Karl R. Brendel

Parallele oder sequentielle Simulationsmethode?

Implementierung und Vergleich anhand eines Multi-Agenten-Modells der Sozialwissenschaft

Herbert Utz Verlag · München

Informatik

Band 88

Zugl.: Diss., München, Univ., 2008

Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN 978-3-8316-4013-3

Printed in Germany
Herbert Utz Verlag GmbH, München
089-277791-00 · www.utzverlag.de

Inhalt

1 Einleitung

1.1 Motivation

Studiert man die gängige Fachliteratur der letzten Jahre zum Thema paralleles und verteiltes Rechnen, kann man den Eindruck gewinnen, dass der Vergleich von sequentiellen und parallelen Simulationssystemen ein wenig lohnenswertes Unterfangen ist. Der Turing-Tradition folgend, scheint es keinen großen qualitativen Unterschied zwischen sequentiellem und parallelem Rechnen zu geben. Als einzigen Vorteil könnte man in einigen Fällen eventuell einen Zuwachs in der Ausführungsgeschwindigkeit beobachten. In Penrose (1990), wird auf Seite 46 argumentiert:

> *"Using more than one Turing device in parallel action - which is an idea that has become fashionable in recent years, in connection with attempts to model human brains more closely – does not in principle gain anything (though there may be an improved speed of action under certain circumstance). Having two separate devices which do not directly communicate with one another achieves no more than having two which do communicate; and if they communicate, then, in effect, they are just a single device!"*

und weiter auf Seite 398 in Penrose (1990):

> *"There is no difference in principle between a parallel and a serial computer. Both are in effect Turing machines."*

Dennett schreibt in Dennett (1993) auf Seite 269 zu diesem Thema:

> *"Given all the ink that has been spilt over this theoretical [sequential vs. parallel] issue, it is important to stress that this is a shift in the balance of power, not a shift to some 'qualitatively different' mode of operation. At the heart of the most volatile pattern-recognition system ('connectionist' or not) lies a von Neumann machine, computing a computable function."*

Und im selben Werk auf Seite 217:

> *"Since any computing machine at all can be imitated by a virtual machine on a von Neumann machine, it follows that if the brain is a massively parallel processing machine, it too can be perfectly imitated by a von Neumann machine."*

Schließlich ist auf Seite 22 in Searle (1990) die folgende Aussage zu finden:

> *"The parallel, ‚brainlike' character of the processing, however, is irrelevant to the purely computational aspects of the process. Any function that can be computed on a*

> *parallel machine can also be computed on a serial machine. Computationally, serial and parallel systems are equivalent: any computation that can be done in parallel can be done in serial."*

Aufgrund der obigen Zitate kann man schnell zu dem Schluss kommen, dass eine Multiagentensimulation auf parallelen und/oder verteilten Computersystemen kaum Vorteile bringen wird. Auf der einen Seite ist jede Computersimulation eine Art von Berechnung, auf der anderen Seite kann jede parallel auszuführende Berechnung anscheinend problemlos auf eine sequentielle Berechnung zurückgeführt werden.

Tatsächlich werden heute vor allem im Bereich der Sozialwissenschaften fast alle existierenden Simulationsprogramme in einer sequentiellen Art und Weise auf einem Einprozessorsystem ausgeführt. Obwohl es schon seit vielen Jahren parallele und verteilte Computersysteme gibt, wurde diese Art von Rechnern von der sozialwissenschaftlichen Simulationsgemeinschaft noch nicht in großem Stil eingesetzt. Es ist anzunehmen, dass dies nicht aufgrund von wissenschaftlichen Überlegungen sondern aufgrund der praktischen Tatsache erfolgt ist, dass Wissenschaftler in den Sozialwissenschaften einfache und weit verbreitete Rechnersysteme verwenden wollen, die nur ein geringes Maß an technischer Unterstützung benötigten. Typischerweise sind Sozialwissenschaftler keine Computerprogrammierer, sie bevorzugen Simulations-Toolkits und fertige „Out-of-the-Box"-Lösungen. Erst in jüngster Zeit, mit der zunehmenden Benutzung des Internets, und mit ersten allgemein verfügbaren Client/Server-Systemen, scheinen sich die Sozialwissenschaftler der Möglichkeiten des verteilten Rechnens bewusst zu werden.

Ziel dieser Arbeit ist es, gerade die Möglichkeiten und Unterschiede einer parallelen im Vergleich zu einer sequentiellen Multiagentensimulation herauszufinden und aufzuzeigen. Hierbei werden die beiden Simulationsmethoden im Rahmen eines Simulationsmodells aus den Sozialwissenschaften auf zwei unterschiedlichen Simulationssystemen abgebildet und verglichen. Da sich die Untersuchung der sequentiellen und der parallelen Multiagentensimulation in dieser Arbeit immer auf die konkrete Implementierung eines Modells bezieht, werden im weiteren Verlauf die Begriffe Multiagentensimulation und Multiagentensystem oft synonym verwendet.

In der wissenschaftlichen Literatur kann derzeit noch kein Ansatz gefunden werden, wie Methoden auf eine theoretische und systematische Art und Weise verglichen werden können (siehe Balzer (1997) und Balzer (2002) für erste Ansätze zu

einem systematisch/theoretischen Methodenvergleich). Daher wird in dieser Arbeit ein empirischer und experimenteller Vergleich durchgeführt.

1.2 Aufbau der Arbeit

In Kapitel 1 wird eine kurze Einführung in die Hauptthemen der Arbeit gegeben. Die Simulation mit zellulären Automaten und der Einsatz von Softwareagenten werden als Grundlagen für die in dieser Arbeit eingesetzten Simulationsmethoden vorgestellt. In diesem Zusammenhang wird in einem kurzen Überblick auf aktuell verfügbare Multiagentensystemplattformen und deren Eignung für parallele/verteilte Simulationen eingegangen.

Als Basis für die Darstellung der in dieser Arbeit verwendeten Simulationssysteme, ist ein Abschnitt mit informationstechnischen Grundlagen (Kapitel 2) eingeschoben. Hier sind allgemeinen Informationen zu parallelen Computersystemen und deren Programmierung zu finden. Auf die teilweise sehr technischen Details wird innerhalb des Dokuments immer wieder referenziert.

In Kapitel 3 wird das Simulationsmodell vorgestellt, das den in dieser Arbeit durchgeführten Simulationen zugrunde liegt. Es handelt sich hierbei um das weithin bekannte Modell zur Entstehung von Solidarnetzwerken von Hegselmann (1997).

