Zusammenfassungen, 2 += alle Events, 3 += alle Messages)
-summary	summaryfile	Während des Shutdown werden hier zentrale Auswertungsdaten der Lokationen geschrieben. (Default: keine Angabe).
-noreproduction		Ausschalten der Reproduktion von Software-Agenten (Default: keine Angabe).

Tabelle 2: Kommandozeilenparameter von G1

4.2 Ausgabedateien

`Syntax: -f filename`
`Beispiel: -f a1`

Dieses Argument bestimmt den grundsätzlichen Namen der Log- bzw. Ausgabedateien. Diese unterscheiden sich durch das dem Dateinamen angehängte Suffix. Die im folgenden beschriebenen Dateien werden gleichzeitig während eines Simulationslaufes ausgegeben.

Anmerkungen:

- Die in den Originaldateien 14-stelligen Fliesskommazahlen wurden zur besseren Darstellung hier auf 3 Nachkommastellen gekürzt.
- Der Agentenname setzt sich zusammen aus [Stammbezeichnung]_[ID]_ [Generationenbezeichner]. Die beim Start der Simulation vorhandenen Software-Agenten gehören zur Generation 0.

4.2.1 Geburt von Software-Agenten

In „*filename*.birth“ werden alle erzeugten Software-Agenten eingetragen mit ihren individuellen Gen-Einstellungen. In der ersten Spalte steht der Name der Software-Agenten, gefolgt von den Namen der beiden Elternagenten, dem Geburtszeitpunkt, und den 6 Genparametern. Software-Agenten der Ausgangspopulation haben „null“ als Elternagenten.

```
name  mother      father      time  p_acquisitiveness del_change
      del_jump    p_satisfaction    w_memory    p_reputation

Pro_10      null  null  15632 0.75  0.25  0.15  0.75  0.2   1.0

Lum_2_0     null  null  15632 0.75  0.25  0.15  0.75  0.2   1.0

Cab_8_0     null  null  15632 0.75  0.25  0.15  0.75  0.2   1.0

Car_5_0     null  null  15913 0.75  0.25  0.15  0.75  0.2   1.0
```

Abbildung 3: filename.birth

4.2.2 Tod von Software-Agenten

In „*filename*.death“ stehen analog zu *filename*.birth alle aus der Simulation entfernten Software-Agenten. In der ersten Spalte steht der Name der Software-Agenten, gefolgt von den 6 Genparametern. Bei Beendigung der Simulation wird in eine Zeile „shutdown“ geschrieben; alle Zeilen danach sind Software-Agenten, die nicht aufgrund ihres eigenen Handelns, sondern aufgrund der korrekten Beendigung der Simulation „sterben“ müssen, und bis zum Ende erfolgreich gehandelt haben.

Name	time	p_acq	d_chg	d_jmp	p_sat	w_mem	p_rep
Con_13	321372	0.3	0.25	0.15	0.75	0.2	1.0
Pro_12	321392	0.3	0.25	0.15	0.75	0.2	1.0
Lum_40_1	321402	0.3	0.25	0.15	0.75	0.2	1.0
Lum_38_2	321412	0.3	0.25	0.15	0.75	0.198	1.0
Car_19_1	321432	0.3	0.25	0.15	0.75	0.2	1.0
Car_41_3	321442	0.3	0.25	0.15			

Abbildung 4: filename.death

4.2.3 Transaktionen

„*filename*.xls“ ist trotz der Dateiendung eine reine ASCII-Datei. Zur schnellen Auswertung unter Microsoft Excel wurde jedoch das Suffix auf *.xls* gesetzt, so kann die Datei durch einfachen Doppelklick geöffnet und importiert werden.

```
date: 04.11.99, time: 21:15, args: -inifile good-bad.ini -t 36000 -f good-bad

nr      time    seller  tribe   type    buyer   location        factor  wood$   board$
        plate$  table$  pro$    lum$    car$    cab$    pro%    lum%    car%    cab%

1       39477   Pro_16_0        1       1       Lum_31_0        L0      1       8.5
                        106.5                           0.65

2       39587   Pro_17_0        1       1       Lum_32_0        L0      1
        7.7250000000000005                              105.725
        0.5725000000000001

3       40458   Pro_15_0        1       1       Lum_37_0        L0      1       10.0
                        108.0                           0.8

4       40529   Pro_11_0        1       1       Lum_31_0        L0      1       9.02875
                        106.02875                               0.7028750000000001
```

Abbildung 5: filename.xls

Die erste Zeile der Datei enthält zu Dokumentationszwecken einen durch G1 generierten Datums- und Zeitstempel sowie die beim Aufruf übergebenen Parameter.

Die zweite Zeile enthält die Spaltenüberschriften zur weiteren Auswertung. Wie auch die Werte in den folgenden Zeilen sind die Spalten in der ASCII-Datei durch Tabs („\t“) getrennt, so daß sie relativ einfach in Auswertungsprogramme eingelesen werden können. Im folgenden werden die einzelnen Spalten beschrieben:

Variable	Beschreibung
Nr	Diese Spalte wird durch den AvInfoserver mit einer fortlaufenden Nummer versehen, die zur Identifizierung einer bestimmten Transaktion dienen kann.
Time	Dieser Zeitstempel wird durch den AvInfoserver zum Zeitpunkt des

Variable	Beschreibung
	Schreibens in die Datei erzeugt. Es ist möglich, daß verschiedene Transaktionen gleiche Zeitstempel erhalten. Diese Spalte dient im Normalfall als X-Achse für Auswertungsgrafiken
Seller	Hier steht der Name desjenigen Software-Agenten, der die Information an den AvInfoserver gesendet hat, in der gegenwärtigen Implementation also der Verkäufer. Der Agentenname setzt sich zusammen aus [Stammbezeichnung]_[ID]_ [Generationenbezeichner]. Die beim Start der Simulation vorhandenen Software-Agenten gehören zur Generation 0. Die ID wird zentral durch den Infoserver vergeben und sollte daher eindeutig sein.
Tribe	Sofern in der Ini-Datei (s.u.) eine weitere Unterteilung der Agententypen in Tribes getätigt wurde, ist sie in dieser Spalte dokumentiert. Der Default ist jedoch tribe == type.
Type	Dies ist der kodierte Agententyp, d.h. 1 für AvTradeAgentProperties.Producer, 2 für Lumberjack usw.
Buyer	In dieser Spalte steht der Name desjenigen Software-Agenten, mit dem diese Transaktion getätigt wurde, die zur Erzeugung des Datensatzes geführt hat. Der Aufbau des Namens ist gleich der Spalte „seller“.
Location	Diese Spalte enthält den Namen des AvLocationAgent, der den Marktplatz repräsentiert, auf dem sich beide Software-Agenten zum Zeitpunkt der Transaktion befunden haben.

