Aktivitätsbasierte Verhaltensmodellierung und ihre Unterstützung bei Multiagentensimulationen

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius–Maximilians–Universität Würzburg

vorgelegt von

Franziska Klügl

aus
Ingolstadt

Würzburg, 2000

Eingereicht am: 6. Oktober 2000

bei der Fakultät für Mathematik und Informatik

1. Gutachter: Prof. Dr. F. Puppe
2. Gutachter: Prof. Dr. S. Kirn

Tag der mündlichen Prüfung:

Aktivitätsbasierte Verhaltensmodellierung und ihre Unterstützung bei Multiagentensimulationen

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius–Maximilians–Universität Würzburg

vorgelegt von

Franziska Klügl

aus
Ingolstadt

Würzburg, 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1

Motivation

1.1 Einführung

1.1.1 Generelle Problematik

Modellbildung und Simulation gehören mittlerweile zu den Standardmethoden bei der Untersuchung von Systemen. Dabei versucht man einen Realitätsausschnitt so abzubilden, daß ein Modell in einem für die aktuelle Aufgabenstellung ausreichenden Maß die Wirklichkeit widerspiegelt. Die Konstruktion eines Modells ist ein aufwendiger Prozeß, bei dem die Vorgehensweise zwischen Deduktion und Induktion liegt [Axelrod 1997a]. Vom Modellbauer wird dabei nicht nur inhaltliches Wissen über das darzustellende System, sondern auch methodisches Geschick verlangt.

Experimente mit Modellen sind aus vielerlei Gründen attraktiv. Dies gilt insbesondere, wenn das reale System noch nicht existiert, nicht direkt zugänglich ist oder Experimente mit diesem zu teuer oder gefährlich sind. Ziel und Zweck von Simulationsstudien reichen dabei von der Design-Analyse, der Ansteuerungsoptimierung bis hin zur Gewinnung von Erkenntnissen über ein zum Teil noch unverstandenes System, sei es in Wissenschaft oder Ausbildung (siehe dazu [Fishwick 1995, Spriet & Vansteenkiste 1982] oder andere Lehrbücher zur Simulation). Die Verteilung der Verantwortlichkeiten und Aufgaben, die beim Durchführen einer Simulationsstudie anfallen, ist abhängig von den Zielen, die mit ihr erreicht werden sollen: Wer ist der Auftraggeber der Simulationsstudie? Wie verteilen sich folgende Rollen auf die beteiligten Personen: der Domänenexperte, der über das Wissen zur Beschreibung des Systems verfügt, der Entwickler des Simulators, der das Modell implementiert und der Experimentator, der mit dem implementierten Modell systematische Experimente durchführt. Diese unterschiedlichen Aufgaben müssen nicht personell getrennt erfüllt werden: Während man in industriellen Anwendungsgebieten häufig eine echte Verteilung findet, ist sie in wissenschaftlichen Anwendungen eher selten, wie im folgenden ausgeführt wird.

Da zur Entwicklung einer Simulation implizit angenommene Sachverhalte konkretisiert werden müssen, stellen sich oft während der Konstruktion eines Modells Fragen, die zum Verständnis des abzubildenden Systems beitragen können. Dabei ist es nur in bestimmten Szenarien sinnvoll, Modelle in “Auftragsarbeit” erstellen zu lassen. Eine personelle Trennung zwischen System– und Modellierexperte ist vor allem bei Anwendungen möglich, in denen es besonders auf das Ergebnis einer Simulationsstudie ankommt, weil dieses zur

Verbesserung eines weitgehend verstandenen Systems beiträgt. Charakteristische Domänen sind dabei Simulationen technischer Anlagen, wie beispielsweise die Optimierung eines Fabrikdesigns. Wenn das Verhalten der Modellbestandteile eindeutig beschrieben werden kann, und deren Konfigurierung ausgetestet werden soll, ist es relativ unproblematisch, die Modellspezifikation von einem Systemexperten festlegen und danach durch spezialisierte Modellbauer implementieren und systematisch austesten zu lassen. Diverse Institute und Firmen haben sich darauf spezialisiert, derartige Simulationsstudien in Auftragsarbeit zu erstellen.

Auf der anderen Seite sind wissenschaftliche Fragestellungen oft dadurch charakterisiert, daß durch die Simulation möglicherweise alternativer Hypothesen ein verbessertes Systemverständnis gewonnen werden soll. Dabei kann allerdings nicht von vornherein klar spezifiziert werden, welches die wirklich relevanten Faktoren bei der Simulation sind. Soll die Simulation als Ergänzung zu empirischen Experimenten dienen, muß der Systemexperte, also der "praktische" Forscher, die Möglichkeit besitzen, selbst das ganze Modell — nicht nur einzelne Konfigurationsparameter — zu manipulieren. So kann der Systemexperte seine Hypothesen eigenständig entwickeln, austesten und dadurch neue Hinweise für praktische Experimente gewinnen. Deshalb sind "Auftragssimulationen" bei Forschungsarbeiten weniger verbreitet. Allerdings hat dies den Effekt, daß ein Systemexperte über zusätzliche Modellier- und Programmiererfahrung verfügen muß. Abhängig vom eigentlichen Ziel der Modellierung ist es also mehr oder weniger zwingend, den Systemexperten direkt an der Modellierung und Experimentation zu beteiligen bzw. ihm im Idealfall passende Werkzeuge zugänglich zu machen, so daß er selbst implementieren und simulieren kann.

Leider findet man diese ideale Form eines wissenschaftlichen Einsatzes von Modellbildung und Simulation in der Realität nur in begrenztem Ausmaß. E. Bruderer und M. Maiers listen in [Bruderer & Maiers 1997] vier Hauptprobleme auf, weshalb Simulation als Methode generell — und insbesondere in den Sozialwissenschaften — immer noch so wenig verbreitet ist:

- Vorhandene Simulatoren verfügen selten über gute Benutzeroberflächen. E. Bruderer und M. Maiers benutzen zur Erläuterung ein Bild aus dem Alltag: Die Benutzbarkeit von Simulatoren wäre vergleichbar damit, wenn man zum Fahren eines Autos zuerst lernen müßte, eines zu bauen. Leider sind gute Fahrer nicht immer gute Mechaniker.

- Generelle Schwierigkeiten beim Benutzen von (fremden) Computerprogrammen hängen eng mit dem obigen Punkt zusammen. Dies liegt auch am Fehlen von Standards für Simulationen. Jeder Modellbauer benutzt nach [Bruderer & Maiers 1997] zum Implementieren seines Modells seine bevorzugte Sprache bzw. benutzt dazu eine andere Entwicklungsumgebung.

- Eine Folge davon sind Schwierigkeiten, die aus Simulationen gewonnenen Resultate anderer Forscher zu reproduzieren. Dies wirkt sich unmittelbar auf die Seriosität der Methodik aus: Was taugen Modelle, die nicht nachvollziehbare Ergebnisse erzeugen?

- Als letzter Punkt wird in [Bruderer & Maiers 1997] angeführt, daß Simulation nicht als Werkzeug in der Lehre oder im Unterricht benutzt wird. Durch Praktika oder Vorlesungen könnte man Studenten an die Methode heranführen und so eine spätere Benutzung von Simulations-Tools erleichtern.

Die Abhilfen, die E. Bruderer und M. Maiers in den übrigen Abschnitten ihres Beitrags vorschlagen, resultieren in der Verbreitung von mehr Programmierkenntnissen unter Sozialwissenschaftlern. Im Grunde ist ihr Ansatz komplementär zu den hier vorgeschlagenen Lösungen. Unabhängig davon, ob ein adäquates Werkzeug zur Verfügung steht, helfen Erfahrungen bei der formalen Spezifikation bzw. bei der Abstraktion auf der Basis höherer Programmiersprachen beim Entwurf von Modellen ohne Einschränkung. Durch höhere Werkzeuge, die die Umsetzung einer Modellspezifikation in eine lauffähige Simulation erleichtern, muß der Aufwand bei der Ausbildung nicht unbedingt einem kompletten Studium zum Software–Entwickler entsprechen.

1.1.2 Multiagentensimulation als Ausgangspunkt

Eine wichtige Frage bei der Modellbildung ist die nach dem angemessenem Abstraktionsniveau der Simulation. Wie kann, ausgehend von Daten und Erfahrungen mit einem realen System, ein Gleichungssystem aufgestellt werden, das die Dynamik der einzelnen Bestandteile und deren Interaktionen in einem ausreichend detaillierten Maß wiedergibt?

Verwendet man als Basis für die Beschreibung eines System das Konzept eines Multiagentensystems, also das einer Gesellschaft aus interagierenden “Agenten”, erhält man dadurch Vorteile auf zwei Ebenen. Zum einen wird eine Metapher verwendet, die deutlich näher an der “natürlichen” Darstellung des Ursprungssystems liegen kann, als dies beispielsweise bei Beschreibungen auf der Makroebene mittels Differentialgleichungen der Fall ist. Dies gilt insbesondere bei Gesellschafts- und Gruppenmodellen, bei denen die Individualität der einzelnen Bestandteile eine wichtige Rolle spielt. Genau an dieser Stelle kommt auch die zweite Ebene der methodischen Vorteile bei der Multiagentenkonzeptionalisierung zum Tragen. Dadurch, daß man jedes Individuum so darstellen kann, wie es für die Reproduktion seines Verhaltens adäquat ist, werden Abstraktionen vermieden, die dem Untersuchungsziel abträglich sein können.

Ein wichtiger Unterschied zwischen Multiagentensystemen zur Problemlösung und Multiagentensimulationen liegt darin, daß letztere immer ein Vorbild oder Originalsystem besitzen. Dieses gibt vor, welche Organisationsformen, Interaktionen, Ziele, usw. im Multiagentenmodell zu finden sind. Insbesondere verfolgt der Modellbauer Ziele, die sich auf seine Arbeit mit dem Modell beziehen: Er will sein Konzeptmodell schnell umsetzen ohne dabei das von ihm gewählte Abstraktionsniveau verlassen zu müssen. Das implementierte Modell soll gut validierbar sein: auf der Ebene der individuellen Agenten und auf der des Gesamtsystems, durch eine verständliche Beschreibung des Verhaltens und durch Beobachtung der durch dieses erzeugten Dynamik. Analog zum Einsatz einer verteilten Problemlösung will ein Modellbauer auch das Modell “laufen lassen” können, also systematische Experimente mit diesem Modell durchführen und dabei aussagekräftige Daten gewinnen und auswerten können.

Diese Aspekte des Arbeitens mit einem Multiagentenmodell können und müssen durch Werkzeuge, wie Modellierungs-Tools oder Simulationsumgebungen unterstützt werden. So kann ein ohnehin leichter zugängliches Modellkonzept durch einen entsprechenden Rahmen so unterstützt werden, daß Modellbildung und Simulation weiter verbreitet werden können, insbesondere ohne Domänenexperten auf die Ebene einer traditionellen Programmiersprache zu zwingen. Genau an dieser Stelle setzt die vorliegende Arbeit an. Im Folgenden wird nun das Ziel der Arbeit verdeutlicht, danach werden in einem kurzen Streifzug die weiteren Kapitel kurz skizziert.

1.2 Ziel der Arbeit

Der Beitrag dieser Arbeit soll darin liegen, durch Kombination bekannter Konzepte aus der Künstlichen Intelligenz einen Rahmen für die Entwicklung von Multiagentensimulationen zu entwerfen. Durch die Integration eines Repräsentationsschemas, das es ermöglicht die Strukturen und das Verhalten eines Multiagentensystems auf eine auf eine strukturierte Art und Weise darzustellen, in eine Modellier- und Simulationsumgebung soll auf dem Weg der Unterstützung der Modellentwicklung und -nutzung durch Domänenexperten ein weiterer Schritt nach vorn getan werden.
Im einzelnen werden dafür folgende Teilschritte zurückgelegt:

1. Schaffung eines Rahmens zur Umsetzung eines Multiagentenmodells: Die Repräsentationsform für Strukturen und deren Dynamik soll es nicht nur erlauben, ein Modell schnell zu implementieren, sondern auch auf eine Weise zu beschreiben, die es unterstützt, das Modell gut verstehbar und somit auch entsprechend analysierbar darzustellen. Das bedeutet im Einzelnen:

   (a) Identifikation von Strukturen in einem Multiagentenmodell, auf denen eine transparente Zustands- und Verhaltensdarstellung basieren kann.

   (b) Deklarative Verhaltensbeschreibung durch Erweiterung des bekannten Konzepts der regelbasierten Verhaltensrepräsentation zu einem aktivitätsbasierten Schema, wie es im Kontext eines agentenbasierten Simulationsmodells sinnvoll ist.

2. Entwicklung eines Interpreters für ein auf diese Weise repräsentiertes Multiagentenmodell.

3. Integration dieses Simulators in den größeren Kontext einer Modellier- und Experimentierumgebung unter Ausnutzung von Erfahrungen mit Wissensakquisitionssystemen.

Bevor nun im Folgenden die Grundlagen für diese Schritte gelegt und dann deren Ausführung vorgestellt wird, soll ein kurzer Streifzug den Aufbau der vorliegenden Arbeit umreißen.

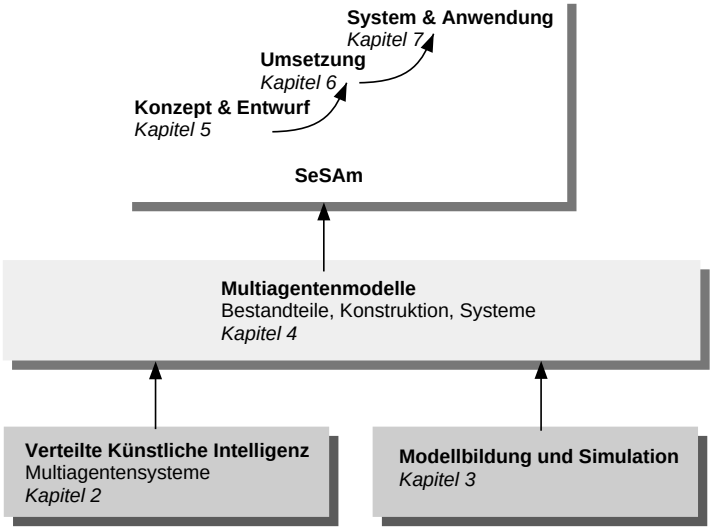
1.3 Streifzug durch die Arbeit

Abbildung 1.1 gibt eine erste graphische Übersicht über die nächsten sechs Kapitel.
Dabei ergibt sich eine grundsätzliche Zweiteilung: In den Kapiteln 2 bis 4 wird nicht nur das hier verwendete Verständnis von Multiagentensimulation ausführlich erläutert, sondern auch die Grundlagen und der Kontext der folgenden Kapitel vorgestellt. In den darauf folgenden drei Kapiteln werden Lösungen für die oben erwähnten Teilschritte dargelegt. Die Arbeit endet mit einen weiteren Kapitel, in dem ein Fazit gezogen wird und weiterführende Arbeiten angerissen werden. Im einzelnen soll wie folgt vorgegangen werden:

1.3.1 Multiagentensysteme + Simulation = Multiagentensimulation?

Man kann sich dem Gebiet der Multiagentensimulation von zwei Seiten her nähern: Aus der Sichtweise der Simulationsmethodik, welche die Multiagentensimulation als neue Form

Abbildung 1.1 Überblick über den Aufbau der Arbeit



der individuenbasierten Modellierung und Simulation identifiziert; oder von der Verteilten Künstlichen Intelligenz her, für die die Simulation eines Multiagentensystems eine klassische Anwendungsdomäne darstellt:

Unter Agenten – wie sie im Bereich agentenbasierter Systeme gesehen werden – sind individuelle Einheiten zu verstehen, denen man Eigenschaften wie "Situatedness"[1], Reaktivität, Autonomie, Sozialität, Zielstrebigkeit oder eine gewisse "Menschenähnlichkeit" — mit den sich daraus ergebenden Konsequenzen — zuordnen kann. Die übliche Vorgehensweise besteht dabei daraus, daß man das lokale Verhalten derartiger Einheiten und ihrer Interaktionen spezifiziert, um ein gewünschtes, kohärentes Globalverhalten zu erzeugen. Die beiden Beschreibungsebenen, die "Architekturebene" des Gesamtsystems und die Ebene der einzelnen Agenten, bestimmen mögliche Vorgehensweisen beim Entwurf und der Umsetzung eines Multiagentensystems: Abhängig von der Art des zu konzipierenden Systems empfiehlt sich beim agenten-orientierten Software-Engineering eher ein "top-down"-Entwurf, bei dem aus dem Zweck des Gesamtsystems, Teilziele bzw. Teildienste abgeleitet werden und diese mit Agenten assoziiert werden. Bei der Beschreibung existierender Multiagentensysteme findet man häufiger, daß ausgehend vom individuellen Agentenverhalten, insbesondere seinen Reaktionen auf Nachrichten usw. aus der Umwelt, das Gesamtsystem "bottom-up" zusammengesetzt wird (Kapitel 2).

Andererseits ist der Entwurf von Modellen, die existierende oder hypothetische Systeme nachbilden, um in Experimenten Aspekte des Orignalsystems zu untersuchen, die Domäne der Simulationstechnik. Mit verschiedenen Modellformen wird dabei auf unterschiedlichen Abstraktionsebenen operiert. Dies gilt insbesondere für Modelle von Systemen von Gruppen, Populationen, o. ä., die aus Individuen bestehen. Bei der individuenbasierten Simulation wird ein System als eine interagierende Menge von "Individuen" aufgefaßt und bei der Spezifikation des Modells ausgehend von diesen Einheiten das Gesamtverhalten dargestellt. Dabei wurden in den letzten Jahren einige unterschiedliche Herangehensweisen eta-

[1] Damit ist gemeint, daß ein Agent in einer Umwelt existiert und mit dieser interagiert.

bliert. Mikroanalytische Modelle, die auf stochastischen Prozessen beruhen oder Zelluläre Automaten, deren Hauptmerkmal die explizite Behandlung von diskreten Raumeinheiten darstellt, sind die beiden am weitesten verbreiteten Formen. Ausgehend von allgemeinen Aussagen über Simulation und Modellbildung werden diese Formen in Kapitel 3 näher besprochen. Das Kapitel endet mit einer Darstellung von Verhaltensmodellen, also von Modellen, die die Aktionen eines Agenten in einer bestimmten Situation fokussieren. Diese sind klar von Modellen abzugrenzen, in denen das Verhalten eines Agenten durch Beschreibungen seiner Ziele, Motivationen, und anderer interner Strukturen begründet wird. Während bei letzteren kognitive Prozesse des Agenten das Verhalten erklären, wird hier sein Verhalten beschrieben. Im entsprechenden Abschnitt wird begründet, warum dabei ein regelbasierter Formalismus als Basis für explizite, deklarative Verhaltensdarstellung das Paradigma der Wahl ist.

In Kapitel 4 werden Multiagentensysteme und Simulation zur *Multiagentensimulation* zusammengeführt: Agenten interagieren in einer simulierten Welt, um so ein vorgegebenes System nachzubilden. Dabei existieren bei der Modellbildung drei Schwerpunkte: Das Verhalten der einzelnen Agenten selbst, ihre Interaktionen in der Gruppe und ein explizites Umweltverhalten, mit dem "externe" Eingaben in das Agentensystem dargestellt werden können. Eine Modellierung als Agentensystem kann dabei in einem gewissen Sinne "intuitiv" sein, wenn das übliche Abstraktionsniveau bei der Beschreibung beobachteten Verhaltens ohne Transformation für die Modellierung beibehalten werden kann. Dies gilt insbesondere, wenn das Verhalten eines Agenten in Bezug zu seiner Umwelt die naheliegende Beobachtungseinheit darstellt. Weitere Möglichkeiten ergeben sich aus der Lokalität des Agentenverhaltens. So können an der Modellierung – bei einer vorgegebenen Umwelt mit entsprechendend auch semantisch klar definierten Schnittstellen – mehrere Modellbauer an der Erstellung des Gesamtmodells beteiligt sein. Interessant ist auch die Möglichkeit, verschiedene Abstraktions- und Aggregationsebenen im Modell zu kombinieren.

1.3.2 Werkzeuge für die Multiagentensimulation

Will man die Umsetzung eines Multiagentenmodells in eine ausführbare Simulation unterstützen, kann man für die drei Schwerpunkte bestimmte Vorgaben und Tools finden: Dabei kann man wiederum zwei unterschiedliche Herangehensweisen identifizieren: Einerseits durch die Formalismen und Strukturen (deklarativer) Spezifikations- und Modellierungssprachen und andererseits durch grafische Modellierungs-Tools für die Implementierung und Experimentierung.

1.3.2.1 Spezifikationssprachen und Repräsentationsstrukturen

Für die oben identifizierten Schwerpunkte von Multiagentenmodellen — Agenten, Interaktionen und Umwelt — gibt es dabei im Bereich der Formalismen und Repräsentationsstrukturen folgende Möglichkeiten:

1. Die Konstruktion der Agenten selbst kann auf vielfältige Weise unterstützt werden. Eine wichtige Rolle spielen dabei vorgegebene Architekturen für die internen Prozesse eines Agenten (aus der Verteilten Künstlichen Intelligenz) und entsprechende Referenzmodelle (aus der Simulationsmethodik). Andererseits kann die Unterstützung durch (deklarative) Beschreibungssprachen geschehen, mit denen das Verhalten eines Agenten z. B. mittels temporaler Logik spezifiziert werden kann.

2. Die Modellierung und Darstellung von Interaktionen innerhalb einer Agentengesellschaft spielt in der Verteilten Künstlichen Intelligenz eine herausragende Rolle. Mit (generischen) Koordinationsmechanismen, Interaktionsprotokollen aber auch Systemarchitekturen oder Aufgabenhierarchien existieren Schablonen, mit dem der Modellierer bei der Konstruktion des intendierten Globalverhaltens unterstützt wird.

3. Die Repräsentation der Bestandteile und Dynamik der nicht in Form von Agenten dargestellten Umwelt kann in einem anderen Formalismus, z. B. in Form eines Makromodells für die relevanten Zustandsgrößen, erfolgen. Allerdings muß ein Agentensystem in diese Umwelt integriert werden können. Wie einfach dies möglich ist, hängt von dem zu modellierenden System ab, aber auch davon, welchen Formalismus die benutzte Modellierumgebung unterstützt.

Der zweite Teil des Kapitel 4 bildet eine Übersicht zum einen über Methoden und Schemata für die Modellierung der Inhalte dieser Schwerpunkte und endet zum anderen mit einer umfangreichen Übersicht über Entwicklungsumgebungen für agentenbasierte Systeme, Simulationssprachen, bzw. deren Aufsätze, bis hin zu generischen Testbeds. Neben einer kurzen Charakterisierung des jeweiligen Systems wird dargelegt, welche Vorgaben für die Agentensteuerung und für die Umweltbeschreibung gemacht werden. Das Fazit dieser relativ vollständigen Aufstellung ist, daß es keine ausgewogene Umgebung für die Entwicklung von Multiagentensimulationen gibt. Bisher ist es nicht gelungen, eine ausbalancierte Methode zu entwickeln, die zwischen Mächtigkeit bei der Verhaltensbeschreibung auf der einen Seite und der Möglichkeit, auch komplexe Modelle zu organisieren, auf der anderen Seite zu vermitteln vermag.

1.3.2.2 Vom Repräsentationsschema zur Entwicklungsumgebung

Ausgehend von diesen Überlegungen wird in Kapitel 5 ein neues Repräsentationsschema für Multiagentenmodelle entworfen. Dieses basiert auf der Beschreibung des internen Zustands eines Agenten durch eine Menge von Zustandsvariablen. Auf diesen Strukturen aufbauend, wird das Verhalten in einem auf Regeln beruhenden Schema dargestellt. Grundidee war es dabei die Verhaltensrepräsentation durch eine mehr-dimensionale Strukturierung so übersichtlich wie möglich zu machen. Dabei wurden folgende Ansätze verfolgt:

- Unterteilung der Regelmenge nach Verhaltenskategorien oder Rollen
- Zusätzliche Unterteilung entlang der Phasen eines Aktualisierungszyklus
- Gruppierung der Aktionen, die typischerweise zusammen ausgewählt werden, zu persistenten Aktivitäten mit Unterteilung der Regelmenge in Regeln, die eine bestimmte Aktivität terminieren oder unabhängig von der aktuellen Aktivität eine neue auswählen.
- Gruppierung von Aktivitäten zu Sequenzen und Alternativen

Durch diese Strukturierungsformen ergeben sich höhere Metaphern für die Verhaltensbeschreibung: Aktivitätsautomat und Quasi-Skelettplan, die ein Modellierer als Basisschablone für die sein Modell benutzen kann. Das Konzept dieses Repräsentationsschemas wird in Kapitel 5 näher erläutert. Kapitel 6 beschäftigt sich damit, wie dieser Entwurf umgesetzt

werden kann – in Strukturen und Algorithmen einer Simulationsroutine für Multiagentenmodelle. Der besondere Schwerpunkt liegt darauf, dieses Schema explizit im Verhaltensmodell abzubilden und so eine deklarative Form zu schaffen.

Eine zweite Ebene der Unterstützung erhält man durch den Einsatz graphischer Werkzeuge auf der Basis einer derartigen Spezifikationsstruktur. Visuelle Programmiersprachen im weiteren Sinne bilden den klassischen Weg, neuen Benutzergruppen ohne besondere Programmierkenntnisse Entwicklungsumgebungen zugänglich zu machen. Allerdings sollte dies nicht auf Kosten einer effizienten Verwendung durch geübte Benutzer gehen, die ihr Modell schnell umsetzen wollen. Deswegen wurde dieser Simulator in eine Modellier- und Experimentierumgebung integriert, die in Kapitel 7 vorgestellt wird. Diese Simulationsumgebung wurde mit dem Namen SESAM bezeichnet: "Shell für Simulierte Agentensysteme".

Damit wurde durch eine Kombination generischer Agentenstrukturen mit graphischen Aggregations- und Abstraktionsmitteln eine Umgebung geschaffen, die Modellbauern aus anderen Fächern ohne Programmiererfahrung an die Hand gegeben wurde. Sie konnten mit SESAM erfolgreich komplexe Verhaltensmodelle repräsentieren und mit diesen experimentieren. Kapitel 8 präsentiert eine Darstellung ausgewählter Anwendungen aus den Bereichen der Simulation sozialer Insekten und als Grundlage für psychologische Experimente. Neben einigen Reimplementierungen von Szenarien, die aus der Literatur bekannt sind, werden auch geplante Anwendungen skizziert. Diese Beschreibungen müssen eine systematische Evaluation ersetzen, die bisher noch nicht durchgeführt werden konnte.

Die Arbeit endet mit einer Zusammenfassung der erreichten Leistungen und einem Ausblick auf mögliche Erweiterungen.

2

Multiagentensysteme

2.1 Allgemeines

Multiagentensysteme bilden das Basiskonzept für die im folgenden dargestellten Simulationsmodelle. Diese können auf zwei Ebenen beschrieben werden: bezüglich des Verhaltens des Gesamtsystems und bezüglich des Verhaltens der einzelnen Agenten in ihrer Umgebung. Bevor die dabei relevanten Aspekte der Beschreibung und der dazu möglichen Mittel zur Unterstützung erläutert werden, soll versucht werden, eine Intuition dafür zu vermitteln, was unter einem Agenten und einem Multiagentensystem zu verstehen ist. Nach einer Behandlung von Modellbildung und Simulation in Kapitel 3 werden im Kapitel 4 die hier relevanten, exakten Definition gegeben.

2.1.1 Das Konzept “Agent”

2.1.1.1 Der allgemeine Agentenbegriff

Der Begriff “Agent” wird nicht nur in vielen Bereichen der Informatik, sondern auch in den Wirtschafts- und Sozialwissenschaften verwendet. Sein Ursprung liegt im Lateinischen, wo die Wortform “agens” mit “handelnd” übersetzt werden kann. Allerdings bleibt bei dieser Begriffsbildung das Problem, daß “Handeln” selbst noch schwerer zu fassen ist, als der Agentenbegriff [Sundermeyer 1993]. Im Bereich der Wirtschaftswissenschaften findet sich die Notation “Agent” für jemanden, der im Auftrag eines anderen handelt, z.B. Verträge abschließt. Diese Bedeutung entspricht dem auch in der Informatik verwendeten Begriff im Bereich der Software-Agenten, z.B. bei Agenten für das Informations-Retrieval.

In der Künstlichen Intelligenz findet man den Begriff “Agent”, bzw. agentenbasiertes System[1] in unterschiedlichen Teilgebieten. Mit verschiedenen Ausgangspunkten und Zielen beim Einsatz von “Agenten” gehen auch verschiedene Auffassungen einher. Diese reichen von Agenten, die – ausgehend von einer Robotik-Sicht – als situierte, reaktive Systeme in einer Umwelt Aufgaben erfüllen und dabei “überleben” können sollen [Brooks 1991b, Brooks 1991a] bis hin zu kooperierenden “Problemlösern” [Bamberger 1999], deren Verhalten auf einer höheren Abstraktionsebene beschrieben werden kann. So bezeichnete schon

[1] Ein agentenbasiertes System ist eines, dessen Basiskonzept das eines Agenten ist [Jennings & Wooldridge 1998]. Jedes Multiagentensystem ist somit ein agentenbasiertes System, aber nicht umgekehrt.

A. Newell [Newell 1982] in seinem Papier "The Knowledge Level" jedes System auf einer abstrakten Ebene oberhalb einer Programmiersprache als "Agent".

Dennoch ist die Frage, wann ein Problemlöser bzw. Systembestandteil als "Agent" bezeichnet werden darf, umstritten. Viele Ansätze werden unter dem Dach der "agentenbasierten" Systeme entwickelt, und manche Forscher befürchten, daß eine Ablehnung bestimmter Teilbereiche auf das gesamte Gebiet der Agentensysteme ausgedehnt werden könnte [Shoham 1999]. Eine Folge dieser Diskussion ist, daß jeder, der ein agentenbasiertes System beschreibt, zuerst seine Vorstellung von "Agent" definiert. In relativ kurzer Zeit entstand so eine Fülle von unterschiedlichsten Definitionen, von "ein Agent ist ein System, das kommuniziert" [Genesereth & Ketchpel 1994] bis zu "ein Agent ist ein System mit mentalen Konzepten" [Shoham 1993]. Das führte wiederum zu Beiträgen, die sich ausschließlich damit beschäftigen, eine nur vage vorhandene Notation mit alternativen Konzepten zu fassen [Huhns & Singh 1999] oder all diese Definitionen zu einer umfassenden zu integrieren, wie [Franklin & Graesser 1997], die Definition aufstellten, die eine große Beachtung gefunden hat.