Die beiden zum Methodenvergleich verwendeten Simulationssysteme werden in Kapitel 4 beschrieben. Zu Beginn wird eine sequentielle Multiagentensimulation konzipiert und auf einem Standardcomputer implementiert. Da das hierfür eingesetzte Simulationssystem *SMASS*[1] bereits in Balzer (2000) detailliert beschrieben ist, wird hier vor allem auf die Aspekte der Abbildung des Simulationsmodells eingegangen. Der Schwerpunkt liegt in diesem Kapitel auf *DMASS*[2], einem neu erstellten Simulationssystem das auf einem massiv parallelen Rechnersystem zum Einsatz kommt. Basierend auf den Systemeigenschaften dieses Parallelcomputers wird untersucht, wie die sequentielle in eine parallele Multiagentensimulation transformiert werden kann. Mit einem standardisierten Verfahren werden die bereits vorhandenen Systemkomponenten der sequentiellen Multiagentensimulation auf die parallele Architektur überführt. Die Notwendigkeit von zusätzlichen, speziell

[1] Sequential Multi-Agent System for Social Simulations.

[2] Distributed Multi-Agent System for Social Simulations.

für die parallele Multiagentensimulation benötigten Konzepten und Mechanismen werden im Zusammenhang mit deren Umsetzung erläutert. Die Beschreibung der Simulationssysteme reicht hierbei von grundlegenden Prinzipien über notwendige technische Details bis hin zur Darstellung der Softwarearchitekturen. Sie wird meist anhand von Quellcodefragmenten durchgeführt, eine formal logische Darstellung erfolgt nicht.[3]

In einem letzten Schritt werden beide Simulationssysteme in Hinblick auf ihre Eigenschaften zur Laufzeit untersucht indem mit ihnen die in Kapitel 5 dargestellten Simulationen durchgeführt werden. Das Kapitel beginnt mit einer Beschreibung der verwendeten Methoden zur Protokollierung, Darstellung und Auswertung der Simulationen. Es folgt ein Abschnitt über die Vorbereitung und Durchführung einzelner Simulationsläufe. Die Darstellung der verwendeten Simulationsszenarien, der darin durchgeführten Simulationsläufe und der erzielten Simulationsergebnisse beschließt das Kapitel.

In Kapitel 6 werden die mit den beiden Simulationsmethoden gemachten Erfahrungen untereinander verglichen und als Ergebnisse festgehalten. In einem ersten Teil werden die methodisch/konzeptionellen Aspekte der Ergebnisse beschrieben, in einem zweiten Teil wird auf deren Relevanz für den Bereich der Sozialwissenschaft eingegangen.

Die Hauptkapitel werden durch einen umfangreichen Anhang ergänzt. In Anhang A wird die Methode zur numerischen Auswertung der durchgeführten Simulation detailliert dargestellt. Anhang B dokumentiert den Einsatz und die Bedienung des DMASS-Simulationssystems. Die dort zu findenden Ausführungen und Abbildungen sollen zu einem erweiterten Verständnis der Funktionsweise des parallelen Multiagentensystems beitragen. Hinweise zur Programmcodedokumentation werden in Anhang C gegeben. Der für SMASS verwendete Quellcode ist in Anhang D aufgeführt. Der für DMASS verwendete Quellcode schließt sich im Anhang E an.

Ein Index und das Literaturverzeichnis sind am Ende des Buches zu finden.

Sowohl für das gedankliche Nachvollziehen einzelner Funktionalitäten in Kapitel 4 und Kapitel 5 als auch zum Studium des im Anhang beigefügten Programmcodes der Simulationssysteme werden Grundkenntnisse in der Programmiersprache *Prolog* vorausgesetzt (siehe z.B. Clocksin und Mellish (1987) oder Sterling und Shapiro (1987) für eine Einführung in die Programmiersprache Prolog).

[3] Siehe Fisher (1994a) und Fisher (1994b) für eine Darstellung von Multi-Agenten Systemen im Rahmen der temporalen Logik von MetateM und Concurrent MetateM.

1.3 Sozialwissenschaftliche Simulationsmethoden

Auf der Suche nach Antworten zu sozialwissenschaftlichen Fragestellungen sind im Laufe der Zeit viele unterschiedliche Simulationsmethoden etabliert und eingesetzt worden. Ein aktueller Überblick über Simulationsmethoden für die Sozialwissenschaften ist in Gilbert und Troitzsch (2005) zu finden. In diesem Kapitel wird kurz auf diejenigen Themenbereiche eingegangen, die für diese Arbeit relevant sind:

- Die Simulation mit zellulären Automaten.
- Simulationen mit Softwareagenten.
- Multiagentensimulationen auf parallelen/verteilten Computersystemen.

1.3.1 Simulation mit zellulären Automaten

Zellularautomaten wurden bereits um 1940 von Stanislaw Ulam in Los Alamos vorgestellt. John von Neumann griff die Idee auf und erweiterte sie zu einem universellen Berechnungsmodell. Er beschrieb damit als erster einen Zellularautomaten, der berechnungs- und konstruktionsuniversell ist.

Anfang der 1970er Jahre gelangte John Conways Game of Life (siehe Gardner (1970)) zu Berühmtheit. Mit einem zweidimensionalen zellulären Automaten und einfachen Regeln konnten verblüffende Strukturen erzeugt werden. In neuerer Zeit hat sich unter anderem Stephen Wolfram die Erforschung von Zellularautomaten zum Ziel gesetzt und dazu zahlreiche Bücher und Artikel veröffentlicht. Einen umfangreichen Einblick in die Welt der zellulären Automaten gibt Wolfram (2002).

In den folgenden Kapiteln werden grundlegende Eigenschaften von Simulation mit zellulären Automaten vorgestellt.

1.3.1.1. Diskreter Raum

Eine der grundlegenden Eigenschaften zellulärer Automaten ist die Anordnung aller Zellen in einem regulären n-dimensionalen Gitter. Diese zugrunde gelegte Gitterstruktur bildet einen diskreten Raum, der als Spielfeld verwendet wird. Abb. 1 zeigt ein Beispiel für ein 2-dimensionales Gitter:

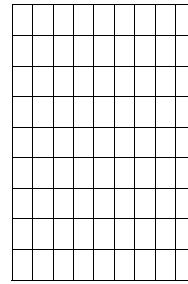

Abb. 1: Zellen in 2-dimensionaler Darstellung.