Variable	Beschreibung
Factor	Die Zahl in dieser Spalte entspricht der Variable AvTradeAgent.outputFactor.type und kodiert das in der Transaktion gehandelte Gut. (1 = Wood, 2 = Board etc.)
Wood_price	In dieser und den folgenden 3 Spalten darf insgesamt nur ein einziger Wert stehen, welcher den Preis anzeigt, zu dem das Gut vom Verkäufer zum Käufer übergegangen ist.
Board_price	
Plate_price	
Table_price	
Pro_cash	In dieser und den folgenden 3 Spalten darf insgesamt nur ein einziger Wert stehen, welcher den Bargeldbestand (AvTradeAgent.cash) des Verkäufers anzeigt.
Lum_cash	
car_cash	
Cab_cash	
Pro_profit	In dieser und den folgenden 3 Spalten darf insgesamt nur ein einziger Wert stehen, welcher den Gewinn (AvTradeAgent.profit) des Verkäufers anzeigt, d.h. den gewichteten Unterschied zwischen dem Lagerpreis und dem Verkaufspreis.

Variable	Beschreibung
Lum_profit	
car_profit	
Cab_profit	
Pro_comeff	In dieser und den folgenden 3 Spalten darf insgesamt nur ein einziger Wert stehen, welcher die Kommunikationseffizienz (AvTradeAgent.tradeCommunication-Counter) des Verkäufers anzeigt, d.h. die Anzahl der gesendeten Messages zwischen zwei Handelsvorgängen.
Lum_comeff	
car_comeff	
Cab_comeff	

Tabelle 3: Spaltenbeschreibungen für filename.xls

4.2.4 Gen-Parameterveränderungen

„*filename*.popgenes“ zeigt die Veränderung des Mittelwertes der Gene, in durch Tabulatoren getrennten Spalten. Ein Typwert in der ersten Spalte zeigt, für welche Agentenklasse die Zeile gilt (1 = Producer, 5 = Consumer). In der zweiten Zeile steht der Zeitpunkt der Ausgabe. Diese Datei ist insbesonders bei Angabe von Varianzen wichtig, da hierdurch die tatsächliche Verteilung der Gene in der Population

betrachtet werden kann. Ausgabezeitpunkt ist jeweils die Erschaffung eines Software-Agenten, d.h. gleichzeitig mit *filename*.birth.

```
Name              p_acq d_chg d_jmp p_sat w_mem p_rep

type 3 genes:     20359 0.3   0.25  0.15  0.75

type 2 genes:     20409 0.3   0.25  0.15  0.75

type 4 genes:     20429 0.3   0.25  0.15  0.75

type 5 genes:     20449 0.3   0.25  0.15  0.75

type 2 genes:     20479 0.3   0.25  0.15  0.75
```

Abbildung 6: filename.popgenes

4.2.5 Anzahl an Software-Agenten

„*filename*.popcount“ zeigt die aktuelle Anzahl an Software-Agenten. Ausgabezeitpunkt ist jeweils die Erschaffung eines Software-Agenten, d.h. gleichzeitig mit *filename*.birth.

Time	pro	lum	car	cab	con
156665	3	6	9	5	3
156705	3	7	9	5	3
159459	3	8	9	5	3
171736	3	8	10	5	3
189853	3	8	11	5	3
201850	3	8	12	5	3

Abbildung 7: filename.popcount

4.3 Laufzeit der Simulation

```
Syntax: -t time
Beispiel: -t 3600
```

Bestimmt die Laufzeit der Simulation in Sekunden (im Beispiel also 1 Stunde Laufzeit). Der Startzeitpunkt ist dabei nicht der Aufruf des G1-Generators, sondern die Übermittlung von AvTradeAgent.go() nach Instanziierung sämtlicher Software-Agenten. Ist die Laufzeit, welche durch den AvInfoServer überwacht wird, beendet, schreibt der Infoserver in sämtliche Logdateien „shutdown“ und ruft die Methode AvLocationAgent.shutdown() auf, welche in einer Kettenreaktion wiederum bei allen bei der jeweiligen Location registrierten Software-Agenten AvTradeAgent.shutdown() aufruft.

4.4 Zahl der Software-Agenten

```
Syntax: -n agentNumber
Beispiel: -n 50
```

In diesem Argument wird die Zahl der von jedem Typ erzeugten Software-Agenten angegeben (im Beispiel 50 Software-Agenten). Die Gesamtzahl an Software-Agenten ergibt sich dann als n * 5 (producer, lumberjack, carpenter, cabinetmaker, consumer, d.h. im Beispiel insgesamt 250 Software-Agenten in der Simulation). Bei Angabe einer Ini-Datei wird dieser Parameter nicht berücksichtigt, da er in der Ini-Datei für jeden Agententyp einzeln angegeben werden muß.

4.5 Tracelevel

```
Syntax: -trace tracelevel
Beispiel: -trace 1
```

Im Source Code sind Programmaufrufe des Typs

```
if (tracelevel>=2) trace( "negotiate(): I want to sell
for $"+ current.offer.price +" and "+msg.sender+" wants
to buy for $"+ offer.price);
```

eingestreut. Sie steuern den Detaillierungsgrad der Logausgabe, die einerseits in der Konsole ausgegeben und andererseits in die Datei *filename.log* geschrieben wird. Grundsätzlich soll tracelevel „1“ eine Zusammenfassung, tracelevel „2“ zusätzlich die eingegangenen Events, und tracelevel „3“ zusätzlich die versendeten Messages mit ihren Parametern loggen.

4.6 Ini-Datei

```
Syntax: -inifile filename
Beispiel: -inifile 2car.ini
```

Dieses Argument erzeugt die Agentenpopulation abhängig von den Einträgen in einer Ini-Datei. Eine Ini-Datei (siehe Beispiel) besteht aus Abschnitten [tribe] und Schlüssel/Wert-Paaren Schlüssel=Wert. Notwendige Einträge in jedem Abschnitte sind die für agentType (zulässige Werte sind: Lumberjack, Carpenter, Cabinetmaker, Producer, Consumer), agentNumber (Anzahl der Software-Agenten dieses

Stammes) und die 5 Gene p_acquisitiveness, del_change, del_jump, p_satisfaction, und w_memory. Einträge für die Varianzen: var_acquisitiveness, var_change, var_jump, var_satisfaction, and var_memory sind optional. Ebenfalls optional ist der Eintrag für reproduction (default ist true, false muss wie im Beispiel angegeben expliziert werden).

```
...

[lum]

agentType=Lumberjack

agentNumber=10

p_acquisitiveness=0.3

del_change=0.25

del_jump=0.15

p_satisfaction=0.7

w_memory=0.7

p_reputation=1.0

reproduction=false



[car1]

agentType=Carpenter

agentNumber=10

p_acquisitiveness=0.3

del_change=0.25

var_change=0.2

del_jump=0.15

p_satisfaction=0.7
```

```
w_memory=0.7

p_reputation=1.0

reproduction=false

...
```

Abbildung 8: Ausschnitt einer Ini-Datei

4.7 Gen-Parameter

4.7.1 Startmittelwerte der Genparameter

```
Syntax: - g p_acq d_chg d_jmp p_sat w_mem p_rep
Beispiel:  -g 0.1 0.26 0.10 0.8 0.2 1
```

Bedeutung: Dieses Argument wird von 6 Zahlen zwischen 0 und 1 gefolgt, die durch Leerzeichen voneinander getrennt werden. Diese 6 Zahlen stellen die Geneinstellungen für die Software-Agenten dar, in der Reihenfolge: p_acquisitiveness, del_change, del_jump, p_satisfaction, w_memory und p_reputation.