> "An agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, ... and so as to effect what it senses in the future"[Franklin & Graesser 1997]

Der Begriff eines Agenten in Multiagentenmodellen beruht auf diesen Konzepten der Verteilten Künstlichen Intelligenz und ist ähnlich schwierig zu fassen. Auch in der Multiagentensimulation existieren unterschiedliche Vorstellungen von Agenten. Diese reichen von Agenten als Modellen menschlicher Akteure – wie sie in der sozialwissenschaftlichen Simulation als Bestandteile von Gesellschaften modelliert werden – oder als abstrakteres Konzept im Sinne einer Erweiterung objekt-orientierter Simulation. Diese Ausführungen stellen einen Vorgriff auf Abschnitt 4.1 dar, wo das Konzept der Multiagentensimulation und dabei auch der dort verwendete Agentenbegriff präzisiert wird. Dort wird auch die für diese Arbeit grundlegende Definition des Agentenkonzepts gegeben.

Im folgenden soll die Betrachtung allgemeiner Eigenschaften von Agenten dieses abstrakte Konzept weiter verdeutlichen; vor allem um eine Grundlage für die folgenden Abschnitte zu schaffen, die verschiedene Beschreibungsebenen in Multiagentensystemen behandeln.

2.1.1.2 Charakteristische Eigenschaften von "Agenten"

Für ein Verständnis der "Essenz" des Agentenkonzepts ist eine Beschreibung der Eigenschaften, die das "Agent-sein" impliziert, sehr hilfreich. Die wichtigsten Charakteristika reichen dabei von "Situatedness", Reaktivität, Autonomie, Fähigkeiten zur sozialen Interaktion, rationales Verhalten bis hin zu einer Menschen-Ähnlichkeit [Ferber 1999, Shoham 1999, Franklin & Graesser 1997, Wooldridge & Jennings 1995, Sundermeyer 1993]. Diese Eigenschaften werden im Folgenden näher betrachtet.

"Situated" Eines der wichtigsten Charakteristika, oft auch als eine definierende Eigenschaft betrachtet, ist die "Situatedness". Damit wird "sich in einer Umgebung befinden" bezeichnet. Ein Agent nimmt seine Umwelt über Sensoren wahr und verändert sie über seine Effektoren. Der Agent lebt in dieser Umgebung, bleibt also in ihr persistent. Wie diese Umgebung beschaffen ist, wird dabei nicht weiter festgelegt.

Reaktiv Ein Agent kann Veränderungen in seiner Umwelt erkennen und sein Verhalten rechtzeitig modifizieren. Damit verbundenen ist auch die Flexibilität im Verhalten, d.h. die Aktionen eines Agenten entsprechen nicht einem vordefinierten, festen Skript, sondern werden durch die Situation, in der sich Agent befindet, bestimmt. Als Synonym wird auch "responsive" verwendet.

Autonom bedeutet, daß der Agent seine Aktionen selbst bestimmt. Er entscheidet also selbst, wie er seine Sensorwerte auf die auszuführenden Aktionen abbildet [Maes 1994]. Allerdings ist es umstritten, was das genau heißt, bzw. wie es realisiert sein kann. [Russell & Norvig 1995] schreiben beispielsweise, daß ein Agent nur dann autonom ist, wenn sein Verhalten durch seine eigenen Erfahrung bestimmt ist. Andere bezeichnen einen Agenten schon als autonom, wenn er ohne oder nur mit sehr wenig Eingreifen eines Menschen (oder anderen Agenten) seine Ziele erfüllen kann: also der Agent nur auf der Basis seiner Sensordaten und seines internen Zustands selbstständig die nächste Aktion auswählt [Castelfranchi & Conte 1996]. Im Grunde werden dadurch zwei unterschiedliche Formen von Autonomie identifiziert, zum einen "Kontroll-Autonomie", zum anderen "Verhaltens-Autonomie": bei ersterem kontrolliert der Agent die Ausführung seines Verhaltensprogramms, bei letzterem kontrolliert er auch die "Programmierung" seines Verhaltens. Eine Beschreibung weiterer Kategorien findet man in [Huhns & Singh 1999], die verschiedene Formen von Autonomie auf unterschiedlichen, mehr oder weniger technischen Ebenen vorschlagen.

Sozial wird manchmal ebenfalls als eine das "Agent-sein" definierende Eigenschaft verwendet. Ein Agent interagiert mit anderen, unabhängig davon, ob er seine Einbindung in soziale Strukturen explizit repräsentiert oder nicht. Für manche Autoren (z.B. [Carley & Newell 1994, Castelfranchi 1998]) kann man nur dann von "sozial" sprechen, wenn sich der Agent anderer Agenten bewußt ist bzw. noch beschränkender, seine sozialen Beziehungen in Form von gemeinsamen Plänen, usw. repräsentiert. Bei anderen Autoren [Genesereth & Ketchpel 1994] ist es für einen "sozialen" Agenten bereits ausreichend, wenn dieser mit anderen über eine Sprache kommunizieren kann.

Rational Gerade wenn das Agentensystem ein sinnvolles kohärentes Verhalten zeigen soll, ist es wichtig, daß die einzelnen Agenten Ziele besitzen und diese verfolgen, also ziel-orientiert vorgehen. In diesem Zusammenhang wird auch oft die Eigenschaft "pro-aktiv" angegeben. Damit ist gemeint, daß das Verhalten eines Agenten mehr als bloße Reaktion auf Umweltgegebenheiten sein sollte. Er sollte auch selbst Initiative ergreifen können, d.h. zum Beispiel aufgrund einer internen Motivation neue Aktivitäten anstoßen können.

Anthropomorph Interessanterweise verbinden viele Forscher mit einem Agenten oftmals das Bild eines Menschen mit mentalen Konzepten, wie "Beliefs", Wünsche oder Absichten. Dieses Konzept findet man vor allem bei der Charakterisierung von Multiagentensimulation in den Sozialwissenschaften (siehe z.B. [Doran 1996]).

Als weitere Eigenschaft wird manchmal, besonders in Virtual Reality Simulationsanwendungen Glaubwürdigkeit genannt. Ein Beobachter sollte den Agenten als das, was er darstellen soll, erkennen. Sein Verhalten sollte für den Beobachter plausibel sein. Ebenso speziell sind die manchmal geforderte Mobilität (z.B. in den Eigenschaften der "strong agency" von

[Wooldridge & Jennings 1995]) oder Adaptivität (wird beispielsweise bei Interface-Agenten oft gefordert).

Die hier besprochenen Eigenschaften sollen die Essenz des "Agent-seins" deutlich machen. Welche Eigenschaften davon Agenten in der Multiagentensimulation besitzen, ist zum einen eine Frage des Abstraktionsniveaus des Modells und wird zum anderen davon bestimmt, welche Eigenschaften bei den Bestandteilen des im Modell abzubildenden Systems identifizierbar sind (siehe dazu auch Abschnitt 4.1).

2.1.2 Der Agent und seine Umwelt

2.1.2.1 Eigenschaften der Beziehung zwischen Agent und Umwelt

Wie in der vorangegangen Diskussion dargestellt, ist die Umwelt in der ein Agent "lebt" eine wichtige Größe bei der Betrachtung eines Agenten. Die wichtigsten Eigenschaften sind dabei die Zugänglichkeit, die Determiniertheit, die Relevanz der Historie, die Dynamik und die Diskretheit der Umwelt. Diese Charakteristika bestimmen zu großen Teilen das Design eines Agenten und werden im folgenden näher dargestellt [Huhns & Singh 1997, Russell & Norvig 1995]:

- Zugänglichkeit
  Welche Information kann der Agent aus seiner Umwelt bekommen? Wie zuverlässig sind diese Wahrnehmungen? Das führt im Endeffekt zu der Frage, wieviel der Agent, insbesondere über längere Zeit gesehen, über seine Umwelt wissen kann.

- Determiniertheit
  Wie weit ist der nächste Zustand der Umwelt durch die vorherige Situation und die Aktionen des Agenten bestimmt? Diese Frage bestimmt die Unsicherheiten, mit denen der Agent in dieser Umwelt umgehen muß. Für das Design eines Agenten ist die Determiniertheit der Umwelt relativ. Wenn die Umgebung nicht zugänglich ist, d.h. der Agent beispielsweise nicht die Ziele anderer Agenten kennt, kann die Umwelt, obwohl objektiv deterministisch, für den Agenten indeteriminstische Züge aufweisen.

- Relevanz der Historie
  Hängt der zukünftige Zustand der Umwelt – der gesamte oder der vom Agenten wahrnehmbare – von der gesamten Geschichte der Interaktion ab oder nur vom aktuellen Zustand? Hier spielt vor allem die Episodenhaftigkeit der Umgebung eine Rolle, kann der Agent von seinen aktuellen Perzeptionen ausgehen, also dem aktuellen Weltzustand, oder hängt die Zukunft von Aktionen ab, die schon länger zurückliegen.

- Dynamik
  Ändert sich die Umwelt, während der Agent noch "überlegt", welche Aktion er als nächstes ausführt sollte, d.h. wie ist die Relation zwischen Dynamik der Umwelt und Dynamik des Agenten? Derartige "Echtzeit"-Anforderungen erschweren das Agentendesign, da ein Beobachten der Umwelt durch den Agenten während einer längeren Planungsphase notwendig ist.

- Diskretheit
  Gibt es bei Wahrnehmungen oder Aktionen einen diskreten Wertebereich oder müssen

beispielsweise bei Aktionen kontinuierliche Werte geregelt werden, wie etwa die Geschwindigkeit eines Fahrzeugs. Damit verbunden ist auch die Abstraktionsebene, auf der der Agent die Umwelt betrachten und kontrollieren kann.

Jede Ausprägung entlang dieser Dimensionen stellt besondere Anforderungen an das Design der Agenten. Ein Agent muß in einer dynamischen Umwelt beispielsweise über eine Architektur-Komponente verfügen, die schnell Aktionen auswählt oder lang erarbeitete Pläne an eine veränderte Situation anpassen kann. Eine Kombination aus Zugänglichkeit, Relevanz der Historie und Dynamik charakterisiert, ob und wieviel der Agent von den Veränderungen in seiner Umwelt vorhersagen kann, usw. Der Zusammenhang zwischen Umweltcharakteristika und Agentendesign war auch ein Untersuchungsschwerpunkt des *Phoenix*-Projekts [Cohen et al. 1989].

Ebenso wie bei den Agenten ergibt sich für die Behandlung von Umwelten relativ zu diesem Agenten die Feststellung, daß diese Eigenschaften erst bei der Beschreibung eines konkreten Multiagentensystems, bzw. Multiagentenmodells relevant werden.

2.1.3 Eigenschaften von Multiagentensystemen

Bei der Betrachtung eines Multiagentensystems – als ein System von mehreren Agenten – spielen nicht nur die Eigenschaften der einzelnen Agenten eine Rolle. Diese Einheiten kommunizieren und interagieren miteinander, möglicherweise auch mit einer gesondert zu betrachtenden Umwelt. Beim Übergang von einem Ein-Agentensystem auf ein Multiagentensystem können weitere Eigenschaften identifiziert werden. Diese Charakteristika ergeben sich vor allem aus der mehr oder weniger ausgeprägten Autonomie der einzelnen Agenten. Eine Darstellung dieser Eigenschaften ist hier schon alleine deshalb interessant, weil sie helfen, ein "natürliches" Agentensystem zu identifizieren. Aus diesen Charakteristika lassen sich auch typische Probleme bei der Konstruktion und Implementierung ableiten.

Folgende allgemeine Charakteristika kann man einem Multiagentensystem zuordnen [Jennings et al. 1998].

- Jeder Agent besitzt nur unvollständige Informationen oder beschränkte Problemlösefähigkeiten und somit eine beschränkte Sicht auf das Gesamtsystem.
- Damit verbunden ist eine dezentrale Datenhaltung, d.h. jeder Agent verwaltet seine Daten lokal.
- Nicht nur die Daten werden lokal gespeichert, auch die Berechnungen, die jeder Agent ausführt, sollten im Idealfall asynchron geschehen. Damit wird die Behandlung von Nebenläufigkeiten wichtig, d.h. Berechnungen und die Ausführung der Aktionen verschiedener Agenten geschehen parallel.
- Im Idealfall verfügen Multiagentensysteme über keine zentrale Kontrolle. Das ist eine direkte Folge aus einem geforderten Maß an Kontroll-Autonomie der einzelnen Agenten.

Manche dieser Eigenschaften, z.B. die der fehlenden zentralen Kontrolle, müssen oft nicht nur für simulierte Multiagentensysteme relaxiert werden. Wenn es z.B. eine zentral gesteuerte Zeitvorgabe gibt oder eine gemeinsame (metrische) Umgebung, könnte man schon von einfachen Formen zentraler Kontrolle sprechen.

Traditionell kann man die so charakterisierten Systeme in Verteilte Problemlöser und Multiagentensysteme im engeren Sinne unterteilen [Bond & Gasser 1988a, Huhns & Singh 1997, Bamberger 1999]. Die Trennung gründet auf der Verteilung von Zielen im System: Während bei Verteilten Problemlösern alle Agenten ein globales Ziel verfolgen, also ihre individuellen Ziele dem Gesamtziel zuarbeiten, muß dies bei einem Multiagentensystem i.e.S. nicht der Fall sein. Jeder Agent ist "egoistisch"; es können demnach auch konfliktionäre Ziele im Gesamtsystem vorhanden sein. Während der Entwurf eines Verteilten Problemlösers demnach eher einer "top-down"-Methodik folgt, wird ein Multiagentensystem i.e.S. eher "bottom-up" entwickelt. Der Fokus liegt dabei bei den einzelnen Agenten und ihrem "Zusammenspiel" und weniger auf einer Verteilung von Aufgaben und der Integration von Teillösungen.

Allerdings – wie auch oben deutlich wurde - verschwimmt diese Unterscheidung immer mehr [Durfee & Rosenschein 1994, Weiss 1999]. Dabei kann man Multiagentenmodelle, wie sie in Kapitel 4 charakterisiert werden, auf der Seite der Multiagentensysteme i. e. S. einordnen – obwohl das konkrete Modell, abhängig vom Originalsystem, durchaus auch Züge eines Verteilten Problemlösers annehmen kann.

Unabhängig, davon welche Ziele die einzelnen Agenten relativ zum Gesamtsystem verfolgen, kann man zwei Beschreibungsebenen identifizieren: Die Ebene der einzelnen Agenten mit der Beschreibung ihres Verhaltens in Reaktion auf ihre "individuelle" Umwelt und ihren internen Zustand und die Ebene des Gesamtsystems mit einer Darstellung von Organisationen, Konstellationen und Interaktionen, die nicht unbedingt explizit auf dieser Ebene repräsentiert sein müssen. Diese beiden Ebenen werden im folgenden näher betrachtet.

2.2 Beschreibungsebenen

2.2.1 Agentenebene

Viele der oben besprochenen Eigenschaften eines Agenten, bzw. der Beziehung eines Agenten zu seiner Umgebung lassen sich zur Beschreibung von Agenten nutzen. [Huhns & Singh 1997] unterscheiden zur weiteren Charakterisierung intrinsische und extrinsische Kategorien. Die ersteren sind durch den Agenten selbst bestimmt, beispielsweise seine Adaptivität, während die letzteren nur in Verbindung mit anderen Agenten Sinn machen. Die Kooperationsbereitschaft ist z.B. ein Charakteristikum, das einen Agenten abhängig von anderen beschreibt. Die Autonomie eines Agenten ist immer nur relativ zu anderen Agenten oder der Umwelt, nie absolut.

Die wohl wichtigste Dimension ist die für das Design, bzw. für die Beschreibung eines Agenten verwendete Kognitionsebene. Dabei wird festgelegt, wie komplex die interne Struktur des Agenten ist und wie aufwendig sein internes Modell behandelt wird. Wieviel Wissen über sich und seine Umwelt, über die Folgen seiner Aktionen benutzt der Agent, um sein Verhalten zu bestimmen? Kann er dieses Verhalten auf der Basis von Lernen adaptieren und seine Performanz verbessern? Die Antworten auf diese Fragen bestimmen, wie differenziert die Beschreibung eines Agenten und seines Verhaltens ist und auf welcher Ebene diese Beschreibung stattfindet: also ob die Ziele eines Agenten beschrieben werden oder sein Eingabe-Ausgabe-Verhalten.

Im folgenden werden zunächst verschiedene Kategorien von möglichen "Schablonen" für die Konzeption eines Agenten, also Agentenarchitekturen betrachtet. Im Anschluß wer-

den anhand einer eigenen Kategorisierung vorhandene Strukturen beschrieben.

2.2.1.1 Bekannte Klassifikationen von Agententypen

Die am häufigsten gemachten Untersuchungen zum Design von Agenten beschäftigen sich mit den Architekturen, also den formalen internen Strukturen. Allerdings findet man in der Literatur keinerlei Einigkeit, wie diese Einteilung konsequent und konsistent aussehen sollte. Wegen der Fülle und Unterschiedlichkeit existierender Architekturen ist es auch kaum möglich, alle bisher entworfenen Formen in einer hierarchisch organisierten Taxonomie unter zu bringen. Bevor eine eigene Einteilung für simulierte Agenten dargestellt wird, werden drei der prominentesten Klassifikationen für Agentenarchitekturen skizziert: die Einteilung nach J. Müller, die formale Klassifikation nach M. Genesereth und N. Nilson und zuletzt die Agentenkategorien nach S. Russell und P. Norvig.

Die in der Verteilten Künstlichen Intelligenz gängigste Unterteilung wurde von J. Müller veröffentlicht [Müller 1996]. Dabei unterscheidet er folgende vier Kategorien.

- *deliberative Architekturen*
  Dazu zählen bei ihm ausschließlich BDI-Agenten (siehe dazu Seite 22).
- *reaktive Architekturen*
  Damit sind die Agenten gemeint, die weiter unten als subkognitive Agenten eingeführt werden.
- *interagierende Agenten*
  Kooperationsformen, gemeinsame Pläne, usw. haben eine direkte Repräsentation in der Architektur
- *hybride Architekturen*
  Damit werden keine Mischformen beliebiger anderer Kategorien bezeichnet, sondern Architekturen, bei denen eine Planer-Komponente mit reaktiven Modulen zusammengeschaltet wird.

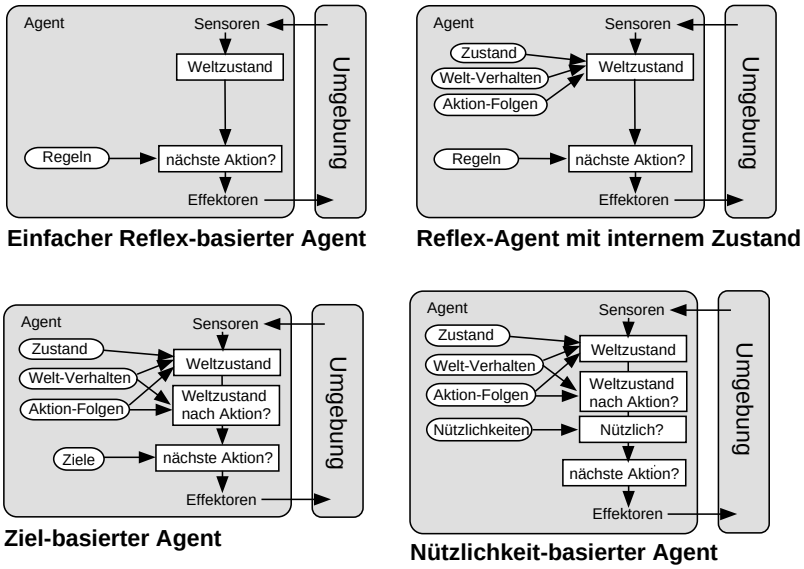
Diese Einteilung spiegelt direkt wider, daß der Schwerpunkt bei Architekturen auf dem gemeinsamen Problemlösen von Agenten liegt. Es wird auch deutlich, daß die Bestandteile einer Architektur, ihre Module und deren Interaktion eng damit verbunden sind, welches Wissen sie beinhalten. In diesen Strukturen sind Weltmodelle oder Wissen über die Kooperationspartner, parametrisierbare Pläne oder fixe Handlungsanweisungen repräsentiert. Form und Inhalt der internen Wissensstrukturen, die der Agent als Basis für seine Verhaltenssteuerung verwendet, bieten eine Grundlage für andere Klassifikationen der internen Strukturen eines Agenten.

Eine der prominentesten der formalen Klassifikationen wurde von M. Genesereth und N. Nilson entwickelt [Genesereth & Nilson 1987]. Sie unterscheiden im Prinzip nur zwischen einem “tropistic agent”, einem reines Stimulus-Response-System ohne irgendeine Form von internem Modell, und einem “hysteretic agent”, der im Gegensatz dazu über einen intern repräsentierten Zustand verfügt. Wenn dieser interne Zustand als Wissensbasis interpretiert werden kann, sprechen [Genesereth & Nilson 1987] von einem “knowledge-level agent”. Sind die Daten in der Wissensbasis mit Zeitstempeln attributiert, dann wird der Agent als ein “stepped knowledge-level agent” bezeichnet. Diese Einteilung dient als Basis für die

Untersuchung von Agenten in [Ferber 1999]. Sie wurde aber im Grunde nur für einen einzigen Agenten in einer deterministischen Umwelt ohne Dynamik, die nicht vom Agenten selbst induziert wird, erstellt. Dennoch kann sie eine formale Basis für eine konsequente Einteilung von Multiagentensystemen bieten, wie in [Fulbright & Stephens 1994] dargestellt (siehe dazu auch Abschnitt 5.1.1.2).

Eine ähnliche Einteilung findet man in [Russell & Norvig 1995], die weniger eine formale Grundlage zur Beschreibung von Agenten legen wollen, sondern "Agentenskelette" darstellen, die die Basis für eine lehrbuchartige Behandlung der Künstlichen Intelligenz bilden. Die einzelnen Agentenstrukturen werden dort folgendermaßen näher charakterisiert (siehe Abbildung 2.1)

Abbildung 2.1 Agententypen nach S. Russell und P. Norvig [Russell & Norvig 1995]



1. Ein einfacher Reflex-Agent besitzt als statisches Wissen nur Situation-Aktion-Regeln. Die Aktionsselektion geschieht zu jedem Zeitpunkt in drei Schritten: Die Sensordaten werden interpretiert und daraus eine abstrakte Situationsbeschreibung erzeugt. Für diese wird eine passende Aktion mit Hilfe von Situation-Aktion-Regeln gesucht.

2. Ein Reflex-Agent mit internem Zustand ist insbesondere in unzugänglichen Umgebungen leistungsfähiger, da er ein Bild der Umwelt, bzw. die relevanten Teile davon, in seinem Speicher haben kann und so mehr Information über die Welt bei der Auswahl der Situation-Aktion-Regeln benutzen kann. Ein wichtiger Punkt dabei ist, wie genau das Bild der Welt mit der Wahrnehmung konsistent gehalten werden kann bzw. mit wieviel Aufwand dies geschehen muß. Grundsätzlich benötigt der Agent dazu zwei Arten von Wissen zur Extrapolation des internen Weltmodells: Wissen über das Verhalten der Welt und Wissen über die Auswirkungen seiner Aktionen.

3. Ein Ziel-basierter Agent besitzt zusätzlich zum Wissen über die aktuelle Situation eine explizite Repräsentation seiner Ziele, die erstrebenswerte Situationen beschreiben.

Zusammen mit dem Wissen über die Folgen seiner Aktionen kann der Agent solche Aktionssequenzen planen, die zu diesen Zielsituationen führen.

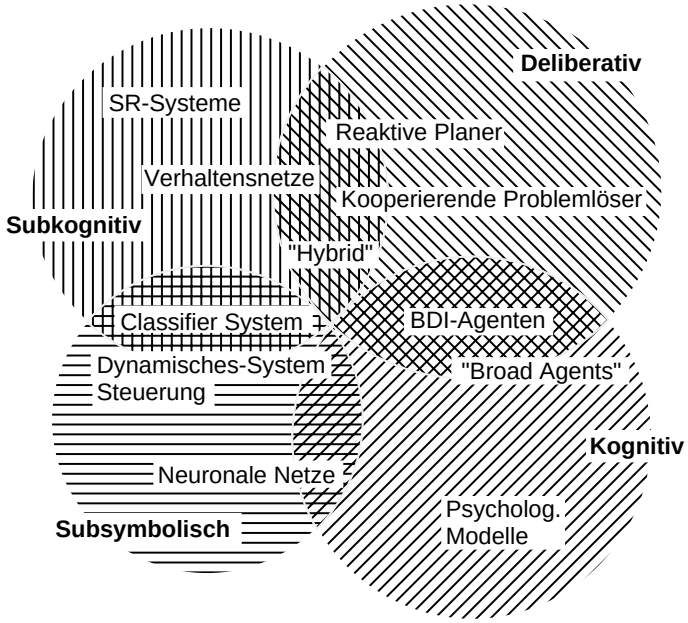
4. Ein Nützlichkeit-basierter Agent verfügt über eine komplexere Situationsbewertung als ein Ziel-basierter Agent, mit der er in der Lage ist, besser zwischen unterschiedlichen Wegen zum Ziel oder auch mehreren Zielen abzuwägen.

Diese Einteilung fokussiert mehr die für die Verhaltenssteuerung zur Verfügung stehenden Wissen und der Komplexität der dieses Wissen nutzenden Mechanismen, als auf konkreten Architekturen. Im folgenden werden nun derartige Formen vorgestellt. Dabei wird eine alternative Einteilung zugrunde gelegt werden, in der auch Ansätze aus dem Artificial Life und anderen Gebieten einbezogen werden können. Ein besonderer Fokus soll darauf liegen, mit welchem Aufwand man Agentenverhalten mit diesen Strukturen beschrieben werden kann.

2.2.1.2 Agentenarchitekturen

In der nun folgenden Einteilung wollen wir auch einfachere Architekturformen mit einbeziehen. Abbildung 2.2 zeigt einen Überblick mit den vier Haupt-Architekturformen, subsymbolische, subkognitive, deliberative und kognitive Strukturen.

Abbildung 2.2 Überblick über Agentenarchitekturen



Durch die sich schneidenden Kreise soll verdeutlicht werden, daß auch diese Einteilung die Fülle der Strukturen nicht klar trennen kann. Außerdem fehlt die Dimension, anhand derer charakterisiert werden kann, wie kompliziert die Formulierung von Verhalten mittels der jeweiligen Strukturen ist: Werden Aktionen ausgewählt, müssen Planschablonen gefüllt oder Pläne generiert werden? Diese Fragen werden im Folgenden bei den jeweiligen Formen erläutert.

Subsymbolische Architekturen findet man vor allem im Bereich Artificial Life und verwandten Gebieten. Subsymbolisch bedeutet hier, daß die nächste Aktion über Mechanismen ausgewählt wird, die keinerlei symbolische Repräsentation manipulieren oder auf symbolischem Schließen beruhen. Im folgenden sollen als repräsentative Beispiele Neuronale Netze, "kybernetische" Regelkreise und andere netzartige Strukturen, bei denen Sensoren direkt mit Effektoren verbunden sind, vorgestellt werden. "Classifier systems" bilden eine letzte Kategorie der subsymbolischen Agentenarchitekturen.

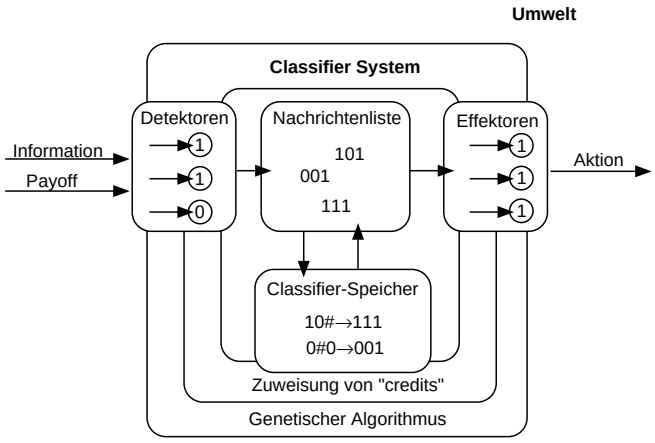
Neuronale Netze assoziieren zur Verhaltenssteuerung Sensorwerte mit Effektorkommandos. So wird die zu evolvierende Verhaltensfunktion der Agenten im ANTFARM-Szenario [Collins & Jefferson 1991b, Collins & Jefferson 1991a] auf diese Weise repräsentiert. Neuronale Netze werden auch deshalb gerne verwendet, da sie ein sehr vereinfachtes Modell für kognitive Prozesse im Gehirn darstellen (siehe dazu auch das Modell von [Nolfi & Parisi 1995]).

Als Steuerung für ein "Dynamisches System" wird eine Architekturform bezeichnet, in der keine expliziten Aktionen auswählt werden. Die Werte der Sensoren, meist als Zahlen- oder Binärwerte, werden manipuliert und kombiniert und die so veränderten Werte als Kontrollparameter an die Effektoren weitergeleitet. Derartige Strukturen findet man oft in einfachen Robotern, bei denen beispielsweise die Geschwindigkeit der Motoren direkt aus der wahrgenommenen Stärke einer Lichtquelle berechnet wird. Die Basismechanismen sind aus der Kybernetik bzw. analogen Regelungstechnik motiviert. Aber auch die über "Potentiale" gesteuerten Agenten, wie zum Beispiel die auf Pheromone reagierende simulierte Ameise bei [Millonas 1994] gehören dazu. Die Hoffnung bei derartigen Steuerungen ist vor allem, das Verhalten des Agenten und die Interaktion mit seiner Umwelt mittels mathematischer Verfahren analysieren zu können [Beer 1995]. Das klassische Beispiel für derartige Einheiten sind allerdings die "Vehicles" von V. Braitenberg [Braitenberg 1984], der damit zeigen wollte, daß Intelligenz vor allem im Auge des Betrachters dieser Roboter liegt. Derartige Strukturen wurden auch in Systemen des Vivarium Projekts [Coderre 1988, Travers 1988] benutzt, bei dem Jugendliche auf spielerische Weise lernen sollten, das "Gehirn" eines Tiers konstruieren, um so das Zusammenspiel von Denken und Verhalten zu lernen.

Bisherige Agentenarchitekturen dieser Kategorie beruhen auf relativ einfachen Mechanismen und "Rechenschritten". Sensorwerte werden modifiziert und quasi zu den Effektoren durchgereicht. Es wird keine vordefinierte Aktion explizit repräsentiert und auswählt – wenn man das Ergebnis der Klassifikation mittels eines Neuronaleb Netzes nicht als direkte Aktionsrepräsentation interpretiert. Allerdings ist es nicht einfach, eine adäquate Struktur für die Anordnung der jeweiligen Bausteine zu finden.

Eine andere Form subsymbolischer Aktionsselektion mit einem expliziten Aktionsauswahlmechansimus findet man bei "Classifier systems" [Goldberg 1989]. Hierbei handelt es sich um regelbasierte Systeme, mit sehr einfachen, auf Bit-Strings basierenden Regeln, deren Kondition gegen die Inhalte einer "Message-List" verglichen werden und diese Liste mit ihren Aktionen verändern können. Das besondere an diesen Systemen ist der eingebaute Lernmechanismus. Die Regeln sind bewertet, diese Präferenzen werden über einen Reinforcement Mechanismus an den Erfolg der Aktion angepasst. Ein weit verbreiteter Algorithmus ist das "Bucket-Brigade": dabei wird das Gewicht der Regeln erhöht, deren Feuern als Sequenz zur Auswahl der letzten Aktion führte. Dabei nimmt die Gewichtsdifferenz abhängig von der Aktualität des Feuerns der Regel ab. In bestimmten Intervallen werden auf der Basis eines Genetischen Algorithmus neue Regeln erzeugt und die Regelmenge aktualisiert.