Im Vergleich zu Zellen in der Mitte einer 2-dimensionalen Gitteranordnung besitzen Zellen am äußeren Rand eine geringere Anzahl von Nachbarzellen. Um diesen Unterschied auszugleichen werden die entsprechenden Enden des 2-dimensionalen Gitters zusammengefügt. Auf diese Weise erhält man ein 3-dimensionales Gebilde Torus, bei dem alle Zellen über die gleiche Anzahl von Nachbarzellen verfügen:

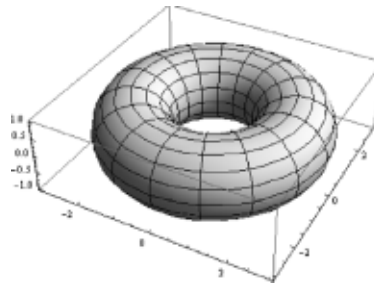

Abb. 2: Zellen auf einem Torus.

1.3.1.2. Lokalität

Eine weitere Eigenschaft von zellulären Automaten ist Lokalität. Das bedeutet, dass alle Interaktionen nur innerhalb einer fest definierten Nachbarschaft stattfinden und die Zellen ihre Zustände internen Regeln entsprechend ändern. Obwohl man sich eine große Anzahl von unterschiedlichen Gitterstrukturen und Nachbarschaften vorstellen kann, sind in der Simulationsszene vor allem die sog. *Von-Neumann*- und *Moore*-Nachbarschaften zu finden (siehe Abb. 3). Beide Nachbarschaften

lassen sich sowohl auf zwei-dimensionale Gitterstrukturen als auch auf einen Torus anwenden. Für die Von-Neumann-Nachbarschaft sind die nördlichen, südlichen, östlichen und westlichen Zellen als Nachbarn definiert. In der Moor-Nachbarschaft kommen die diagonalen Zellen in nordöstlicher, nordwestlicher, südöstlicher und südwestlicher Richtung hinzu. Laut Definition ist die mittlere Zelle immer Teil ihrer eigenen Nachbarschaft. Bei einer Reichweite von einer Zelle umfasst die Von-Neumann-Nachbarschaft 5 Zellen, die Moor-Nachbarschaft 9 Zellen. Die Größe einer Nachbarschaft ist nur durch die Dimension des verwendeten Spielrasters limitiert. Obwohl die meisten Simulationen mit zellulären Automaten auf Von-Neumann- und Moor-Nachbarschaften mit regulären Gitterstrukturen basieren, ist auch die Verwendung anderer Raster möglich (siehe Flache und Hegselmann (2001) für eine Untersuchung von zellulären Automaten auf irregulären Gitterstrukturen).

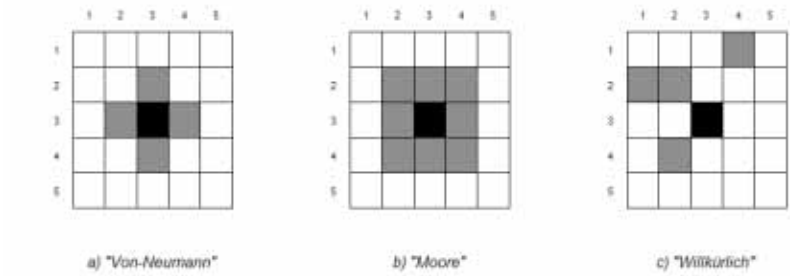

Abb. 3: Verschiedene Nachbarschaftsarten.

1.3.1.3. Zustandsübergänge

In einem zellulären Automaten nehmen alle Zellen nur diskrete Zustände aus einer vorgegebenen, endlichen Zustandsmenge ein. Die Zustandsübergänge werden durch vorgegebene Regeln beschrieben, die entweder simultan oder in einer zufälligen Auswahl sequentiell auf jede Zelle innerhalb eines Simulationsschrittes angewandt werden.[4] Statt kontinuierlicher Zustandsübergänge werden Übergänge in diskreten Zeitschritten durchgeführt, in denen sich die Zellzustände „sprungartig“

[4] Siehe Hegselmann (1996) für eine ausführliche Diskussion der Frage von Artefakten im Kontext der synchronen und asynchronen Aktualisierung in zellulären Automaten.

ändern. Die Zustandsübergangsregeln gelten für alle Zellen gleichermaßen (*Homogenität*) und sind lokal, das heißt im Kontext einer wohl definierten Nachbarschaft beschrieben.

1.3.1.4. Zelluläre Automaten und Multiagentenmodelle

Interessante Anwendungen ergeben sich durch die Kombination von zellulären Automaten mit Multiagentenmodellen. Dabei wird die Umgebung mit ihren spezifischen Gegebenheiten für das Agentensystem als zellulärer Automat abgebildet. Auf diese Art und Weise wird z.B. im Sugarscape-Modell von Epstein und Axtell (1996) die Verteilung von Energieressourcen repräsentiert.

Auf den ersten Blick scheint die Verbindung eines zellulären Automaten mit einem Multiagentenmodell einen Widerspruch darzustellen. Auf der einen Seite wird ein zentraler Mechanismus zur Steuerung von Zellzuständen verwendet, auf der anderen Seite werden autonome Softwareagenten mit individuellen Charakteristiken modelliert.

Der vermeintliche Widerspruch löst sich auf, wenn die folgenden Punkte betrachtet werden:

- Für jede Zelle können Regeln definiert werden, mit denen die Zelle auf die Zustände in der Nachbarschaft reagieren kann.
- Der aktuelle Zustand einer Zelle und die Summe ihrer Regeln können alle relevanten Informationen zum Verhalten eines Agenten repräsentieren.
- Schließlich kann in einem weiteren Schritt komplett von der Zelle abstrahiert werden und deren Zustand als der Zustand eines Agenten angesehen werden, der auf einer betrachteten Zelle positioniert ist.

Das in dieser Arbeit verwendete Simulationsmodell geht die oben beschriebene Verbindung aus einem zellulären Automaten und einem Multiagentenmodell ein.

Im nächsten Kapitel wird detaillierter auf den Agentenbegriff eingegangen.

1.3.2 Simulation mit Softwareagenten

Das Konzept eines Softwareagenten ist sowohl in der „Verteilten Künstlichen Intelligenz“ (*VKI*) als auch in der allgemeinen Computerwissenschaft im Bereich der Agenten-orientierten Programmierung (*AOP*) zu einem wichtigen Bestandteil geworden (siehe z.B. Shoham (1993) und Huntbach und Ringwood (1999)). Von der Objekt-orientierten Programmierung (*OOP*) ausgehend wird hier ein Agent ganz

allgemein als ein abgeschlossenes Gebilde betrachtet, das im Idealfall eigenständig agieren und kommunizieren kann.

Hewitt (1977) erläutert sein Konzept vom selbständigen, interaktiven und dauerhaft agierenden Objekt, dem sog. *Actor*, mit einem atomaren inneren Zustand und der Möglichkeiten anderen ähnlichen Objekten auf Nachrichten zu antworten.