4.7.2 Startverteilung der Genparameter

```
Syntax: -var var_p_acq var_d_chg var_d_jmp var_p_sat
var_w_mem p_rep
Beispiel: -var 0 0 0.2 0 0 0
```

Bedeutung: Dieses Argument wird ebenfalls von 6 Fließkommazahlen im Wertebereich zwischen 0 und 1 gefolgt, die durch Leerzeichen voneinander getrennt werden. Diese 6 Zahlen stellen die Varianzen der Geneinstellungen für die Software-

Agenten dar, in der Reihenfolge: p_acquisitiveness, del_change, del_jump, p_satisfaction, w_memory, und p_reputation.

Im o.a. Beispiel werden alle Software-Agenten mit identischer, nicht veränderter Genaustattung erzeugt, lediglich der Parameter del_jump variiert mit einer Normalverteilung von 0.2 um den im Argument -g vorgegebenen Wert für del_jump.

4.7.3 Detaillierte Übersicht der vorhandenen Genparameter

4.7.3.1 Acquisitiveness (p_acquisitiveness)

Das Acquisitiveness-Gen bestimmt während einer Verhandlung, ob der Software-Agent beim folgenden „offer“ seinen Preis verändert. Die tatsächliche Entscheidung wird durch eine stochastische Probe bestimmt, der Wert dieses Gens determiniert also nicht die Handlung, sondern gibt eine Wahrscheinlichkeit an. Der Wertebereich des Gens liegt zwischen 0 und 1, ein Wert von 0.7 bedeutet eine 70%ige Wahrscheinlichkeit, daß der Software-Agent seinen Preis in Richtung auf den Verhandungspartner anpasst. Tritt der Software-Agent als Verkäufer auf, so wird er, sofern `checkAgainst(p_acquisitiveness)` „false“ liefert, seinen Preis in einer bestimmten Höhe senken, wie im folgenden Codebeispiel gezeigt. Tritt der Software-Agent als Käufer auf, so wird er sein Angebot erhöhen. Der Betrag der Änderung stepSize wird durch das Gen del_change gesteuert.

```
// check whether we will make a different offer
 if (!checkAgainst(p_acquisitiveness)) {
      // if I overcome my greed...
      sellActPrice = sellActPrice - current.stepSize;
                  // ...then lower price by step size
```

4.7.3.2 In-Negotiation Delta Price Change (del_change)

Dieses Gen dient der Berechnung des jeweiligen preislichen Entgegenkommens zwischen zwei Verhandlungsrunden. Nachdem die Überprüfung von p_acquisitiveness ergeben hat, daß beim nächsten Gegengebot der Preis verändert wird, wird an dieser Stelle der tatsächliche Wert der Preisänderung berechnet.

Dieser ergibt sich aus der Differenz zwischen dem eigenen Preisangebot und dem des potentiellen Transaktionspartners. Ein Wert von del_change=0.25 bedeutet in diesem Fall das Erreichen des gegnerischen Preisangebots in 4 Verhandlungsrunden (sofern der Gegner nicht ebenfalls entgegenkommt, sondern bei seinem Preisangebot verharrt).

```
// update the negotiation object
      if (current.stepSize==0) {
// we need to determine the step size for this negotiation
      // (if we haven't already)
      current.stepSize = (sellActPrice - offer.price) * del_change;
```

4.7.3.3 Pre-Negotiation Delta Price Change (del_jump)

Dieses Gen bestimmt den jeweiligen Startwert bei Neuverhandlungen. Das Verhandlungsergebnis der letzten Transaktion wird gespeichert. Durch Multiplikation mit einem prozentualen Aufschlag del_jump errechnet sich der Ausgangswert für die nächste Transaktion. Ein Beispiel für del_jump=0.15: ein Software-Agent hat beim letzten Mal ein Gut für 100 GE verkauft. Beim nächsten Mal wird er mit 115 GE in die Verhandlungen einsteigen.

```
// new starting price for negotiations is agreement price plus
del_jump%

            lastSellPrice = agreement;

            sellMaxPrice = agreement * (1.0 + del_jump);
```

4.7.3.4 Satisfaction (p_satisfaction)

Dieser Parameter spiegelt das durch Verhalten p_acquisitiveness gesteuerte Verhalten des Transaktionspartners. Falls die Verhandlungsrunden nicht zur Zufriedenheit des Software-Agenten verlaufen (d.h. daß der Transaktionspartner seinen Preis nicht ändert), dann wird durch stochastische Prüfung gegen den Parameter p_satisfaction eine Entscheidung über den Abbruch der Verhandlungen (d.h. ein Senden von *reject-offer* anstelle eines (counter-)*propose* getroffen. Ein Wert von p_satisfaction=0.75 bedeutet dabei eine 75% Chance eines Verhandlungsabbruchs.

```
if (current.lastPriceOffer >= offer.price) {

      // if other party doesn't raise price offer

      if (!checkAgainst(p_satisfaction)) {

      // then check against our satisfaction and reject evtl.
```

4.7.3.5 Price Memory Weight (w_memory)

Um feststellen zu können, ob der vom Transaktionspartner im allerersten Angebot (offer) angegebene Preis überhaupt in einen Bereich fällt, der eine erfolgreiche Verhandlungslösung zuläßt, prüft der Käufer diesen vor Aufnahme von Verhandlungen. Er vergleicht den erhaltenen Preis dabei mit denjenigen Preisangeboten, die er in der Vergangenheit von anderen Software-Agenten erhalten hat, unabhängig davon, ob diese Angebote später zu Abschlüssen geführt haben oder nicht.

Das Gen w_memory dient in diesem Zusammenhang der Gewichtung von Preisinformationen. Ein Wert w_memory=0.2 rechnet den aktuell erhaltenen Preis mit einem Gewicht von 20% in die Variable memory ein. Je höher der Gewichtungsfaktor w_memory ist, desto stärker wird der aktuelle Wert einbezogen, und desto schneller wird sich der Wert von memory einer aktuellen Marktlage anpassen.

```
if (offer.price >= memory) {

      // ...then use memory to check for too high prices

            if ( !checkAgainst(p_satisfaction)) {

                  reject = true;

                  }

                  if (memory != 0 && offer.price > 2 * memory)

                  reject = true;

                  // reject exorbitant prices, introduced in combi-
                  nation with

                  // brenner's learning rule, see postTransaction()

                  }

          memory = (1-w_memory) * memory + w_memory * of-
        fer.price;

                  // update our memory of initial prices
```

4.7.3.6 Reputationsinformationen (w_adjust_reputation; p_curiosity; p_communicativeness)

Es bestehen drei Möglichkeiten Reputationsinformationen zu verarbeiten:

1. Keinerlei Verarbeitung

2. Dezentrale Verarbeitung

3. Zentrale Verarbeitung

Die Auswahl der drei Möglichkeiten wird über die Parameter w_adjust_reputation, p_curiosity und p_communicativeness gesteuert.