Abbildung 2.3 Schematische Darstellung eines “classifier systems” [Goldberg 1989] und seiner Funktionsweise: Die “classifier” operieren auf der sogenannten Nachrichtenliste. Dorthin werden auch die von den Sensoren ermittelten Werte geschrieben, die dabei ein (mehrstufiges) Regelfeuern anstoßen. Erfolgen keine Änderungen mehr in dieser “message list”, wird eine Aktion ausgewählt und an die Effektoren weitergeleitet. Die Gewichte der Regeln werden auf der Basis des Erfolgs der zugehörigen Aktion adaptiert. In gewissen Zeitabständen verändert ein Genetischer Algorithmus die Regelmenge selbst.



Subkognitive Architekturen werden auch als “reaktiv”[2] bezeichnet. Bei ihrer Entwicklung stand meist das Ziel im Vordergrund, einen einfachen Agenten zu schaffen, der vor allem in einer realitätsnahen, oder zumindest interessanten Umwelt überlebt. Eine darüber hinaus gehende Zielerfüllung war dabei nur zweitrangig bzw. ergibt sich aus der Interaktion einfacher Strategien mit einer komplexen Umwelt [Agre 1989, Brooks 1991b, Brooks 1991a].

Subkognitive Architekturen unterscheiden sich von den subsymbolischen, die ebenso ohne ein internes Weltmodell auskommen, vor allem dadurch, daß zumindest im Laufe der Implementierung symbolische Repräsentationen verwendet werden. Da die zugrundeliegenden Verhaltensmodelle von Menschen konstruiert werden, tragen subkognitive Architekturen deutlich antropogenere Züge. Ein anderes wichtiges Unterscheidungsmerkmal, vor allem zu den deliberativen Architekturen, liegt darin, daß die Auswahl der nächsten Aktion auf der Basis von aktuellen Sensorwerten geschieht. Es wird keinerlei explizites internes Weltmodell aufgebaut, aktualisiert und dieses zur Aktionsselektion benutzt. Es werden also keine Aktionen deshalb ausgeführt, weil man einem bestimmten geplanten Zustand herstellen will. Im folgenden sollen verschiedene Formen wie Stimulus-Response-Regeln mit unterschiedlich komplexen Auswahlmechanismen oder Verhaltensnetze dargestellt werden.

Die einfachste subkognitive Architektur sind Stimulus-Response-Regeln. Eine in der Umwelt wahrgenommene (Teil-)Situation wird direkt auf eine Aktion abgebildet, in dem eine aus einem Fundus vordefinierter Aktionen ausgewählt wird. Sind mehrere Aktionen möglich, ist eine wie auch immer geartete Konfliktauflösung nötig, z.B. über eine Verrechnung von Aktionsgewichten mit Reizstärken. Eine Architektur, die auch für die Simula-

[2] Obwohl der Begriff “reaktiver” Agent in der Literatur deutlich häufiger als “subkognitiver” Agent für Agenten mit derartigen Architekturstrukturen benutzt wird, möchte ich ihn dennoch in dieser Arbeit als Architekturbezeichnung vermeiden, da er treffender eine Eigenschaft als eine Struktur bezeichnet.

tion von Ameisenstaaten verwendet wird, ist das “ECO Modelling Frame” [Ferber & Jacopin 1991, Drogoul et al. 1991]. Dabei wird — vereinfacht dargestellt — jedem Reiz, der pheromonartig über die diskrete Karte propagiert wird, eine bestimmte Aufgabe und eine Routine zu ihrer Erfüllung zugeordnet. Ein Agent, der eine Menge von Reizen auf seiner aktuellen Position wahrnimmt, berechnet Bewertungen für die Tasks basierend auf der Stärke des wahrgenommenen Reizes und dem Gewicht der Task. Die höchstbewertete Tätigkeit wird ausgewählt und im aktuellen Zeittakt ausgeführt, danach beginnt das Spiel von vorne.

Andere subkognitive Architekturen kann man unter dem Stichwort “Verhaltensnetze” zusammenfassen. Eine netzartige Struktur ist oft schon dadurch gegeben, daß sich die verschiedenen Aktivitäten gegenseitig beeinflussen und dies bei deren Bewertung in Rechnung gestellt wird. Ein Beispiel ist die “dynamic action selection” von P. Maes [Maes 1991], bei der verschiedene Aktivitäten durch Kanten verbunden sind, deren Existenz aus STRIPS-Operatordarstellungen abgeleitet werden kann. Über diese Kanten wird bei der Aktionsselektion Aktivierungsenergie von externen Sensorknoten und internen “Motivationen”, z.B. Hunger, durch das Netz propagiert. Wenn das Netz zur Ruhe gekommen ist, stellt der “Gewinner–Knoten” die nächste Verhaltensweise. C. Oechslein [Oechslein 1997] implementierte ein Beispiel für ein “dynamic action selection”–Netz, mit dem (gezieltes) Verhalten in einem dynamischen Supermarktszenario nachgebildet werden sollte. Dabei mußte man feststellen, daß das Netz relativ lange braucht, um zur Ruhe zu kommen und auf einer balancierten Aktivierungsverteilung aufbauend, die nächste Aktion auszuwählen.

In anderen Verhaltensnetzen, wie z.B. der Subsumptionsarchitektur [Brooks 1986] oder den emergenten Robotersteuerung von L. Steels [Steels 1994], sind verschiedene Aktivitäten, wie “Kollisionsvermeidung” und “Navigation” gleichzeitig und ständig aktiv. Durch ihr Zusammenspiel werden die Steuerbefehle für die Effektoren erzeugt bzw. parametrisiert. Auch schaltkreis–ähnliche Strukturen gehören zu diesen subkognitiven Formen, unabhängig davon, ob sie nach bestimmten Zielvorgaben mit Wissen über das Erreichen dieser Ziele kompiliert werden [Kaelbling & Rosenschein 1991, Kaelbling 1988] oder per Hand entworfen [Agre & Chapman 1987] sind. P. Agre benutzt zur Ansteuerung seines Schaltkreises “indexical–functional aspects”, die die aktuell relevanten Bestandteile der Umwelt beschreiben.

Deliberative Architekturen sind im Grunde alle Architekturen, die ein internes Weltmodell benutzen, um mit dessen Hilfe eine Folge von Aktionen zu planen. Meist besitzt der Agent darüber hinaus eine explizite Repräsentation seiner Ziele.

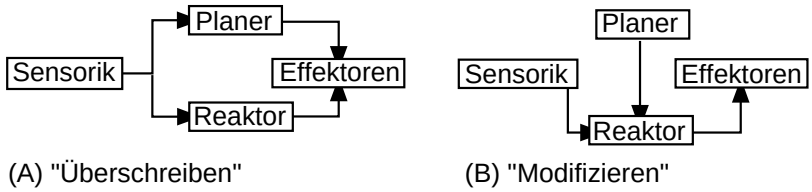
Besonders verbreitet sind derartige Agenten in den Bereichen, in denen agentenbasierte Systeme in Szenarien zum verteiltem Problemlösen eingesetzt werden, beispielsweise zur koordinierten Steuerung verteilter Systeme, sei es in Transportunternehmen [Fischer et al. 1995] oder Elektrizitätswerken [Jennings 1994]. Jeder Agent verfügt über Fähigkeiten zum eigenständigen Problemlösen, und kann über eine Art Interaktionsmodul mit anderen in Verbindung treten. Verhandlungsmechanismen, Erzeugung und Verwaltung von gemeinsamen Plänen, usw. treten bei der Beschäftigung mit derartigen Agenten in den Vordergrund [O’Hare & Jennings 1996, Müller 1993]. Für die Anwendung in Simulationsszenarien, bei denen derartiges Problemlösen in einer gegebenen Umwelt abstrahiert werden soll, kann man einfachere Architekturen identifizieren, wie reaktive Planer und hybride Systeme. Eigentlich müßte man in dieser Kategorie auch *BDI*-Agenten beschreiben, die nach A. Rao und M. Georgeff die einfachste Architekturform, mit der rationales Verhalten erzeugt werden kann

[Rao & Georgeff 1995]. Allerdings kann man mit *BDI*-Agenten auch eine (psychologische) Theorie zur Intentionalität verbinden. Aus diesem Grunde werden sie im nächsten Abschnitt behandelt.

Statt wie “klassische Planer” zuerst einen vollständigen Plan aufzustellen und diesen in einem zweiten Schritt abzuarbeiten, berücksichtigen reaktive Planer, daß sich die Umwelt während der Planerstellung und -ausführung ändern und somit der Plan inkorrekt werden kann. Zur Realisierung der dabei notwendigen Reaktivität gibt es mehrere Möglichkeiten. Eine Möglichkeit ist es, die Ausführung des Plans zu überwachen und vor jedem Schritt festzustellen, ob dieser durchführbar ist. Derartig erweiterte Planungsverfahren werden in Lehrbüchern zur Künstlichen Intelligenz ausführlich besprochen, z. B. in [Russell & Norvig 1995]. Als Beispiel sei hier nur das Planen mit *Reactive Action Packages* (RAPs) [Firby 1989] erwähnt. Ein RAP stellt dabei einen Planbaustein dar, der entsprechend des Ziels, das er erfüllen kann, ausgewählt wird. Neben dem Zieltest und einer Kontextrepräsentation besteht ein RAP aus einem Netz weiterer abstrakter Planbausteine oder konkreten Aktionen. Abhängig von der aktuellen Situation des Agenten wird dieses Netz expandiert, in dem es in eine “Task-Agenda” einsortiert bzw. wird an die Effektoren geschickt wird. Auf diese Weise geschieht ein situationsabhängiges Expandieren vordefinierter Planbausteine. Dadurch daß abstraktere Planbausteine direkt mit Zielsituationen verknüpft sind, kann man *RAP* eher den einfacheren Mechanismen zur Planauswahl zuordnen.

Weitere Ansätze zur Kombination von reaktivem und zielgerichtetem Verhalten findet man in den sog. “hybriden” Architekturen. Dabei werden verschiedene Module, eines für schnelles reflexartiges Reagieren, ein anderes für geplantes Vorgehen zusammengeschaltet. Abbildung 2.4 zeigt zwei mögliche Formen derartiger Systeme [Parunak 1996]. H. van Dyke Parunak benutzt in seiner Klassifikation von Agentenarchitekturen diese beiden Formen als Zwischenstufen zwischen rein “reaktiven” Architekturen und puren Planern, wobei die jeweiligen Planer-Module durchaus die Mächtigkeit eines Plan-Generierers verwenden können.

Abbildung 2.4 Formen “hybrider” Architekturen nach [Parunak 1996]



In der Architekturform (A) gibt nach [Parunak 1996] der Reaktor die Aktionen an die Effektoren. Das Planungsmodul überwacht den Reaktor und überschreibt seine Aktionen, falls diese das Ziel nicht erfüllen. Der Sinn liegt in einer Echtzeitumgebung darin, daß der Reaktor schneller Aktionen produzieren kann als ein Planer, dessen Aktionen aber für ein zielgerichtetes Verhalten sinnvoller sind. Eine Kombination in der Form von (A), allerdings mit vertauschten Rollen, also ein Reaktor, der eingreift, wenn die geplanten Aktionen fehlschlagen, ist ebenfalls denkbar; sie würde den weiter oben besprochenen reaktiven Planern entsprechen. Bei der Architekturform (B) modifiziert ein Planer die Parameter der Aktionen des reaktiven Moduls, wie es bei beispielsweise bei den verschiedenen Ebenen der Subsumptionsarchitektur ([Brooks 1986], siehe auch Seite 20) der Fall sein soll. Derartige “hybri-

de" Systeme können um weitere Schichten erweitert werden. Ein Beispiel sind die Touring-Machines [Ferguson 1995], die über eine dritte Ebene der Umweltmodellierung verfügen. In dieser werden Modelle über das Verhalten der Umwelt, die Effekte der eigenen Aktionen, erstellt und aktualisiert. Agenten mit der InteRRaP–Architektur [Müller 1996] können mittels einer dritten Schicht ihre Kooperation bzw. andere soziale Interaktionen steuern.

Kognitive Architekturen im engeren Sinne[3] beruhen auf kognitionswissenschaftlichen oder psychologischen Modellen und Vorstellungen. Zu dieser Klasse gehören einige bedeutende Architekturen und Architekturformen: BDI-Agenten folgen einer Theorie zur Intentionalität und sind beim Einsatz in verteilten Problemlöseszenarien sehr erfolgreich. In "virtual reality" Umgebungen, o. ä. findet man immer mehr vielseitige Agenten, kurz "broad agents", die plausibles Verhalten produzieren sollen und dazu auf kognitionswissenschaftlichen oder ethologischen[4] Modellen beruhen. Eine derartige Architektur, die gerade Simulationen von menschlichen Gesellschaften häufig zugrunde liegt, ist *Soar*. Diese wird im übernächsten Paragraph näher besprochen.

Die Klasse der sogenannten BDI (Belief–Desire–Intention)– Agenten beruht auf den Arbeiten von Dennett, der 1978 das Verhalten von "intentional systems" durch Attribute wie "beliefs", "preference" oder "intention" beschrieb [Haddadi & Sundermeyer 1996]. Derartige "mental states" kann man nach [Kiss 1992] in drei Kategorien unterteilen. Jeweils ein Repräsentant dieser Klassen stellt einen Bestandteil einer BDI-Architektur.

1. *kognitive* Attribute repräsentieren und manipulieren das Wissen, das der Agent über seine Umwelt sammelt, dazu zählen die "Beliefs".

2. *konativ* sind Konzepte, die direkt mit den Aktionen eines Agenten zu tun haben, also "intention". Unter diesem Begriff werden in BDI-Systemen vor allem die konkreten Pläne des Agenten verwaltet, die untereinander konsistent die Erfüllung der aktuellen Ziele ermöglichen.

3. *affektive* Attribute stehen für relative unbestimmte "Gefühle", also "desires".

In BDI-Architekturen werden damit Aspekte der Motivation des Agenten bezeichnet [Haddadi & Sundermeyer 1996], meist jedoch werden unter diesem Attribut die Ziele des Agenten beschrieben.

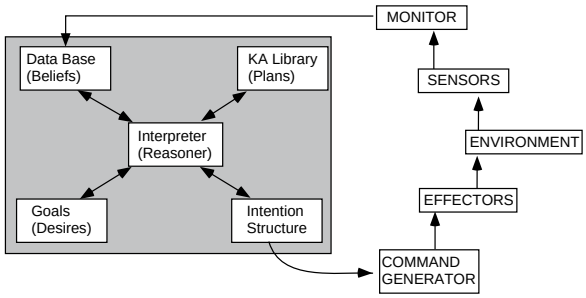
"procedural reasoning system" als eine der erfolgreichsten BDI-Architekturen Die wohl einflußreichste BDI-Architektur ist das "procedural reasoning system (PRS)" [Georgeff & Ingrand 1989, Ingrand et al. 1992], dessen Struktur in Abbildung 2.5 dargestellt wird.

Auf der Basis aktiver Ziele und Daten aus dem internen Weltrepräsentation (in der "data base") sind "knowledge areas" (KAs) anwendbar. Diese sind Planbausteine, die eine deklarative Spezifikation von prozeduralem Wissen darüber beinhalten, wie Ziele erreicht werden können und in bestimmten Notfallsituationen reagiert werden kann. Sie bestehen aus einem "Body", der die Verfeinerung des KA beschreibt und "invocation condition", einer Anwendbarkeitsbedingung. Der Interpreter wählt eine KA aus und plaziert sie auf der "intention structure". Intentions sind dabei Aufgaben, die das System zur Ausführung ausgewählt hat.

[3] Wie oben bereits erwähnt, wird manchmal keine Trennung zwischen kognitiven und deliberativen Architekturen gemacht.

[4] Ethologie ist die Tierverhaltensforschung.

Abbildung 2.5 Struktur des “procedural reasoning system” [Georgeff & Ingrand 1989]: Die “data base” enthält dabei die verfügbare Information über die Umwelt, Die “KA library” ist eine Sammlung möglicher “knowledge areas”(KAs), während die “Intention structure” alle gerade aktiven KAs beinhaltet.



Sie bestehen aus einer Start-KA und ihren Verfeinerungen. Über Meta-KAs werden nicht nur Präferenzen zur Sortierung der Intentionen bestimmt, sondern auch Konflikte zwischen auszuwählenden KAs aufgelöst, Daten in der “Data base” modifiziert, usw.

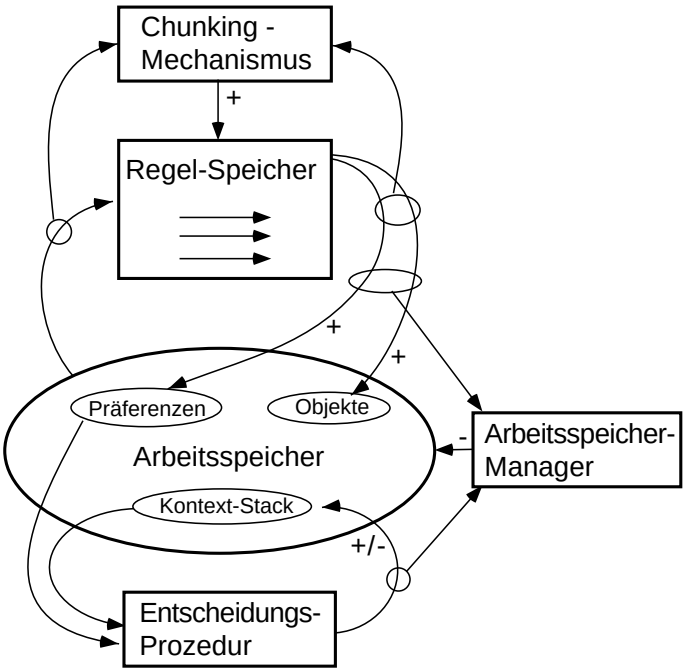
Die grundlegende Planrepräsentation in den “knowlegde areas” und ihre Behandlung durch Auswahl und sukzessive Verfeinerung ist wohl vergleichbar mit den RAPs (siehe oben). Allerdings werden die Ziele des Agenten explizit repräsentiert, demzufolge können im Gegensatz zu RAP, in denen die Ziele implizit in den Planbausteinen codiert werden, mehre Ziele mit unterschiedlichen Präferenzen gleichzeitig aktiv sein. Obwohl das PRS-System schon etwas älter ist, ist es zum einen wegen seines erfolgreichen Einsatzes in verschiedenen Anwendungen und zum anderen wegen seiner Reimplementierung im *dMars*-System [AAII 1996] immer noch wichtig und aktuell.

Während man BDI-Agenten wie deliberative Architekturen vor allem im Anwendungsbereich kooperierender wissensbasierter Systeme findet, gehören die Anwendungsgebiete der “broad agents” in den Bereich der Umgebungen für Virtuelle Realitäten. Damit wird kein spezielles Architekturmuster charakterisiert, sondern Ansätze, die für menschliche Benutzer möglichst plausible Interaktionsmöglichkeiten erzeugen und dabei deutlich komplexer als die BDI-Architekturen sind: Oft werden Module für reaktives Verhalten, Plangenerierung und Motivationsmodelle mit komplexen graphischen Animationen oder Verstehen und Generieren natürlicher Sprache gekoppelt (für einen Überblick siehe [Maes 1995], aktuelle Arbeiten sind [Dryer 1999, Hayes-Roth & Doyle 1998]). Folgende Architekturen sind interessante, noch relativ einfache Beispiele: [Terzopoulos et al. 1994] konstruierten detaillierte Verhaltensmodelle für Fische, mit der Motivation, eine realistische Darstellung der Tiere in einem simulierten Meer zu schaffen. Ein weiteres Beispiel findet sich in [Bates et al. 1994]; hier werden Emotionen als wichtige verhaltenstreibenen Faktor in die Architektur miteinbezogen. Auch für “klassische” Simulationsanwendungen, wie die Nachbildung von Produktions- und logistischen Prozessen werden Anstrengungen unternommen, beteiligte Menschen detaillierter nachzubilden. So entwickelte C. Urban eine Architektur für simulierte (menschliche) Agenten [Urban 2000] (*PECS*, siehe dazu Abschnitt 4.4.1.2), die die psychologischen Theorien von D. Dörner [Dörner 1999] nachbildet. Zu den “broad agents” kann man zuletzt auch Spielcharaktere, wie *Creatures* zählen [Grand et al. 1997], auch wenn ihre Architektur kaum symbolische Strukturen beinhaltet.

Soar als besondere Rahmenarchitektur Wie oben schon erwähnt, gehören zu den kognitiven Architekturen diejenigen, die auf einer kognitionswissenschaftlichen Theorie basieren oder ihre Implementierung darstellen. Soar [Laird et al. 1987, Rosenbloom et al. 1991] ist dabei die auch in Simulationsanwendungen am weitesten verbreitete. Sie beruht auf mehreren Hypothesen über intelligentes Handeln. Dazu gehört unter anderem, daß jedes Problemlösen als (heuristische) Suche in Problemräumen formuliert werden kann, und eine regelbasierte Implementierung in einem Symbolsystem das geeignete Mittel zur Repräsentation derartiger intelligenter Aktionen ist [Newell & Simon 1976, Laird et al. 1987].

Soar bietet eine einheitliche Repräsentation, die über eine Kombination von domänenunabhängigem und spezifischem Wissen eine Integration von schwachen (Suche) und starken (direkte wissensbasierte Lösung) Problemlösungsmethoden zuläßt. Abbildung 2.6 zeigt den inneren Aufbau eines Soar-Systems.

Abbildung 2.6 Aufbau und Vorgehensweise von *Soar* nach [Laird et al. 1987]



Alles langfristige Wissen ist in Form von Regeln abgelegt, deren Aktionen Elemente in den Arbeitsspeicher eintragen: Problemräume, Zustände, Operatoren und Präferenzen. Darüber hinaus gibt es einen Kontext-Stack mit den aktuell verfolgten Zielen. Soar geht in zwei Phasen vor: In einer “Elaboration Phase” feuern Regeln aus dem Regel-Speicher, bis keine mehr anwendbar ist. In der zweiten, der “Decision Phase” wird auf der Basis der vorhandenen Präferenzen die nächste Situation oder Operation ausgewählt. Dabei wird automatisch eine Sackgasse[5] erkannt und ein neues Ziel generiert, z.B. um eine Ordnung in den Präferenzen zu finden oder einen Operator zu implementieren. Über den sogenannten “Chunking”–Mechanismus wird der Weg zur Lösung dieser Sackgasse in neuen Regeln zusammengefaßt und kann so in Zukunft abgekürzt werden. Auf diese Weise wird, z.B. wie

[5] d.h wenn dabei keine Auswahl möglich ist

in [Laird et al. 1991] beschrieben, eine Robotersteuerung gelernt. Auch für Simulationsanwendungen ist diese Architektur wichtig. Nicht nur Steve (siehe oben) wurde auf der Basis von Soar implementiert, sondern auch Gegner und Partner in simulierten Flugmanöver werden beeindruckend erfolgreich mit TacAir-Soar–Agenten [Tambe et al. 1995] nachgebildet. Für Untersuchungen zur Organisationstheorie wurden Plural-Soar–Agenten [Prietula & Carley 1994] benutzt, um verschiedene individuelle Kooperationseigenschaften und ihre Auswirkungen auf die Entstehung von Organisationsstrukturen zu testen.

Wenn man davon ausgeht, daß jede Operator-Anwendung als regelbasierte Zustandsänderung, ebenso wie jede Problembereich- und Zielauswahl, in Form von Regeln repräsentiert werden muß, ergibt sich eine riesige Menge von Regeln, die zu verwalten ist. Obwohl durch Problembereiche die Regelmenge strukturiert werden kann, ist sie allein wegen der Menge der Regeln nicht besonders wartungsfreundlich. Auch der Chunking-Mechanismus ist nicht ganz unproblematisch, da er unter anderem die Regelmenge weiter aufbläht [Laird et al. 1991]. Vor wenigen Jahren wurde *Soar* mit Schnittstellen versehen, was eine Interaktion mehrerer *Soar*-Agenten, aber auch mit anderen “Umwelt”-Programmen ermöglicht [Soar 2000].

Kein anderes kognitionswissenschaftliches Modell hat einen mit *Soar* vergleichbaren Verbreitungsgrad erreicht; es gibt aber weitere Modelle, die ebenso auf der Annahme beruhen, daß kognitive Fähigkeiten durch Produktionsregeln realisiert werden können (siehe dazu [Anderson 1993]).

Während diese Theorien davon ausgehen, daß die angemessene Ebene, auf der kognitive Prozesse erklärt werden sollten, die funktionale sei, und deshalb auch auf dieser Ebene Implementierungen vorgenommen werden sollten, wenden sich konnektionistische Modelle dagegen. D. Rumelhart [Rumelhart 1989] argumentiert, folgendermaßen: “it is the architecture of the machine that determines the essential nature of the program itself” (Seite 207). Er schlägt deshalb konnektionistische Ansätze mit abstrakten Neuronen als interagierende Basiseinheiten vor. Diese Agentenarchitekturen nach konnektionistischen Theorien, also Neuronale Netze, schließen auch wieder den Kreis zu den subsymbolischen Agentenarchitekturen. Ähnliches gilt für Modelle zur Verhaltensforschung bei Tieren (Ethologie), z.B. [Blumberg 1994, Heisenberg 1994], da auch sie auf subsymbolischer Signalverarbeitung basieren.

2.2.2 Agentensystemebene

Es gibt sehr unterschiedliche Ausprägungen von Gesellschaften, die als Multiagentensysteme betrachtet werden können. Dabei stellt sich die Frage nach einer Analyse von Multiagentensystemen. Eine einfache, eindimensionale Charakterisierung ist dabei nicht möglich, vielmehr ist eine Untersuchung auf der Basis verschiedener Dimensionen denkbar. Die dafür sinnvollen Kriterien, die Granularität und Heterogenität der Agenten sowie die Komplexität der Wechselwirkungen und das dabei vorgegebene Maß an Kooperation, werden im folgenden nacheinander betrachtet [Sundermeyer 1993, Moulin & Chaib-Draa 1996, Huhns & Singh 1997, Weiss 1999].

Granularität und Anzahl der Agenten. Man findet gerade in dieser Hinsicht ein weites Spektrum von Multiagentensystemen. An einem Ende mit wenigen, komplexen Agenten kann man Systeme wie ARCHON–Anwendungen finden, bei denen beispielsweise sieben Agenten ein gesamtes Elektrizitätsnetz oder zwei kooperierende Expertensysteme einen Teilchen-

beschleuniger steuern [Jennings 1994]. Bei Simulationsanwendungen existieren dagegen Systeme mit mehreren Hunderten oder Tausenden relativ einfacher Agenten. So betrachten [Hogeweg & Hesper 1985] die Darstellung von Interaktionen sehr vieler Agenten mit gleichen Verhaltensprogrammen, aber lokalem Wahrnehmungen als Voraussetzung für eine erfolgreiche Modellierung sozioinformatischer Prozesse.

Heterogenität der Agenten und Verteilung der Aufgaben oder Ziele der Agenten. Wie unterschiedlich sind die Agenten des Systems betreffend ihrer Struktur, aber auch im Hinblick auf ihre Ziele? Das ist eine interessante Frage, denn sie betrifft auch die Frage nach der Offenheit des Systems. Je freier die Entwickler eines einzelnen Agenten in ihrer Wahl der Architektur, Programmiersprache, usw. sind, desto verschiedener werden im Endeffekt die implementierten Agenten. Eine weitere Folge ist, daß es schwer wird, ein kohärentes Verhalten des gesamten Systems zu erreichen, gerade wenn die Agenten nicht nur sehr heterogen sind, sondern auch sehr unterschiedliche, möglicherweise konfliktionäre Ziele verfolgen. Da dies aber die Voraussetzung für eine Verwendbarkeit von Multiagentensystemen zur Steuerung, aber auch Nachbildung verteilter Systeme ist, wird gerade in die Entwicklung von Mechanismen viel Forschungsarbeit gesteckt, die ein sinnvolles Verhalten des Gesamtsystems unterstützen. Ein weiterer, damit zusammenhängender Schwerpunkt liegt in der Formalisierung des Verhaltens und der Ziele der Agenten, um idealerweise ein gewisses Globalverhalten daraus abzuleiten und zu verifizieren (siehe dazu auch Abschnitt 2.3.3).

Koppelung der Agenten und Komplexität ihrer Wechselwirkungen. Auch dieser Punkt ist mit dem Ziel verbunden, ein kohärentes Globalverhalten aus dem Verhalten individueller, egoistischer Agenten zu erzeugen. Gerade die Frage der Koppelung der Agenten hat Einfluß auf die Realisierung eines derartigen Globalverhaltens. Wie stark sind die Wechselwirkungen, d.h. wie hoch ist das Verhältnis Kommunikation – Inferenz der einzelnen Agenten? Wie geschieht die Interaktion der Agenten: durch explizite Kommunikation mittels Nachrichten oder durch Modifikation einer gemeinsamen Umwelt? Wie hoch ist das Abstraktionsniveau der Kommunikationsinhalte? Wie weitreichend und variabel sind die Wechselwirkungen?

Organisationsstruktur und Kooperationsbereitschaft. Diese Untersuchungsdimension ist auch mit der Verteilung der Ziele und den Wechselwirkungen in einer Agentengesellschaft verbunden. Welchen Grad an Autonomie besitzt jeder Agent? M. Wooldridge und N. Jennings [Wooldridge & Jennings 1995] forderten für ihren strengen Agentenbegriff (“strong agency”), daß Agenten “wohlwollend” und somit kooperationsbereit sein sollten. Derartige Überlegungen kommen eher aus dem Bereich des Verteilten Problemlösens.

Diese Untersuchungsdimensionen können nicht unabhängig voneinander betrachtet werden. So ist beispielsweise die Heterogenität der Agenten mit ihrer Zahl verbunden. Wenn ein Agentensystem nur wenige Einheiten umfaßt, sind diese meist sehr spezialisiert. Interessant ist es zudem, einem Agentensystem mit konkreten Charakteristika bestimmte Ausprägungen von Eigenschaften zuzuweisen. Ein Beispiel ist die Robustheit. Je spezialisierter die wenigen beteiligten Einheiten sind, umso kleiner wird die Toleranz des Gesamtsystems gegenüber Ausfällen. Man kann die Simulation von Multiagentensystemen sehr gut dafür benutzen, derartige globale Eigenschaften mit Merkmalen dieser Untersuchungsdimensionen

zu korrelieren, um so Heuristiken für die Konstruktion von Lösungen für konkrete Anwendungsgebiete zu finden. Derartige Szenarien findet man nicht nur bei dem Entwurf verteilter Programmsysteme, sondern auch beim Design von Organisationen (für einen Überblick siehe [Prietula et al. 1998]).

