Intelligente Software-Agenten

Stefan Schreinert
<stefan@intelligent-agents.info>

17. Juni 2002

Zusammenfassung

Neben dem Forschungsgebiet der *Künstlichen Intelligenz* (KI) versucht auch die *Verteilte Künstliche Intelligenz* (VKI) eine Maschine intelligent handeln zu lassen. Im Vergleich zur KI erreicht ein Netzwerk in der VKI seine Intelligenz dadurch, dass viele, auch geringer intelligente Software-Programme zusammenarbeiten: Die Software-Agenten. Diese sind dadurch intelligent, dass sie mit anderen Agenten *kommunizieren*, sich untereinander *koordinieren* und miteinander *kooperieren*.

Dieser Artikel ist ein Überblick über die Architekturen intelligenter Agenten, die Vorteile mobilen Codes und die Verteilte Künstliche Intelligenz. Dabei wird gezeigt, welche Vorteile eine Kooperation bietet. Zugleich aber sieht der Leser, warum eine *bedingungslose* Kooperation nachteilig ist und Egoismus zum *intelligenten* Handeln erforderlich ist. Dazu werden die Grundzüge der strategischen Verhandlungen samt der benötigten Spieltheorie beschrieben.

1 Software-Agenten

Der Begriff des Agenten hat in unserem Sprachgebrauch mehrere Bedeutungen. So spricht man von dem *Geheimagenten*, der seinen Martini geschüttelt und nicht gerührt trinkt. Oder von dem *Musik-Agenten*, der für einen bekannten Künstler die Auftritte und Tourneen plant. Auch der Kaufmann im Reisebüro, der für den Kunden zum Beispiel eine Städtereise durch Italien zusammenstellt, wird im englischen als *Travel Agent* bezeichnet. Und nicht zuletzt findet sich im Computerbereich eine weitere Bedeutung von Agent: Dem *Netzwerk-Agenten*, der für seinen Administrator das Netzwerk überwacht und bei Problemen Alarm schlägt. Aber alle diese Agenten haben nur wenig mit den *Software-Agenten* zu tun, mit denen sich diese Arbeit beschäftigt.

Auch wenn es verschiedene Definitionen über Software-Agenten gibt, ist es am vielversprechensten, Agenten über ihre Charakteristika zu beschreiben. So muss ein Software-Programm mindestens die nachfolgenden sechs Eigenschaften erfüllen, um als Software-Agent zu gelten:

- **Autonomie**: Agenten können ohne direktes Eingreifen eines Anwenders agieren und ihre Aufgaben erfüllen.
- **Soziale Fähigkeiten**: Besonders wenn Agenten mit dem Anwender kommunizieren, müssen sie soziale Kompetenz besitzen.
- **Reaktives Verhalten**: Agenten beobachten ständig ihre Umwelt und reagieren zeitnah auf Veränderungen.
- **Proaktives Verhalten**: Agenten können ihre Umwelt auch selber beeinflussen, um den gewünschten Zustand herzustellen.
- **Kontinuität**: Agenten bleiben über sehr lange Zeit hinweg aktiviert.
- **Zielorientiertheit**: Agenten brauchen ein klares Ziel, welches sie erfüllen sollen. Hier ist besonders rationales Handeln nötig.

Darüberhinaus gibt es noch weitere Eigenschaften, die wünschenswert sind. So sollen Agenten selbständig ihre aktuelle Umgebung verlassen können und zu anderen Rechnern migrieren können.

Wichtig ist auch, dass sie die aufgetragenen Aufgaben erfüllen und dabei Charakter zeigen. Nur so vertrauen Anwender dem Agenten. Ebenso wichtig sind die Fähigkeiten zum Schlussfolgern, Lernen und zur sinnvollen Zusammenarbeit.