- **w_adjust_reputation**

  Der Parameter w_adjust_reputation steuert die generelle Berücksichtigung von Reputationsinformationen:

w_adjust_reputation = 0	Es werden keine Reputationsinformationen berücksichtigt.
w_adjust_reputation = 1	Es werden Reputationsinformationen berücksichtigt.

- **p_communicativeness**

  Der Parameter p_communicativeness steuert die Nutzung von Reputationsinformationen der zentralen Reputationsinstanz.

p_communicativeness = 0	Die zentrale Instanz wird nicht genutzt.
p_communicativeness = 1	Die zentrale Instanz wird genutzt.

- **p_curiosity**

  Der Parameter p_curiosity steuert die Nutzung eigener Reputationsinformationen.

p_curiosity = 0	Es werden keine Reputationsinformationen genutzt.

p_curiosity = 1	Es werden eigene Reputationsinformationen genutzt.

4.8 Evolutionäre Algorithmen

4.8.1 Costs Of Life

```
Syntax: -col costsOfLife
Beispiel: -col 0.02
```

Die „Lebenskosten“ werden in gleichen Zeitabständen vom Bargeld des Software-Agenten („cash“) als Prozentsatz abgezogen. Im o.a. Beispiel also immer 2% des Bargelds, mindestens jedoch 1 GE. Software-Agenten, die kein Bargeld mehr besitzen, gehen „bankrott“ und werden die Simulation verlassen.

4.8.2 Reproduction

```
Syntax: -noreproduction
```

Schaltet die agent reproduction (d.h. die evolutionary algorithms) aus. Wenn dieser Parameter gesetzt ist, sollte die Endpopulation eines Simulationslaufs identisch mit der Ausgangspopulation sein, sofern nicht während des Simulationslaufs Software-Agenten gestorben sind. Neue Software-Agenten kommen hingegen nicht dazu.

4.8.3 Courter Threshold

```
Syntax:  -ct courterThreshold
Beispiel: -ct 12
```

Das Konzept der hier verwendeten dezentralen evolutionären Algorithmen sieht die Aussendung von sogenannten „Plumages“ durch die Software-Agenten vor, in de-

nen diese ihre Gene und ihre Fitness beschreiben. Jeder Software-Agent sammelt diese Plumages, die er von anderen Software-Agenten seines Agenttyps bzw. seines „Tribes“ bekommt. Wenn er genug Plumages erhalten hat (d.h. mehr als in der courterTreshold angegeben, im Beispiel also 13), sucht er sich den besten Partner aus, mischt die aus dessen Plumage gewonnenen Gene mit seinen eigenen und erschafft einen neuen Software-Agenten.

4.8.4 Maturity Threshold

```
Syntax:    -mt maturityThreshold
Beispiel: -mt 10
```

Da Software-Agenten bei Beginn der Simulation noch ohne Erfahrungswerte handeln müssen, wird auch bei den evolutionären Algorithmen eine Latenzzeit gewährt. Anhand des Wertes der maturityThreshold wird festgelegt, wieviele erfolgreiche Transaktionen ein Software-Agent abgeschlossen haben muß, bevor er anfängt, Plumages auszusenden bzw. empfangene Plumages zu verarbeiten.

4.9 Summaryfile

```
Syntax: -summary summaryfilename
```

Während des Shutdown des Systems werden Auswertungszahlen in diese Datei geschrieben. Im summaryfile werden z.B. die gesendeten proposes und rejects mitgezählt, was ein guter Ansatzpunkt für eine Sensitivitätsanalyse sein könnte. Um ein Summaryfile zu schreiben, müssen aber alle Messages über den LocationAgent (als EventListener, s.u.) geroutet werden, was Auswirkungen auf das Laufzeitverhalten von AVALANCHE hat, verglichen mit dem direkten Senden.

```
if (direct)
      ((AvTradeAgent)msg.payload).AvAgentEventFired(e);
else
      notifyAvAgentEventListeners(e);
```

5 Klassenhierarchie

5.1 Die AvAgent-Klassen

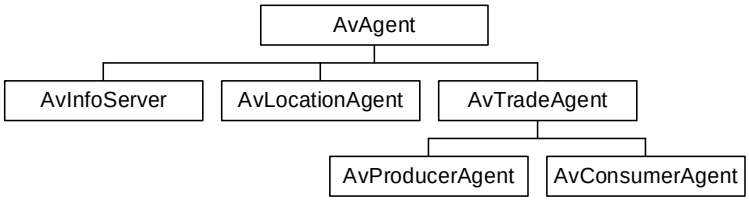


Abbildung 9: Klassenhierarchie AvAgent

In diesem Abschnitt wird nur die Funktionalität in groben Zügen dargestellt und auf wichtige Eigenschaften verwiesen. Methoden werden ohne Parameter angegeben, genauere Informationen sind im Quelltext bzw. im javadoc (avalanche/Docs/index.html) ersichtlich.

5.1.1 AvAgent

Diese Klasse dient der Sammlung verbindender Variablen und Methoden aller abgeleiteten Klassen. Sämtliche Middleware-Funktionalität (z.B. von Voyager) sollte ebenfalls hier implementiert sein, so daß die abgeleiteten Klassen systemunabhängig gestaltet werden können.

Wichtige Methodenblöcke sind die Generierung von Zufallszahlen (generateRandom(), checkAgainst()) und von laufenden Nummern für Messages (genID()), sowie die Behandlung von Java-Threads (sleep()) und Java-Events (addAvAgentEventListener()).

5.1.2 AvTradeAgent

Diese Klasse implementiert die wesentliche Funktionalität von AVALANCHE. Jeder AvTradeAgent versucht, unter Berücksichtigung folgender Vorgehensweise sein Ziel der Gewinnmaximierung zu verfolgen.

Er versucht seine Waren (outputFactors) zu verkaufen.

Besitzt er genügend Ressourcen (inputFactors), produziert er Waren.

Besitzt er keine Ressourcen, versucht er diese zu kaufen.

Ist keines dieser Ziele auf dem aktuellen Marktplatz (AvLocationAgent) erreichbar, versucht er auf einen anderen Marktplatz zu wechseln. Ist auch diese nicht möglich, stirbt er.