2.3 Agenten-orientiertes Software-Engineering

Gerade im Bereich der Software-Entwicklung hofft man, durch das Konzept eines agentenbasierten Systems eine neue Abstraktionsform gefunden zu haben, die es erleichtert, die Form komplexer Software zu entwickeln, wie sie etwa agentenbasierte Systeme darstellen [Wooldridge et al. 1999, Jennings 2000]. Durch eine passende Methodik mit entsprechenden Werkzeugen kann der Entwicklungsprozeß für agentenbasierte Systeme und möglicherweise analog für Multiagentenmodelle unterstützt werden. Trotz, oder gerade aufgrund verschiedenartiger Ansätze fehlt bisher eine etablierte Methodologie.

In den nächsten Abschnitten sollen diese Ansätze zum agenten-orientierten Software-Engineering präsentiert werden: zum einen sind dies Methoden, die als Erweiterung objekt-orientierter und “knowledge engineering”-basierter Techniken gesehen werden können. Diese gehen von der Agentensystemebene aus. Dagegen setzen formale Spezifikationsansätze an der Agentenebene an. Diese werden im Anschluß daran vorgestellt.

2.3.1 Erweiterungen zu objekt-orientierten Techniken

Die Sicht von Agenten als “aktive” oder abstrakte Objekte mit speziellen mentalen Zuständen, führt direkt zum Versuch, für agentenbasierte Systeme objekt-orientierte Methoden zu verwenden bzw. zu erweitern. Die Weiterentwicklung von objekt-orientierten Methoden zu agenten-spezifischen Techniken ist demnach nicht nur wegen ihrer weiten Verbreitung und der so erleichterten Integration in vorhandene Softwarestrukturen interessant, sondern sie bietet wichtige Sichtweisen auf das zu analysierende und konstruierende System [Iglesias et al. 1999]:

- Statische Modelle beschreiben die strukturellen Relationen zwischen den verschiedenen Agenten (Agententypen).
- In dynamischen Modellen können die Interaktionen zwischen Agenten bzw. deren Kommunikation spezifiziert werden.
- Funktionale Modelle beschreiben das Verhalten bzw. die Aktivitäten der einzelnen Agenten.

Allerdings ist eine Anpassung der Methoden aus vielerlei Gründen notwendig: So werden Agenten im Prinzip auf einem höheren Abstraktionsniveau als “Objekte” behandelt. Das sollte auch die Analyse- und Designmethodik widerspiegeln: Ein Entwickler sollte weniger das Verhalten der Agenten, als ihre Ziele in einem bestimmten Kontext beschreiben müssen. Auch für die Spezifikation der internen Attribute eines Agenten ist ein höheres Abstraktionsniveau auf der Basis von mentalen Zuständen sinnvoll. Analoges gilt für die Interaktionen und Relationen der Agenten untereinander. Die richtige Beschreibungsebene ist nicht die des einfachen Verschickens von inhaltlich nicht weiter spezifizierten Nachrichten, sondern die der Sprechakte in Verhandlungs- und anderen Interaktionsprotokollen.

Agenten werden also durch ihre Einbindung in eine Gesellschaft charakterisiert [Wooldridge 1997, Iglesias et al. 1999, Burmeister 1996]. Grundsätzlich können Werkzeuge und Methoden aus dem Standard-Software-Engineering auch deshalb nur bedingt zum Entwurf eines Agenten herangezogen werden, da ein Agent als "reaktives System" gesehen werden kann. Deswegen passt eine herkömmliche, funktionale Sicht mit vorgegebenen Start- und Zielsituationen nicht so richtig. Ein Agent muß als reaktives System vor allem eine permanente Interaktion mit seiner Umwelt sicherstellen können [Fisher & Wooldridge 1995].

Vor allem wegen der konzeptionellen Nähe zwischen "Agent" und "Objekt" ist – wie oben erwähnt – die Erweiterung einer objekt-orientierten Methodik oft der erste Ansatz zur Entwicklung einer agenten-orientierten Technik. Im folgenden sollen drei ausgewählte Beispiele kurz skizziert werden: agenten-orientierte Analyse und Design von B. Burmeister [Burmeister 1996], die Modellierungstechnik für Systeme von BDI-Agenten von D. Kinny et al. [Kinny & Georgeff 1997, Kinny et al. 1996] und die Analyse- und Designmethode von M. Wooldrigde et al. [Wooldridge et al. 1999]. Dabei fokussiert jede dieser Methoden jeweils einen anderen, relevanten Aspekt: Burmeister's Schwerpunkt liegt auf der Beschreibung eines schrittweisen Vorgehens, Kinny et al. entwerfen vor allem graphische Notationen, während bei Wooldrigde et al. die Sicht als Konstruktion einer Gesellschaft besonders interessant ist.

2.3.1.1 Agenten-orientierte Analyse und Design nach B. Burmeister

Ausgangspunkt der Methode von B. Burmeister [Burmeister 1996] ist die Beschreibung dreier Aspekte eines Agentensystems durch folgende Teilmodelle.

- Das *Basis-Agentenmodell* beinhaltet die Agenten und ihre interne Beschreibung.
- Die statischen Beziehungen zwischen den einzelnen Agentenklassen werden im *Organisationsmodell* dargestellt. Diese Beziehungen können sowohl Vererbungsrelationen, aber auch funktionale Rollen in einer Organisation betreffen.
- Das *Kooperationsmodell* beschreibt die dynamischen Interaktionen und Kooperationsbeziehungen zwischen den Agenten.

Wie oben erwähnt, fokussiert B. Burmeister die Vorgehensweise bei der Analyse und dem Design eines agentenbasierten Systems. Dabei werden für jedes Modell ausführlich die Schritte angegeben, mit denen es erstellt werden kann.

- Man beginnt mit der Identifikation der Agenten als aktive Einheiten im System und ihrer Umweltobjekte als passive Objekte. Allerdings geschieht dies nur für Analysezwecke, in späteren Phasen werden diese passiven Einheiten möglicherweise auch als Agenten realisiert. Für die Dokumentation der Agenten werden CRC-Karten ("Classes - Responsibilities - Collaborations") benutzt. In diese werden bei weiteren Schritten Motivationen, Pläne und das dabei notwendige Wissen und die Informationen der einzelnen Agenten eingetragen.
- Den Ausgangspunkt für das Organisationsmodell bildet eine Identifikation der möglichen Rollen und Verantwortlichkeiten im System. Agenten bzw. Rollen mit den gleichen Informationen, Plänen und Verhalten werden zu Agententypen zusammengefasst, mittels Vererbungshierarchien strukturiert und in Standard-Diagrammen (z.B. in OMT-Objektdiagrammen) dargestellt.

- Für das Kooperationsmodell werden die Ziele der Agenten, die durch Kooperation erreicht werden sollen sowie dabei notwendige Partner identifiziert. Ausgehend hiervon werden relevante Nachrichtentypen und Kooperationsprotokolle definiert.

Insbesondere die Ergebnisse der beiden letzten Punkte können in graphischen Repräsentationen aus objekt-orientierten Standardtechniken, wie z.B. das Kooperationsmodell in Interaktionsgraphen dargestellt werden. Allerdings macht B. Burmeister dabei nur Vorschläge [Burmeister 1996].

2.3.1.2 Modellierungsmethode für BDI-Agenten nach D. Kinny et al.

Die Modellierungstechnik für Systeme von BDI-Agenten von D. Kinny et al. [Kinny & Georgeff 1997, Kinny et al. 1996] basiert auf einer expliziten Trennung zwischen externer und interner Sicht auf die Agenten. Davon ausgehend wird hier eine Vorgehensweise in mehreren Schritten und Modellen mit jeweils konkreten Diagrammvorgaben beschrieben.

Bei der externen Sicht werden zwei Teilmodelle spezifiziert: Agentenmodell und Interaktionsmodell. In ersterem werden die hierarchischen Beziehungen zwischen abstrakten und konkreten Agentenklassen und die Anzahl der notwendigen Agenteninstanzen festgelegt. Die Basis für die Aufstellung dieses Diagramms ist wie bei B. Burmeister eine Analyse der (funktionalen und organisatorischen) Rollen in der Anwendungsdomäne. Danach werden im Interaktionsmodell die diesen Rollen zuweisbaren Verantwortlichkeiten und Funktionen spezifiziert. Der Schwerpunkt liegt dabei auf den notwendigen Interaktionen. Diese werden sehr detailliert spezifiziert — bis zur Ebene der Syntax und Semantik für die Kommunikation zwischen den Agenten — und durch eine Analyse und Verfeinerung der Dienstleistungen, die für die Erfüllung der Verantwortlichkeiten einer Rolle notwendig sind, festgestellt.

Auch die Inhalte der internen Sicht eines Agenten können ausgehend von den Verantwortlichkeiten und Services einer Rolle bzw. Agentenklasse, die den Agenten zugewiesen wurden, abgeleitet werden. Dazu analysiert man die Mittel, d.h. Pläne und infolgedessen Information, die dabei jeweils notwendig sind. Konkret werden entsprechend des BDI-Konzepts (siehe Seite 22) drei Teilmodelle, das “belief”- das “goal”- und ein “plan”-Modell spezifiziert und graphisch dargestellt. Ziel ist dabei zum Beispiel eine ausführbare, graphische Beschreibung, wie sie in Abbildung 2.7 als Rahmen für einen Plan, dargestellt wird.

Interessant ist dabei, daß die über Belief-, Ziel-, und Planmodelle spezifizierten BDI-Agenten direkt mittels der Strukturen der PRS-Architektur (“Procedural Reasoning System” siehe auch Abschnitt 2.2.1.2) oder über ihre kommerzielle Entwicklungsumgebung *dMars* [AAII 1996] realisiert werden können.

2.3.1.3 Analyse und Design nach M. Wooldrigde et al.

In der noch relativ neuen, auf Erfahrungen mit diversen agentenbasierten Projekten beruhenden Methode von M. Wooldridge et al. [Wooldridge et al. 1999] findet man einen stärkeren Fokus auf das höhere Abstraktionsniveau von Agenten im Vergleich zu Objekten. Dabei wird das agentenbasierte System, das analysiert und erstellt werden soll, als Gesellschaft behandelt und konstruiert. Allerdings ist es nicht Ziel dieser Methode, ein System auf der Basis der, in der Analyse-Phase erstellten, abstrakten Modelle soweit zu verfeinern, bis es direkt implementierbar ist. Sie beschränken sich im Grunde auf die externe Sicht des Gesamtsystems. In Abbildung 2.8 werden die an der Analyse- und Design-Phase beteiligten Modelle dargestellt.

Abbildung 2.7 Generisches Plan-Diagramm nach [Kinny & Georgeff 1997] Die Elemente eines derartigen Graphen enthält drei Typen von Knoten: *Startzustände* (ausgefüllte Kreise), *Endzustände* ("pass" als der Kreis mit dem gefüllten Inneren und "fail" dargestellt als Kreis mit Querstrich) und *interne Zustände* ("passiv" ohne feinere Struktur, dargestellt durch einen leeren Kreis und "aktiv", die mit einer "Aktivität" assoziiert werden, und Teilpläne usw. repräsentieren können). Transitionen bilden die einzige Art von Kanten.

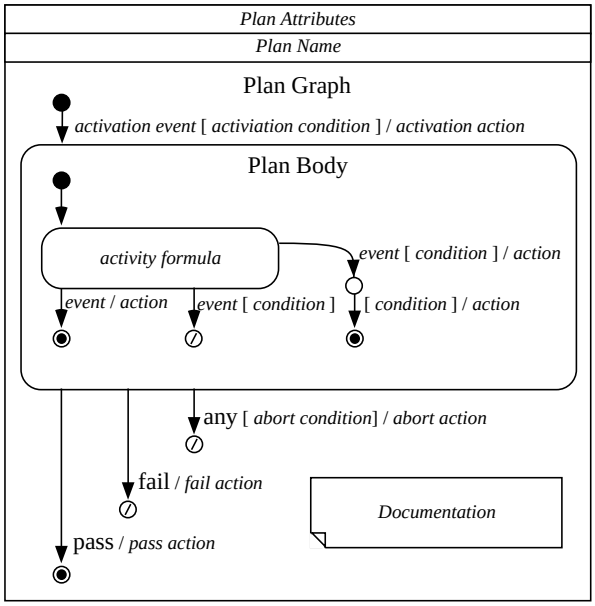
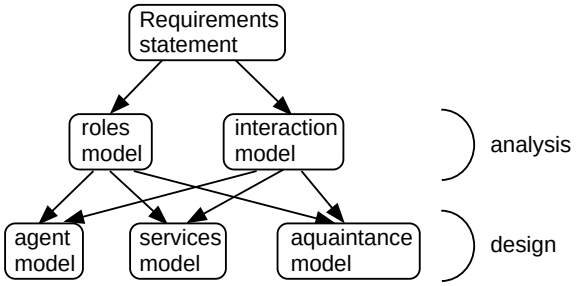


Abbildung 2.8 Modelle der agenten-orientierten Methode nach [Wooldridge et al. 1999]



Das zentrale Konzept ist auch hier das einer Rolle in einer Organisation. Die Analyse des zu erstellenden Systems wird durchgeführt, indem mögliche Rollen mit Verantwortlichkeiten und einer Art von Zielen und deren Interaktionen in Form von Protokollen identifiziert werden. In der zweiten Phase des Designs werden die Rollen zu Agententypen mit qualitativer Beschreibung ihrer Aufgaben und der möglichen Verbindungen zu Interaktionspartnern zusammengefasst. Für die konkrete Umsetzung der Modelle wird auf für die Anwendungsdomäne passende Standard-Software-Engineering-Methoden verwiesen.

2.3.1.4 Anwendungsbereich dieser Methoden

Nur in [Wooldridge et al. 1999] wird thematisiert, für welche Art von Multiagentensystemen die Vorgehensweise wirklich geeignet ist. Der von M. Wooldrigde et al. identifizierte Anwendungsbereich kann aber ohne weiteres auf die anderen beiden Methoden übertragen werden. Hier bestehen die Zielsysteme aus wenigen, komplexen, heterogenen Agenten in Anwendungsszenarien, die a priori kooperativ sind, d.h. unter den Agenten keine echten Konflikte implizieren. Ebenso sollten die Agenten mit institutionalisierten Protokollen auskommen können. Damit charakterisieren [Wooldridge et al. 1999] eine typische Anwendung von agentenbasierten Systemen, wie sie beispielsweise mit ARCHON [Jennings 1994] erstellt werden können. Die Methode könnte sich auch bei der Zusammenstellung eines Multiagentensystems aus vorhandenen intelligenten Systemen mit deren Erweiterung zu kooperierenden Problemlösern als passend erweisen.

Ausgangspunkt der Methode bildet dabei immer die Architekturebene mit Betrachtung von höheren Interaktionsformen und Organisationsstrukturen. Entsprechend der top-down-Vorgehensweise werden dabei identifizierte Aufgaben, usw. auf der Agentenebene verfeinert. Dabei stellt sich die Frage, wie geeignet ein derartiges Schema für die Nachbildung eines Originalsystems in einer Multiagentensimulation ist. Für die Analyse des Systems ist eine Untersuchung, welche Organisationsstrukturen, Rollen usw. vorhanden sind und nachgebildet werden sollen, sicher sinnvoll. Allerdings ist die Erstellung eines Simulationsmodells nicht unbedingt ein mit der Entwicklung eines agentenbasierten Softwaresystems vergleichbarer Konstruktionsprozess.

2.3.2 Ansätze aus dem Knowlegde Engineering

Intelligente Agenten werden meist mit wissensbasierten Systemen assoziiert, die mit anderen interagieren können. Dementsprechend naheliegend ist es, für das Modellieren des Wissens eines Agenten bekannte Methoden und Prozesse aus der Wissensakquisition und -repräsentation anzupassen. Diesen Methoden fehlt aber fast immer die Möglichkeit, verteilte oder soziale Aspekte im Wissens der Agenten zu modellieren. Zudem sind sie meist nicht so einfach erweiterbar wie die objekt-orientierten Methoden [Iglesias et al. 1999], was wohl auch an ihrem a priori höherem Abstraktionsniveau liegt.

Im Gegensatz zu objekt-orientierten Methoden gibt es nur wenige etablierte Methoden und Modelliersprachen speziell für wissensbasierte Systeme, bei denen der Versuch unternommen wurde, sie für die Entwicklung von Multiagentensystemen anzupassen. *MAS-CommonKADS* [Iglesias et al. 1998] ist ein Beispiel für eine Erweiterung der *CommonKADS* Methodologie. Ähnlich wie bei den oben beschriebenen Anpassungen von objekt-orientierten Methoden, liegt das Ziel dabei, die Erstellung kooperierender Problemlöser mit einer Fülle von einzeln zu spezifizierenden “Modellen” zu unterstützen: “agent model”, “task model”, “expertise model”, “organisation model”, usw. Wenn in einem Simulationsmodell das Verhalten komplexer, kooperierender und wissensbasierter Agenten nachgebildet werden soll, ist der Einsatz dieser Methode für die Erstellung eines Simulationsmodells möglicherweise angebracht, ansonsten findet man in den anderen hier vorgestellten Methoden und Sprachen einfachere Möglichkeiten.

2.3.3 Formale Methoden

Eine formale Repräsentation eines zu entwickelnden Multiagentensystems deckt meist nur die Spezifikationsphase ab und nicht die gesamte Entwicklung, wie in den oben besprochenen Methoden. Allerdings findet man in dieser Kategorie auch Formalismen, die direkt ausführbar bzw. automatisch in eine ausführbare Form transformiert werden können.

Ausgangspunkt ist dabei die Darstellung des Agentenverhalten bzw. der Agentenarchitektur, also eine Spezifikation auf der Agentenebene – im Gegensatz zu den obigen Methoden, welche von der Gesamtarchitektur ausgehen. Auf der Basis einer formalen Logik oder anderer Beschreibungssprachen hofft man – wie bei jeder Form der Spezifikation – vor allem eine exakte Darstellung des zu entwickelnden Systems zu gewinnen und darauf aufbauend eine Verifikation und Validation des implementierten Systems durchführen zu können. Durch eine Analyse der Spezifikation kann es möglich sein, bestimmte Eigenschaften des Agenten, aber auch des Gesamtsystems bereits vor dessen Implementierung zu beweisen. Auch für die a posteriori–Generierung einer Erklärung des beobachteten Verhaltens können formale Spezifikationen wichtig sein [Fisher 1995].

Es gibt verschiedene Formen der Repräsentation einer Spezifikation von Multiagentensystemen. Im Folgenden sollen die dabei vorhandenen Kategorien anhand von Beispielen dargestellt werden. Die Einteilung erfolgt nach der Basisrepräsentation: Zunächst sollen logik-basierte Ansätze, danach Spezifikationsformalismen aus dem Knowlegde und Software Engineering vorgestellt werden.

2.3.3.1 Temporale Logik

Temporale Logik wurde entwickelt, um Zeitabhängigkeiten und Dynamik zu beschreiben zu können [Fisher 1997a]. Sie ist eine nicht-klassische Logik, deren Basis ein Zeitmodell bildet und die durch Operatoren, die Zeitabhängigkeiten darstellen, erweitert wird. Ein Beispiel für eine temporale Logik ist METATEM, die im Folgenden näher beschrieben wird. Dabei wird die Zeit durch eine Folge von Situationen repräsentiert [Fisher 1995]. METATEM ist vor allem wegen zwei Erweiterungen für die Spezifikation von Multiagentensystemen interessant: Zum einen gibt es die Möglichkeit, die so formulierten Beschreibungen direkt auszuführen, zum anderen ist diese Ausführung in *Concurrent* METATEM auch parallel möglich. Der Einsatz temporaler Logik hat an dieser Stelle den Vorteil, die stringente, klare Semantik einer formalen Logik mit Möglichkeiten zur Beschreibung von Dynamik und reaktivem Verhalten zu verbinden.

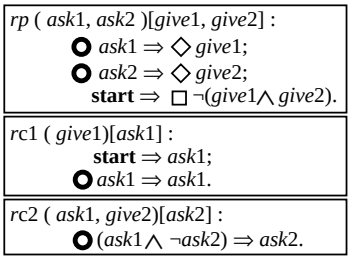
Auf der Basis des $\Diamond$ - Operators[6] läßt sich diese temporale Logik auch zu einer BDI-Logik erweitern, da sich so Ziele elegant als irgendwann zu erreichende Situationen formulieren lassen. Zusätzlich benötigt man Möglichkeiten, um Prioritäten für Ziele und Aktionen formulieren zu können, und so eine Grundlage für die Auswahl des aktuell verfolgen Ziels bzw. der auszuführenden Aktion zu schaffen [Fisher 1997b].

In *Concurrent* METATEM wird ein Agent durch die Festlegung von zwei Aspekten spezifiziert: durch seine Schnittstellen nach außen, und durch einen Satz temporaler Regeln, welche sein Verhalten bestimmen (siehe dazu auch das in Abbildung 2.9 dargestellte Beispiel).

Für die Darstellung der Schnittstelle eines Agenten zu seiner Umwelt, dem “Interface”, existiert folgender Formalismus: Direkt nach dem Agentennamen werden in runden Klam-

[6] $\Diamond$=*sometimes* charakterisiert eine Situation, die irgendwann in der Zukunft eintreten wird bzw. soll.

Abbildung 2.9 Beispiel für eine Verhaltensspezifikation in *Concurrent* METATEM [Fisher & Wooldridge 1997]. Das System besteht aus drei Agenten: *rp*, einem Ressourcen-Produzent und zwei Verbrauchern *rc*1 und *rc*2. Der Agent *rp* gibt jeweils dem der beiden Agenten eine Ressourceneinheit ("*give*"), der danach gefragt hat. Der Agent *rc*1 fragt in jedem Zyklus mit der Nachricht "*ask*1". Die Regel von Agent *rc*2 bedeutet, daß er in jedem Takt, in dessen vorherigem Takt er keine "*ask*2"-Nachricht geschickt, aber eine "*ask*1"-Nachricht empfangen hat, eine "*ask*2"-Nachricht sendet. Die einzelnen Operatoren haben dabei folgende Bedeutung: $\Diamond$=*sometimes*, $\Box$=*always* und $\bigcirc$=*strong last*.



mern die "environment predicates" angegeben. Das sind die Nachrichten, die der Agent aus seiner Umwelt akzeptieren kann. In eckigen Klammern findet man die "component predicates", also die Nachrichten, die der Agent verschicken kann. Unter dieser Spezifikation der Schnittstellen des Agenten stehen die temporalen Regeln, mit denen das Verhalten des Agenten beschrieben werden kann.

Auf der Basis einer derartigen Spezifikation können nicht nur Eigenschaften, wie Fairness oder Sicherheits-Charakteristika des so formalisierten Systems verifiziert werden. Mittels eines automatischen Beweisers konnte realisiert werden, daß die Regeln direkt ausführbar sind. Man kann also man auf dieser Basis das System prototypisch implementieren, oder zumindest sein Verhalten simulieren [Fisher 1995]. Allerdings sollte man beachten, daß derartige automatische Beweiser und die damit verbundene direkte Ausführbarkeit bei einer temporalen Logik, insbesondere wenn sie durch andere Modalitäten erweitert wird, ineffizient sind [Fisher 1997a].

Kommunikation geschieht über einen Broadcast-Mechanismus. Die Folge davon ist, daß relativ einfach neue Agenten hinzugefügt oder entfernt werden und so innerhalb eines gewissen Rahmens variable Strukturen dargestellt werden können. Die Mengen der Nachrichten, die ein so spezifizierter Agent verstehen bzw. versenden kann, sind konstant. Die Beschreibung eines komplexeren Agentenverhaltens könnte allerdings aufwendig werden, da keine weitere Strukturierung in der Menge der temporalen Regeln vorgesehen ist. Auch sollte man nicht unterschätzen, welche Schwierigkeiten es für einen menschlichen Entwickler bedeuten kann, Agentenverhalten *vollständig* und *konsistent* u.a. auf der Basis einer derartigen "low-level" Logik zu beschreiben.

2.3.3.2 Andere modale Logiken

Formalismen, die weniger auf zeitliche Dynamik des Systems ausgerichtet sind, bieten "Belief"-, Aktions- oder andere Logiken. Dabei wird Prädikatenlogik, wie bei der temporalen Logik, um entsprechende modale Operatoren erweitert. Nach [Rao & Georgeff 1991] reichen zur Spezifikation von rationalem Verhalten in echten Anwendungsdomänen die drei

mentalen Kategorien der Beliefs, Desires und Intentions, die jeweils die Informationen, Motivationen und Intentionen eines Agenten repräsentieren. Jeder dieser Aspekte kann auf der Basis entsprechender Logiken formal dargestellt und kombiniert werden.

Der BDI-Formalismus von A. S. Rao und M. P. Georgeff [Rao & Georgeff 1991] gilt wegen seiner Implementierung in PRS (Abschnitt 2.2.1.2) als der bekannteste. Dennoch soll an dieser Stelle als Beispiel die einfachere "Programmiersprache" *Agent0* betrachtet werden. Auf die Darstellung einer multi-modalen BDI-Logik möchte ich verzichten, da diese extrem komplex werden kann und ihr praktischer Nutzen ohne vereinfachende Annahmen umstritten ist [Rao & Georgeff 1995, Dignum & Linder 1997]. Zudem existieren automatische Beweiser für multi-modale BDI-Logiken derzeit nur für vereinfachte Formen [Wooldridge 1997], was eine automatische Ausführung so spezifizierter Programme nicht möglich macht.

Agent0 von Y. Shoham [Shoham 1993] war die erste "höhere" agenten-basierte Sprache. Sie ist keine aximomatisierte Logik mit entsprechenden Beweismethoden, sondern eine interpretierte Programmiersprache mit dem formalen Hintergrund einer (beschränkten) BDI-Logik. Dennoch wird dadurch schon deutlich, wie eine Spezifikation bzw. in diesem Falle auch eine entsprechende Implementierung aussehen könnte. *Agent0* beruht auf einer expliziten Modellierung von mentalen, intentionalen Konzepten, für deren Beschreibung eine formale, zeitpunkt-basierte Sprache mit einem Belief-Operator $B_a^t \phi$ (Agent a glaubt Fakt ϕ zum Zeitpunkt t) zur Verfügung steht. Auch Aktionen werden als Fakten repräsentiert: Ein Fakt wird wahr, wenn die Aktion ausgeführt ist. Weitere Operatoren sind für Verpflichtungen, Entscheidungen und Fähigkeiten zuständig. Es fehlt eine Repräsentation von Motivation.

Das Verhaltensprogramm eines Agenten besteht dabei wie in *Concurrent*METATEM aus einer Schnittstelle zur Umwelt, die mögliche Aktionen des Agenten und für ihn identifizierbare Information enthält. Das Verhalten wird durch "Commitment-Regeln" repräsentiert, mittels derer sich ein Agent selbst und auch anderen Agenten gegenüber zu Aktionen verpflichtet. Aktionen können auch aus dem Versenden sprechakt-basierter Nachrichten bestehen.

Die Bewertung dieser Sprache für den Einsatz bei der Modellierung von Agentensystemen kommt zu einem ähnlichen Ergebnis wie bei *Concurrent*METATEM: Einfaches internationales Verhalten ist relativ schnell in dieser Sprache formuliert. Die Umwelt muß dabei über einen weiteren Agenten dargestellt werden. Allerdings sind komplexere Verhaltensmodelle wegen der Unstrukturiertheit und dem möglicherweise sehr niedrigen Abstraktionsgrad der Verhaltensbeschreibung weniger praktikabel. Dennoch besticht auch hier der Vorteil einer klaren Logik mit einer einfachen Syntax und klaren Semantik.

2.3.3.3 "Compositional Reasoning Architectures"

DESIRE ("framework for DEsign and Specification of Interacting REasoning components") [Gavrila & Treur 1994] bietet einen Rahmen zur Spezifikation wissensbasierter Systeme, die aus interagierenden Komponenten konstruiert sind. Ausgehend davon kann man in diesem Rahmen nicht nur zusammengesetzte Architekturen einzelner Agenten sondern auch aus Agenten zusammengesetzte Systeme darstellen. Auf diese Weise wird ein Formalismus, der sich bei der Spezifikation anderer Systeme als erfolgreich erwiesen hat, für agentenbasierte Systeme wiederverwendet.

Der Rahmen basiert auf verknüpften Komponenten, wobei jeweils eine einem "Task" entspricht. Komponenten können primitiv oder zusammengesetzt sein. Primitive Komponenten besitzen eine Wissensbasis, mit der eine bestimmte primitive Aufgabe gelöst wer-

den kann. Die Wissensbasis einer zusammengesetzten Komponente besteht dagegen aus explizit formuliertem Kontrollwissen zur Aktivierung der Subkomponenten. Der zugrundeliegende Formalismus ist dabei eine dreiwertige Prädikatenlogik ("true", "false" und "unknown"). Jede Komponente verfügt zusätzlich zu ihrer Wissensbasis über ein Eingabe- und ein Ausgabe-Interface. Die Komponenten sind über Informationsverbindungen verknüpft, in denen spezifiziert wird, wie die Wahrheitswerte der Atome der jeweiligen Schnittstellen aufeinander abgebildet werden.

Die Wissensbasen der Komponenten können dabei auf mehreren Ebenen formuliert werden: Objekt-Ebene, Meta-Ebene, Meta-Meta-Ebene . . . . Die Informationsverbindungen zwischen den Schnittstellen der Komponenten können dabei auch unterschiedliche Ebenen verknüpfen. So entspricht z.B. die Verbindung einer zusammengesetzten Komponente mit ihren Basis-Komponenten – zu deren Steuerung – einer Verbindung der Objekt-Ebene der zusammengesetzten Komponente mit der Meta-Ebene dieser untergeordneten Komponenten. Diese Verknüpfungen können dabei auch die aufwendige Übersetzung zwischen den verschiedenen "signatures" leisten, d.h. die Transformation zwischen den Wertemengen oder Sprachen, in denen die Wissensbasen der unterschiedlichen Komponenten definiert werden.

Ausgehend von dieser Spezifikationsumgebung können nun sowohl Agenten als auch Agentensysteme als zusammengesetzte Architekturen repräsentiert werden [Dunin-Keplicz & Treur 1995]. Mulder et al. [Mulder et al. 1998] spezifizieren einen Agenten mit einer PRS-Architektur (siehe Abschnitt 2.2.1.2) in DESIRE. Die fünf Bestandteile der Architektur, Belief-Datenbank, Plan-Bibliothek, Ziel- und Intentions-Strukturen und der zentrale Interpreter können dabei direkt als DESIRE-Komponenten repräsentiert werden. Ein anderes Beispiel für eine konkrete DESIRE-Spezifikation findet man in [Brazier et al. 1995], wo die formale Darstellung einer ARCHON-Anwendung beschrieben wird.