Als einer der ersten verwendet Minsky (1985) den Begriff *Agent*. Ein Agent wird hier jedoch mehr im Sinne eines mentalen Prozesses zur Konstituierung einer Intelligenz verstanden und entspricht somit nicht der meist verwendeten Bedeutung eines autonomen Agenten.

1.3.2.1. Was ist ein Agent?

Eine allgemeingültige Definition des Agentenbegriffs kann in der Literatur nicht gefunden werden. Die Nutzung des Agentenbegriffs ist zu weitläufig um in einer einheitlichen Definition zusammengefasst zu werden. Als Agenten werden einfache motorische Steuerungseinheiten ebenso wie kommunikationsfähige und Dialog führende Programme gesehen. Mit Sensoren in einer Umgebung agierende Systemeinheiten werden mit dem Agentenbegriff belegt wie dies auch für selbständig handelnde, wissensbasierte und teilweise auch mobile Systeme zutrifft. Häufig wird Agent nicht als alleinstehender Begriff, sondern zusammen mit Attributen wie *intelligent* oder *autonom* verwendet.[5] Franklin und Graesser (1997) untersuchten eine Vielzahl von Agentendefinitionen bezüglich ihrer Gemeinsamkeiten und leiteten daraus ab, was sie als „*the essence of being an agent*" betrachten:

> *„An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future".*

Wie sieht es aber mit normalen Programmen aus? Können diese nicht auch als Agenten betrachtet werden? Nach Franklin und Graesser könnte ein Gehaltsabrechnungsprogramm derart betrachtet werden, dass es die Umwelt, in der es eingesetzt wird, über die Eingabedaten wahr nimmt und mit Ausgabedaten darauf reagiert. Da aber dessen Ausgabedaten normalerweise keine Änderungen bewirken, die das Programm zu einem späteren Zeitpunkt wieder wahrnehmen kann, fällt es

[5] „Intelligent" beispielsweise in Crowston und Malone (1988), bzw. „autonom" in Ferguson (1992). Die IFAAMAS (International Foundation for Autonomous Agents und Multiagent Systems) enthält den Begriff autonomous in ihrem Namen, während das Standardisierungsgremium FIPA (The Foundation for Intelligent Physical Agents) das Attribut intelligent nutzt.

nicht in die Kategorie von Agenten. Auch die Anforderung, dass ein Agent seine Aktionen für einen gewissen Zeitraum ausübt, wird von einem Gehaltsabrechnungsprogramm nicht erfüllt. Dieses wird einmal aufgerufen, beendet daraufhin seine Aktionen und wartet darauf, dass es ein weiteres Mal aufgerufen wird. Nach Franklin und Graesser fallen die meisten der üblichen Programme aufgrund einer der beiden obigen Kriterien aus der Agentenklassifizierung heraus: „*All software agents are programs, but not all programs are agents*".

1.3.2.2. Klassifikation von Agenten anhand ihrer Eigenschaften

Verschiedene Einsatzgebiete und damit verbundene Anforderungen führen zu einer *Klassifikation von Agenten*. Anhand ihrer Eigenschaften können die unterschiedlichen Agenten differenziert werden. Zur Definition von Agenteneigenschaften werden im Wesentlichen die Arbeiten von Franklin und Graesser als auch die von Jennings und Wooldridge (1996) referenziert. Agenten werden hierbei als Hard- und Softwareeinheiten gesehen, die folgende, konstituierende Eigenschaften aufweisen:

- „*Situatedness*". Dass sich ein Agent in einer Umgebung befindet, wird als wichtige Eigenschaft angesehen. Innerhalb dieser Umgebung bleibt der Agent persistent und ist mit den Änderungen konfrontiert, die durch ihn selbst oder durch andere Agenten verursacht werden. Der Agent nimmt seine Umwelt, die real oder ebenfalls simuliert sein kann, über Sensoren wahr und kann über seine Effektoren Veränderungen darin vornehmen. Softwareagenten verfügen nicht über physikalisch wirkende Sensoren und Effektoren, ihre Aktorik ist algorithmischer Natur.
- *Reaktivität*. Agenten befinden sich in einer dynamischen Systemumgebung und sind befähigt, ihre Umgebung wahrzunehmen und auf deren Veränderung zu reagieren. Eine Änderung in der Umgebung kann durch Umwelteinflüsse wie z.B. der Anpassung eines Ressourcenbestandes als auch durch Aktionen anderer Agenten hervorgerufen sein. Um effektiv in das Systemgeschehen eingreifen zu können muss die Reaktion des Agenten auf Ereignisse in seiner Umwelt unmittelbar erfolgen können. Das Reagieren auf neue Situationen setzt eine gewisse Flexibilität des Agenten voraus. Sein Verhalten kann nicht vollständig von einem immer gleich ablaufenden Mechanismus bestimmt werden, es muss an unterschiedliche Situationen anpassbar sein. Die Reaktivität alleine setzt beim Agenten jedoch

noch nicht die Fähigkeit voraus, seine internen Regeln oder Pläne selbst abändern zu können.

- *Autonomie.* Die Autonomie stellt die zentrale Eigenschaft dar, mit der sich Agenten gegen andere Softwareeinheiten abgrenzen. Die Abkehr vom Paradigma der direkten Manipulation wird durch die Hervorhebung der Kontrolle über die eigenen Aktionen und den internen Zustand der Agenten vollzogen. Jeder Agent bestimmt selbst, wie er auf Ereignisse reagiert und welche Aktionen er dabei ausführt. Betrachtet man die Möglichkeiten eines Agenten genauer, kann zwischen zwei unterschiedlichen Arten von Autonomie unterschieden werden. Bei der *Kontrollautonomie* steuert der Agent die Ausführung seines Verhaltensprogramms. Er kann ohne oder nur mit einem geringen Eingreifen eines Benutzers oder anderer Agenten seine Ziele erreichen. Dies ist für einige Autoren wie Jennings und Wooldridge (1996) und Castelfranchi et al. (1996) als Kriterium für die Autonomie eines Agenten bereits ausreichend. Andere Autoren wie z.B. Russell und Norvig (1995) bezeichnen einen Agenten nur dann als autonom, wenn er sein Verhalten in Abhängigkeit von gemachten Erfahrungen selbständig abändern kann. In diesem Fall spricht man von *Verhaltensautonomie*.
- *Sozialität.* Soziales Verhalten kennzeichnet die Fähigkeit eines Agenten zur Interaktion mit anderen Agenten. Die Fähigkeit der Kommunikation ist mit dieser Eigenschaft untrennbar verbunden. Im einfachsten Fall kommuniziert und interagiert ein Agent mit anderen Agenten unabhängig davon, ob er die sozialen Strukturen, in die er eingebunden ist, erkennt oder sie repräsentiert. Für einige Autoren ist diese Eigenschaft für eine Zuschreibung von Sozialität für Agenten bereits ausreichend. So wird in Genesereth und Ketchpel (1994) die Fähigkeit der Beherrschung einer Agentenkommunikationssprache als die zentrale Eigenschaft von sozialen Agenten angeführt. Andere Autoren stellen höhere Anforderungen. So erwarten z.B. Carley und Newell (1994) und Castelfranchi (1998) von einem sozialen Agenten, dass er zusätzlich über ein explizites Modell der anderen Agenten verfügt. Schlussfolgerungen sollen auf der Basis von Wissen über dieses Modell durchgeführt werden. Hierdurch soll ein Agent in die Lage versetzt werden, über die Ziele anderer Agenten zu urteilen,

deren Erwartungen, Motive und Möglichkeiten in eigenen oder in gemeinsamen Plänen und Aktionen zu berücksichtigen.