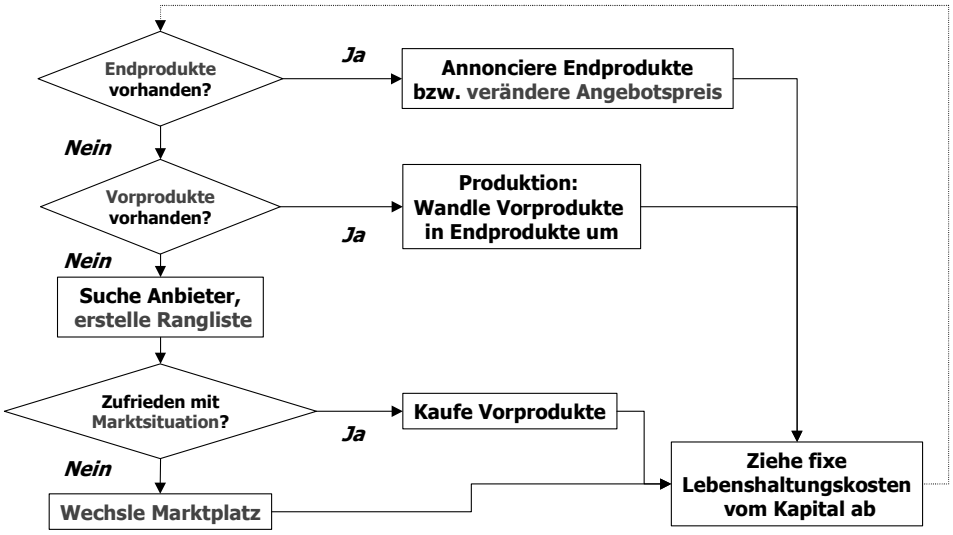


Abbildung 10: Hauptschleife eines AvTradeAgent

Jeder AvTradeAgent wird mit einem AvTradeAgentProperties-Objekt initialisiert, welches charakterisierende Variablen für die Stammdaten des Software-Agenten enthält, z.B. den Namen und Typ, die Art der Input- und Outputfaktoren sowie die Genparameterwerte.

Die Methoden des AvTradeAgent lassen sich in 3 große Gruppen einteilen:

Proaktives Wirtschaften und Verhandeln (work(), trySelling(), tryProduction(), tryBuying(), processMessage(), negotiate())

Beobachtung und Steuerung von Geld- und Warenvermögen (addItemToInventory(), removeItemFromInventory(), amountOfItemInInventory(), capital(), payCostsOfLife(), updateProfit(), updateSales())

Vererbung (growOlder(), mate(), selectNameForOffspring(), getMate(), create-Plumage(), estimateFitness())

5.1.3 AvTradeAgentProperties

Die Klasse AvTradeAgentProperties

Abgeleitet von AvAgentProperties

Initialisierung mit einem AvAgentServer-Objekt, dem Server auf dem der Software-Agent starten soll, sowie zwei AvFactor-Objekten, welche die **input**- und **output**-Faktoren des TradeAgents spezifizieren.

Eigenschaften: Dem AvTradeAgent können folgende Attribute mitgegeben werden:

- Das anfängliche Geldvermögen mit **setCash().**
- Die Anzahl an input-Faktoren zu Beginn mit **setInputUnits()**
- Die Anzahl an output-Faktoren zu Beginn mit **setOutputUnits()**
- Die Kosten zur Produktion eines output-Faktos mit **setProductionCosts()**
- Die Zahl an input-Faktoren, die zur Produktion mindestens nötig ist mit **setProductionInputRelation()**
- **Die Zahl an output-Faktoren, die in einerm Produktionszyklus erstellt werden mit setProductionOutputRelation()**
- Die Zeit, die für die Produktion in einem Produktionszyklus erforderliche Zeit mit **setProductionTime()**

- Die maximale Anzahl an Versuchen, input-Faktoren auf dem Markt zu erwerben mit **setBuyMaxAttempts()**

- Die maximale Anzahl an Angeboten, die eingeholt werden mit **setBuyMaxOffers()**

- Die Zeit, die nach einem erfolglosen Kaufversuch gewartet werden soll mit **setBuyWait()**

- Der Preisaufschlag, der den gewünschten Verkaufspreis pro output-Einheit ergibt mit **setSellCharge()**

- Die maximale Anzahl an Versuchen, output-Faktoren auf dem Markt zu verkaufen mit **setSellMaxAttempts()**

- Die Zeit, die gewartet werden soll, bevor die output-Faktoren ggf. billiger nochmals angeboten werden sollen mit **setSellWait()**

- Der minimale Preis, der für eine output-Einheit erzielöt werden soll mit **setSellMinPrice()**

- Der Prozentsatz, um den der Verkaufspreis bei einem gescheiterten Versuch gesenkt werden soll, falls er dadurch nicht unter den Mindestpreis fällt mit **setDiscount()**

5.1.4 AvConsumerAgent

Der AvConsumerAgent überschreibt einige Methoden des „normalen“ AvTradeAgent, um folgende Eigenheiten zu besitzen: Er kann nichts produzieren, also auch keine Waren verkaufen. Außerdem besitzt er immer gleichviel Geld (**cash**), kann

also - in gewissen Grenzen - beliebig viele Waren einkaufen. Dies soll ein Endkonsumentenverhalten simulieren.

5.1.5 AvProducerAgent

Der AvProducerAgent überschreibt einige Methoden des „normalen“ AvTradeAgent, um folgende Eigenheiten zu besitzen: Er benötigt keine Ressourcen (**input**), um Waren zu produzieren und er kann keine Waren einkaufen. Dies soll einen Rohstofflieferanten simulieren, der - allerdings auch nur abhängig vom Verkaufserfolg - Rohstoffe liefern kann.

5.1.6 AvLocationAgent

An einem AvLocationAgent können sich AvAgents mittels **register()** und **deregister()** an- bzw. abmelden. Bei einer Anmeldung sorgt der AvLocationAgent automatisch dafür, daß der Software-Agent beim bisherigen Server abgemeldet wird - ein Software-Agent soll sich nur an einem Server zur gleichen Zeit befinden können.

Der AvLocationAgent implementiert die **register()** und **deregister()**-Methoden mit Hilfe einer **Hashtable**.

Ein registrierter Software-Agent kann mit **setOffer()** ein Angebot auf den Markt bringen.

Ein registrierter Software-Agent kann mit **getOffers()** eine Liste (genauer einen **Vector**) aller am Markt vorhandenen Angebote zu einem gewünschten Faktor erhalten.

Andere AvLocationAgent können mittels **link()** und **unlink()** als Nachbarn an- bzw. abgemeldet werden. Auch hier wird wieder ein **Hashtable**-Objekt verwendet.