Diese Spezifikationssprache ist auch deshalb sehr attraktiv, weil für sie sowohl eine (einfache) graphische Oberfläche zur Komponentenspezifikation, als auch ein Code-Generator für Prototypen des spezifizierten Systems entwickelt wurden. Problematisch ist dabei, wie bei vielen Spezifikationssprachen, das fehlende "symbol grounding", also der nicht vorhandene Bezug zwischen den in der Spezifikation verwendeten Symbole und der in einer konkreten Programmierumgebung vorhandenen Funktionen, Prädikate, usw. Ebenso eignet sich die Spezifikationssprache wohl weniger für Agentensysteme mit variablen Strukturen, da die definierten Informationsverbindungen einen wichtigen Bestandteil der Spezifikation darstellen. Die Spezifikation der Dynamik des Systems geschieht demnach auf der Basis einer fixen Struktur. Das ist für die interne Zusammensetzung eines Agenten noch unproblematisch. Bei Agentensystemen ist dies wohl auf Anwendungen im Bereich verteilten Problemlösens mit einer vorgegebenen, durchdachten Menge von Agenten angepasst. Bei einem Multiagentenmodell müßte auch die Umwelt als eine Komponente beschrieben werden, zu der alle Agenten Verbindungen unterhalten.

2.3.3.4 Formale Spezifikation mit algebraischen Spezifikationssprachen

Algebraische Spezifikationsrahmen, wie beispielsweise *Z*, werden mit wachsender Beliebtheit beim Design und bei der Verifikation von Softwareentwürfen eingesetzt (siehe dazu [Sommerville 1996]). Dabei werden auf Mengen basierende formale Notationen dazu benutzt, die Vorgänge in einem Programm zu beschreiben, z.B. Wertebereiche und Constraints für Variablen. *Z* ist dabei wegen seiner Strukturierungsmöglichkeiten beliebt, welche es erlauben, auch umfangreichere Softwaresysteme zu beschreiben.

Ein Spezifikationsrahmen, in dem Agenten so spezifiziert werden können, daß sie direkt in Programmcode umsetzbar sind, ist das Ziel von M. Luck et al. [Luck et al. 1997]. In Abbildung 2.10 werden die grundlegenden Schemata dieses Spezifikationsrahmens dargestellt. Sie können ad hoc in Klassendefinitionen einer objekt-orientierten Programmiersprache umgesetzt werden. Ein Agent ist dabei als Objekt mit Zielen definiert, ein autonomer Agent besitzt zusätzlich noch eigene Motivationen.

Abbildung 2.10 *Z*-Rahmenwerk zur Spezifikation eines (autonomen) Agenten (nach [Luck et al. 1997]). Jedes "Kästchen" entspricht einem *Z*-Schema: der obere Teil bestimmt die "Klasse", von der dieses Schema erbt sowie weitere Attribute; der untere Teil enthält die Constraints, die bei diesen Attributen erfüllt sein müssen.

Entity

$attributes : \mathbf{P}\ Attribute$

$attributes \neq \{\}$

Object

$Entity$
$capableof : \mathbf{P}\ Action$

$capableof \neq \{\}$

Agent

$Object$
$goals : \mathbf{P}\ Goal$

$goals \neq \{\}$

Autonomous Agent

$Agent$
$motivations : \mathbf{P}\ Motivation$

$motivations \neq \{\}$

Allerdings bleibt fraglich, was durch eine Spezifikation auf dieser niedrigen Ebene gewonnen wird, besonders da sie vieles nicht explizit macht, z. B. Sensor-Ausstattung, Darstellung der Reasoning-Fähigkeiten, mentale Konzepte, usw. [Fisher 1997a, Wagner 1997]. Die fehlende "high-level" Darstellung von Zielen u.a. führt dazu, daß ein Designer diese ohne weitere Unterstützung spezifizieren muß, die ein derartiger Rahmen eigentlich bieten sollte. Das in [Luck et al. 1997] dargestellte Simulationsbeispiel - ein "Pursuit"-Problem - ist zwar automatisch generiert, aber doch sehr einfach. Hauptkritikpunkt an einem *Z*–basierten Spezifikationsrahmen ist allerdings die fehlende Möglichkeit, (zeitliche) Dynamik zu repräsentieren und auf dieser Basis Agenten als reaktive Systeme mit komplexem rationalem Verhalten zu spezifizieren.

R. Goodwin stellt in [Goodwin 1994] einen weiteren, in *Z* formulierten Modellierrahmen für Agenten vor. Dieser ist deutlich detaillierter und stellt auch abstraktere Konzepte dar. In Abbildung 2.11 ist dieser Beschreibungsrahmen für einen Agenten nach [Goodwin 1994] dargestellt, um zu zeigen, daß auch eine Beschreibung auf *Z*-Basis komplexe Inhalte for-

malisieren kann. An sich wäre zum Verständnis dieser Darstellung eine Kenntnis der *Z*-Notationen notwendig, dennoch ist der abgebildete Teil der Spezifikation doch relativ klar.

Abbildung 2.11 Formale Darstellung eines Agenten im *Z*-Rahmenwerk von R. Goodwin [Goodwin 1994].

$$
\begin{array}{|l}
\textit{Mechanism} \\
\hline
\textit{percieve} : \textit{ExternalState} \pfun \textit{PERCEPT} \\
\textit{effects} : (\textit{COMMAND} \times \textit{CONFIGURATION}) \pfun \textit{INTERACTION} \\
\hline
\end{array}
$$

$$
\textit{Controller}\,[S] == (\textit{PERCEPT} \times S) \pfun (\textit{COMMAND} \times S)
$$

$$
\begin{array}{|l}
\textit{Agent}\,[S] \\
\hline
\textit{controller} : \textit{Controller}\,[S] \\
\textit{mechanism} : \textit{Mechanism} \\
\textit{initialInternalState} : S \\
\hline
\textit{ctrCommands}\,(\textit{controller}) \subseteq \textit{mechCommands}\,(\textit{mechanism}) \\
\textit{mechCommands}\,(\textit{mechanism}) \subseteq \textit{ctrPercepts}\,(\textit{controller}) \\
\textit{initialInternalState} \in \textit{ctrStates}\,(\textit{controller}) \\
\hline
\end{array}
$$

Ein Agent ist demnach vollständig durch “Mechanismus”, “Controller” und Startzustand dargestellt. Der erstere bestimmt, wie der Agent seine Umwelt wahrnehmen und verändern kann. Er besteht aus einem Paar von Funktionen: “percieve” ist eine Abbildung des aktuellen externen Zustand zu den “Wahrnehmungen” des Agenten; “effects” eine Abbildung eines Befehls des Controllers mit der aktuellen Konfiguration des Agenten, auf eine “interaction”, die die Veränderungen an der Umwelt ausführt. Die Konfiguration stellt dabei dar, aus welchen Bestandteilen der Agent besteht. Der Controller entspricht der Aktionsselektion: er ist also eine Abbildung zwischen den Wahrnehmungen und Kommandos des Mechanismus. Das dabei verwendete Verfahren ist abhängig vom internen Zustand des Agenten, den Goodwin nicht weiter spezifiziert.

Über diese Formalisierung eines Agenten hinaus bietet R. Goodwin auch detaillierte Darstellungen von Agentenaufgaben und Umwelten. Diese Art der Repräsentation soll nach R. Goodwin nur als Analyse- und Kommunikationshilfe dienen. Das Ziel dieser Formalisierung ist dabei, nicht wie bei G. Luck automatisch ausführbare Agentenprogramme zu erzeugen, sondern bestimmte Eigenschaften von Agenten, wie “erfolgreich” oder “fähig (capable)”, möglichst exakt darzustellen. Allerdings bleibt er auf der Ebene eines Ein-Agentensystems und untersucht die Koppelung zwischen einem Agenten, der Aufgabe des Agenten sowie der Umwelt, in der diese Aufgabe erfüllt werden soll.

An dieser Stelle sollen die generellen Betrachtungen zu Multiagentensystemen beendet werden. Bevor diese Konzepte im Kapitel 4 wieder aufgegriffen und auf die Nutzung agentenbasierter Konzepte bei der Modellbildung und Simulation angewendet werden, sollen im nächsten Kapitel Basisannahmen und Grundlagen aus der Simulationsmethodik vorgestellt werden.

3

Verhaltensmodellierung und Simulation

3.1 Modellbildung

Die Konstruktion von Modellen und das Experimentieren mit diesen ist eine schon lange bekannte und oft benutzte Vorgehensweise, um die Wirklichkeit zu verstehen, und somit besser steuern zu können. Durch ein Modell, d.h. eine abstrakte Beschreibung eines Systems bzw. eines Teils eines Systems werden Hypothesen über das reale System formalisiert und idealerweise die essentiellen Merkmale erfaßt [Spriet & Vansteenkiste 1982].

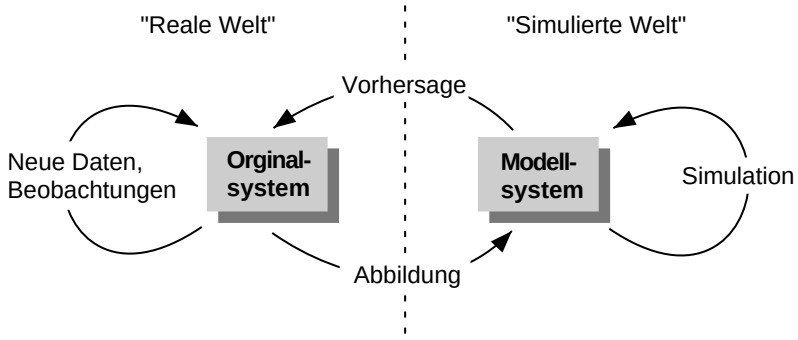
3.1.1 Der System- und Modellbegriff

Zunächst sollte der Begriff "System", wie er hier verwendet wird, geklärt werden. Darunter versteht man "eine Menge von Objekten mit einer Struktur" [Monsef 1997]. Diese Struktur besteht z.B. aus regelhaften Interaktionen oder Abhängigkeiten zwischen den Teilen im System. Entscheidend ist, daß das System von seiner Umwelt trennbar gleichzeitig jedoch durch Ein- und Ausgaben mit ihr verknüpft ist. Die Definition eines System macht demnach nur zusammen mit seiner Umwelt – über das Modell der Eingabedaten – Sinn. Die Grenzen sind vor allem dadurch bestimmt, zu welchem Zweck das System identifiziert wurde; so kann man einen Realitätsausschnitt mittels unterschiedlicher "Systeme" beschreiben. Dieser Begriff eines Systems ist also ein Hilfsmittel zur Beschreibung der Realität. Er ist nicht mit dem Konzept eines formalen Rahmens gleichzusetzen, welcher auch als "formales System" bezeichnet wird.

Formal gesehen kann man ein (deterministisches) "System" S der Systemtheorie durch $S = \langle T, U, Y, Q, \Omega, \sigma, \lambda \rangle$ definieren [Spriet & Vansteenkiste 1982, Fishwick 1995], mit
T : Zeit-Basis - Repräsentation der Uhr; Grundlage für Sortierung der Ereignisse, z.B. $T = R$ (kontinuierliches Zeitsystem) oder $T = I$ (diskretes Zeitsystem).
U : Input-Menge, enthält alle möglichen Eingabewerte, Schnittstelle der Umgebung zum System; Menge der nicht von S kontrollierbaren Eingaben, denen S zu jedem Zeitpunkt ausgesetzt ist.
Y : Output-Menge, Schnittstelle, über die S seine Umwelt modifiziert; Menge aller möglichen Ausgabewerte.
Q : Zustandsmenge - Speicher des Systems; Repräsentation der Vergangenheit, die aktuelle und zukünftige Ausgaben beeinflußt.
Ω : Menge aller zulässigen Input-Funktionen ("Input-Segment"); Menge aller möglichen

Eingabe-Muster.
σ : Transitionsfunktion, $\sigma : Q \times T \times T \times \Omega \rightarrow Q$, bei Zeit-Invarianz vereinfacht: $\sigma : Q \times \Omega \rightarrow Q$.
λ : Output-Funktion, $\lambda : Q \rightarrow Y$, nicht unbedingt zeit-invariant.
Ein Modell kann darauf aufbauend folgendermaßen charakterisiert: "Any image which can be considered as a *System* and is used by a *subject* to obtain information about another system can be thought of as a *model*." M. Minsky (nach [Monsef 1997], Seite 1) Abbildung 3.1 verdeutlicht den Zusammenhang zwischen Modell und System.

Abbildung 3.1 Modellierung als Abbildung zwischen Originalsystem und Modell – nach [Casti 1997] übersetzt und leicht modifiziert



3.1.2 Phasen der Modellbildung

Der Prozess der Konstruktion und Simulation eines Modells kann in mehrere Phasen unterteilt werden. Dabei scheint es so, daß jeder Autor seine eigene grundlegende Methodologie und Terminologie entwickelt hat. Man kann allerdings folgende gemeinsame Schritte identifizieren:

1. Am Anfang steht grundsätzlich die Entwicklung eines Modellkonzepts [Bossel 1994]. Es wird dabei festgelegt, auf welcher Detaillierungsebene und zu welchem Zweck ein Modell entwickelt werden soll. Das wichtigste Ergebnis dieser Phase sollte eine qualitative Beschreibung der Wirkungszusammenhänge im Modell sein. In einem Multiagentenmodell wäre dies eine noch relative abstrakte Darstellung der Verhaltensweisen der einzelnen Agenten(typen), der Umweltbestandteile und -dynamik. Ein derartig abstraktes Modell wird von P. Fishwick [Fishwick 1995] als "Konzeptmodell" bezeichnet.

   Dabei ist wichtig, daß ein Modell nur in Zusammenhang mit einem vorgegebenen Experiment-Rahmen sinnvoll ist. Mit dem Modell sollten also die zulässigen Experimente spezifiziert werden, denn nur in Relation zu diesen Experimenten kann es auf Validität untersucht werden [Monsef 1997]. Das (hypothetische) Ideal für Simulationsexperimente ist das sogenannte "Basismodell"; ein Modell, das für alle möglichen Experimente gültig ist. Die Konstruktion eines derartigen Modells ist in den seltensten Fällen praktikabel. Spezifiziert man aber den Experimentrahmen, kann man ein einfacheres ("lumped") Modell erstellen, das für diese Experimente gültig ist [Zeigler 1976].

2. Die darauf folgende größere Phase ist die Implementierung des Simulationsmodells. Dafür ist die Festlegung auf ein bestimmtes Modellierungsparadigma bzw. auf einen konkreten Formalismus notwendig. Eine dabei bevorzugte Vorgehensweise ist es, zuerst die Modellstruktur ausreichend genau darzulegen und anschließend, in Verbindung mit Simulationsläufen, das Modell so einzustellen (zu "kalibrieren"), bis das Zielverhalten ausreichend genau produziert wird. Bei Multiagentenmodellen kann sich diese Kalibrierung auf mehrere Betrachtungsebenen beziehen. Das Verhalten eines simulierten Individuums, aber auch das Gesamtverhalten, kann vorgegebenen Richtwerten entsprechen.

3. An die Phase der Modellkonstruktion schließt sich üblicherweise eine Phase mit Simulationsexperimenten an. Diese können zweierlei Bedeutung besitzen. Zum einen sind Simulationsläufe notwendig, um Daten zum Einstellen des Modells bzw. als Grundlage für dessen Validation zu generieren. Andererseits bilden Daten aus der Simulation die Basis für jede Untersuchung des Modells. Eine Kombination aus Einstellen des Modells und Bewertung durch Simulationsläufe findet man vor allem in Optimierungsaufgaben, bei denen die optimale Steuerung oder Konfiguration von Prozessen, u.ä. gesucht wird.

4. In der letzten Phase werden die Daten aus den Simulationsexperimenten ausgewertet. Das kann dazu führen, daß für das reale System eine neue Steuerung entwickelt wird und demzufolge das Modell geändert werden muß.

Es wurde bei dieser Aufstellung schon deutlich, daß die Phasen der Modellbildung und Simulation nicht klar voneinander trennbar sind. So ist die Kalibrierung des Modells meist mit etlichen Simulationsexperimenten verbunden. Gerade, wenn Modellbildung und Simulation mit dem Ziel der Optimierung von realen oder zu planenden Systemen oder Prozessen verbunden ist, können die variablen Aspekte alle Bestandteile des Modells betreffen.

Das Modell muß ausreichend genau dem Ursprungssystem entsprechen. Deshalb muß in jeder der oben skizzierten Phasen sichergestellt werden, daß diese Relation noch gut genug gegeben ist. Dazu sind zwei Teilaspekte wichtig: Validation und Verifikation. Validation bezieht sich darauf, sicher zu stellen, daß das richtige Modell konstruiert wird. Mit Methoden der Verifikation kontrolliert man, ob das Modell richtig konstruiert wird, also bei der Implementierung keine Fehler gemacht wurden [Balci 1997].

Wie oben erwähnt, ist die Validität eines Modells immer relativ zu den beabsichtigten Experimenten. Abhängig vom Modellzweck kann man dabei unterschiedliche Ebenen der Validität fordern. Die grundlegendste Form ist die Ebene der Reproduktion des Eingabe-Ausgabe-Verhaltens als Blackbox ("reproductive"). Verlangt man zusätzlich, daß Modell und System so synchronisiert werden können, daß das Modell Eingaben so speichert, daß auch zukünftige Ausgaben äquivalent sind, erreicht man "predictive" Validität. Die andere Form der Validität ist die strukturelle Validität: Das Modell bildet die interne Struktur des Ausgangssystems ausreichend genau nach [Spriet & Vansteenkiste 1982]. Es gibt diverse Techniken zur Validierung und Verifikation von Simulationsmodellen, die abhängig vom Stadium der Modellkonstruktion, vom Formalismus, vom Grad der gewünschten Validität sind. Diese reichen von informellen Techniken, wie Anhörungen der Systemexperten, über dynamische Tests, wie zum Beispiel dem Vergleich über Animationen bis hin zu aus der Software-Verifikation bekannten formalen Techniken [Balci 1994]. Grundsätzlich kann dabei

Ansätze unterscheiden, die vom nicht ausgeführten Modell – also der Verhaltensbeschreibung – ausgehen und andererseits Methoden, die über die Beobachtung und Auswertung eines Simulationsexperiments benutzen.

3.1.3 Modellformalismen und Simulationsmethodik

Abhängig von Eigenschaften des zu modellierenden Systems und der Zielsetzung der Simulationsstudie stehen dem Modellbauer verschiedene Methoden und damit verbundene Modellformalismen zur Verfügung[1]. Häufig wird bezüglich der Art des Zeitfortschritts im Modell unterschieden, mit der jeweils unterschiedliche Modellparadigmen verbunden sind [Monsef 1997], die kontinuierliche und die ereignisbasierte Simulation: Kontinuierliche oder "time-stepped" Modelle werden in jedem Takt komplett durchgerechnet. Dies ist etwa der Fall bei Makromodellen, die in Form von Diffentialgleichungen repräsentiert sind. Der Begriff "kontinuierlich" ist hierbei etwas irreführend, da für die Simulation eine iterative Berechnung der Differentialgleichungen durchgeführt wird. Dazu muß die Zeitmenge diskretisiert werden, allerdings kann das Zeitintervall – abhängig vom verwendeten Integrationsschema – beliebig klein gewählt werden.

Bei ereignisbasierter Simulation werden die Änderungen, die ein Ereignis verursacht, durch das Modell propagiert. Dabei neu auftretende Ereignisse werden in die Ereigniskette einsortiert, in der jedem Ereignis seine Auftrittszeit zugeordnet wird. Danach wird die Simulationsuhr bis zu dem Zeitpunkt weiter geschaltet, an dem das nächste Ereignis auftreten soll [Fishwick 1995]. Derartige Modelle können dabei z.B. als Petri-Netze repräsentiert werden.

Über eine andere Dimension der Klassifikation von Simulationsmodellen, nämlich die Art der abgebildeten Objekte, kann man die Modelle, die in der vorliegenden Arbeit näher betrachtet werden sollen, charakterisieren. Betrachtet man das Gesamtsystem als ein einziges Objekt, beschreibt dessen Zustand durch Variablen und bringt diese unter Berücksichtigung bestimmter Parameter miteinander in Relation, so spricht man von einem Makromodell. Identifiziert man dagegen mehrere interagierende Teilsysteme, spricht man von einem Mikromodell [Troitzsch 1990]. Diese unterschiedlichen Formen werden im Folgenden näher betrachtet.

3.1.3.1 Makromodelle

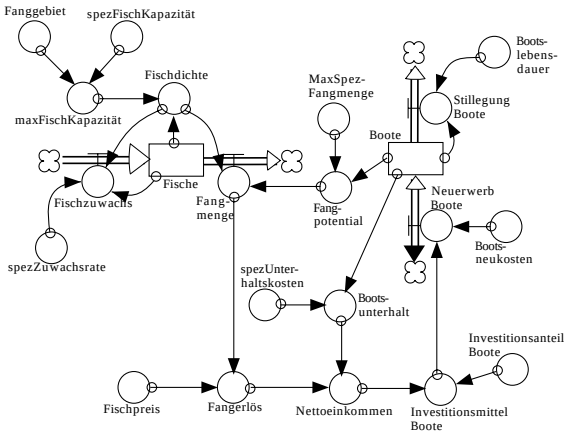
Bei einem Makromodell charakterisiert man das System, z.B. ein Ökosystem, über einige Zustandsvariablen, wie etwa Populationsgrößen. Deren voneinander abhängige zeitliche Dynamik wird in einem Gleichungssystem, üblicherweise mit Differentialgleichungen, abgebildet. Ein prominentes Beispiel ist die Beschreibung von Räuber-Beute-Systemen durch die Lotka-Volterra-Gleichungen [Bossel 1994]. Generell ist bei derartigen Makromodellen die Zeit die einzige unabhängige Variable, der Raum wird ebenso wie die Population als homogen vorausgesetzt. Bei der Abbildung der Interaktionen zwischen den verschiedenen Gruppen geht man von einer Gleichverteilung der Einheiten aus.

Für derartige Modelle gibt es seit den 60iger Jahren etablierte Methoden, wie zum Beispiel *system dynamics* [Richardson 1991]. Auf der Basis eines kausalen Graphen wird über wenige Analyseschritte ein "Level-Raten"-Modell entwickelt, das quasi einer graphischen

[1] Eine schön strukturierte, knappe Auflistung findet man in [Troitzsch 1990].

Notation einer Differentialgleichung entspricht. Die Werte der Zustandsvariablen (“Level”) werden über Zu- und Abflüsse geregelt. Aus der dabei gängigen, graphischen Darstellung der einzelnen Elemente kann man direkt das zugehörige Gleichungssystem[2] ablesen. DYNAMO [Richardson & Pugh 1981] oder STELLA [Richmond et al. 1987] sind die bekanntesten Modellierumgebungen, mit denen die Entwicklung und Implementierung eines *system dynamics*-Modells unterstützt wird.

Abbildung 3.2 Beispiel für ein *system dynamics* -Modell in STELLA-Notation: das “Fischfangsystem” (nach [Bossel 1994]). Dabei deuten die viereckigen “Level” die Zustandsvariablen für die Fischpopulation und die Zahl der Boote an. Die Flußrichtung von Quelle zur Senke (jeweils als “Wolke” dargestellt) deutet an, welche Raten hinzugefügt bzw. von der Zustandsvariablen abgezogen werden. Kreise stehen für Hilfsparameter oder Konstanten, falls nur Pfeile ausgehen (in *DYNAMO* gibt es dafür unterschiedliche graphische Notationen, ebenso wie für verschiedene Qualitäten von Verbindungen)



Boote(t) = Boote(t-dt) + (Neuerwerb_Boote - Stillegung_Boote)*dt
INIT Boote = 100
INFLOWS:
Neuerwerb_Boote = Investitionsmittel_Boote / Bootsneukosten
OUTFLOWS:
Stillegung_Boote = Boote / Bootslebensdauer

Fische(t) = Fische(t-dt) + (Fischzuwachs - Fangmenge)*dt
INIT Fische = 5000
INFLOWS:
Fischzuwachs = spezZuwachsrate*Fische*(1-Fischdichte)
OUTFLOWS:
Fangmenge = Fangpotential*Fischdichte

Bootslebendauer = 15
Bootsneukosten = 100000
Bootsunterhalt = spezUnterhaltskosten*Boote
Fangerlös = Fischpreis*Fangmenge
Fanggebiet = 100
Fangpotential = MaxSpezFangmenge*Boote
Fischdichte = Fische / maxFischKapazität
Fischpreis = 1000
Investitionsanteil_Boote = 0,5
Investitionsmittel_Boote = Nettoeinkommen*Investitionsanteil_Boote
maxFischKapazität = Fanggebiet*spezFischkapzität
MaxSpezFangmenge = 100
Nettoeinkommen = Fangerlös-Bootsunterhalt
spezFischkapazität = 100
spezUnterhaltskosten = 50000
spezZuwachsrate = 1

In Abbildung 3.2 wird ein komplettes Modell mit den in STELLA üblichen graphischen

[2]Interessanterweise beschreibt [Fishwick 1990] *system dynamics* -Modelle als qualitativ, weil sie auf graphischen Repräsentationen beruhen und nicht wie andere Autoren wegen des üblicherweise verwendeten, im Vergleich zu anderen Algorithmen für Diffentialgleichungen groben Integrationsschemas.

Mitteln und den zugehörigen Gleichungen dargestellt. Die Unterstützung durch derartige Werkzeuge führte dazu, daß man diese Methodik heute als etabliert bezeichnen kann.

3.1.3.2 Mikromodelle

Das andere Extrem bei einer Einteilung von Modellformen anhand der Form der betrachteten Objekte sind Mikromodelle. Dabei werden im Modell explizit mehrere Einheiten mit ihren Eigenschaften und Relationen voneinander getrennt betrachtet und simuliert. Das Verhalten des Gesamtsystems ergibt sich aus der Interaktion der zugrundeliegenden Objekte. In diesen Bereich gehört auch der Begriff "objekt-orientierte Simulation", womit eine objektorientierte Sichtweise auf das Mikromodell verbunden ist. Eine besondere Form von Mikromodellen sind individuenbasierte Modelle. Dabei entsprechen die interagierenden Einheiten "Individuen". Diese Sichtweise hat sich vor allem für die Anwendung derartiger Modelle bei der Simulation von Gesellschaftssystemen als sinnvoll erwiesen [Judson 1994, Prietula et al. 1998].

In Mehrebenenmodellen[3] wird das Verhalten desselben Systems auf unterschiedlichen Ebenen dargestellt. Dabei werden verschiedene Mikromodelle auf unterschiedlichen Aggregationsebenen untersucht. Die Eigenschaften von Aggregaten ergeben sich idealerweise aus der Kombination der Eigenschaften der zugrundeliegenden Einheiten. Derartige Mehrebenenmodelle findet man vor allem in sozialwissenschaftlichen Simulationen [Troitzsch 1996, Möhring 1996]. Auch DEVS[4] [Zeigler 1984] – als ein allgemeines Spezifikationsschema für ereignisbasierte Modelle – sieht eine zusammenfassende Modellierung von Systemen auf aggregierten ("compositional system") Ebenen vor.

3.2 Individuenbasierte Modelle

Es gibt verschiedene Formen der individuenbasierten Simulation und, daraus abgeleitet, verschiedene Formalismen für die Modellierung. Die dabei verwendeten Bezeichnungen sind nicht immer eindeutig, manchmal sogar umstritten (siehe [DeAngelis & Gross 1992]). Die folgenden Abschnitte sollen einen Überblick über die hier wichtigsten zwei Ansätze liefern, um die im nächsten Kapitel vorzustellende Multiagentenmodelle in den Kontext individuenbasierter Simulation integrieren zu können.

Zu individuenbasierten Modellen zählen dabei alle, die das Konzept eines Individuums, insbesondere bei der Konzipierung des Modells, verwenden. Ziel dieser Modelle ist also, das nachzubildende Verhalten auf der Ebene des Individuums zu erfassen und abzubilden[5]. Man findet derartige Modelle vor allem in Anwendungsgebieten, in denen Gesellschaften simuliert werden sollen – in der Biologie bei Tier- und Planzenpopulationen oder in den Sozialwissenschaften bei der Simulation von Bevölkerungsentwicklungen.

Im folgenden sollen zwei signifikante Beispiele individuenbasierter Simulation herausgegriffen werden:

[3] Nicht zu verwechseln mit "Multimodellen", [Fishwick & Zeigler 1992], bei denen das gleiche System entsprechend seines Zustands durch unterschiedliche Modelle dargestellt wird.

[4] Discrete Event Specification ist eine stark an den Systembegriff angelehnte, allgemeine Spezifikationsform für ereignisbasierte Modelle.

[5] Bei der Umsetzung des Modells muß entsprechend dieser Definition nicht unbedingt die Repräsentation eines Individuums auch explizit sein. Auch Modelle, die individuelles Verhalten auf Raten, o. ä. in einem Gleichungssystem abbilden, können als individuenbasiert betrachtet werden.

- Mikroanalytische Modelle (als prozessorientierte Simulationsmodelle ohne Repräsentation von Raum) und als zweites
- Zelluläre Automaten (wegen ihrer expliziter Raumrepräsentation).

3.2.1 Prozessmodelle und Mikroanalytische Modelle

3.2.1.1 Prozessorientierte Simulation

Die prozessorientierte Sicht der Simulation findet man bei ereignisbasierten Modellen, bei denen ein Prozess als eine zeitlich geordnete Folge von Ereignissen definiert ist. Das Konzept von "Prozessen" wird eingeführt, um die Menge der zu verwaltenden Ereignisse auf "natürliche" Weise zu strukturieren: So wird z.B. ein Ereignis erzeugt, wenn der Prozess entsteht, die nächsten Ereignisse jeweils, wenn der Prozess seinen Zustand wechselt, und ein letztes, wenn der Prozess das System verläßt [Monsef 1997].

Eine individuenbasierte Sicht erhält das Modell dann, wenn mit einem Prozess jeweils ein Individuum assoziiert wird. Dies gilt vor allem für Modelle, bei denen sich die Fragestellung auf Eigenschaften der einzelnen Prozesse bezieht und dementsprechend die Konstruktion des Modells vorantreibt. Im Gegensatz dazu bilden bei anderen prozessorientierten Modellen, wie beispielsweise Warteschlangenmodellen primär Fragestellungen auf der Ebene des Gesamtsystems den Fokus der Simulation, z.B. die maximale Warteschlangenlänge [Vetterle 1986] im Gegensatz zur durchschnittlichen Zeit im Wartezustand bei einer bestimmten Einheit bzw. dem zugeordneten Prozess.