1.1 Architekturen

Für den inneren Aufbau der Software-Agenten gibt es drei grundlegende Architekturen: *Deliberativ*, *reaktiv* und *hybrid*. Während die deliberativen Agenten über ein *Planungs-* und *Schlussfolgerungssystem* verfügen, fehlen bei den reaktiven Agenten sämtliche aus der Künstlichen Intelligenz bekannten Methoden. Beide Modelle haben ihre Vorteile, jedoch auch erhebliche Nachteile. Die hybride Architektur vereinigt die beiden anderen Architekturen in einer.

Deliberative Agenten werden häufig auch als BDI-Agenten bezeichnet: Sie haben *Überzeugungen* (englisch beliefs), *Wünsche* (englisch desires) und *Intentionen* (englisch intentions). Stellt ein BDI-Agent eine Umweltveränderung fest, so greift er auf sein *inneres Modell* der Umwelt zu und passt dieses an die neue Umweltsituation an. Im Anschluss kann der Agent die für ihn beste Reaktion generieren. Die erweiterte BDI-Struktur besteht nun aus den Komponenten:

- **Überzeugungen** spiegeln die grundlegenden Ansichten und Erwartungen des Agenten wider.
- **Wünsche** sind die Überzeugungen, dessen Erfüllung sich der Agent wünscht.
- **Ziele** sind die Wünsche des Agenten, die er selber zu verwirklichen versucht.
- **Intentionen** sind kleine, geplante Schritte, um die Ziele zu erfüllen.
- **Pläne** sind mehrere Intentionen, die zu konsistenten Einheiten kombiniert sind.

Werden diese Erkenntnisse zu einer konkreten Architektur deliberativer Agenten kombiniert, so entsteht die in Abbildung 1 dargestellte Struktur. Dabei hat der *Scheduler* die Aufgabe, die vom *Planer* erstellten Pläne zum passenden Zeitpunkt zu starten — etwa dann, wenn genügend Ressourcen vorhanden sind. Der *Ausführer* ist dann für die

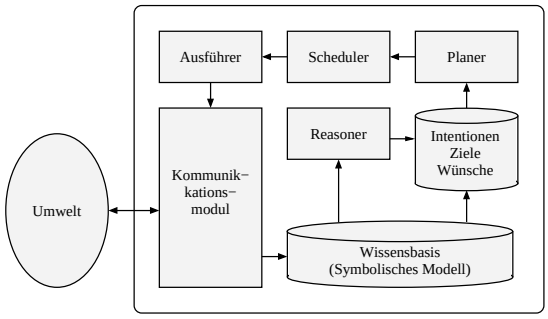

Abbildung 1: Architektur deliberativer Agenten

konkrete Umsetzung der Intentionen zuständig. Scheitert dieser, so muss die Aufgabe zurück an den Scheduler oder gar Planer gegeben werden. Diese müssen dann die neue Umweltsituation verarbeiten und die Aufgabe modifizieren. Da jede Reaktion auf ein Ereignis diesen aufwändigen Prozess durchläuft, kann sich die Umwelt bereits geändert haben, bevor die erste Reaktion beim Ausführer angekommen ist.

Anders bei den **reaktiven Agenten**, die sofort auf ein Ereignis reagieren. Denn sie besitzen weder einer Planer, noch einen Scheduler oder Ausführer. Auch haben sie kein inneres Modell und höchstens *implizite*, *fest definierte* und *unveränderliche* Ziele. Reaktive Agenten erstellen folglich ihre Reaktionen nur aufgrund des aktuellen Ereignisses. Andere Daten stehen ihnen nicht bereit. Dennoch gibt es Belege dafür, dass diese einfache Architektur dennoch zu hoher Intelligenz bemächtigt. Beispielsweise verfügt eine einzelne Ameise über nur sehr wenig Intelligenz. Dennoch handelt der gesamte Ameisenstaat intelligent. In der Robotik gibt es beispielsweise die Subsumption-Architektur von Brooks [1], welche mehrere, hierarchisch angeordnete Kompetenz-Module enthält. Jedes Modul kann dabei die Ein- und Ausgabe der darunterliegenden manipulieren.