5.1.7 AvInfoServer

Der AvInfoServer stellt den AvInformants eine Möglichkeit zur Verfügung, Informationen über ihren aktuellen Zustand in einem Fenster, der Unterklasse **AvInfoServerWindow**, auszugeben. Für jeden registrierten AvInformant wird dabei eine eigene Statuszeile **(**ein **AvInfoServerStatusPanel**) angezeigt. Zudem sind alle Meldungen in einem größeren Textfeld zu sehen.

beforeShutdownTimeout() und **beforeShutdown()** sind implementiert: Nach Betätigen des Schließ-Gadgets (von der Unterklasse **AvInfoServerWindowListener** erkannt) wird der Server erst nach einiger Verzögerung heruntergefahren und gibt noch entsprechende Meldungen von sich.

Mit **addInformant()** und **removeInformant()** können AvInformants an- und abgemeldet werden.

Mit **inform()** können registrierte AvInformants ihren aktuellen Status in der Ihnen zugeordneten Zeile anzeigen. Zudem können allgemeine Meldungen mit der zweiten Form der Methode im größeren Textfeld von beliebigen Klassen angezeigt werden.

5.2 Verhandlungen

```
class AvFactor (implements java.lang.Cloneable, ja-
va.io.Serializable)
class AvOffer (implements java.io.Serializable, ja-
va.lang.Cloneable)
class BasicNegotiation (implements java.io.Serializable)
```

5.2.1 AvFactor

Initialisierung mit einem String, der die Bezeichnung (den Typ) des Faktors darstellt.

Eigenschaften: AvFactor stellt ein Produkt dar. Bis jetzt ist dieses nur durch seine Bezeichnung definiert, Merkmale wie z.B. Qualität, Farbe o. Ä. können später hinzugefügt werden.

Die öffentliche String-Variable **type** läßt sich neu belegen.

5.2.2 AvOffer

Initialisierung mit einem AvTradeAgent-Objekt, dem Anbieter des Angebots, einem AvFactor-Objekt, dem Angebotsgegenstand, sowie zwei **int**s, die Preis und Menge angeben. Wird nur der Anbieter angegeben, wird ein „leeres“f Angebot erzeugt (Preis und Menge sind 0).

Eigenschaften: AvOffer stellt ein Angebot dar.

- Mit **isEmpty()** läßt sich feststellen, ob es sich um ein „leeres“ Angebot handelt.
- **supplier()** gibt den anbietenden AvTradeAgent an.
- Mit **sign()** läßt kann ein Software-Agent ein Angebot annehmen/unterzeichnen. Ein Angebot kann nur einmal unterzeichnet werden !
- Mit **signed()** läßt sich feststellen, ob das Angebot schon unterzeichnet wurde.
- Mit **signatory()** läßt sich der Unterzeichner feststellen.

5.3 Nachrichten

`class AvAgentMessage (implements java.io.Serializable)`

`class AvInfo (implements java.io.Serializable, java.lang.Comparable)`

5.4 Exceptions

Diese Klassen wurden in früheren Versionen von AVALANCHE eingesetzt. Sie sind derzeit nicht in den Quellcode eingebunden, es ist jedoch möglich, daß Kommentare auf diese Klassen verweisen. Daher sind die Beschreibungen dieser Klassen hier mit aufgeführt.

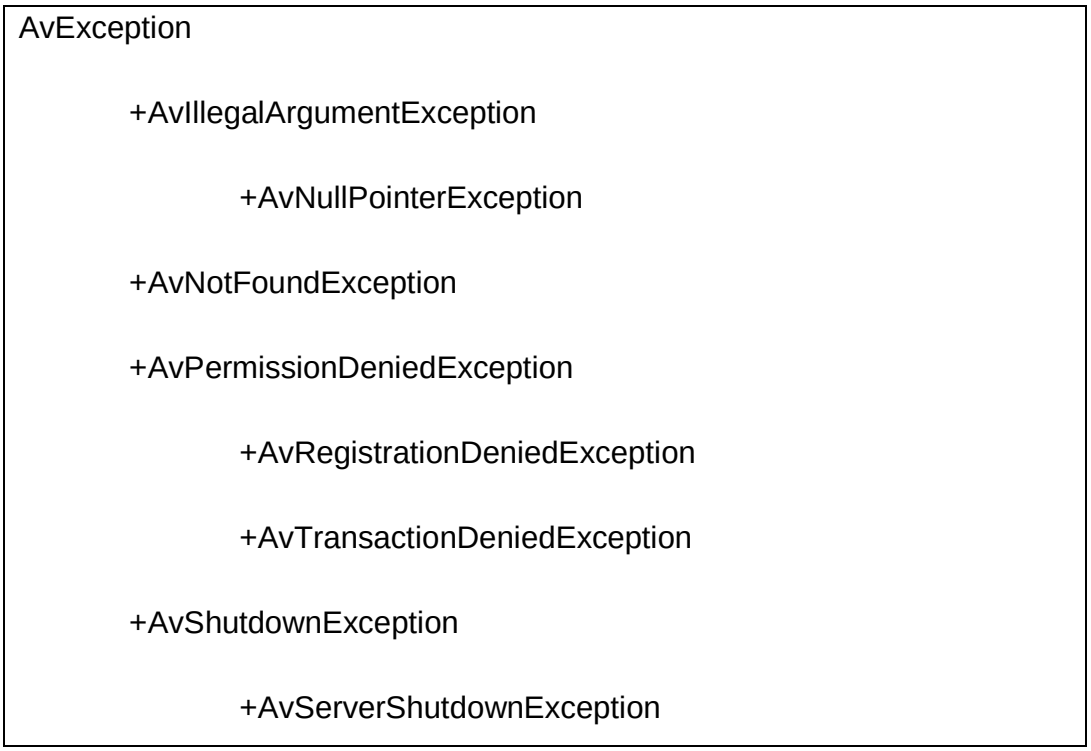


Abbildung 11: Klassenhierarchie Exceptions (z.Zt. nicht eingesetzt)

5.4.1 AvException

Abgeleitet von Exception

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält.

Eigenschaften: Dies ist die Oberklasse aller AvException-Klassen zur Definierung spezieller AVALANCHE -Ausnahmen. Die genaue Ableitung der jeweiligen AvException ist in der Fehlermeldung enthalten.

5.4.2 AvPermissionDeniedException

Abgeleitet von AvException

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält.

Eigenschaften: Diese Exception wird geworfen, wenn ein Zugriff oder eine Anfrage verweigert wird.

5.4.3 AvRegistrationDeniedException

Abgeleitet von AvPermissionDeniedException

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält, sowie optional einem **int**, welcher ein Zulassungslimit angibt.

Eigenschaften: Diese Exception wird geworfen, wenn ein Registrierung verweigert wird.

5.4.4 AvTransactionDeniedException

Abgeleitet von AvPermissionDeniedException

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält.

Eigenschaften: Diese Exception wird geworfen, wenn eine Transaktion nicht erfolgreich durchgeführt wurde.

5.4.5 AvIllegalArgumentException

Abgeleitet von AvException

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält.

Eigenschaften: Diese Exception wird geworfen, wenn ungültiger Wert übergeben wird.