- *Intentionalität.* Als Grundlage für ein kohärentes Multiagentensystems wird oft vorausgesetzt, dass die einzelnen Agenten Ziele besitzen und diese in einer rationalen Art und Weise verfolgen. Nach Simon sind Agenten aufgrund von Ressourcenknappheit und unvollständigen Informationen begrenzte rationale Systeme (der Begriff der „*bounded rationality*" wurde bereits in den 1950er Jahren eingeführt, siehe Simon (1957)). Bei Bedarf ergreifen sie eigenständig die Initiative und nehmen dabei nicht nur Anweisungen entgegen, sondern unterbreiten auch proaktiv Vorschläge für ein weiteres Vorgehen. Damit ist das Verhalten eines Agenten mehr als eine bloße Reaktion auf die Veränderungen in seiner Umwelt. In Abhängigkeit von weiter fortgeschrittenen Architekturen und komplexeren Funktionen werden Agenten immer mehr anthropomorphe Eigenschaften zugeschrieben. Unter Zuhilfenahme von Begriffen wie z.B. Überzeugungen (engl. *beliefs*), Wünsche (engl. *desires*) und Absichten (engl. *intentions*) werden Agenten als immer komplexere artifizielle Systeme modelliert (siehe Rao et al. (1991) für eine Einführung in die BDI-Architektur). Vor allem bei Multiagentensimulationen in den Sozialwissenschaften soll sich entsprechend Doran (1996) ein Agent ähnlich einem Menschen verhalten. Der wesentliche Unterschied zu anderen Systemen liegt darin, dass Agenten derartige Begriffe explizit darstellen und auch nutzen. Nach Dennett (1987) ist die Zuschreibung mentaler Attribute zur Erklärung des Agentenverhaltens angemessen. Mit Schlussfolgerungen aus eigenen Absichten, Überzeugungen und internen Motivationen sollen Agenten Einfluss auf die Ausführung und Änderung von Plänen und Aktionen nehmen.

1.3.2.3. Agentenkommunikation

Agenten, die in einem Verbund mit anderen Agenten arbeiten, benötigen Mechanismen zur Kommunikation mit ihrer Umgebung. In der Informatik wird der Kommunikationsbegriff bislang entweder mit einem Datenaustausch oder dem Funktionsaufruf nach dem Client-Server-Prinzip in Verbindung gebracht (siehe z.B. Orfali et al. (1999) für eine Einführung in Client-/Server-Systeme). Bei einer Agentenkommunikation steht meist eine Informationsübermittlung im Zentrum des Interesses, die auf einer abstrakteren, oft linguistischen Ebene stattfindet.

Kommunikationsinhalte sind ebenso Teil eines Multiagentenmodells wie das Agentenverhalten selbst. Im Folgenden wird ein kurzer Überblick über unterschiedliche Kommunikationsformen zwischen Agenten gegeben:

- *Keine explizite Kommunikation*. In einfachen Multiagentensimulationen können Informationen auch ohne explizite Kommunikation ausgetauscht werden. So wird z.B. bei zellulären Automaten die Aktualisierung einer Auszahlungsmatrix in einem iterierten Gefangenendilemma-Spiel von jedem Agenten selbständig durchgeführt. Der im Spiel involvierte Nachbaragent wird hierbei nicht kontaktiert, es kommt zu keinem Informationsaustausch.
- *Shared Memory (Blackboard)*. In früheren Anwendungen der verteilten künstlichen Intelligenz kamen vorwiegend Blackboard-Mechanismen als Kommunikationsformen zum Einsatz (siehe z.B. Carver und Lesser (1992)). Hier findet eine Nachrichtenübertragung durch Veröffentlichung in einem für alle Agenten sichtbaren Speicherbereich statt. Als abstrakte Form des *shared memory* kommunizieren Agenten in einer indirekten Art und Weise miteinander. Sie schreiben Daten auf das Blackboard bzw. lesen Daten von ihm. Für die direkte Adressierung von Nachrichten an einen bestimmten Agenten bietet diese Kommunikationsmethode keine Unterstützung. Mit der zunehmenden Realisierung echt verteilter Systeme geriet der Blackboard-Ansatz zunehmend in den Hintergrund und wurde durch direkte, oft sprechaktbasierte Kommunikationsformen ersetzt.
- *Einfache Kommunikation*. In der Informatik existieren viele etablierte Kommunikationstechniken für unterschiedliche Einsatzzwecke. Für einfache Anwendungen ist z.B. die Kommunikation über geteilte Variablen ausreichend. In verteilten Systemen wird Kommunikation oft durch Aufrufe entfernter Funktionen (engl. *remote procedure call*, *RPC*) oder durch den Einsatz einer Kommunikations-Middleware wie *DCE* („*Distributed Computing Environment*“, siehe z.B. Brando (1995)) oder *CORBA* („*Common Object Request Broker Architecture*“, siehe z.B. Yang und Duddy (1996)) realisiert. Mit Brokering-Diensten wird damit für eine einheitliche Sicht auf die in einem Netzwerk verteilten Programmelement gesorgt und deren einfache Nutzung ermöglicht.
- *Kommunikation über Nachrichten*. Bei der Kommunikation über Nachrichten (engl. *message passing*) erfolgt eine zielgerichtete Auswahl eines Empfängers

durch einen Sender. Durch eine Folge von Nachrichten, bei denen sich Sender und Empfänger abwechseln, kann es zu einem Dialog zwischen Agenten kommen. In noch weiter fortgeschrittenen Agentensystemen werden zusätzlich zu Nachrichten auch ganze Pläne untereinander ausgetauscht.