Hybride Agenten vereinigen nun beide Architekturen in einer. Dabei sind die reaktiven Agenten für Aufgaben zuständig, bei denen keine Planung notwendig ist. Beispielsweise können häufig auftretenden Situationen über einen einfachen und schnellen Algorithmus abgewickelt werden. Erst wenn längerfristige Entscheidungen notwendig sind, muss der deliberative Teil aktiv werden. In der Interrap-Architektur von Müller [2] gibt es

zudem noch eine kooperative Planungsschicht für den Fall, dass mit anderen Agenten kommuniziert und kooperiert werden muss. Die Entscheidung, welche Aufgaben von welcher Architektur übernommen wird, hängt stark von der gegeben Anwendung ab. Typischerweise erledigt die reaktive Architektur die gesamte Kommunikation und die deliberative Architektur die restlichen, komplexeren Aufgaben.

1.2 Mobile Agenten

Software-Agenten lassen sich für unterschiedlichste Aufgaben eingesetzten. Beispielsweise sollen sie Informationen aus einer umfangreichen Datenbank filtern, CPU-intensive Berechnungen durchführen oder mit anderen Agenten auf Marktplätzen verhandeln. So kann es vorteilhaft sein, den Agenten zum Ort des Geschehens zu übertragen — etwa zur Datenbank, zum leistungsstarken Rechner oder zum Marktplatz zu transportieren. Besonders im Bereich des *Mobile Computing* ist diese Möglichkeit von hohem Nutzen. Typischerweise verfügen mobile Geräte, wie beispielsweise PDA's, nur geringe Bandbreite, Rechenleistung und Hauptspeicher. Auch sind sie nicht permanent mit einem Netzwerk verbunden. Mobile Agenten könnten das Gerät verlassen und zu geeigneten Rechnern migrieren. Daneben lassen sich dank mobiler Agenten Nachrichten aktiv machen. Statt einer reinen Text-Nachrichten, werden Agenten mit der Botschaft zum Empfänger geschickt. Dort kann er beispielsweise den Text an die Zielgruppe anpassen oder Multimedia-Präsentationen synchronisieren.

Allerdings birgt der mobile Code auch Nachteile. So ist es nicht immer effizient, einen ganzen Agenten zu verschicken und auf einem anderen Rechner migrieren zu lassen. Um beispielsweise eine Web-Seite abzufragen ist es noch immer effizienter, diese per Client-Server-Modell herunterzuladen. Daneben ist die hohe Komplexität zu nennen. Denn ist ein Agent einmal abgeschickt, so lässt er sich kaum noch kontrollieren. Erst wenn sich der Agent wieder beim Heim-System meldet, kann er neue Anweisungen entgegen nehmen. Folglich können Programmierfehler folgenschwere Probleme erzeugen, da der Agent nicht einfach terminiert werden kann.

Das größte Problem mobiler Agenten betrifft jedoch die geringe Sicherheit. So kann erstens ein *Agenten-System* bedroht werden, wenn ein Agent vertrauliche Daten liest oder eine Denial-of-Service-Attacke startet.

Zweitens kann das Computer-Netzwerk von mobilen Agenten bedroht werden. So könnte sich ein Agent kontinuierlich replizieren und über das Netzwerk transportieren. Dadurch würde dieses geflutet und folglich überlastet werden.

Als Drittes kann ein Agent bedroht werden. So könnte ein anderer Agent mit diesem kommunizieren und so versuchen, ihn negativ zu beeinflussen. Beispielsweise könnten so vertrauliche Daten entlockt werden oder aber falsche Börsenmeldungen verbreitet werden. Auch Dritte könnten den Agenten bedrohen, indem sie beispielsweise Nachrichten verändern oder unterdrücken. Die aber schwerwiegendste Gefahr kommt vom Agenten-System. Dieses stellt alle benötigten Ressourcen und kann diese auch kontrollieren. So hat das System Einblick in den Code- und Datenbereich des Agenten. Damit kann es den Agenten sowohl umprogrammieren als auch die transportierten Geheim-Schlüssel extrahieren. Elektronisches Geld ist ebenfalls eine leichte Beute eines kriminellen Agenten-Systems.