5.4.6 AvNullPointerException

Abgeleitet von AvIllegalArgumentException

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält.

Eigenschaften: Diese Exception wird geworfen, wenn null-Pointer unzulässigerweise übergeben wird.

5.4.7 AvNotFoundException

Abgeleitet von AvException

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält.

Eigenschaften: Diese Exception wird geworfen, wenn ein Objekt nicht gefunden wurde, falls z.B. ein Software-Agent nicht registriert ist und sich dennoch abmelden will.

5.4.8 AvShutdownException

Abgeleitet von AvException

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält.

Eigenschaften: Diese Exception wird geworfen, wenn ein Server oder auch ein Software-Agent in Kürze deaktiviert wird.

5.4.9 AvServerShutdownException

Abgeleitet von AvShutdownException

Initialisierung optional mit einem String-Objekt, das eine Fehlermeldung enthält.

Eigenschaften: Diese Exception wird geworfen, wenn speziell ein Server in Kürze heruntergefahren wird.

6 Kommunikationsprotokolle

Die zwischen den AvTradeAgents und AvLocationAgents verschickten Nachrichten lassen sich in 4 Protokolle unterteilen, die unterschiedliche semantische Bedeutung für die Software-Agenten haben. Sämtliche Nachrichten werden syntaktisch in der Klasse AvAgentMessage verpackt. Die semantische Bedeutung der Nachrichtentypen orientiert sich an den Standardisierungsbemühungen der FIPA (Foundation

for Intelligent Physical Agents) in Bezug auf eine „Agent Communication Language“ (ACL). Für die Nutzung in AVALANCHE wurde eine Untermenge der vorhandenen Nachrichtentypen implementiert.

6.1 Die Klasse AvAgentMessage

Die unterschiedlichen Variablen entsprechen wie folgt den Vorschlägen in der ACL:

AVALANCHE Implementation	ACL Spezifikation	Bedeutung
String performative	Benennung des „communicative act“	In dieser Variable steht der Nachrichtentyp, z.B. propose, reject, accept-offer.
String content	:content	Diese Variable enthält die eigentliche Information der Nachricht, bei einem "propose" z.B. wird hier das Gut und der verlangte Preis kodiert.
String inReplyTo	:in-reply-to	Hier wird eine

		Referenz auf die vorausgegangene Nachricht übermittelt, um die Verhandlungssequenz verfolgen zu können.
String language	:language	Diese Variable wird nicht genutzt. In der ACL Spezifikation ist dieses Feld für die Festlegung der Codierung der Variable „content“ vorgesehen.
String protocol	:protocol	Diese Variable wird zur Einordnung der empfangenen „communicative acts“ verwendet. In AVALANCHE gibt es 4 Protokolle,

		die dem Handel, der Bewegung mobiler Software-Agenten, der Suche nach Anbietern und der Reproduktion der Software-Agenten dienen.
String receiver	:receiver	In dieser Variable steht die eindeutige Identifikation des Empfängers.
String replyWith	:reply-with	Diese Variable wird nicht genutzt. In der ACL Spezifikation ist dieses Feld als Vorschlag des Senders für die Festlegung des „in-reply-to“ vorgesehen.
String sender	:sender	In dieser Variab-

		le steht die eindeutige Identifikation des Senders.
Object payload		Diese Variable kann je nach Nachrichtentyp unterschiedliche Bedeutung haben.

Tabelle 4: AvAgentMessage und ACL

6.2 Das Protokoll „trade“

Die innerhalb dieses Protokolls (im Code noch fälschlicherweise als Protokoll bezeichnet) versendeten Nachrichtentypen werden bilateral zwischen zwei Instanzen von AvTradeAgent (oder daraus abgeleiteten Klassen) verschickt. Ziel der Kommunikation ist die Durchführung einer ökonomischen Aushandlungsphase mit Angebot und Gegenangebot bis zum Erreichen einer Vereinbarung über Güter- und Wertaustausch.

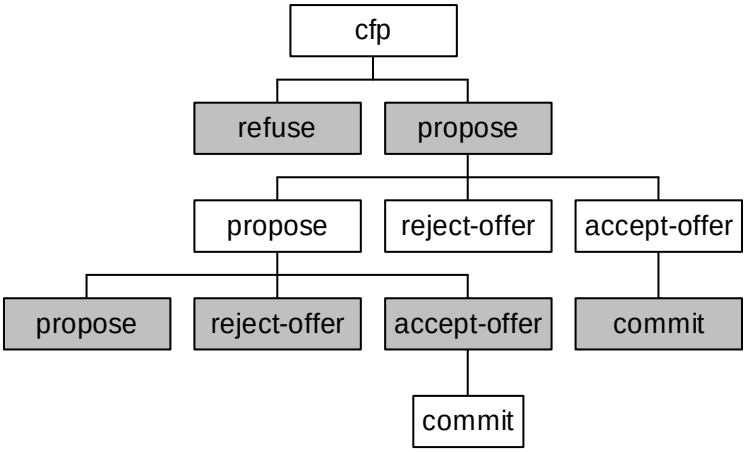


Abbildung 12: Das Protokoll “trade”

In Abbildung 12 werden weiße Nachrichten durch Käuferagenten, graue Nachrichten durch Verkäuferagenten verschickt.

Das Verhandlungsprotokoll beginnt mit dem Abschicken einer Nachricht vom Typ „cfp“. Diese wird vom Käuferagenten direkt an einen Verkäuferagenten geschickt, dessen Adresse der Käuferagent durch eine vorangegangene Ausführung des Protokolls „recommend-seller“ ermittelt hat.

Der Verkäuferagent sendet daraufhin durch ein „propose“ genauere Angaben zu seinem Angebot, insbesondere den Preis. Sendet er ein „refuse“, ist dies üblicherweise ein Zeichen dafür, daß das Gut bereits verkauft wurde, der Software-Agent sich jedoch noch nicht an der Location wieder ausgetragen hat.

Im Normalfall folgen auf ein „propose“ wechselseitig weitere „proposes“, in denen sich Käufer und Verkäufer in ihren Preisvorstellungen langsam einander annähern. Schließlich sendet einer von beiden ein „accept-offer“, auf welches der Verhandlungspartner mit einem „commit“ antwortet – jetzt erst ist die Transaktion abgeschlossen.

Sollte während der Verhandlung ein Software-Agent beschliessen, die Kommunikation abzubrechen, sendet er ein „reject-offer“. Daraufhin werden beide Software-Agenten zurückgesetzt und können sich auf dem Markt neu orientieren.

6.3 Das Protokoll „recommend-seller“

Dieses Protokoll dient der Ermittlung eines passenden Verkäuferagenten durch einen Käufer. Der Käuferagent sendet eine Anfrage an den AvLocationAgent, bei dem er gerade registriert ist. Dieser sendet ihm die Liste der gerade registrierten Software-Agenten, bei denen der gesuchte AvFactor als outputFactor angegeben ist. Daraufhin wird der Käuferagent zufällig oder gezielt Software-Agenten aus dieser Liste mittels eines „cfp“ ansprechen (siehe Protokoll „trade“).