3.2.1.2 Mikroanalytische Modelle

Eine spezielle Form derartiger Prozessorientierung, insbesondere mit stochastischen Prozessen findet man in den etwa um 1960 auftauchenden, sogenannten mikroanalytischen Modellen[6] [Troitzsch 1997]. Diese werden vor allem in den Wirtschafts- und Sozialwissenschaften benutzt, um die Auswirkungen von politischen (Detail-)Entscheidungen zu untersuchen. Die ersten mikroanalytischen Modelle dienten dabei vor allem zur Voraussage der Bevölkerungsentwicklung. Die Vorgehensweise ist dabei wie folgt: eine Menge von "Komponenten", z.B. Einzelpersonen, Haushalte oder Firmen wird durch Merkmale, wie Alter oder Bevölkerungsdichte des Wohnorts beschrieben. Auf der Basis von "Komponentengruppen, d. h. Komponenten mit ähnlichen Merkmalsausprägungen, wird jeweils eine Verhaltenscharakteristik, z.B. für die Einkommensentwicklung einer Person in dieser Kategorie, angegeben. So wird spezifiziert, wie sich die Merkmalsausprägungen abhängig von unveränderter Dynamik oder modifiziert durch politische Vorgaben ändern. Die Darstellung kann auf der Basis von Interaktionsmatrizen abhängig von anderen Komponenten geschehen [Vetterle 1986]. Meistens enthält die Verhaltenscharakteristik Terme mit stochastischem Einfluß, um die Individualität der Komponenten nachzubilden. Als Ausgangsbestand an Komponenten nimmt man üblicherweise einen Ausschnitt aus der aktuellen Bevölkerung, wie man sie direkt aus Volkszählungen oder anderen Datenquellen erhält, die von Statistik-Instituten erhoben werden. Diese Daten werden im sogenannten "Micro data file" [Gilbert & Troitzsch 1999] gespeichert. In einer derartigen Tabelle wird nun jeder Eintrag entsprechend des zu untersuchenden Modells entwickelt.

[6] Manchmal werden derartige Modelle auch als "Mikro-Simulationsmodelle" bezeichnet [Troitzsch 1990], was zu Verwechslung mit den allgemeiner zu verstehenden "Mikromodellen" führen kann.

3.2.1.3 Beispiel aus der Biologie

Individuenbasierte Modelle, die dieser Kategorie der mikroanalytischen Modelle zugeordnet werden können, werden in theoretischen Biologie häufig verwendet. Der besondere Fokus liegt darin, auf eine abstrakte Weise, Interaktionen zwischen den einzelnen Tieren darzustellen [DeAngelis & Gross 1992]. Das folgende Beispiel verdeutlicht diese Möglichkeit.

So untersucht z.B. D. Gordon et al. [Gordon et al. 1992] die Verteilung von Arbeiterinnen von Ernteameisen (*Pogonomyrmex*) auf verschiedene Aufgaben. Dabei wird der Zustand, d. h. die Aktivität und Aufgabe einer Ameise k durch ein Tripel (a_k, b_k, c_k) mit jeweils möglichen Werten 1 oder -1 dargestellt. Dabei bedeutet $a_k = 1$, daß die Ameise aktiv ist, $a_k = -1$ dagegen, daß die Ameise inaktiv ist. b_k und c_k codieren, ob die Arbeiterin eine Fouragier-Ameise, Soldatin, eine Nestbau- oder eine Müllabfuhr-Arbeiterin ist. In jedem Takt entscheidet jede Arbeiterin abhängig von Interaktionen mit anderen Arbeiterinnen, ob sie ihren Zustand wechselt oder beibehält, also welche Werte (a_k, b_k, c_k) im nächsten Takt besitzt. Für jedes der Elemente dieses Tripels wird eine Entscheidungsfunktion definiert, z.B. wenn $\sum_{j \neq k} \alpha_{k,j} a_k - \theta_k > 0$, dann wird oder bleibt die Ameise k aktiv. Der interessante Punkt dabei ist die Spezifikation der Interaktionsmatrix $(\alpha_{k,j})$ und der Schwellwert θ_k (bzw. die entsprechenden Matrizen $(\beta_{k,j})$ oder $(\gamma_{k,j})$). Ein Beispiel ist folgende Matrix $(\alpha_{k,j})$, die darstellt, daß nur Tiere derselben Kategorie (z.B. nur aktive und inaktive Fouragier-Arbeiterinnen) miteinander interagieren:

	P	F	N	M	p	f	n	m
P	-1				-1			
F		-1				-1		
N			-1				-1	
M				-1				-1
p	-1				-1			
f		-1				-1		
n			-1				-1	
m				-1				-1

Dabei steht bei jedem Element dieser Matrix wiederum eine Matrix der Interaktionen innerhalb von Ameisen dieser Kategorien, so sind P bzw. p alle aktiven bzw. inaktiven Soldatinnen (“Patroler”), F bzw. f sind alle aktiven bzw. inaktiven Fouragier-Arbeiterinnen, usw. Wenn also r Ameisen zu einem Zeitpunkt aktive Soldatinnen sind, dann kann man den Eintrag oben links durch eine $r \times r$ Matrix ersetzen. Dabei entspricht jede Spalte bzw. Zeile einem Tier und die Einträge an sich beliebigen, genauer zu spezifizierenden Gewichten für die Interaktion mit den anderen aktiven Soldatinnen.

Die Entscheidungsfunktionen für b_k und c_k werden auf der Basis von ähnlichen Matrizen berechnet. Im Grunde besteht dieses Modell aus drei großen, meist schwach besetzten Matrizen. Während eines Simulationsexperiments wird in jedem Zeittakt für jede Ameise in einer zufällig gewählten Reihenfolge der Zustand neu berechnet und die Matrizen so aktualisiert, bis eine stabile Verteilung der Arbeiterinnen in die Kategorien erreicht war [Gordon et al. 1992].

Bei diesem Modell ließ sich direkt bestimmen, an welcher Stelle der Matrix ein Wert ungleich 0 stehen sollte, also welche Art von Ameise mit welcher anderen interagieren darf. Aber wie hoch genau dieser Zahlenwert, also das Gewicht dieser individuellen Interaktion für die Entscheidung der Ameise bei jeder einzelnen der drei Matrizen sein muß, um nach endlich vielen Takten eine stabile Verteilung zu erhalten, bedurfte – wie [Gordon et al. 1992]

zugeben – einiger Kalibrierung. Bei der Simulation muß in jedem Aktualisierungszyklus für jede Einheit der nächste Zustand auf der Basis von 3 dünn besetzten großen Matrizen berechnet werden. Diese Matrizen jeweils müssen aktualisiert werden, da die Einträge vom Zustand des Individuums abhängen. Derartige, relativ einfache Berechnungen können mit leistungsfähigen Computern sehr schnell durchgeführt werden. Damit sind auch große Individuenzahlen – [Gordon et al. 1992] simulierten 1600 Tiere – behandelbar. Allerdings liegt der Preis dieser umfangreichen Modelle darin, daß die Entscheidungsfunktion der Einheiten keine weitere Zustandsinformation, wie z.B. Energieniveaus der einzelnen Modellameisen, beinhaltet und sind somit eigentlich trotz der globalen Information, die von jeder Ameise bei der Zustandsberechnung zur Verfügung steht, relativ einfach.

Ohne eine explizite Raummodellierung ist eine Definition von Lokalität nur indirekt und sehr beschränkt möglich, da die Position eines Individuums nicht als zweite unabhängige Variable zur Verfügung steht. Man könnte beispielsweise in einem Modell einer menschlichen Gesellschaft, ein Attribut “Wohnort” einführen und so eine Ortsabhängigkeit über zusätzlich repräsentierte Wohnortabstände, oder ähnliches einbringen. Distanz- und raumabhängige Interaktionen und Merkmalsänderungen sind dennoch im Rahmen eines solchen Modells schwer modellierbar. Über die Probleme einer impliziten Raummodellierung hinaus, sind auch Formen komplexer Interaktionen in mikroanalytischen Modellen nur umständlich handhabbar. Ein Beispiel ist das “Heiraten”, bei dem zwei Haushalte zu einem zusammengeführt werden müssen, der dann Charakteristika beider Ausgangsprozesse aufweist. Eine Form individuenbasierter Modelle, die direkt an eine (diskrete) Raumrepräsentation gebunden sind, sind Zelluläre Automaten, die im folgenden erläutert werden.

3.2.2 Zelluläre Automaten

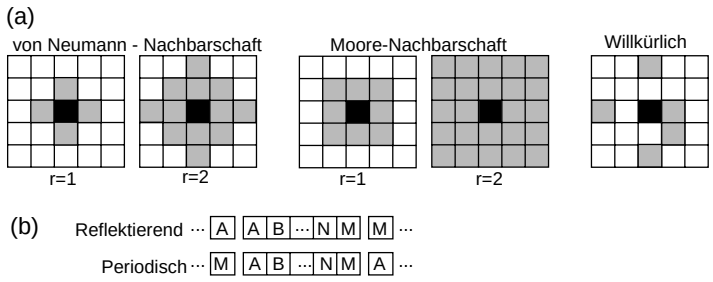
3.2.2.1 Definitionen

Im Gegensatz zu den mikroanalytischen Modellen basieren Zelluläre Automaten auf einer expliziten Darstellung des Raums. Dieser wird dabei als einheitliches d-dimensionales Gitter aus “Zellen” repräsentiert. Jede Zelle besitzt einen diskreten Zustand, der auf der Basis des Zustands seiner Nachbarzellen in jedem Zeitschritt berechnet wird. Diese Regeln verwenden dabei per definitionem nur lokale Informationen und sind für alle Zellen einheitlich. Als Zellulärer Automat wird dabei sowohl dieses uniforme Regelwerk, als auch das gesamte System bezeichnet [Toffoli & Margolos 1987, Bagnoli 1998]. Manchmal wird die Diskretisierung auch als definierendes Merkmal verwendet. So schreiben [Ermentrout & Edelstein-Keshet 1993], daß jedes System, das in Zeit, Raum und Zustand diskret sei, ein Zellulärer Automat wäre. Die Zustandsübergangsregeln müssen nicht deterministisch sein, auch stochastische Übergänge sind möglich. Auf spezieller Hardware, der sogenannte “Cellular Automata Machine” lassen sich Zelluläre Automaten besonders schnell simulieren [Toffoli & Margolos 1987], allerdings sind derartige Maschinen nicht besonders weit verbreitet.

In Abbildung 3.3 sind die häufigsten Nachbarschafts- und Randformen dargestellt. Üblicherweise werden periodische oder reflektierende Randbedingungen verwendet, um die Unendlichkeit des Gitters zu simulieren, wie in Abbildung 3.3 dargestellt. Bei 2-dimensionalen Zellulären Automaten entsprechen periodische Randformen einem Torus.

Einfache Zelluläre Automaten kann man als eine Darstellung der räumlichen Diskretisierung von partiellen Differentialgleichungen interpretieren. Dabei werden zusätzlich zur unabhängigen Variable der Zeit eine oder mehrere unabhängige Dimensionen des Raums

Abbildung 3.3 Verschiedene Formen von Nachbarschaften (a) und Randbedingungen (b).



eingeführt. Auf dieser Basis können Methoden für die Analyse eines möglichen asymptotischen Verhaltens von Automaten entwickelt werden. Gerade in der Physik gibt es dazu diverse Arbeiten [Wolfram 1986, Bagnoli 1998], die unter anderem Verfahren wie "mean field approximation" zur Bestimmung des asymptotischen Verhaltens einfacher Ökosystemmodelle anwenden.

3.2.2.2 Anwendung in der Simulation

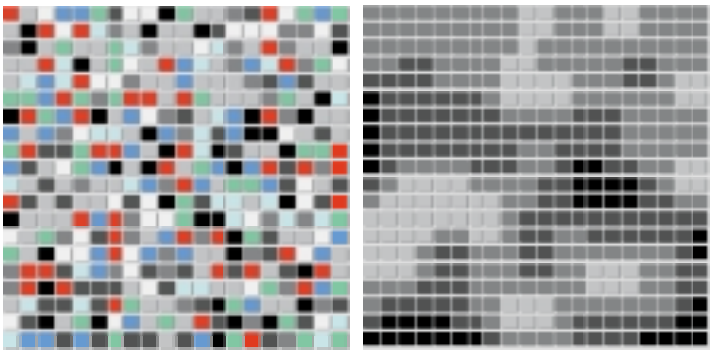
Zelluläre Automaten bilden die Basis für viele Modelle mit unbeweglichen, räumlich verteilten Einheiten mit lokalen Interaktionen. Klassische Beispiele sind Modelle von Kontaktprozessen, wie die Ausbreitung von Epidemien. Auch Waldbrände können auf (probabilistischen) Zellulären Automaten basieren: Der Zustand einer Zelle ändert sich mit einer bestimmten Wahrscheinlichkeit, wenn genügend Nachbarzellen von der Krankheit infiziert sind bzw. brennen [Bagnoli 1998].

Auch in den Sozialwissenschaften tauchten derartige Modelle schon sehr bald auf und wurden vor allem unter dem Namen "checkerboard"[7]-Modelle bekannt. In [Hegselmann & Flache 1998] findet man einen Überblick über derartige sozialwissenschaftliche Anwendungen. Der Startzustand und stationäre Endzustand eines Zellulären Automaten zur Simulation der Verbreitung von Meinungen wird in Abbildung 3.4 gezeigt. Zu Beginn der Simulation erhält jede Zelle einen zufällig gewählten Zustand, also einen diskreten Zahlenwert, der die Meinung der Person auf dieser Zelle widerspiegelt. In jedem Schritt wird diese Meinung an die der benachbarten Zellen angepaßt, in dem als neuer Meinungswert der Durchschnitt der Nachbarmeinungen (inklusive der eigenen) angenommen wird. Nach wenigen Schritten ist ein stabiler Zustand erreicht.

Interessante Phänomene weist auch das "Game of Life" auf. Es handelt sich hier um einen zweidimensionalen Zellulären Automaten mit 2 Zuständen: 0 für "tot" und 1 für "lebendig". Eine Zelle wird oder bleibt lebendig abhängig davon, ob ausreichend Nachbarzellen ebenfalls lebendig sind. Auf der Basis dieser einfachen Regeln lassen sich interessante stabile, periodische und sogar sich bewegende Strukturen erzeugen. Diese Phänomene inspirierten zahlreiche Arbeiten zur Dynamik von Zellulären Automaten (siehe [Wolfram 1986, Langton 1991, Bagnoli 1998])

[7] Schachbrett

Abbildung 3.4 Zustände eines Modells zur Propagierung von Meinungen



zufälliger Startzustand nach 4 Takten

3.2.2.3 Komplexere Formen von Zellulären Automaten

Bei der Modellierung physikalischer Systeme reicht eine derartige Lokalität und Parallelität der Regeln nicht aus, um z.B. die Partikel-Bewegung bei Diffusion darzustellen. Wird ein Partikel als ein bestimmter Zustand der Zelle, auf der es sich befindet, modelliert, müssen bei der Bewegung dieses Partikels in einem Schritt zwei Zellen abhängig voneinander geändert werden: Die Zelle, die das Partikel verläßt und die, auf die es neu kommt. Eine Lösung dafür bietet die Margolos-Umgebung [Toffoli & Margolos 1987]. Dabei werden Regeln nicht mehr für eine einzelne Zelle, sondern für die gleichzeitige Zustandsänderung mehrerer Zellen formuliert. Die Partitionierung des Gesamtautomaten in Teilbereiche, für die jeweils ein Regelsatz gilt, wird im nächsten Schritt geändert, um die Interaktionen zwischen Nachbarzellen zu gewährleisten.

Allerdings verläßt man bei dieser Art der Modellierung immer weiter die Zone, in der die Darstellung eines Systems als Zellulärer Automat “intuitiv” ist und in der Transformation der Regeln vom realen System in das simulierte einer “natürlichen” Konzeptionalisierung folgt. Ein Beispiel in diesem Grenzbereich ist das sogenannte Nagel-Schreckenberg-Modell für Straßenverkehr [Esser et al. 1998, Nagel & Rasmussen 1994]. In einem eindimensionalen Zellulären Automat repräsentiert der Zustand einer Zelle die Geschwindigkeit des Fahrzeuges auf dieser Zelle. Abhängig davon, wieviele Zellen zwischen zwei belegten Zellen liegen, wird diese Geschwindigkeit verändert, einer bestimmten “Trödelwahrscheinlichkeit” unterworfen und danach eine Bewegung simuliert.

Bei diesem Modell wird deutlich, daß der Transfer von der Sicht auf die Fahrzeuge – die Regeln werden für die aktiven Einheiten formuliert – auf die raumbasierte Implementierung, bei der ein Fahrzeug durch einen Zustand der Raumeinheit dargestellt wird, doch einiges an Umdenken erfordert. Ein Fahrzeug nimmt nicht mehr die vor ihm fahrende Einheit wahr, vielmehr überprüft jede Zelle in einer bestimmten Umgebung, ob andere Zellen einen Zustand ungleich 0 besitzen. Auch die Bewegung läßt sich nicht mehr einfach formulieren. In Abschnitt 8.2.3 wird eine Reimplementierung und Erweiterung dieses Modells als Multiagentenmodell in SESAM beschrieben, das bereits dadurch näher an der “natürlichen” Konzeptionalisierung ist, daß die Einheit, deren Verhalten in der Realität beobachtet wird, an sich das einzelne Fahrzeug und nicht das Straßenstück ist, auf dem es sich befindet.

Komplexere Formen von Modellen auf der Basis von Zellulären Automaten werden auch

in den Wirtschaftswissenschaften und verwandten Gebieten immer bedeutender. Es sollen insbesondere solche Systeme untersucht werden, bei denen räumliche Interaktionen nicht vernachläßigbar sind. Ein Beispiel wird in [Balmann 1997] beschrieben. Dieses Modell hat mit den oben beschriebenen Zellulären Automaten kaum mehr als den diskretisierten Raum als Fokus der Modellierung gemein. Die betrachtete Fläche wird in Parzellen unterteilt, die entweder frei sind, von einem bestimmten landwirtschaftlichen Betrieb bewirtschaftet werden, oder selbst die Hofstelle beherbergen. Über letztere wird fast die gesamte Dynamik des Agrarsektors nachgebildet, über Entscheidungen zur Anpachtung neuer, verfügbarer Parzellen, dem Einsatz der Produktionsmittel, bis hin zur Aufnahme von kurz- und langfristigen Krediten, usw. Die Entscheidungen der Betriebe werden streng rational auf der Basis von Kosten-Nutzen-Berechnungen gefällt. Der Raum wird nicht nur über die Behandlung der Parzellen in die Modellierung miteinbezogen, sondern auch über entfernungsabhängige Transportkosten.

Der Übergang zu anderen Formen raumbasierter Modelle ist nicht klar abgrenzbar. Die Aktualisierungsfunktionen für den Zustand einer bestimmten Zelle können beliebig komplex sein. In [Maxwell & Costanza 1995] wird ein Ökosystemmodell für ein konkretes Sumpfgebiet in Louisiana beschrieben. Das Areal wird in ein regelmäßiges Gitter mit etwa ein Quadratkilometer großen Zellen eingeteilt. Jede Zelle wird zu einem bestimmten Habitat-Typ, z.B. Brackwasser oder Salzmarsch, usw. klassifiziert, für den jeweils ein spezielles lokal homogenes Ökosystem-Makromodell entwickelt wurde. Diese Modelle für die Zustandsänderung einer Zelle waren z.T. komplexe *system dynamics*-(STELLA)-Modelle. Die Zellen sind über Zu- und Abflüsse von Wasser, organischem Material, usw. miteinander verbunden. T. Maxwell und R. Costanza bezeichnen ihr Modell zwar nicht als Zellulären Automaten, aber es gehört eindeutig in den Bereich komplexer raumbezogener Modelle. Im Grunde kann man es auch als eine besondere Form eines Kompartment-Modell[8] interpretieren.

3.2.2.4 Zelluläre Automaten und individuenbasierte Simulation

An dem zuletzt beschriebenen Modell eines Sumpfgebietes kann man deutlich den Unterschied zwischen Mikrosimulation und individuenbasierter Simulation erkennen. Das Modell kann eindeutig als Mikromodell identifiziert werden, da das Gesamtsystem auf der Basis vieler kleinerer Einheiten untersucht wird. Diese Einheiten stehen allerdings nicht für interagierende Individuen, sondern für lokale räumliche Prozesse. Somit kann es nicht als ein individuenbasiertes Modell betrachtet werden.

Auf der anderen Seite kann ein Zellulärer Automat durchaus ein individuenbasiertes Modell darstellen. Für jede Zelle kann ein beliebiges Regelwerk definiert sein, das auf die Zustände in einer großen Nachbarschaft reagiert. Im aktuellen Zustand einer Zelle können ebenso alle relevanten Informationen zum Verhalten eines Individuums codiert werden. Dennoch bleibt die Frage, wann ein derartiges Modell als ein individuenbasiertes betrachtet werden kann. Die Antwort darauf ist: wenn der Zustand der Zelle den Zustand eines darauf positionierten Individuums repräsentiert, wie z.B. im Nagel-Schreckenberg-Modell, bei dem der Zustand einer Zelle die Geschwindigkeit des darauf positionierten Fahrzeuges darstellt. Allerdings ist es, wie oben beschrieben, bei dieser Fokussierung auf den zugrundeliegenden Raum schwierig, die Bewegung von Einheiten zu formulieren. Dies gilt ebenso für Systeme mit wirklich individuellem Verhalten der Einheiten, aber bis zu einem bestimmten Punkt

[8] Kompartment-Modelle folgen der Metapher eines System aus Containern und Flüssen, die diese verbinden. Prominente Beispiele findet man in der Medizin bei Modellen des Herz-Lungen-Kreislaufs [Fishwick 1995].

auch für Modelle mit variablen Individuenzahlen – wenn während der Simulation Einheiten erzeugt oder entfernt werden. Nicht-trivial ist auch das Formulieren von Modellen, bei denen mehrere Einheiten, die eigentlich unterschiedliches Verhalten aufweisen sollten, eine Zelle besetzen können. All diese Information muß im Zustand der einzelnen Zellen codiert werden.

Zelluläre Automaten spielen in der individuenbasierten Simulation unabhängig von der gerade beschriebenen Problematik eine wichtige Rolle. Sie stellen eine etablierte Methodik zur Untersuchung raumbezogener Prozesse nicht nur in der Physik dar. Das liegt zum einen an der Entwicklung aus partiellen Differentialgleichungen, aber auch an der Möglichkeit effizienter Simulation, insbesondere auf Parallelrechnern.

Interessante Anwendungen von Zellulären Automaten finden sich auch in Kombination mit Multiagentenmodellen. Dabei wird die dynamische Umgebung für das Agentensystem, z.B. die Waldbrände in *Phoenix* [Cohen et al. 1989] oder die Verteilung von Energieressourcen in *Sugarscape* [Epstein & Axtrell 1996], selbst als Zellulärer Automat repräsentiert.

3.3 Verhaltensmodelle

Nach P. Fishwick [Fishwick 1995] kann Verhaltensmodellierung wie folgt charakterisiert werden: Es wird versucht, eine Abbildung zwischen den Eingaben und Ausgaben des Systems zu finden, ohne daß auch interne Systemzustände explizit nachgebildet werden. Das System an sich wird als “black box” behandelt. Bei Fishwick werden zur Darstellung von Verhaltensmodellen Methoden verwendet, die keinen direkten Zugriff auf den internen Systemzustand zulassen, bzw. bei denen die Abbildung zwischen Eingaben und Ausgaben aus Beispielen gelernt – Neuronale Netze – oder automatisch adaptiert wird – z.B. in Regressionsmodellen. Ich möchte den Begriff etwas weiter fassen und nur den Fokus auf das Systemverhalten legen. Der interne Zustand und seine Repräsentation soll dabei nicht beschränkt sein.

Ein Verhaltensmodell fokussiert demnach das Eingabe-Ausgabe-Verhalten eines Systems. Interne Strukturen werden dabei nicht betrachtet oder nur soweit ein interner Zustand für die Reproduktion der Eingabe-Ausgabe-Abbildung notwendig ist. Dabei stellt sich die Frage, wie sich ein Verhaltensmodell mit einer individuenbasierten Simulation verträgt, bei der die Zusammensetzung einer Gesellschaft – also die interne Struktur des Gesamtsystems – grundlegend ist.

3.3.1 Verhaltensmodelle und individuenbasierte Simulation

Dem Zusammenspiel von Verhaltensmodell und individuenbasierter Simulation kann man sich von zwei Ebenen nähern – ähnlich wie bei der Beschreibung eines Multiagentensystems (Abschnitt 2.2): Für ein Modell des Eingabe-Ausgabe-Verhaltens des Gesamtsystems kann es – abhängig von der für diese Abbildung notwendigen Detaillierung – erforderlich sein, den internen Zustand des Gesamtsystems auf verschiedene Untereinheiten zu verteilen. Dennoch kann man an dieser Stelle nur sehr beschränkt von einen individuenbasierten Modell sprechen, da das Verhalten des Gesamtsystems im Modell codiert wird.

Für diese Arbeit ist die zweite Seite relevant: Anstatt das Verhalten des Gesamtsystems direkt zu betrachten, werden Verhaltensmodelle für die Individuen, also für die Teilsysteme, die die Struktur des Gesamtsystems ausmachen, entwickelt und simuliert. Durch deren

Interaktion wird das Verhalten der gesamten Gesellschaft nachgebildet.

Für ein Verhaltensmodell eines Individuums als Bestandteil eines Gesamtsystems gelten an sich dieselben Prinzipien wie für ein Verhaltensmodell auf höherer Ebene – der Fokus liegt auf dem Eingabe-Ausgabe-Verhalten des Individuums. Allerdings muß man sich einen wichtigen Unterschied bei den Schnittstellen der Teilsysteme klarmachen:

Das Modell der Eingabedaten für die Teilsysteme ist zweigeteilt. Das Individuum erhält Eingaben von anderen Individuen und von außerhalb des Gesamtsystems. Ebenso wird ein Teil der Ausgaben zu Eingaben anderer Teilsysteme. Dabei ist ein wichtiger Punkt die Variabilität dieser Verbindungen. Gerade bei Gesellschaftsmodellen ist ein Erzeugen und Löschen neuer Individuen häufig, die in ein bestehendes Gefüge integriert oder aus demselben entfernt werden müssen.

Derartige Überlegungen sind hier nur auf einer sehr abstrakten Ebene möglich, konkretisiert man individuenbasierte Modelle zu Multiagentenmodellen, wie es im nächsten Kapitel gemacht wird, kann man nicht-triviale Aussagen machen. Wichtig ist hier zunächst, daß das Eingabe-Ausgabe-Verhalten der Individuen fokussiert werden soll. Im Folgenden werden kurz Mittel für die Darstellung von Verhaltensmodellen angerissen und Regeln als die Beschreibungsform der Wahl eingeführt.

3.3.2 Beschreibungsformen für Verhaltensmodelle

Bei einem Verhaltensmodell geht es darum, Eingabe- mit Ausgabedaten zu verknüpfen. Dafür gibt es so viele verschiedene Möglichkeiten, daß sie hier nicht vollständig aufgelistet, sondern nur angedeutet werden können.

Dabei ist die Unterscheidung nach der Art der Eingabedaten wichtig: Bei kontinuierlichen Daten kann man beispielsweise mit Interpolationsverfahren Funktionen zwischen Ein- und Ausgabeverhalten erzeugen oder die Struktur des Zusammenhangs schätzen. Dadurch kann man die Parameter der Gleichungen so anpassen, daß Eingaben auf entsprechende Ausgaben abgebildet werden. Bei diskreten Daten kann man auch versuchen, geschlossene mathematische Gleichungen aufzustellen. Die einfachste Lösung (bei einer endlichen Eingabemenge) ist es jedoch, Assoziationslisten für Eingabe-Ausgabe-Wertpaare zu verwalten. Kann man interne Zustände des Systems formulieren, bietet sich das weite Feld automatenartiger Repräsentationen an: Zustandsautomaten oder Ereignisautomaten, Petri-Netze, usw.

3.3.3 Regeln als deklaratives Beschreibungsmittel

3.3.3.1 Deklarative Modellierung und ihre Vorteile

Grundsätzlich verbindet man mit deklarativer Modellierung und Repräsentation eine explizite Trennung von verwendetem Wissen und einem dieses Wissen verarbeitenden Interpreter. Bei einer deklarativen Programmierung beschreibt man das Problem und nicht den prozeduralen Weg zur Lösung desselben [Schneider 1991].

In einer deklarativer Repräsentation wird also, im Gegensatz zu prozeduralen Formen, Wissen unabhängig von seinem Gebrauch, also ohne Kontrollinformation, z.B. zur Ausführungsreihenfolge usw. spezifiziert [Rich & Knight 1991]. Das bedeutet hier, daß ein Modell aus einer expliziten Beschreibung des Verhaltens in einem abstrakten Rahmen besteht und nicht aus der in einer Programmiersprache implementierten Verknüpfung von Eingaben zu Ausgaben. Ein Simulator interpretiert diese Beschreibung bei der Ausführung des

Modells, das somit ohne Information über Aspekte wie Zeitfortschritt, Ereigniskettenverwaltung, Animationsansteuerung, Datensammelaufrufe usw. auskommen kann. Auf diese Weise besteht bei einem Simulator, der eine deklarative Repräsentation interpretiert, die Möglichkeit, das Modell näher am Konzeptmodell zu implementieren. Das ist eine wichtige Voraussetzung für eine Modell-Entwicklungsumgebung für Domänenexperten. Das Simulationssystem nimmt die wichtigsten technischen Arbeiten ab und der Modellierer kann sich – wenn er mit dem Beschreibungsrahmen ausreichend vertraut ist – auf das Modell an sich konzentrieren. Die Basis für "rapid prototyping" wird damit gelegt. Weitere Darstellungen, welche Vorteile eine deklarative Repräsentation eher genereller Natur für ein wissensbasiertes System besitzt – auch ein Modell kann als Wissensbasiertes System gesehen werden – findet man in [Bamberger et al. 1997, Russell & Norvig 1995, Rich & Knight 1991, Luger & Stubblefield 1993].

3.3.3.2 Grundlagen regelbasierter Verhaltensbeschreibung

Ein regelbasiertes Schema ist *das* deklarative Mittel zur Beschreibung von Verhalten, da auch prozedurale Aspekte auf dieser Basis ausgedrückt werden können. G. L. Luger und W. A. Stubblefield [Luger & Stubblefield 1993] charakterisieren ein Regelsystem in diesem Zusammenhang wie folgt: "The production system imposes a procedural semantics on the declarative language used to form the rules" (Seite 176). Das bedeutet, daß eine regelbasierte Modelliersprache auf einem deklarativen Formalismus beruht, inhaltlich aber dennoch prozedurale Vorgehensweisen beschreibt.

Bereits im Abschnitt 2.3.3 wurden verschiedene regelbasierte Formalismen zur Beschreibung von Agentenverhalten erläutert – von Regeln auf der Basis von Ausdrücken in temporaler Logik (*Concurrent*METATEM) bis hin zu "Programmierung" bzw. Kontrolle von Agenten durch die Formulierung von "Commitment"-Regeln (*Agent0*). Eine regelbasierte Darstellung der Beziehung zwischen Eingabe- und Ausgabedaten eines Agenten kann in Zusammenhang mit der Modellierung von Agentensystemen als eine etablierte Form der Verhaltensdarstellung gesehen werden. Dabei ist zu beachten, daß eine Verwendung des Regelschemas dann besonders geeignet ist, wenn die Ein- und Ausgabedaten auf einem gewissen Abstraktionsniveau vorhanden sind. Bei rein numerischen Ein- und Ausgaben sind regelkreisartige Strukturen oder mathematische Gleichungen besser geeignet.