2 Multi-Agentensysteme

Nachdem der erste Abschnitt über einen *einzelnen* Agenten handelte, rückt nun die Kooperation zwischen *mehreren* Agenten in den Vordergrund. Es geht nicht mehr darum, *einen* Agenten möglichst intelligent zu machen. Vielmehr existieren *mehrere* Agenten, die jeweils auf ein Thema spezialisiert sind. Brauchen sie Wissen aus einem anderen Themen-Gebiet oder Mithilfe bei einer Aufgabe, so müssen sie mit anderen Agenten *kommunizieren*, *kooperieren* und sich *koordinieren*. Zuvor müssen jedoch die Bedingungen für eine Kooperation festgelegt werden.

2.1 Verteiltes Problemlösen

Im Bereich des Verteilten Problemlösens arbeiten mehrere Agenten an einer großen, komplexen Aufgabe. Ein Agent alleine könnte sie nicht lösen, da diese beispielsweise seine Fähigkeiten oder

sein Wissen übersteigt. Aus diesem Grund kommen die Strategien des *Task Sharings* und des *Result Sharings* zum Einsatz.

Beim **Task Sharing** wird ein großes Problem in viele kleine, leichter zu lösende Teilprobleme aufgeteilt. Anschließend werden *passende* Agenten zu diesen Teilproblemen gesucht und mit der Lösung beauftragt. Jeder dieser Agenten erledigt seine Aufgabe, wobei er sein Teilproblem in weitere Sub-Probleme unterteilen kann. In der letzten Phase werden alle Teillösungen zu einer Gesamt*lösung* kombiniert, wodurch die Gesamt*aufgabe* gelöst ist. So lässt sich mit dem Task Sharing beispielsweise das (ideale) Problem der *Türme von Hanoi* in logarithmischer Zeit lösen — im Unterschied zur exponentiellen Komplexität auf herkömmliche Weise.

Auch das **Result Sharing** bietet zahlreiche Vorteile. Statt Aufgaben werden Ergebnisse mit denen der anderen Agenten geteilt. Dadurch erhalten die Agenten Gewissheit, dass ihre eigene Lösung mit denen der anderen übereinstimmt und folglich korrekt sein könnte. Auch lassen sich dadurch Lösungen vervollständigen, wenn zuvor nur mit unzureichenden Daten gerechnet werden musste. Dringend benötigte Daten müssen geschätzt werden und sind folglich nur ungenau. Durch das Result Sharing würden die geschätzten Daten mit der Zeit durch die exakten Werte ersetzt. Letztendlich lässt sich noch Zeit sparen, wenn eine benötigte Berechnung bereits durch einen anderen Agenten vorgenommen wurde.

Problem ist aber, dass alle Ergebnisse an die interessierten Agenten zu übermitteln sind. Auch müssen die Ergebnisse der anderen Agenten integriert werden, was nicht immer sinnvoll ist — dafür aber aufwändig. Deshalb muss genau abgewägt werden, welches Ergebnis zu teilen ist und welches nicht.

Ein weiteres Problem betrifft die Kommunikation zwischen den Agenten. Die Frage ist nämlich, wie ein Agent die zu lösende Teilaufgabe beschreiben kann. Hierfür werden **Sprachen** zur Wissensrepräsentation eingesetzt, wie *KIF*, *RDF/XML* oder das klassische *LISP*. Neben einer verständlichen Sprache muss der Agent noch die *Bedeutung* jedes Wortes kennen. Eine **Ontologie** sorgt dabei für ein gemeinsames Verständnis. Diese Informationen samt der eigentlichen Nachricht werden in der *Knowledge Query and Manipulation Language* (KQML) oder der *FIPA-ACL* in einem Daten-Paket vereinigt und mit obligatorischen Informationen versehen, wie Empfänger, Absender und Art der Anfrage.