7 Directory Services in AVALANCHE

Damit die Software-Agenten sich gegenseitig ansprechen zu können, wird über die Simulation verteilt ein Verzeichnisdienst implementiert. Die Wurzel dieses Verzeichnisses wird durch den Infoserver geführt. Die Einträge, die durch den Infoserver lokal geführt werden, sind Verweise auf entfernte Verzeichnisse in den Marktplätzen (LocationAgents). Im Infoserver handelt es sich daher nur um eine einstufiges Verzeichnis.

Die Verzeichnisbäume, die durch die LocationAgents geführt werden, sind dagegen aufwendiger. Die einzelnen Ebenen sind wie aus der folgenden Grafik ersichtlich eingeteilt:

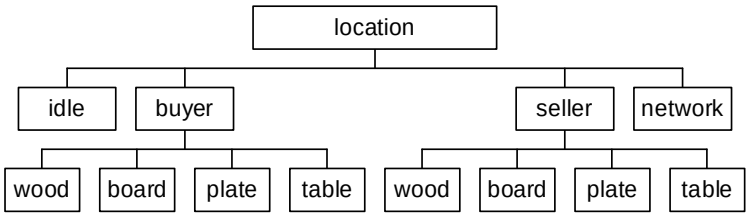


Abbildung 13: Directory Service Struktur

Die Eintragsebenen, die mit bind(), und unbind() angesprochen werden können, sollen folgendermaßen mit Leben gefüllt werden: Ein neu erschaffener Software-Agent trägt sich selbst über Directory.put ("/idle/"+agentName, (IAvTradeAgent) this) in das Verzeichnis ein. Sofern er im weiteren Verlauf als Käufer auftreten möchte, muß er per Namespace.bind (locationName+"/buyer/"+inputFactor+agentName, (IAvTradeAgent) this) diesen Eintrag verändern und über Directory.remove(...) den alten Eintrag löschen. Um sich als Verkäufer einzutragen, ist spiegelbildlich ein Namespace.bind (locationName+ "/buyer/" +inputFactor +agentName, (IAvTradeAgent) this) notwendig.

Sofern ein Software-Agent jetzt auf seinem Marktplatz nach Anbietern oder Nachfragern eines bestimmten Gutes sucht, kann er beispielhaft über den Aufruf von getDirectoryEntries("/locationName /buyer/ board") sämtliche im Moment der Abfrage gültigen Einträge in dieser Klasse abfragen und zur Verarbeitung lokal speichern.

LocationAgents tragen sich in ähnlicher Weise über Namespace.bind("locationName/network/",...) bei den angrenzenden Lokationen ein.

Verzeichnis der Initialisierungen der Testreihen

I. Testreihe

Initialisierung durch Kommandozeileneingabe:

```
-n 50 -t 3600 -g 0.7 0.25 0.15 0.75 0.2 1 -mt 25 -ct 12 -f just50
```

II. Testreihe

Initialisierung durch Kommandozeileneingabe:

```
-n 50 -t 3600 -g 0.7 0.25 0.15 0.75 0.2 1 -mt 25 -ct 12 -f just50 –rp
```

III. Testreihe

(Drei kooperative Zimmermann-Agenten; keine Reputationsverarbeitung)

Initialisierung durch Initialisierungsdatei:

Kooperative Zimmermann-Agenten (Carpenter)

```
[car]
agentType=Carpenter
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Holzfäller-Agent (Lumberjack)

```
[lum]
agentType=Lumberjack
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Schreiner-Agent (CabinetMaker)

```
[cab]
agentType=CabinetMaker
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Konsumenten-Agent (Consument)

```
[con]
agentType=Consumer
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Produzenten-Agent (Producer)

```
[pro]
agentType=Producer
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

IV. Testreihe

(Ein unkooperativer Zimmermann-Agent; zwei kooperative Zimmermann-Agenten; keine Reputationsverarbeitung)

Initialisierung durch Initialisierungsdatei:

Unkooperativer Zimmermann-Agent (Carpenter)

```
[carcheater]
agentType=Carpenter
agentNumber=1
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=0.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Zimmermann-Agent (Carpenter)

```
[car]
agentType=Carpenter
agentNumber=2
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Holzfäller-Agent (Lumberjack)

```
[lum]
agentType=Lumberjack
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Schreiner-Agent (CabinetMaker)

```
[cab]
agentType=CabinetMaker
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Konsumenten-Agent (Consument)

```
[con]
agentType=Consumer
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Produzenten-Agent (Producer)

```
[pro]
agentType=Producer
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

V. Testreihe

(Ein unkooperativer Zimmermann-Agent; zwei kooperative Zimmermann-Agenten; Nutzung eigener Reputationsinformationen)
Initialisierung durch Initialisierungsdatei:

Unkooperativer Zimmermann-Agent (Carpenter)

```
[carcheater]
agentType=Carpenter
agentNumber=1
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=0.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Zimmermann-Agent (Carpenter)

```
[car]
agentType=Carpenter
agentNumber=2
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Holzfäller-Agent (Lumberjack)

```
[lum]
agentType=Lumberjack
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=1.0
p_communicativeness=0.0
p_curiosity=1.0
reproduce=false
```

Kooperativer Schreiner-Agent (CabinetMaker)

```
[cab]
agentType=CabinetMaker
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=1.0
p_communicativeness=0.0
p_curiosity=1.0
reproduce=false
```

Kooperativer Konsumenten-Agent (Consument)

```
[con]
agentType=Consumer
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Produzenten-Agent (Producer)

```
[pro]
agentType=Producer
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

VI. Testreihe

(Ein unkooperativer Zimmermann-Agent; zwei kooperative Zimmermann-Agenten; Nutzung der zentralen Reputationsinstanz)
Initialisierung durch Initialisierungsdatei:

Unkooperativer Zimmermann-Agent (Carpenter)

```
[carcheater]
agentType=Carpenter
agentNumber=1
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=0.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Zimmermann-Agent (Carpenter)

```
[car]
agentType=Carpenter
agentNumber=2
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Holzfäller-Agent (Lumberjack)

```
[lum]
agentType=Lumberjack
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=1.0
p_communicativeness=1.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Schreiner-Agent (CabinetMaker)

```
[cab]
agentType=CabinetMaker
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=1.0
p_communicativeness=1.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Konsumenten-Agent (Consument)

```
[con]
agentType=Consumer
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```

Kooperativer Produzenten-Agent (Producer)

```
[pro]
agentType=Producer
agentNumber=3
p_acquisitiveness=0.7
del_change=0.25
del_jump=0.15
p_satisfaction=0.7
w_memory=0.7
p_reputation=1.0
w_adjust_reputation=0.0
p_communicativeness=0.0
p_curiosity=0.0
reproduce=false
```