Bevor im nächsten Kapitel nach der Einführung des Konzepts eines Multiagentenmodells verschiednene Details der regelbasierten Beschreibung von Agentenverhalten erläutert werden, sollen im Folgenden allgemeine Betrachtungen regelbasierter Verhaltensrepräsentation gegeben werden.

Darstellung einzelner Regeln Eine Regel besteht grundsätzlich aus zwei Teilen: Kondition ("linke Seite") und Aktion ("rechte Seite"). Diese kann man in einer einfachen Form wie folgt interpretieren: Wenn *Kondition* erfüllt ist, dann führe / wähle *Aktion* aus. Dabei können sowohl Kondition als auch Aktion unterschiedliche Formen annehmen. Die Folge davon sind unterschiedlich komplexe Mechanismen mit Auswirkungen auf die Mächtigkeit des jeweiligen Regelformalismus. Diese Stufen unterscheiden sich vor allem bei der Feststellung des Wahrheitsgehalts der Kondition bzw. bei der Parametrisierbarkeit der Aktion[9].

[9]Bei der Formulierung von Agentenverhalten auf dieser Ebene kann es zu Verwirrung bezüglich des Begriffs "Aktion" kommen: Mit demselben Begriff wird sowohl die rechte Seite einer Regel, als auch ein Effektorkom-

Bei ersterem besteht die einfachste Form darin, den aktuellen Wahrheitsgehalt einer atomaren Aussage zu berechnen, indem die aktuelle, konkrete Situation des Agenten überprüft wird. Dabei wird jedes Atom für sich betrachtet. Variablen werden nicht verwendet. Bei vorgegebenen Elementen wird deren Existenz geprüft, konkrete absolute oder relative Werte werden verglichen. Der Wahrheitsgehalt derartiger Prädikate kann über die üblichen logischen Konnektoren (und, oder, Negation) zu einer komplexeren Kondition kombiniert werden.

Beinhaltet die Kondition eine Existenzaussage, wie z.B. $\exists x : wahrnehmbar(ich, x) \wedge isa(futter, x)$, dann reicht der Mechanismus des einfachen Ausrechnens nicht mehr aus. Der Wahrheitsgehalt dieser Teilkondition läßt sich mittels Unifikation[10] feststellen. Dieser Mechanismus liefert eine Ersetzung für die Variablen in den Ausdrücken, z.B. x. Diese Zuordnung "macht" die beiden Ausdrücke "gleich", soforn eine derartige Belegung der Variablen existiert, d.h. wenn ein Ausdruck $wahrnehmbar(ich, futter1)$ in der aktuellen Eingabemenge aller wahrnehmbaren Umweltobjekte existiert, liefert $unify(wahrnehmbar(ich, x), wahrnehmbar(ich, futter1))$ die Ersetzung bzw. Variablenbindung $\{x/futter1\}$.

Eine weitere Möglichkeit, die in Verbindung mit graphischen Simulatoren, wie beispielsweise *Kidsim* [Smith et al. 1994] verwendet wird, ist die Bestimmung des "Erfülltseins" einer Kondition mittels "Pattern Matching". Bei den diesen Simulatoren zugrundeliegenden (graphischen) Ersetzungsregeln wird getestet, ob ein bestimmtes Muster – auch mit "don't care"-Zeichen – in der Eingabemenge vorhanden ist, das in der Aktion der Regel durch ein anderes ersetzt werden kann, um so die Ausgabe zu bestimmen.

Die Behandlung der Vorbedingung einer Regel beeinflußt auch die Komplexität der Nachbedindungen, also der Regelaktionen. Während bei Regeln, deren Kondition über Nachschauen getestet wird, die Aktion in einem getrennten zweiten Schritt weiter behandelt werden kann, kann bei einem Verfahren, das eine Variableninstantiierung vornimmt, diese Bindung in der Aktion benutzt werden. Bei einer logischen Inferenz ist es selbstverständlich, daß eine Regel wie $\exists x : wahrnehmbar(ich, x) \wedge isa(futter, x) \rightarrow bewegenZu(ich, x)$ formulierbar ist. Ob eine entsprechende Parametrisierung der Aktion beim "Pattern matching" möglich ist, hängt davon ab, ob "don't care"–Zeichen in der Aktion erlaubt sind – dann können Teile der getesteten Muster in der Regelaktion eingesetzt werden. Wird durch eine Regel allerdings eine nicht-parametrisierbare Folge ausgewählt, beispielsweise ein Standard-Bewegungsmuster, muß keine Variablenbindung an die Aktionsausführung weitergegeben werden, was diese vereinfacht. Bei der Betrachtung von Regelsystemen kommen zusätzliche Aspekte, insbesondere bei der möglichen Interferenz von Aktionen, dazu.

Behandlung einer Regelmenge Der Unterschied zu Verhaltensbeschreibungen auf der Basis von Skriptsprachen, die Bedingungen in Form von If-Then-Konstrukten beinhalten können, liegt hauptsächlich in der Vorgehensweise bei der Interpretation des Programms. Während bei einer Skriptsprache, wie z.B. *MAML* [CEU 2000], eine Anweisung nach der anderen die Aktionenfolge eines Agenten bestimmt, spezifiziert man in einem Regelsystem, welche Situation welche Aktion erfordert. Alle Regeln sind quasi gleichzeitig "aktiv", d.h. wird die Situation, die in der Regelkondition beschrieben wird, als vorhanden bewertet, kann eine Re-

mando oder eine andere Form einer Ausgabe eines Agenten bezeichnet. Allerdings wird bei Regelaktionen wie wir sie betrachten wollen, tatsächlich ein Effektorkommando ausgewählt. Somit ist hier keine weitere Unterscheidung notwendig, wenn durch den Kontext der jeweiligen Betrachtung klar ist, was gemeint ist.

[10] Dieser Algorithmus wird in vielen Lehrbüchern zur Künstlichen Intelligenz, zum Beispiel in [Russell & Norvig 1995], behandelt.

gel an sich unabhängig von anderen Regeln feuern. Diese Spezifikation einer Regel ist schon allein dadurch explizit, daß keine Annahmen über den aktuellen Zustand des Agenten gemacht werden. Die Situation, in der die Aktion ausgeführt werden soll, muß stattdessen vollständig in der Kondition der Regel formuliert werden.

Bei der Modellierung mittels einer Regelmenge kann man zwei Ziele verfolgen. Man hat demzufolge zwei Möglichkeiten, die Regeln anzuordnen: Zur Abbildung verschiedener Eingabemuster auf unterschiedliche Ausgaben, kann man die Regeln "parallel" behandeln. Auf der anderen Seite kann man durch eine kaskadenartige Anordnung der Regeln Zwischenstufen herleiten, um die Regelmenge besser handhaben zu können.

Bei letzterem gibt es grundsätzlich zwei Behandlungsformen: Rückwärts- oder Vorwärtsverkettung. Bei der Rückwärtsverkettung wird von den Ausgaben auf die dazu notwendigen Eingaben geschlossen. Bei der Vorwärtsverkettung werden ausgehend von den Eingaben die sukzessiven Ausgaben betrachtet. Vorwärtsverkettung paßt dabei besser zum Konzept der Verhaltensmodellierung, da dabei die Abbildung der Sensoreingaben auf die Ausgaben des Agenten im Mittelpunkt steht. Bei der Rückwärtsverkettung werden eher Entscheidungsfindungsprozesse – ausgehend von Zielen oder Zielverhalten – nachgebildet.

Interessanter ist hier die Behandlung einer Regelmenge, bei der jede Regel die gleiche Aufgabe – Auswahl einer Aktion – erfüllen kann. Dabei wird bei der Simulation basierend auf einer Regelmenge die Folge der Aktionen durch folgenden "klassischen Zyklus" aus "match", "conflict-resolution" und "act" bestimmt:

- Die "match"-Funktion sucht die Regeln, deren Kondition in der aktuellen Eingaben als wahr festgestellt werden kann.
- Die "conflict-resolution" wählt aus dieser Menge eine (oder mehrere) Regeln aus, die im nächsten Schritt konfliktfrei feuern können. In diesem Schritt wird somit die Reihenfolge der Regelausführung bestimmt. Dabei gibt es diverse Ansätze, die mehr oder weniger Wissen über die möglichen Effekte von Aktionen mit einbeziehen. Die einfachste Form der Konfliktauflösung besteht darin, die erste Regel, deren Kondition als zutreffend berechnet wurde, zu feuern und alle anderen zu ignorieren. Andere Möglichkeiten bestehen darin, eine Regel aus den feuerbaren zufällig oder die spezifischste – d.h. die die detaillierte Situationsbeschreibung in der Kondition enthält – auszuwählen. Andererseits kann man Wissen über Priorität der jeweiligen Regeln benutzten, um die feuerbaren Regeln zu gewichten. Ein weiterer Ansatz besteht darin, die Konfliktauflösung auf die Aktionsdurchführung zu verlagern, indem alle Regeln gleichzeitig ausgewählt werden.
- Die "act"-Funktion führt die Aktion der Regel aus.

Eigenschaften von Regelmengen Auf diesen Zyklus haben verschiedene Merkmale der Regelmenge Einfluß: Kontextfreiheit und Eindeutigkeit der Regel, bzw. Kommutativität und Reversibilität der Aktionsanwendung.

Ist die Regel kontextfrei, genügt das Testen ihrer Vorbedingung, um bestimmen zu können, daß die Regel feuern könnte. Die Regel ist unabhängig von anderweitig repräsentierten Situationsaspekten. Die Konsequenz daraus ist Modularität in der Darstellung: Jede kleinste Einheit, also jede Regel, kann unabhängig von anderen Regeln bzw. von nicht direkt repräsentierten Kontexten dargestellt werden. Abhängigkeiten zwischen den Regeln kommen

dadurch zustande, daß durch die Aktion einer Regel die Situation dahingehend geändert wird, daß die Vorbedingungen anderer Regeln "wahr" werden. Die Kontextunabhängigkeit kann allerdings bei der Unterteilung der Regelmenge und einer diese ausnutzende Indizierung nicht mehr gegeben sein. Das Ziel der Regelindizierung liegt genau darin, Regeln nur in einem sinnvollen Kontext zu testen, d.h. (möglicherweise implizites) Wissen über die Anwendbarkeit einer Teilmenge von Regeln zu nutzen, um die Regelmenge zu strukturieren. Tiefergehende Betrachtungen zur Strukturierung der Regelmenge werden in Abschnitt 5.3.1 behandelt.

Eindeutigkeit kann in einem Regelsystem auf die Konditionen der Regeln bezogen werden[11]. Ist die Situationsbeschreibung in der Vorbedingung so detailliert, daß in einer konkreten Situation nur genau eine Regel feuern kann, ist eine Konfliktauflösung unnötig. Allerdings ist es in den wenigsten Szenarien möglich, Situationen völlig eindeutig zu beschreiben – in Planungsanwendungen wird die Konsequenz dieses Sachverhalts als "Frameproblem" bezeichnet. Können wegen uneindeutiger Konditionen in einer konkreten Situation mehrere Regeln feuern, ist, wie im obigen Zyklus aufgeführt, eine Konfliktauflösung notwendig. In Simulationsszenarien findet man dazu zwei Ansätze. Bei sogenannter qualitativer Simulation [Iwasaki 1989, Kuipers 1986] wird die Konfiguration des aktuellen Situation kopiert und verschiedene "Zweige" der Simulation gleichzeitig oder nacheinander entwickelt. Ohne weiteres Wissen ist dies zum Beispiel auch die Konfliktauflösung bei *Soar* (siehe Abschnitt 2.2.1.2). Kann der Agent mit seinen Regeln keine eindeutige Aktion auswählen, wird versucht, über Prioritäten eine Ordnung zwischen den möglichen Aktionen zu finden. Ist dazu kein Wissen in Form von Regeln vorhanden, werden per default die möglichen Effekte der Aktionen untersucht. Dies kann soweit führen, daß zur Auswahl einer Aktion intern solange "simuliert" wird, bis feststeht, ob die Aktion zur aktuell gewünschten Zielsituation führen kann. Die zufällige Auswahl einer Regel-Aktion aus einer Menge der aktuell möglichen Aktionen, wird zum Beispiel in SESAM realisiert. Allerdings ist die Folge davon, daß bei derartigen möglicherweise nicht-deterministischen Modellen mehrere Simulationsexperimente für ein Szenario notwendig sind, um sichere Aussagen über das Modellverhalten treffen zu können.

Die anderen beiden zu betrachtenden Eigenschaften beziehen sich auf Aktionen. Ist die Ausführung der Aktionen unabhängig von ihrer Reihenfolge, kann man die Regelmenge als kommutativ bezeichnen. Allerdings findet man dieses Merkmal in regelbasierten Agentensteuerungen kaum. Dabei wird meist eine Aktion ausgewählt, mit deren Ausführung die Umwelt verändert wird. In den wenigsten Umwelten – schon in einfachen Szenarien wie der "Blocksworld" ist dies nicht mehr der Fall – die Reihenfolge der Umweltänderungen ist unwichtig.

Analoges kann man für die Reversibilität der Regelanwendung feststellen. Nur in sehr einfachen, simulierten Umwelten, können die Effekte von Aktionsausführungen langfristig mitgeführt werden, so daß diese rückgängig gemacht werden können. Dies wird umso aufwendiger, je mehr Agenten Änderungen in der gleichen Umwelt vornehmen.

Neben der Natürlichkeit der Darstellung und der Möglichkeit, prozedurales Wissen explizit und einheitlich zu repräsentieren, zählt gerade die Modularität der Regeln zu den großen Vorteilen dieser Repräsentationsform. Jede Regel kann als ein Wissensfragment unabhängig von anderen formuliert werden.

[11] Nicht eindeutige Aktionen kann man – abhängig vom Konfliktauflösungsmechanismus – auch als Disjunktion der Vorbedingungen behandeln

Mit dieser Isoliertheit der einzelnen Regeln ist allerdings auch ein wichtiger Nachteil der regelbasierten Darstellung verbunden. Keine Regel ruft eine andere direkt auf, sondern die Aktion einer Regel verändert die aktuelle Situation, wodurch die Konditionen anderer Regeln wahr werden können. Die Formulierung von algorithmischem Wissen ist somit problematisch, da eine Reihenfolge bei der Aktionsausführung nur indirekt über die Wirkung der Aktionen auf Regelvorbedingungen dargestellt werden kann. Die Formulierung von derartigen Aktionssequenzen muß auch Rücksicht auf den Konfliktauflösungsmechanismus nehmen. Auf diese Weise kann die Darstellung von Aktionssequenzen undurchsichtig werden und nur schwer nachvollziehbar sein.

Ein weiterer Nachteil ist üblicherweise die Ineffizienz, die mit der Behandlung großer Regelmengen verbunden ist. In jedem Zyklus müssen alle "relevanten" Regeln ausgetestet werden, um die Regelmenge festzustellen, deren Vorbedingung erfüllt ist. Entscheidend ist dabei, die Zahl der in der aktuellen Situation in Frage kommenden Regeln – die "relevanten" – so gering wie möglich zu halten. Der Königsweg dazu ist, die Strukturierung der gesamten Regelmenge vor der eigentlichen Nutzung, indem man Zusammenhänge zwischen den Regeln, sei es in Vorbedingung oder Aktion, und zusätzliches Wissen dazu nutzt, durch eine geeignete Vorauswahl so wenige Regeln wie möglich genauer testen zu müssen. Ein Verfahren hierfür ist die Indizierung von Regeln bei Objekten der Aktion oder Aspekten der Situation – hier Sensoren usw. Eine ausgefeilte Vorgehensweise zeigt der *rete*-Algorithmus[12], der in einem Kompilierungsschritt gleiche Vorbedingungsteile zusammenfaßt und so die Regelmenge in ein Netz übersetzt. Dieses kann bei Änderungen im aktuellen Kontext schnell durchlaufen werden.

[12] Der *rete*-Algorithmus ist in Lehrbücher zur Künstlichen Intelligenz, wie [Russell & Norvig 1995] beschrieben

4

Multiagentenmodelle und Simulation

4.1 Definition und Bestandteile

4.1.1 Begriffe

Multiagentensimulation oder agentenbasierte Simulation sind Begriffe, die man mittlerweile recht häufig findet. Dabei kann man verschiedene, mehr oder weniger ähnliche Richtungen identifizieren: Zum einen ist dies der Ansatz, das Konzept eines Multiagentensystems bei der Formulierung des Modells zu nutzen. Das Spektrum reicht dabei von Modellen, die nur auf dem abstrakten Konzept eines Agenten beruhen bis zu Modellen, die ausschließlich menschliche Akteure als Agenten nachbilden. In dieser Arbeit sollen letztere unter die ersteren subsumiert werden. Eine grundlegend andere Verwendung dieser Bezeichnungen findet man bei komplexeren Simulationssystemen, deren einzelne Module (wie Simulatoren oder Optimierer) als Agenten behandelt werden. Diese Richtung ist für die vorliegende Arbeit nicht relevant.

Charakteristika, die allgemein mit Agenten assoziiert werden, wurden bereits in Abschnitt 2.1.1.1 eingeführt, diese sollen nun für Agenten, wie sie im Folgenden verwendet sind, konkretisiert werden.

4.1.1.1 Simulierte Agenten

Im Folgenden soll ein simulierter Agent als aktiver Bestandteil eines Modells gesehen werden. Er zeichnet sich durch verschiedene charakteristische Merkmale aus:
Ein Agent

1. verändert nicht nur sich selbst, sondern wirkt auf seine Umwelt und bleibt in dieser persistent. Aktionen geschehen dabei nicht nur als passive Antwort auf Umwelteinflüsse.

2. agiert in Relation zu seiner Umwelt – er verändert sie nicht nur, sondern kann Information über seine Umwelt beziehen.

3. besitzt einen beschränkten Wahrnehmungs- und Aktionsradius (Lokalität des Verhaltens).

4. verfügt über ein nicht-triviales Verhaltensrepertoire. Dieses Verhalten ist auf einer höheren Ebene beschreibbar.

Gerade der letzte Punkt – obwohl nicht sonderlich konkret – ist eigentlich ausschlaggebend. "Agent" ist auch in Simulationsanwendungen ein abstrakteres Konzept als "Objekt". Häufig [Wooldridge 1997, Jennings et al. 1998] wird zwischen "Agent" und "Objekt" dadurch getrennt, daß ein Agent zusätzlich zu seinem internen Zustand auch sein Verhalten kapseln soll, im Gegensatz zu einem Objekt, das nur seinen Zustand kontrolliert. Diese Unterscheidung ist ebenfalls für agentenbasierte Simulation relevant, allerdings handelt es sich in meinem Augen eher um eine "technische" Charakterisierung, die in diesem Zusammenhang weniger relevant ist.

4.1.1.2 Multiagentenmodell

Ein agentenbasiertes System ist nach [Jennings et al. 1998] ein System, dessen Schlüsselabstraktion die eines Agenten ist. [Jennings 2000] macht die Essenz eines agentenbasierten Systems an drei Bestandteilen fest: Agenten, High-Level-Interaktionen und organisatorische Beziehungen.

Analog dazu wird hier ein agentenbasiertes Modell, also ein Modell, dessen Basiskonzept das eines Agenten ist, eingeführt. Ein Multiagentenmodell ist demnach ein agentenbasiertes Modell mit mehr als einem Agenten. Multiagentensimulation ist dementsprechend die Ausführung bzw. Experimentation mit einem Multiagentenmodell. Man kann ein Multiagentenmodell zudem auch als Multiagentensystem, das ein "reales" Multiagentensystem in einer simulierten Umwelt abbildet, betrachten.

Ein Modellbauer identifiziert passende Teile des zu modellierenden Systems als Agenten, die miteinander in Wechselwirkung stehen, um so das globale Verhalten des Gesamtsystems zu erzeugen. Die Umwelt eines einzelnen Agenten besteht dabei möglicherweise nicht nur aus anderen Agenten, sondern kann eine anderweitig generierte Dynamik besitzen. Diese Umwelt stellt einen wichtigen Teil des Modells das, d.h. auch sie muß mit viel Sorgfalt analysiert und nachgebildet werden.

Was macht nun die Essenz eines Multiagentenmodells aus? Für die vorliegende Arbeit kommen drei Faktoren in Frage:

- Agenten,
- ihre Interaktionen und
- eine simulierte Umwelt.

Organisationsstrukturen können dabei für die Agenten explizit vorgegeben und beschrieben werden oder aus den Interaktionen resultieren und während der Simulation bewertet werden. Insgesamt ist die Aufgabe, ein Multiagentenmodell zu entwickeln, gerade bei den Aspekten Interaktionen und Organisation wohl einfacher als die Entwicklung eines Multiagentensystems, da das Originalsystem meist bereits über Interaktionen und Strukturen verfügt, die identifizierbar sind oder über die Hypothesen gebildet werden können.

Dieses Konzept eines Multiagentenmodells soll im Folgenden in den Kontext der bisher beschriebenen Bereiche der individuenbasierten Simulation und der Verteilten Künstlichen Intelligenz gesetzt werden. Zunächst aber sollen einige Beispiele für Multiagentensimulationen dargestellt werden, um das Konzept plausibler zu machen.

4.1.1.3 Beispiele für Multiagentensimulationen

In der Literatur findet man sehr unterschiedliche Ausprägungen von Modellen, die man als Multiagentenmodelle bezeichnen kann. Die Zielsetzung bei den im Folgenden beschriebenen Modellen geht allerdings über die einer interessanten Testumgebung hinaus, wie sie etwa bei Testbeds der Fall ist.

Man kann im Bereich des "Artificial Life" viele Modelle als Multiagentenmodelle identifizieren. Eine konkrete Anwendung aus der Zoologie, die auch in Kreisen der Verteilten Künstlichen Intelligenz Beachtung gefunden hat, ist das MANTA-Projekt ("Modelling Anthill Activity") [Drogoul et al. 1991, Drogoul & Ferber 1994], mit dem die Aktivitäten innerhalb eines Ameisenhügels nachgebildet werden sollten. Die Implementierungsbasis für dieses Agentensystem war die Stimulus–Response–basierte Architektur des "Eco Modelling Frame" (EMF) (siehe auch Seite 20), die auch für verteiltes Problemlösen von klassischen KI-Problemen, wie das N-Puzzle, eingesetzt wurde. Diese Anwendung zeigt, daß auch mit einfachen, subkognitiven Agenten komplexes, emergentes Verhalten nachgebildet werden kann. Konkrete biologische Fragestellungen waren dabei allerdings eher zweitrangig.

Die ersten Modelle mit kognitiven Agenten entstanden im EOS-Projekt, in dem die Entwicklung von sozialen Strukturen, z.B. die Ausbildung von Hierarchien im steinzeitlichen Südfrankreich, untersucht wurde [Doran et al. 1994, Doran & Palmer 1995]. Dies geschah auf der Basis von internen Modellen der einzelnen Agenten. Die "Umweltmodelle" der Agenten beinhalten die Erfahrungen der Agenten mit Gruppenführern, Freunden usw. Dieses Modell gilt als eines der ersten Multiagentenmodelle überhaupt – insbesondere mit kognitiven Agenten – und wurde in den Sozialwissenschaften stark beachtet. Es führte dabei zur Überzeugung, daß Multiagentensimulation besonders gut zur Nachbildung sozialer Prozesse in menschlichen Gesellschaften geeignet ist. Mittlerweile gibt es Zeitschriften, wie JASSS ("Journal of Artificial Societies and Social Simulation"), die sich quasi ausschließlich mit Multiagentensimulationen beschäftigen.

All diesen Modellen ist gemeinsam, daß die wenigsten mit einer speziellen Methodik entworfen wurden. Die EOS-Modelle wurden beispielsweise direkt in Prolog implementiert. Bei organisationstheoretischen Untersuchungen existieren einige, die als Basis Soar (siehe Seite 24) benutzen.

Das Thema der Validierung und Kalibrierung wird kaum angesprochen, meist genügt den Modellierern das Augenmaß. Einzig von R. Axtrell et al. [Axtell et al. 1996] wird dieses Problem thematisiert. Ein Lösungsansatz wird durch das "model alignment" vorgestellt. Man versucht ein Multiagentenmodell so einzustellen, daß es die Ergebnisse eines anderen Modells reproduziert. Das Ziel dabei ist, festzustellen, ob die globalen Phänomene reproduziert werden können. R. Axtrell et al. schafften dies für das oben angesprochene Sugarscape-Modell und für ein Modell von R. Axelrod [Axelrod 1997b] zur Verbreitung von kulturellen Werten.

4.1.2 Kontext

4.1.2.1 Multiagentenmodelle und individuenbasierte Simulation

Der wichtigste Unterschied zu den im Abschnitt 3.2 vorgestellten individuenbasierten Simulationsformen ist dabei, daß das Konzept "Agent" allgemeiner anwendbar ist als "Individuum". Individuen sind atomare Bestandteile von Gesellschaften, Agenten dagegen Aktoren – auch Gruppen von Individuen können als Agenten behandelt werden. Auf der anderen

Seite können Agenten als komplexere Einheiten als die Individuen betrachtet werden, die üblicherweise in der individuenbasierten Simulation verwendet werden. Das Verhaltensrepertoire von Individuen ist meist jedoch deutlich einfacher als das, das man von einem "Agenten" erwarten würde.

Ein eher technischer Unterschied liegt darin, daß es für das Verhalten von Agenten keine aus einer Methodik vorgegebenen Einschränkungen gibt: Man kann beliebige lokale (dynamische) Strukturen in das Verhaltensmodell der Agenten mit einbeziehen und muß nicht wie bei Zellulären Automaten eine feste Nachbarschaftsform verwenden. Der Raum muß nicht als einheitliches Gitter modelliert werden, sondern kann durchaus Inhomogenitäten aufweisen. Ebenso wie das Einbeziehen räumlicher Information, ist auch die Darstellung heterogener Gesellschaften und dabei auftretender co-evolutionärer Prozesse möglich.

Somit kann man Multiagentensimulation als ein abstrakteres und allgemeineres Konzept als individuenbasierte Simulation betrachten. Man kann zudem alle im Abschnitt 3.2 besprochenen Formen individuenbasierter Modellierung in dieses Konzept integrieren. Diese Allgemeinheit bedeutet im Endeffekt aber auch, daß man jede der oben dargestellten Techniken als eine Multiagentensimulation interpretieren bzw. in eine solche übersetzen kann. So lassen sich beispielsweise Zelluläre Automaten dadurch, daß jede Zelle als Agent dargestellt wird, in ein Multiagentensystem übersetzen. In manchen Fällen, wie zum Beispiel beim Nagel-Schreckenberg-Automaten zur Verkehrssimulation (siehe 8.2.3), repräsentiert der Zustand einer Zelle Eigenschaft der Einheit auf dieser Zelle. Das Verhalten des Automaten läßt sich einfach in einem Multiagentenmodell reproduzieren, bei dem die Fahrzeuge als Agenten dargestellt sind [Bazzan et al. 1999].

Die Flexibilität von Multiagentenmodellen ist aber zugleich eines ihrer größten Probleme. Mit den Voraussetzungen und Annahmen, die die Darstellung des Agentenverhaltens erfüllen muß, geht auch die Prägnanz, wie sie z.B. diskretisierte partielle Differentialgleichungen bieten würden, verloren.

4.1.2.2 Einordnung von Multiagentenmodellen in die VKI

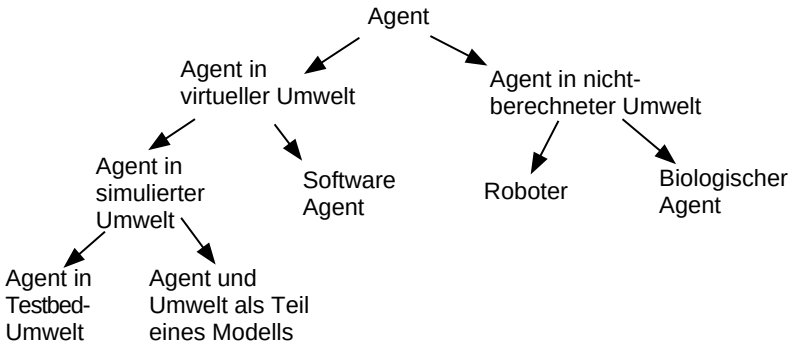
Wie oben erwähnt, kann man ein Multiagentenmodell als ein Multiagentensystem in einer simulierten Umwelt identifizieren, das nur durch Bezug zu einem anderen System interpretierbar ist. Fraglich ist nun, in welchen Kontext der VKI (Verteilten Künstlichen Intelligenz) Multiagentensimulationen eingeordnet werden können: Dazu sollen zunächst Überlegungen zu Agenten und ihren "Lebensräumen" gegeben werden. Wichtig ist dabei insbesondere auch inwieweit die Umwelt des Agenten so kontrollierbar ist, daß sie als Bestandteil in das Modell eingehen kann.

Abbildung 4.1 zeigt eine Taxonomie, in welcher konkreten Umwelt ein Agent existiert.

Grundsätzlich gibt es die Unterscheidung, ob ein Agent in einer berechneten oder realen Umwelt existiert. Trotz immensen Fortschritten in der Robotik stellt eine reale Umgebung, unabhängig davon, wie kontrolliert und beschränkt sie im Endeffekt ist, immer noch sehr hohe Anforderungen an Sensoren und Effektorsteuerung. Beim Design oder der Untersuchung eines Agenten sind deshalb sehr viele technische Probleme zu lösen. Dennoch kann man einige Robotikanwendungen als Multiagentensimulationen betrachten, da die Umwelt künstlich modelliert wird. Beispiele sind die Experimente von K. Dautenhahn mit Roboter-Agenten in Hügellandschaften, mit denen soziale Interaktionen nachgebildet werden [Dautenhahn 1997, Dautenhahn 1995].

Bessere technische Voraussetzungen existieren, wenn der Lebensraum des Agenten ein

Abbildung 4.1 Taxonomie der konkreten Umgebung eines Agenten



Computer ist. Allerdings kann man auch hier weitere Unterscheidungen machen. Software-Agenten sind eigenständige Programme, die im Dateisystem oder Internet ihren Aufgaben nachgehen. Der wichtige Unterschied dabei ist, daß der Konstrukteur des Agenten keinen Einfluß auf die Umwelt hat und nur Vermutungen über die Struktur und weitere "Bevölkerung" dieses offenen Systems anstellen kann. Auch bei simulierten Umwelten, die speziell für den Agenten konstruiert wurden, gibt es Unterschiede in der Kontrollierbarkeit durch den Entwickler des Agenten. In Testbeds, also Experimentierumgebungen mit vorgegebenem Szenario, liegt gerade der Sinn der Untersuchungen darin, den Agenten in standardisierten Umgebungen agieren zu lassen. Nur so bleibt eine Vergleichbarkeit gewahrt. Multiagentenmodelle gehören an sich zu den Systemen, in denen sowohl der Agent, als auch seine Umwelt Teile desselben Modells sind. Der Modellbauer hat volle Kontrolle über die Umwelt des Agenten. Das Verhalten der Agenten ist nur dann sinnvoll, wenn auch die Umwelt ausreichend detailliert und ausreichend komplex repräsentiert wurde.