2.2 Kooperationen

Aus dem Gebiet des Verteilten Problemlösens sind einige Strategien hervorgegangen, wie mehrere Agenten miteinander zusammenarbeiten können. Die zwei wichtigsten Kooperationsstrategien sind das *Kontraktnetz-Protokoll* und das *Partitial Global Planning*. Während beim **Kontraktnetz-Protokoll** die Aufgaben per öffentlicher Ausschreibungen verteilt werden, gibt es beim **Partial Global Planning** einen gemeinsamen Plan, der alle Aktivitäten der einzelnen Agenten enthält.

Es hat sich jedoch gezeigt, dass kooperative Agenten ein schlechteres Ergebnis liefern, als unkooperative Agenten. Diese handeln im hohen Maße egoistisch und sind nur daran interessiert, ihre eigenen Ziele zu verwirklichen. Deshalb werden unkooperative Agenten auch als **egoistischen Agenten** bezeichnet. Damit ein Agent jedoch *intelligent* handelt, muss er zusätzlich noch streng rational handeln: Er muss seinem Ziel immer einen Schritt näher kommen.

Ein Beispiel soll die Vorteile der egoistischen Agenten verdeutlichen. Dazu dienen zwei Menschen, die in der Wüste gestranden sind und nur noch wenig Trinkwasser haben. Kooperative Agenten würden das verbliebene Wasser gerecht untereinander aufteilen. Jedoch könnte es dadurch sein, dass *beide* kurz vor der rettenden Oase verdursten. Egoistische Agenten würden hingegen nur dann das Wasser teilen, wenn ihnen dadurch ein Nutzen entstünde. Kennt beispielsweise nur der zweite Agent den Weg zur Oase, so hat der erste ein starkes Interesse daran, dass dieser überlebt. Zusätzlich könnte noch brücksichtigt werden, dass beide Agenten die Wasserstelle nicht erreichen würden, da ihnen zuvor das Wasser ausginge. Ein Agent *alleine* könnte es jedoch schaffen und anschließend Hilfe für den anderen anfordern und beide könnten ihre Ziele noch verwirklichen. Der Unterschied zu kooperativen Agenten ist nun, dass diese niemals solche komplexen Überlegungen anstellen, sondern nur das Wasser ‚gerecht' aufteilen.

2.3 Strategische Verhandlungen

Damit Agenten sowohl egoistisch als auch rational handeln können, müssen sie sich zuvor über ihre möglichen Aktionen im Klaren sein. Nur so kann ein Agent mit einem für ihn guten Ergebnis aus der Verhandlung herauskommen.

Treffen nun mehrere Agenten aufeinander und wollen miteinander kooperieren, so müssen sie zuvor alle Bedingungen der Synergie *verhandeln*. Dabei versucht jeder Teilnehmer das für ihn beste Verhandlungsergebnis zu erzielen. Die Grundlage, wie verschiedene Verhandlungsergebnisse bewerten werden können, bildet dabei die **Spieltheorie**.

Ein Beispiel hierfür ist das *Hostage Crisis Scenario* nach [3], bei dem ein Flugzeug mit Geiseln entführt wird. Nun stehen den beteiligten Parteien mehrere Optionen zur Verfügung, wie sie reagieren sollen. Beispielsweise könnte die Regierung mit den Terroristen *verhandeln* oder die Maschine *stürmen*. Die Entführer könnten ihrerseits *verhandeln* oder das Flugzeug *sprengen*. All diese Entscheidungen lassen sich mit Hilfe eines Entscheidungsbaums darstellen, wie in Abbildung 2 gezeigt.