Ein besonderer Schwerpunkt der Forschungsarbeit im Themengebiet Softwareagenten liegt im Bereich der Agentenkommunikationssprachen (engl. *agent communication languages*, *ACL*). Diese Sprachen stellen ein standardisiertes Begriffssystem und eine formale Semantik für den Austausch von Nachrichten dar und können unter Bezugnahme auf eine gemeinsame Ontologie auch zwischen heterogenen Agenten eingesetzt werden. Üblicherweise wird die Sprechakttheorie als Grundlage herangezogen auf der Dialoge und Kommunikationsprotokolle aufsetzen. Die aktuell bedeutendsten Kommunikationssprachen sind *KQML* („*Knowledge Query and Manipulation Language*“, siehe z.B. Finin et al. (1994)) und die ACL der *Foundation for Intelligent Physical Agents* (*FIPA*, siehe http://www.fipa.org).

1.3.2.4. Agentenkoordination

Zeitliche Zusammenhänge spielen in vielen Agentenszenarien eine wichtige Rolle. Ein Indiz hierfür liefert die zentrale Bedeutung der Agentenkoordination: Aktionen unterschiedlicher Agenten stehen häufig in einem zeitlichen Zusammenhang. Zur Erreichung eines Zieles müssen mehrere Agenten entweder gleichzeitig handeln, oder geplante Aktivitäten sind aufgrund gemeinsam genutzter Ressourcen zu synchronisieren.

Nach Malone und Crowston (1991) kann Koordination als ein Prozess verstanden werden, der mit den Abhängigkeiten zwischen Aktivitäten umgehen kann. Hierbei gibt es zwei Sichten auf die im Zusammenhang mit der Koordination möglichen Problematiken: eine lokale Sicht, bei der es um das Management der Aktivitäten innerhalb eines Agenten geht und eine globale Sicht, die auf Abstimmung von Handlungen zwischen mehreren Agenten abzielt.

Ein lokales Koordinationsproblem liegt vor, wenn:

- Ein Agent zwischen alternativen Aktionen wählen kann, die in unterschiedlicher Weise Einfluss auf die Umgebung haben.

oder

- Die Reihenfolge und Zeit der Ausführung von Aktionen den Agenten und seine Umgebung unterschiedlich beeinflussen.

Entsprechend beschäftigen sich lokale Koordinierungstechniken mit der Auswahl geeigneter Aktionen und deren zeitlicher Anordnung.

Demgegenüber hat globale Koordinierung das Erreichen eines kohärenten Systemverhaltens zum Ziel. In Szenarien mit autonomen, parallel arbeitenden Agenten können viele Situationen auftreten, die die Kohärenz beeinträchtigen. Folgenden Arten der Koordination können unterschieden werden:

- *Zentrale Koordination*. Dies ist die traditionelle Form der Koordination, in der die Aktivitäten der einzelnen Agenten im Bedarfsfall von einer zentralen Stelle aus gesteuert werden.
- *Dezentrale Koordination*. Hier erfolgt eine Delegation von Entscheidungskompetenzen an einzelne Agenten oder Agentengruppen. In einer Selbstabstimmung regeln die einzelnen Mitglieder der betroffenen Einheiten die Koordination ihrer Aktivitäten eigenständig.

1.3.2.5. Eigenschaften von Multiagentensystemen

In Szenarien mit mehreren Agenten treten Merkmale verteilter Systeme auf, die sich im Verhalten eines einzelnen, isoliert arbeitenden Agenten nicht zeigen. Doch auch diese Systemeigenschaften, die sich vor allem aus der mehr oder weniger ausgeprägten Autonomie der einzelnen Agenten ergeben, haben einen Einfluss auf die Gestaltung der Agenten. Die Agenten kommunizieren und interagieren miteinander und mit einer zusätzlich zu betrachtenden Umwelt. Im Unterschied zu monolithischen Systemen sind die Informationen in einem Multiagentensystem normalerweise intransparent über die einzelnen Agenten verteilt. Jeder Agent besitzt nur ein Wissen über den für ihn relevanten Ausschnitt der ihn umgebenden Welt. Dieses Wissen muss nicht notwendigerweise exakt und konsistent sein. Durch das gleichzeitige Wirken mehrerer autonomer Einheiten ist das Wissen über den Zustand der Umgebung generell temporärer Natur. Aufgrund der Informationsfülle und deren Dynamik ist es für einen Agenten erforderlich, auch mit unvollständigem, möglicherweise inkorrektem Wissen arbeiten zu können.

Die obigen Ausführungen zusammenfassend, kann man Multiagentensystemen die folgenden allgemeinen Eigenschaften zuordnen (vgl. Jennings et al. (1998)):

- Jeder Agent ist in seinen Informationen und Problemlösefähigkeiten beschränkt. Er hat damit nur eine eingeschränkte Sicht auf das Gesamtsystem.

- Aufgrund der Individualität und Autonomie eines Agenten verwaltet er seine Daten lokal. Aus Sicht des Multiagentensystems erfolgt hierdurch eine dezentrale Datenhaltung.
- Die Aktionen, die jeder Agent ausführt, sollten nach Möglichkeit asynchron geschehen. Der Aufbau von komplexen, parallel miteinander kommunizierenden und interagierenden Agenten wird erst durch die Behandlung von Nebenläufigkeit als wichtige Eigenschaft eines Multiagentensystems ermöglicht.
- Idealerweise verfügen Multiagentensysteme über keine zentrale Kontrolle. Dies verbietet die grundsätzlich geforderte Autonomie der Agenten. In einem gewissen Maße soll jeder Agent selbst Kontrolle über seine durchgeführten Handlungen und Interaktionen übernehmen.

In vielen Fällen müssen manche dieser Eigenschaften relativiert werden. So könnte man bereits von einer zentralen Kontrolle sprechen, wenn es eine zentral weiter geschaltete Uhr für das Gesamtsystem gibt.

1.3.3 Agentensimulation auf parallelen Computersystemen

Der generelle Einsatz von parallelen Computern ist keine neue Idee, tatsächlich lässt er sich zeitlich weit zurückverfolgen. So schreibt Gill (1958) bereits in den 1950er Jahren über parallele Programmierung. Holland (1959) schreibt über einen „*computer capable of executing an arbitrary number of sub-programs simultaneously*“ im Jahre 1959. Conway (1963) beschreibt 1963 das Design eines parallelen Computers und dessen Programmierung. Veröffentlichungen mit vergleichbaren Titeln werden auch weitere 30 Jahre später noch gemacht (siehe z.B. Howe und Moxon (1987) und Karp (1987)). Ungeachtet der langen Historie argumentiert Flynn und Rudd (1996): „*the continued drive for higher- and higher-performance systems […] leads us to one simple conclusion: the future is parallel*“.

Sowohl in Forschung und Wissenschaft, als auch in der industriellen Praxis werden parallele und verteilte Computersysteme seit vielen Jahren erfolgreich eingesetzt.

Betrachtet man den Bereich der Multiagentensimulationen zeigt sich jedoch ein anderes Bild. Nach wie vor sind nur sehr wenige Systeme im Einsatz, die tatsächlich parallel oder verteilt arbeiten. Bei den meisten Simulationssystemen handelt es sich um Softwaresysteme, die in einer sequentiellen Art und Weise programmiert sind und auf einer klassischen Von-Neumann-Rechnerarchitektur zum Einsatz

kommen. In einigen Fällen wird auf derartigen Computersystemen durch die Verwendung von Threads eine Nebenläufigkeit von Prozessen (und damit auch Agenten) unterstützt, die sich für den Betrachter als scheinbare Parallelität zeigt (siehe Kapitel 2 für Grundlagen zu parallelen Computersystemen und paralleler Programmierung.). Viele der heutigen Simulationsumgebungen und selbstentwickelten Multiagentenanwendungen basieren auf der Programmiersprache Java (siehe z.B. Bigus und Bigus (2001) für eine Realisierung von intelligenten Agenten in Java).

Wooldridge und Jennings (1998) wiesen schon früh auf eine offensichtliche Schwierigkeit hin, die in vielen Tools zur Erstellung von Multiagentensystemen beobachtet werden kann: die Abwesenheit oder der mangelhafte Einsatz von Nebenläufigkeit. Dabei wird Nebenläufigkeit oft als eine der wichtigeren Qualitäten eines Multiagentensystems angesehen. In neuerer Zeit wies Duvigneau et al. (2003) nach, dass die meisten Multiagentenframeworks nur eine eingeschränkte Unterstützung der Programmierung von Nebenläufigkeit anbieten.

In diesem Zusammenhang wird den derzeit existierenden Programmbibliotheken für sozialen Simulationen von einem technischen Standpunkt aus oft vorgeworfen, dass sie auf die Implementierung von „Spielzeugsimulationen" ausgerichtet sind, in denen sehr einfache Agenten und Interaktionsregeln eingesetzt werden.

Nebenläufige Programme sind durch die mehrfache Verwendung von Leichtgewichtsprozessen charakterisiert, mit denen auf einem Einzelprozessorsystem der Anschein erweckt wird, dass sie tatsächlich parallel ausgeführt werden. Als Vorteile der Verwendung von Nebenläufigkeit im Zusammenhang mit sozialen Simulationen können unter anderem die folgenden Punkte aufgeführt werden:

- *Verbesserte Nutzung der eingesetzten Prozessoren.* In einer Multiprozessorsystemumgebung werden die Vorteile besonders deutlich: jeder Prozessor kann hier einen unterschiedlichen Prozess übernehmen, wodurch eine echte Parallelverarbeitung ermöglicht wird. Einerseits kann hierdurch eine Erhöhung der Ausführungsgeschwindigkeit erzielt werden, andererseits können komplexere Agenten mit einer umfangreicheren Kommunikationsstruktur erstellt werden. Aber auch in Einprozessorsystemen kann es durch eine optimale Nutzung der verfügbaren Ressourcen durch das Betriebssystem zu einer verbesserten Nutzung des verwendeten Prozessors kommen.
- *Modellierung der realen Parallelität.* Nebenläufige Modelle können die verteilte Natur von sozialen Systemen repräsentieren. Darüber hinaus führt die

Einführung von Nebenläufigkeit zur Aufhebung einer strikten zeitlichen Ordnung von Ereignissen. Einzelne Agenten können im Vergleich zu anderen Agenten ihre Aktionen in unterschiedlichen Zeitabschnitten durchführen. Dies ermöglicht den Entwurf von komplexen Simulationsumgebungen, die über einen erheblichen Freiheitsgrad in der Ereignisabfolge verfügen.

In einer kurzen Übersicht werden im Folgenden aktuelle Modellierungs- und Programmierwerkzeuge für soziale Simulationen auf die Möglichkeiten zur parallelen/verteilten Verarbeitung hin untersucht:

- Obwohl die ursprüngliche Sprache Logo um Konstrukte zur Parallelverarbeitung erweitert wurde, handelt es sich bei dem Nachfolger *NetLogo* (siehe Wilensky (1999) und http://ccl.northwestern.edu/netlogo) um eine reine Einzelprozessoranwendung. Es können unterschiedliche Instanzen von NetLogo gleichzeitig gestartet werden, eine parallele/verteilte Verarbeitung eines gemeinsamen Simulationsmodells ist jedoch nicht möglich.
- *Swarm* (siehe Minar et al. (1996), Sen (1998) und http://www.swarm.org/wiki/Main_Page) unterstützt eine gewisse Form von Nebenläufigkeit bei der Aktivitäten als Leichtgewichtsprozesse instantiiert und zugeteilt werden können, eine Existenz von autonomen, nebenläufigen Agenten wird aber nicht ermöglicht.
- *Repast* (siehe North et al. (2006) und http://repast.sourceforge.net/) erlaubt die Simulation von Nebenläufigkeit durch dessen *discrete event scheduler*, eine direkte Implementierung von tatsächlich nebenläufigen Prozessen ist nicht möglich.
- Mason (siehe Luke et al. (2004) und http://cs.gmu.edu/~eclab/projects/mason/) entspricht im grundlegenden Aufbau Swarm und erlaubt daher keine nebenläufigen Agenten.
- *MACE* (siehe Gasser et al. (1987)) und sein Nachfolger *MACE3J* (siehe Gasser und Kakugawa (2002)) sind Entwicklungssysteme für Multiagentensysteme, die speziell für den Einsatz einer sehr großen Gemeinschaft komplexer Agenten geeignet sind. Das originale MACE war ein echt verteiltes, objektorientiertes Multiagentenentwicklungssystem, das auf dedizierten Multiprozessorsystemen und ersten Netzwerken aus Arbeitsplatzrechnern lief. MACE3J ist vollständig Java basiert und läuft auf Einzel- und Multiprozessorsystemen genauso wie auf großen Workstation-Clus-

tern. Die MACE-Systeme sind vor allem auf maximale Skalierbarkeit und Performanz ausgelegt.