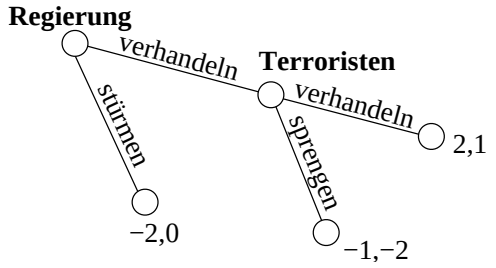

Abbildung 2: Entscheidungsbaum

Jede Verhandlungssituation lässt sich auf das in der Abbildung gezeigte synchrone Protokoll zurückführen: Es kann immer nur eine Partei zu einem Zeitpunkt handeln. Die anderen Teilnehmer können darauf nur reagieren. In der Abbildung 2 beginnt die Regierung zu taktieren und kann beispielsweise versuchen zu *verhandeln*. Die Terroristen könnten darauf mit *sprengen* oder *verhandeln* reagieren.

An den Blättern des Entscheidungsbaums befinden sich die **Erlöswerte** (englisch payoffs) für jedes Handlungs-Ablauf. Ist beispielsweise die Regierung *verhandlungsbereit* und die Entführer entscheiden sich für *sprengen*, so ist das Ergebnis für die Regierung -2 und für die Entführer -1.

Neben der Darstellung als *Entscheidungsbaum* lassen sich auch nur die *Erlöswerte* betrachten, wie in der Tabelle 3 dargestellt. Für *automatisch*

		Regierung	
		verhandeln	stürmen
Terroristen	verhandeln	2,1	-2,0
	sprengen	-1,-2	0,-3

Abbildung 3: Erlöswerte

verhandelnde Agenten ist besonders das Nash-Gleichgewicht wichtig [4]. Eine Verhandlungsstrategie s ist dann im **Nash-Gleichgewicht**, wenn die Annahme gilt: Wählt ein Agent a die Strategie s, so ist es für einen Agenten b immer schlechter, eine andere Strategie außer s zu wählen. Im *Hostage Crisis Scenario* ist die Strategie (verhandeln,verhandeln) im Nash-Gleichgewicht. Allerdings ist es für die Terroristen günstiger, die Maschine zu *sprengen*, wenn die Regierung *stürmt*.

Mit der Spieltheorie ist es somit möglich, eine optimale Verhandlungsstrategie zu finden. Gleichzeit werden die alternativen Aktionen bewertet. Wird nun noch eine Grenze definiert, die nicht unter- bzw. überschritten werden darf, so ist ein Raum definiert, in dem der Agent einer Einigung zustimmt. Selbstverständlich versucht der Agent zuerst, die für ihn beste Lösung zu erzielen.

Das eigentliche **Verhandlungsprotokoll** basiert auf dem Modell der abwechselnden Angebote von Rubinstein [5]. Dabei kann zu einem Zeitpunkt t immer nur ein Agent ein Angebot unterbreiten. Alle anderen teilnehmenden Agenten können dieses Angebot *annehmen*, *ablehnen* oder aus der Verhandlung *aussteigen*. Haben alle Teilnehmer das Angebot angenommen, so wurde zum Zeitpunkt t das Ergebnis s erreicht. Lehnt auch nur ein Agent das Angebot ab, so kommt zum Zeitpunkt t keine Einigung zustande. Nun macht ein anderer Agent ein Gegenangebot zum Zeitpunkt $t + 1$. Steigt nur ein Agent aus der Verhandlung aus, so gilt diese als gescheitert. In diesem Fall endet die Verhandlung so, wie zuvor für den Konfliktfall definiert. Bei der *Hostage Crisis* wäre diese, dass die Maschine *gestürmt* und *gesprengt* wird.

Für Verhandlungen sind zudem noch vier Re-

geln wichtig, an die sich jeder Agent halten muss:

1. Jeder Agent handelt rational. Das bedeutet, dass jeder Teilnehmer versucht, seinen Nutzen entsprechend seiner Präferenzen zu maximieren.

2. Agenten vermeiden es, aus einer Verhandlung auszusteigen. Ist es für einen Teilnehmer gleich gut, aus einer Verhandlung auszusteigen oder die Offerte abzulehnen, so entscheidet er sich immer für *ablehnen*.