- *Mozart* (siehe Roy und Haridi (1999) und http://www.mozart-oz.org) liegt die Programmiersprache Oz zugrunde. Sie ermöglicht neben funktionaler, objektorientierter und logischer Programmierung auch Konstrukte, die nebenläufige und verteilte Ausführung unterstützen. Da Mozart ursprünglich für die Implementierung von mobilen Agenten und zur Lösung von Problemen der künstlichen Intelligenz entwickelt wurde, enthält Oz netzwerktransparente Erweiterungen. Plattformunabhängiger Bytecode, der von einer virtuellen Maschine ausgeführt wird, macht Mozart zu einer Umgebung für verteilte Simulationssysteme in heterogenen Netzwerken.
- Bei *AgentBuilder* (siehe http://www.agentbuilder.com) handelt es sich um ein integriertes Software-Toolkit um intelligente Softwareagenten und agentenbasierte Applikationen zu entwickeln. AgentBuilder basiert auf Java und unterstützt CORBA und TCP/IP Sockets. Kommuniziert wird mittels KQML.
- *MadKit* (siehe http://www.madkit.org) ist eine Multi-Agenten-Plattform und ein Entwicklungsframework, das vollständig auf Java basiert und daher auf einer Vielzahl von Plattformen einsetzbar ist. MadKit erlaubt das Erstellen von verteilten Multi-Agenten-Systemen. Aufgrund des zentral eingesetzten, synchronen Schedulers ist die Nebenläufigkeit jedoch eingeschränkt.

Auf die teilweise schon in diesem Abschnitt erwähnten Eigenschaften von parallelen/verteilten Computersystemen und den Möglichkeiten der nebenläufigen Programmierung wird im nächsten Kapitel genauer eingegangen.

Informatik

Band 88: Karl R. Brendel: **Parallele oder sequentielle Simulationsmethode?** · Implementierung und Vergleich anhand eines Multi-Agenten-Modells der Sozialwissenschaft
2010 · 316 Seiten · ISBN 978-3-8316-4013-3

Band 87: Daniel Motus: **Referenzmodell für die Montageplanung in der Automobilindustrie**
2009 · 204 Seiten · ISBN 978-3-8316-0860-7

Band 86: Joachim Wolfgang Kaltz: **An Engineering Method for Adaptive, Context-aware Web Applications**
2006 · 196 Seiten · ISBN 978-3-8316-0647-4

Band 85: Stephanie Spranger: **Calendars as Types** · Data Modeling, Constraint Reasoning, and Type Checking with Calendars
2006 · 308 Seiten · ISBN 978-3-8316-0564-4

Band 84: Sascha Vogel: **Eine Methode zur Entwicklung flexibler Dienste auf der Basis von Interaktionsmustern**
2004 · 202 Seiten · ISBN 978-3-8316-0334-3

Band 83: Andreas Rausch: **Componentware** · Methodik des evolutionären Architekturentwurfs
2004 · 205 Seiten · ISBN 978-3-8316-0326-8

Band 82: Peter Birke: **Und es geht doch! Wie rational ist irrational?** · Näherungsweise Berechnung von Wurzeln natürlicher Zahlen mittels eines divisionsfreien Algorithmus auf eine Genauigkeit von 1000 geltenden Ziffern
2004 · 64 Seiten · ISBN 978-3-8316-0299-5

Band 81: Claudia Gold: **Framework-basierte Unterstützung bei der Realisierung von Lastverteilung**
2003 · 193 Seiten · ISBN 978-3-8316-0272-8

Band 80: Bärbel Ripplinger: **Linguistic Knowledge in Cross-Language Information Retrieval**
2002 · 182 Seiten · ISBN 978-3-8316-0181-3

Band 79: Andreas Dehmel: **A Compression Engine for Multidimensional Array Database Systems**
2002 · 170 Seiten · ISBN 978-3-8316-0139-4

Band 78: Helmut Reiser: **Sicherheitsarchitektur für ein Managementsystem auf der Basis Mobiler Agenten**
2002 · 259 Seiten · ISBN 978-3-8316-0108-0

Band 77: Peter M. Borst: **An Architecture for Distributed Interpretation of Mobile Programs**
2002 · 576 Seiten · ISBN 978-3-8316-0103-5

Band 76: Holger Schmidt: **Entwurf von Service Level Agreements auf der Basis von Dienstprozessen** · 2. Auflage
2005 · 301 Seiten · ISBN 978-3-8316-0455-5

Band 75: Rainer Hauck: **Architektur für die Automation der Managementinstrumentierung bausteinbasierter Anwendungen**
2001 · 201 Seiten · ISBN 978-3-8316-0056-4

Band 74: Marco Pötke: **Spatial Indexing for Object-Relational Databases**
2001 · 226 Seiten · ISBN 978-3-8316-0043-4

Band 73: Michael Bader: **Robuste, parallele Mehrgitterverfahren für die Konvektions-Diffusions-Gleichung**
2001 · 152 Seiten · ISBN 978-3-8316-0040-3

Band 72: Robert Müller: **Fingerprint Verification with Microprocessor Security Tokens**
2001 · 170 Seiten · ISBN 978-3-8316-0015-1

Band 71: Katharina Spies, Bernhard Schätz (Hrsg.): **Formale Beschreibungstechniken für verteilte Systeme** · 9. GI/ITG Fachgespräch, München, Juni 1999
1999 · 260 Seiten · ISBN 978-3-89675-918-4

Band 70: Ricarda Weber: **Accounting and Payment Concepts for Fee-Based Scientific Digital Libraries**
2000 · 360 Seiten · ISBN 978-3-89675-875-0

Band 69: Dieter A. Bartmann: **Benutzerauthentisierung durch Analyse des Tippverhaltens mit Hilfe einer Kombination aus statistischen und neuronalen Verfahren**
2000 · 204 Seiten · ISBN 978-3-89675-836-1

Band 68: Anton Christian Frank: **Organisationsprinzipien zur Integration von geometrischer Modellierung, numerischer Simulation und Visualisierung**
2000 · 166 Seiten · ISBN 978-3-89675-796-8

Band 67: Markus Podolsky: **Ein durchgängiger, integrierter Ansatz für den Entwurf objektorientierter, Workflow-basierter Systeme**
2000 · 292 Seiten · ISBN 978-3-89675-771-5

Band 66: Thomas Paintmayer: **Einbettung von Sicherheitsverfahren in ein offenes heterogenes Sicherheitsmanagement auf der Basis der OSI Managementarchitektur**
2000 · 250 Seiten · ISBN 978-3-89675-770-8

Band 65: Stephen Heilbronner: **Konzeption einer Architektur für das integrierte Management der Ressourcennutzung nomadischer Systeme in Datennetzen**
2000 · 248 Seiten · ISBN 978-3-89675-733-3

Band 64: Boris Gruschke: **Entwurf eines Eventkorrelators mit Abhängigkeitsgraphen**
2000 · 301 Seiten · ISBN 978-3-89675-725-8

Erhältlich im Buchhandel oder direkt beim Verlag:
Herbert Utz Verlag GmbH, München
089-277791-00 · info@utzverlag.de

Gesamtverzeichnis mit mehr als 3000 lieferbaren Titeln: www.utzverlag.de