3. Die Agenten honorieren die Ergebnisse der Verhandlung und halten sich daran.

4. Jede Verhandlung steht für sich selbst. Es können keine Zusagen für zukünftige Handlungen gemacht werden.

Im Laufe einer Verhandlung können sich die Erlöswerte ändern und damit auch der Zustimmungsraum, in dem der Agent einer Einigungen zustimmen würde. Diese Veränderungen lassen sich über eine **Utility-Funktion** darstellen. Beispielsweise könnte die Utility-Funktion angeben, dass mit zunehmendem Zeitpunkt t der Erlös sinkt oder aber steigt. Verkauft ein Agent beispielsweise ein verderbliches Gut, so ist er daran interessiert, schnellstmöglich eine Einigung zu erzielen. Neben diesen *fixen Gewinnen/Verlusten pro Zeit* gibt es noch die drei Kategorien *fixer Diskontsatz*, *Finanzsystem mit Verzinsung* und *Begrenzung mit fixen Verlusten pro Zeit*.

Jeder Agent kennt dabei seine Utility-Funktion und hat zumindest eine Vermutung über die Funktion der anderen Agenten. Aufgrund dieser Annahmen kann der Agent die Verhandlungen positiv für sich beeinflussen, da er beispielsweise die Einigung hinauszögern kann, wenn es für ihn von Vorteil ist.

3 Ausblick

Software-Agenten sind ein weiterer, vielversprechender Weg, um Computer intelligent zu machen. Die Einsatzmöglichkeiten sind dabei weitreichend. Beispielsweise kann ein persönlicher Einkaufsagent seinem Anwender Kaufvorschläge entsprechend seiner Präferenzen unterbreiten. Hat nun ein anderer Anwender mit ähnlichen Präferenzen ein neues Produkt erworben und positiv bewertet, so dürfte es auch für den anderen Anwender interessant sein. Somit hat der Agent eine Grundlage für seine Kaufvorschläge.

Darüberhinaus kann ein Software-Agent für seinen Anwender die Kauf- oder Verkaufsverhandlungen übernehmen. Beispielsweise für Online-Auktionen eignen sich Verhandlungs-Agenten sehr gut. Diese bieten auf einer Auktion für ein Produkt mit, bis die vom Anwender gesetzte Preisobergrenze erreicht ist. Ebenso kann der Agent versuchen, in mehreren Auktionshäusern für genau ein Produkt zu bieten.

Desweiteren helfen Software-Agenten auf virtuellen Marktplätzen und Business-to-Business (B2B) Portalen Personalkosten einzusparen. Auf solchen Marktplätzen treffen sich derzeit noch menschliche Operatoren, um den günstigsten Anbieter oder zahlungsfreudigsten Abnehmer zu finden. Die Automobil-Industrie spart heute durch sogenannte Branchen-Portale wie Covisint bis zu tausend Dollar pro Fahrzeug ein. Agenten könnten zukünftig die Aufgaben der Operatoren übernehmen und selbständig über Preise und Konditionen verhandeln. Zudem könnten sie mit anderen Einkäufern kooperieren und gemeinsam eine größere Menge abnehmen, wodurch sich bessere Einkaufspreise erzielen liesen.

Literatur

[1] Rodney A. Brooks. Intelligence without representation. Number 47 in Artificial Intelligence, pages 139–159. 1991.

[2] Jürgen Müller. *The Design of Intelligent Agents: a Layered Approach*. Springer Verlag, Berlin, Heidelberg, 1996.

[3] Sarit Kraus and Jonathan Wilkenfeld. A strategic negotiations model with applications to an international crisis. Technical report, 1993. UMIACS TR 90–20, CS TR 2407.

[4] John F. Nash. Two-person cooperative games. In *Econometrica*, volume 21, pages 129–140, 1953.

[5] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50:97–109, 1982.