Bevor in den nächsten Kapiteln eine konkrete Möglichkeit vorgestellt wird, den Entwicklungsprozess eines Multiagentenmodells zu unterstützen, werden dessen Bestandteile ausführlicher betrachtet. Anstelle von taxonomischen Details soll dabei der Fokus eher auf die konkrete Realisierbarkeit gelegt werden: Wie oben angedeutet, besitzt die Formulierung einer Multiagentensimulation drei "Schwerpunkte": die Agenten, ihre Umgebung und eine Beschreibung der Interaktionen zwischen den Agenten und ihrer Umwelt [Epstein & Axtrell 1996]. Diese sollen im folgenden ausführlich dargestellt werden.

4.1.3 Design der Agenten

Aus den obigen Betrachtungen zur Essenz eines Agenten wird deutlich, daß man zumindest drei Aspekte festlegen muß, um einen einzelnen Agenten darzustellen.

- Welche Sensoren besitzt ein Agent?
- Über welche Effektoren verfügt ein Agent?
- Welche Mechanismen benutzt der Agent, um seine nächste Aktion auf der Basis der Wahrnehmungen auszuwählen?

4.1.3.1 Welche Information benutzt ein Agent?

Die Basis für die Auswahl der nächsten Aktion bildet die Information, die ein Agent zur Verfügung hat. In einem Multiagentensystem ist diese grundsätzlich lokal. Kein Agent kann einen globalen Sichtwinkel auf das System, dessen Teil er ist, einnehmen. Selbst wenn ein Agent als zentraler Informationsserver dient, z.B. als "Yellow Pages" für die Adressen der anderen Kommunikationsteilnehmer, kann er nur die Information weitergeben, die er irgendwann von anderen Agenten oder durch eigene Erfahrung gesammelt hat. Welche Art von Information kann nun ein Agent für seine Aktionsselektion benutzen? In Abschnitt 2.2.1.2 werden Strukturen vorgestellt, mit denen Agenten ihr Wissen auf unterschiedliche Weise verwalten können bzw. Architekturen, die verschiedenartig repräsentiertes Wissen benutzen. Dabei wird aber offen gelassen, woher der Agent dieses Wissen bezieht.

Grundsätzlich hat der Agent mehrere Informationsquellen, die zum Teil auf verschiedenen Ebenen, aber nicht unabhängig voneinander darstellbar sind: Information von Sensoren, aus Kommunikationsverbindungen, die eine spezielle Form von Sensoren darstellen, der interne Zustand des Agenten und sein Gedächtnis, als besonderer Teil des internen Zustands. Diese Informationsquellen werden im Folgenden näher dargestellt.

Sensoren Es gibt diverse Schnittstellen zwischen einem Agenten und seiner Umgebung, die man als Form von Sensoren bezeichnen könnte. In diversen Simulationsumgebungen gibt es explizit repräsentierte Sensoren, um die für den Agenten zugängliche Information konfigurierbar beschränken zu können, wie es z.B. in XRaptor [Mössinger et al. 1995] gemacht wird, um Studenten in Programmierpraktika die Effekte beschränkter Sensoren deutlich zu machen. Ein andrer Grund liegt im Einsatzgebiet des Simulators. Sollen damit Roboter–Architekturen in einer simulierten Umwelt getestet werden, dann müssen die Sensoren als solche nachgebildet werden, wie z.B. in INSIGHT [Strippgen 1995]. In den meisten Tools aber gibt es keine explizite Sensorrepräsentation. Informationen aus der Umwelt können entweder nur über Nachrichten von anderen Agenten oder von dem Objekt, das die Umwelt repräsentiert, empfangen und als solche verarbeitet werden. Noch einfacher in bezug auf die vorgegebene Struktur der Sensoren sind Systeme, die Schnittstellen-Funktionen bereitstellen, mit denen der Agent eigenständig Merkmale seiner Umwelt abfragen kann.

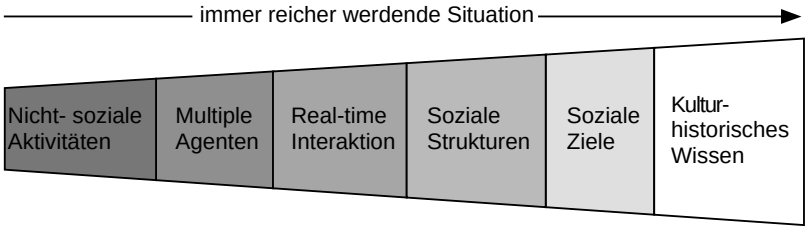
Ein wichtiger Punkt in diesem Zusammenhang ist die Art und Weise, wie die wahrgenommenen Objekte für den Agenten repräsentiert werden und wie er auf sie zugreifen kann. Unabhängig von zeitlicher Attributierung usw. gibt es zwei grundsätzliche Indexierungsformen: absolut und deiktisch. Absolute Indizierung bedeutet, daß jedes Objekt bzw. jeder Agent der Umgebung über seinen Namen, bzw. seine Referenz angesprochen wird. Auf diese Weise ist — bei eindeutigen Namen der Agenten — keine Mehrdeutigkeit möglich. Um festzustellen, welche Objekte in welcher Relation zum wahrnehmenden Agenten stehen, ist allerdings eine aufwendige Instantiierung notwendig. Die andere Möglichkeit ist eine funktionale Indizierung, die sogenannte "deiktische" Repräsentation [Agre & Chapman 1987, Agre 1997]. Der Agent referenziert die Objekte seiner Umwelt durch ihre Bedeutung für seine Aktivitäten. So kann ein anderer Agent als "Der-Hund-der-mich-beißt" in der Wahrnehmung repräsentiert werden, statt aufwendig nach "Waldi" zu suchen. Allerdings sind derartige Referenzierungen nicht eindeutig, z.B. wenn zwei Hunde auf den Agenten zulaufen. Sie sind auch weniger flexibel, da sie direkt an bestimmte Aktivitäten gebunden sind, im Beispiel das Spazierengehen im Park. Für die Generierung derartiger Referenzierungen ist ein ständiges Monitoring der Umgebung notwendig, um die für die aktuelle

Aktivität relevanten “Aspekte”, d.h. Teilkonfigurationen der Umwelt auszufiltern. P. Agre schlägt dazu aufwendige “visual routines” vor, die ständig überprüfen, welche (vorgegebenen) Aspekte in der Welt auffindbar sind [Agre & Chapman 1987, Agre 1989]. Aber auch bei absoluter Referenzierung kann ein ständiges Monitoring der Umgebung wichtig sein, wenn etwa die Bewegung eines Objekts verfolgt werden muß oder wenn die Situation nicht anders erfaßt werden kann.

Information durch Kommunikation Eine andere Informationsquelle sind andere Agenten in der Umwelt, die ihr Wissen über Kommunikationskanäle mitteilen können. Auch hier ist nicht gewährleistet, daß diese Daten unverrauscht ankommen bzw. die Sensoren der anderen Agenten korrekt arbeiten. Grundsätzlich trifft man in diesem Zusammenhang auf Terminologie-Probleme. Diese kommen allerdings in Multiagentenmodellen nicht in dem Ausmaß wie bei Multiagentensystemen in (realen) verteilten Anwendungsszenarien vor, da meist die Kommunikationsinhalte ebenso Teil eines Modells sind, wie das Agentenverhalten selbst. Sie unterliegen deshalb der Kontrolle des Modellbauers. Geschieht die Interaktion zwischen den modellierten Agenten über standardisierte Kommunikationssprachen, wie zum Beispiel *KQML* [Finin et al. 1997, Cohen & Levesque 1995], können verschiedene Modellbauer ihre unterschiedlich konstruierten Agenten zu einem System zusammenbauen, wenn zumindest die zugrundeliegende Semantik geklärt ist.

Gedächtnis Einmal gemachte Wahrnehmungen kann der Agent in einer beliebigen Form – von logischen Aussagen bis hin zu Zählern – in seinem Gedächtnis ablegen. Eine Folge davon ist, daß nicht mehr jede aktuell relevante Information aus der Umwelt geholt werden muß, sondern aus dem subjektiv erstellten internen Modell der Umwelt bezogen werden kann.

Abbildung 4.2 Bestandteile eines internen Umweltmodells mit immer weiterreichenden Informationen über seine (soziale) Umwelt und seine Einbindung in diese (nach [Carley & Newell 1994])



Es gibt, wie in Abbildung 4.2 dargestellt, verschieden detaillierte interne Weltmodelle, die von dem bloßen Wissen um seine Aufgaben und Ziele in der Welt und Information über soziale Strukturen und die eigene Einbindung in eben diese Strukturen, bis hin zu vollständigem Wissen über kultur-historische Zusammenhänge reichen. Je mehr von diesem Wissen dem Agenten zur Verfügung steht und je mehr er davon verarbeiten kann, desto eher kann man von einem “sozialen” Agenten sprechen: Derartig explizites Wissen um soziale Kontakte, Dynamik, mehr oder weniger gemeinsam verfolgte Ziele, werden oft als die Essenz eines “sozialen” Agenten betrachtet [Carley & Newell 1994, Castelfranchi 1998].

Interner Zustand Technisch betrachtet, bildet das Gedächtnis des Agenten den Teil eines weiter faßbaren internen Zustands (siehe auch 2.2.1.1). Attribute können auch interne Motivationen, veränderliche oder statische Eigenschaften des Agenten beschreiben. “Hunger” kann z.B. über einen Energiewert, der kontinuierlich verringert wird, bestimmt werden. “Augenfarbe” oder andere Merkmale gehören auch zum Zustand eines Agenten, sind aber nicht Teil des Gedächtnisses oder eines anders gearteten internen Weltmodells. Auch die aktuelle Aktivität oder der aktuell verfolgte Plan sind Bestandteile des internen Zustands eines Agenten und können Auswirkungen auf die Auswahl der nächsten Aktion haben.

Im Zusammenhang mit dem von den modellierten Agenten verarbeiteten Wissen ist es wichtig, daß die gewählte Wissensrepräsentation und der zugehörige Verarbeitungsmechanismus auch Bestandteil des Modells sind oder zumindest eine beträchtlichen Einfluß auf das Modell haben. Wenn diese also in Form einer Spezifikationssprache oder Architektur vorgeben sind, sollten sie für den Modellbauer transparent sein und möglichst wenig Einschränkungen vornehmen.

Bevor auf Aktionsselektionsmechanismen und das resultierende Verhalten eingegangen wird, soll ein kurzer Überblick über mögliche Formen der Aktionsrepräsentation gegeben werden.

4.1.3.2 Darstellung von Aktionen

Agenten verfügen über zwei Schnittstellen zur Umwelt, einerseits die Sensoren, über die der Agent Information über seine Umwelt erhält, andererseits die Effektoren, mit denen er versucht, seine Umwelt zu verändern. Effektoren führen die vom Agenten ausgewählte Aktion aus. In erster Linie haben also Aktionen immer mit Modifikation zu tun, allerdings müssen sie nicht auf Standardaktionen, wie Bewegen oder das Aufnehmen von Lasten beschränkt sein. Auch das Absenden einer Nachricht kann als eine Aktion verstanden werden.

Erstaunlicherweise gibt es in der Literatur zu Multiagentensystemen kaum Aufsätze zur Aktionsdarstellung, die über die klassische Repräsentation als zustandsverändernde Operatoranwendung hinaus gehen. Welche Form eine Aktion annimmt, scheint bei den Problemen, die sich aus Koordination und Multiagentenplanen oder der Behandlung von internen Weltmodellen usw. ergeben, zweitrangig zu sein.

Die am weitesten verbreitete, “klassische” Form von Aktionen ist mit einer funktionalen Sichtweise verbunden. Eine Aktion ist dabei die Ausführung eines parametrisierten Operatoraufrufs. Ausgangspunkt ist dabei eine Repräsentation des (Teil-)Zustands der Welt und eine ebenso explizite Darstellung der Ziele des Agenten, meist in Form einer Zielsituation. Durch eine Aktion soll ein Übergang in einen anderen Zustand, welcher dem Zielzustand näher ist, erreicht werden. Ein Operator kann dabei explizit mit “Vor- und Nachbedingungen” repräsentiert werden.

Diese explizite Operatordarstellung bildet die Voraussetzung für jede Art von Planen mit Aktionen als die direkt ausführbaren, primitivsten Teile des resultierenden Plans. Wenn der Agent die Effekte einzelner Aktionen nicht abschätzen kann, kann er in Verbindung mit seiner internen Beschreibung der Welt keine “Simulationen” durchführen. Diese sind aber zur Generierung bzw. Bewertung von Aktionsfolgen notwendig. Weitere Ausführungen zu dabei relevanten Formen von Situations- und Aktionslogiken und -repräsentationen und den dabei auftretenden Problemen, findet man zum Beispiel in [Genesereth & Nilson 1987, Georgeff 1998], aber auch in jedem Lehrbuch zur Künstlichen Intelligenz. Einen sehr umfassen-

den Überblick über mögliche Aktionsdarstellungen in Multiagentensystemen findet man in [Ferber 1999].

Eine vorgegebene Agentenarchitektur beeinflußt die möglichen Aktionsformen. Die dabei unterstützten Strukturen rangieren von Aktionen als abstrakte logische Aussagen mit Variablen bis hin zu Aufrufen von Schnittstellenfunktionen. Grundsätzlich haben dabei die Merkmalsdimensionen der Parametrisierbarkeit, Diskretheit und der potentiellen Parallelität einen Einfluß auf die Verhaltensmodellierung.

Parametrisierbarkeit Müssen die Aktionen parametrisiert werden? Parametrisierte Aktionen besitzen den Charakter der oben beschriebenen Operatoren. Der Modellbauer muß explizit repräsentieren, ob und mit welchen Instanzen die Aktion ausführbar ist. Dabei ist ein Schließen über durchführbare Aktionen möglich. In nicht-parametrisierten Aktionen ist dagegen das Wissen, welche Teile der Umwelt wie verändert werden, implizit und möglicherweise absolut codiert. Die Selektion einer Aktion wird somit zum einstufigen Auswahlproblem. So ist beispielsweise bei einer nicht-parametrisierten Kommunikationsaktion der Inhalt Bestandteil der Aktion. Die Aktion wird ausgewählt, es kann kein Schließen darüber stattfinden, mit welchem Inhalt die Nachricht geschickt wird. Eine Flexibilisierung derartiger Aktionen kann man dadurch ermöglichen, indem man zuläßt, daß während der Ausführung der Aktion auf bestimmte Parameter des Agenten zugegriffen wird. Die Aktion ist somit nur indirekt parametrisiert. Allerdings müssen derartige Aktionen transparent und gut dokumentiert sein, da das Wissen, welche Parameter verwendet werden, nicht explizit sein muß. Nicht-parametrisierte Aktionen sind eher auf dem Niveau von Bewegungssteuerungen u. a. verbreitet und nehmen dabei die Form von Kommandos an. Ein Beispiel sind Modelle, die mit MICE [Durfee & Montgomery 1989] erstellt wurden: Die Aktualisierungsfunktion der einzelnen Agenten muß einen Token zurückliefern, der repräsentiert, in welche Richtung sich der Agent bewegt. Parametrisierte Aktionen findet man dagegen eher bei logik-basierten Umgebungen.

Diskrete Aktionen oder kontinuierliche Regelung Eine andere wichtige Unterscheidung bei der Darstellung von Aktionen liegt darin, ob die Aktion diskret oder kontinuierlich ist, also ob die Aktion zu einem Zeitpunkt eine diskrete Änderung in der Umwelt bewirkt oder in beliebig kleinen Zeitintervallen entsprechend kontinuierliche Änderungen bewirkt. Der Übergang zwischen diesen beiden Aktionsformen ist fließend und abhängig von dem zugrundeliegenden Zeitmodell. Im Prinzip ist das Schließen über diskrete Aktionen einfacher, da sie zum einen auf einer abstrakteren Ebene formuliert werden und zum anderen weniger Flexibilität bezüglich des Ausführungszeitpunkts und des Zeitpunkts, an dem die Effekte der Aktion sichtbar sein müssen, besteht. Ein Aktionsselektionsmechanismus, der kontinuierliche Aktionen liefert, besitzt eher den Charakter eines Regelkreises (siehe Seite 18). Kontinuierliche Aktionen findet man daher eher in Roboter-Steuerungen, diskrete eher in Simulationsumgebungen mit abstrakterer Aktionsdarstellung.

Sequentielle oder Parallele Ausführung Ein weiteres Merkmal einer Aktion bzw. Menge von Aktionen liegt in ihrer Ausführung. Werden die Aktionen sequentiell ausgeführt oder müssen sie parallel behandelt werden? Das Planen von parallelen Aktionen ist komplexer als das einer Aktionsfolge, ist aber bei der Steuerung von Robotern (gleichzeitige Regelung von Geschwindigkeit und Richtung der Räder) notwendig.

Aus diesen Überlegungen ergibt sich die Frage, auf welchem Abstraktionsniveau die Aktionen bestimmt werden, also wieviel die Simulationsumgebung dem Agenten abnimmt bzw. inwieweit sie ihn gleichzeitig beschränkt. Das ist ein Aspekt, der auch bei der Besprechung von vorhandenen Simulationsumgebungen für Multiagentensysteme in Abschnitt 4.4.2 behandelt wird.

4.1.3.3 Formen der Aktionsselektion

Die Aktionsselektion hat die Aufgabe, die Wahrnehmungen des Agenten mit den auszuführenden Aktionen in Beziehung zu setzten. Im einfachsten Fall wird die zu jedem Zeitpunkt auszuführende Aktion ausgewählt. Andererseits kann dahinter auch ein Planungsmechanismus verborgen sein. Die konkrete Vorgehensweise bei der Bestimmung der nächsten Aktion ist dabei von der Agentenarchitektur abhängig. In Abschnitt 2.2.1.2 wurde bereits ein umfassender Überblick über mögliche Strukturen gegeben. In diesem Abschnitt sollen ein paar grundsätzliche Beziehungen zwischen Aktionsselektion und dem zu modellierenden Verhalten der Agenten besprochen werden.

Beschränkt man sich auf die Beschreibung von Verhalten, kann man die Aktionsselektion eines Agenten in vier Kategorien mit unterschiedlichen Auswirkungen auf die Anforderungen an die Verhaltensmodellierung einteilen [Klügl et al. 1998]. Diese Kategorien charakterisieren ein Spektrum von einfach zu implementierenden, aber eher unflexiblen zu komplexen flexiblen Verhaltensmodellen.

- Jeder Agent hat eine bestimmte Rolle bzw. eine Aktivität, die er ohne Rücksicht auf eine möglicherweise veränderte Umweltdynamik ausführt. Ein Wechsel dieser Aktivität, also der Folge seiner Aktionen, ist nicht möglich. Derartige Agenten sind völlig inflexibel, wohingegen die Vorgabe eines derartigen Agentenprogramms relativ einfach ist.

  Beispiele dafür findet man in sehr einfachen Simulationen, bei denen genau ein Phänomen, z.B. das Nestbauverhalten von Wespen [Deneubourg et al. 1991], untersucht werden soll. Das Verhalten eines Agenten wird durch genau eine Regel gesteuert, anderes Verhalten ist nicht vorgesehen. In diese Kategorie kann man auch andere Formen von Agenten einordnen, die durch fixe Skripten gesteuert werden.

- Eine derartige Spezialisierung der Agenten kann ein wenig gelockert werden, indem man bei bestimmten, besonders relevanten Stimuli einen Wechsel der Aktionsskripten zuläßt. Derartige Steuerungen bieten immer noch eine einfache Modellierung, da zusätzlich zur Formulierung der Skripten nur wenige Regeln für “Notfälle” spezifiziert werden müssen.

- Der Übergang zu flexibleren, aktivitätsgebundenen Agentensteuerungen ist fließend. Es existieren keine strengen Rollen mehr, alle Aktionen sind in Form von Aktivitäten organisiert. Der Agent befindet sich solange in einem “Aktivitätszustand”, bis eine bestimmte Situation eintritt, die eine andere Aktivität zur Folge hat. Durch ein komplexeres Verhaltensprogramm mit mehreren Aktivitäten und deutlich mehr Übergängen zwischen diesen kann der Agent flexibler reagieren; es muß aber mehr Sorgfalt bei der Modellierung aufgewendet werden. Man könnte auch Formen von deliberativen Agenten, die sich in Verhandlungen mit anderen Agenten verpflichten, bestimmte Teil-Pläne auszuführen, dieser Kategorie zuordnen.

- In die flexibelste Kategorie gehören die klassischen subkognitiven Agenten. Zu jedem Zeittakt muß neu entschieden werden, welche Aktion ausgeführt werden soll. Derartiges Verhalten ist im Modell handhabbar, wenn das Repertoire eines Agenten beschränkt ist und die Situation, in der die Aktionen durchzuführen sind, relativ gut strukturiert betrachtet werden kann. In manchen Multiagentensimulationen findet man auch komplexe Verrechnungsschemata, um ausgehend von einem Stimulus die passende Aktion zu finden. Allerdings müssen derartige Schemata explizit und transparent sein, ansonsten kann die Erstellung des Verhaltensprogramms für einen Modellierer, der das Verrechnungsschema nicht entworfen hat, kompliziert sein und unter Umständen überraschende Effekte erzeugen.

Eine interessante Frage bei der Betrachtung von Aktionsselektionsformen ist die Frage nach der Bedeutung von Lernmechanismen. Bestimmte Formen von Lernen, z.B. Reinforcement Learning [Sutton & Barto 1998], setzen bestimmte Selektionsmechanismen voraus.

4.1.4 Formen von Interaktion

Eine wichtige Frage bei Multiagentenmodellen ist die, welche Interaktionen vorkommen können und wie diese repräsentiert werden. Gerade die vielfältigen Möglichkeiten der Interaktionen zwischen den einzelnen Einheiten und zwischen den Agenten und ihrer Umwelt machen derartige Modelle so attraktiv. Darüber hinaus ist die Betrachtung von Interaktion für Multiagentensysteme von herausragender Bedeutung. So gründen zum Beispiel M. N. Huhns und M. Singh [Huhns & Singh 1999] ihren Agentenbegriff darauf, wie wichtig die Interaktion mit einem Agenten-Kandidaten für das restliche System ist.

Grundsätzlich geschieht Interaktion immer dann, wenn zwei oder mehr Agenten in eine dynamische Beziehung gesetzt werden, weil sie durch ihre Aktionen oder andere Ereignisse direkt oder indirekt über andere Agenten oder die Umwelt miteinander in Kontakt kommen [Ferber 1999]. Dabei sind folgende Annahmen für die allgemeine Situation notwendig:

1. Es sind mehrere Agenten anwesend, die kommunizieren oder Aktionen ausführen können.

2. Es gibt eine Situation, die als “Treffpunkt” dienen kann, also eine räumliche und zeitliche Einheit, in der die Agenten “interagieren”.

3. Es gibt dynamische Elemente, die eine lokale oder temporale Beziehung der Agenten zulassen, z.B. Ressourcen, die sie benutzen müssen oder Nachrichten, über die kommuniziert werden kann.

4. Es gibt ein gewisses Maß an (relativer) Autonomie der Agenten, d.h. die Agenten verfügen über keine fixierte Beziehung zueinander.

Darüber hinaus stellt J. Ferber [Ferber 1999] eine Klassifikation von Interaktionssituationen vor. Diese beruht auf drei Kriterien:

- Sind die Ziele der Agenten kompatibel oder inkompatibel? Zwei Agenten verfügen über inkompatible Ziele, wenn das Ziel-Erreichen eines Agenten zu einem Zustand führt, in dem der andere sein Ziel nicht mehr erreichen kann.

- Damit sind folgende Fragen verbunden: Wie stehen die Agenten zu den vorhandenen Ressourcen? Sind diese für die Ziele aller Agenten ausreichend?
- Eine analoge Frage betrifft die Fähigkeiten der Agenten. Sind diese ausreichend, um alleine das gegebene Ziel zu erfüllen?

Auf der Basis dieser drei Kriterien kann J. Ferber eine systematische Aufstellung von Interaktionstypen aufführen, wie sie in Tabelle 4.1 zusammengefaßt ist.

Tabelle 4.1 Interaktionstypen nach [Ferber 1999]

Kategorie	Typ der Interaktion	Ziele	Ressourcen	Fähigkeiten
Gleichgültig	Unabhängigkeit	kompat.	ausreichend	ausreichend
	Einfache Zusammenarbeit	kompat.	ausreichend	nicht ausr.
Kooperation	Behinderung	kompat.	nicht ausr.	ausreichend
	koordinierte Kooperation	kompat.	nicht ausr.	nicht ausr.
	rein indivd. Wettbewerb	inkomp.	ausreichend	ausreichend
Antagonismus	rein kollekt. Wettbewerb	inkomp.	ausreichend	nicht ausr.
	indiv. Resourcen-Konflikte	inkomp.	nicht ausr.	ausreichend
	kollekt. Resourcen-Konfl.	inkomp.	nicht ausr.	nicht ausr.

In Multiagentensimulationen kann man auch andere, nicht direkt an kooperative Szenarien gebundene Formen von Interaktionen finden. Das ist gerade in Simulationen von sozialen Systemen, die Vorgänge wie Effekte von “Freundschaften”, “Heiraten” u.ä. nachbilden, der Fall. Generelle Aussagen über diese Interaktionsformen kann man nicht treffen, da sie als Bestandteile des Modells bzw. der Fragestellung, variierbar sind.

Im Bereich der Multiagentensysteme findet man diverse Ideen für Interaktionsmuster oder Sprachen für die Darstellung von Kooperativen Aktivitäten. Diese kann man dazu benutzen, Interaktionen der oben charakterisierten Formen zu realisieren. Derartige Möglichkeiten zur Unterstützung der Modellierung werden in Abschnitt 4.3.2.2 ausführlich behandelt.

4.1.5 Design der Umgebung

Betrachtet man einen Agenten in einem Multiagentensystem, dann bilden zumindest die anderen Agenten dessen Umgebung. Derartige Szenarien findet man oft bei der Simulation von Wirtschafts- oder politischen Prozessen [Axelrod 1997a, Axelrod 1995]. Darüber hinaus kann die Umgebung auch ohne das Agentenverhalten eine interessante Dynamik besitzen. Davon leben “Testbeds”, d.h. Experimentierumgebungen für Agenten und Agentensysteme [Hanks et al. 1993]. In einer “interessanten” Umgebung müssen Agenten ihre Aufgaben erfüllen, auch wenn ihre Aktionen indeterministische Effekte haben oder unerwartete Ereignisse dazu führen, daß keine verläßlichen Daten über Zustände der Umwelt möglich sind. In Tileworld [Pollack & Ringuette 1990] entstehen und verschwinden zufällig Löcher, die der Agent mit Blöcken verstopfen soll.

Ein mit sehr hohem Aufwand erzeugtes Umweltverhalten besitzt das Phoenix-Testbed [Cohen et al. 1989]. Das Umweltmodell, das diesem Testbed zugrunde liegt, ist ein realistisches Waldbrand-Modell. Auf der Basis eines Zellulären Automaten bietet es auf einer detailgetreuen Karte des Yellowstone-Nationalparks ein Szenario für die Steuerung von Teams aus Bulldozern zur Feuerbekämpfung. Auch Sugarscape [Epstein & Axtrell 1996] besitzt

einen Zellulären Automaten als Umweltmodell. Auf jeder Zelle liegt eine bestimmte (veränderliche) Menge an Zucker, also der Ressource, die die Agenten benötigen. Auf eine sehr abstrakte Weise werden in diesem Szenario soziale Verhaltensweisen nachgebildet. Sugarscape ist kein Testbed, in das verschiedene Agentenarchitekturen hineingesetzt werden, sondern ein generisches Modell für soziale Verhaltensweisen.

Aus diesen Beispielen wird deutlich, daß das Umweltmodell für ein Agentensystem keinen formalen Beschränkungen ausgesetzt ist. Auf diese Weise lassen sich verschiedene Simulationsansätze gut kombinieren: so ist es vorstellbar, die Dynamik einer bestimmten Zustandsgröße des Gesamtsystems durch ein Makromodell zu repräsentieren. Der Wert dieser Zustandsgröße kann für Agenten zugänglich sein, oder Ereignisse mit für die Agenten wahrnehmbaren Effekten triggern.

4.2 Charakteristika von Multiagentensimulationen

4.2.1 Ausgangspunkt: Geeignete Systeme

Ausgehend von der Behandlung der Bestandteile einer Multiagentensimulation werden kurz die Charakteristika erläutert, die bei einem zu modellierenden System identifizierbar sein sollten, damit eine Verwendung von Multiagentensimulation attraktiv bzw. sinnvoll ist.

- Das Ursprungssystem besteht aus Einheiten, deren individuelle Unterschiede nicht "weg-gemittelt" werden können, ohne das Ergebnis der Experimente zu sehr zu verfälschen. Beispiele sind biologische Systeme mit relativ wenig Agenten. Bei Gesellschaften mit großen Populationen sind meist andere Simulationsformalismen, wie Makromodelle, sinnvoller, da individuelle Merkmale bei sehr vielen Agenten – statistisch gesehen – nicht mehr relevant sind.

- Die Interaktionen zwischen den Einheiten sind zu komplex, als daß man sie durch eine Formel ausdrücken könnte, ohne dabei wichtige Effekte zu ignorieren. Das kann beispielsweise bei soziologischen Modellen der Fall sein. Während Interaktionen bei einfachen, rein rational handelnden Agenten[1] auch mit einfacheren Techniken abbildbar und analysierbar sind, ist die Kombination von Einheiten, die auf ihre lokale Situation reagieren und sich so an die Dynamik ihrer Umwelt anpassen, wohl am besten auf der Basis interagierender Agenten modellierbar. Ähnliches gilt bei wirtschaftswissenschaftlichen Modellen, wenn ein Agent nicht nur seinen wirtschaftlichen Erfolg maximieren will, sondern darüber hinaus weitere Faktoren in seine Entscheidungen miteinbezieht.

- Spielt der Raum mit seinen Inhomogenitäten eine wichtige Rolle, wie zum Beispiel in Ökosystem-Modellen, sind Modelle ungeeignet, in denen diese Unterschiede nicht abbildbar sind. Multiagentenmodelle bieten über die Zellulären Automaten hinaus die Möglichkeit, nicht nur ein interessantes Raummodell zu entwickeln, sondern auch Agenten in einer organischen Weise in diesem inhomogenen Raum handeln zu lassen. Ein derartiges Umweltmodell kann dabei zudem stark variierende Umwelteinflüssen, z.B. kurzfristige Katastrophen, zusammen mit langfristiger Dynamik beinhalten.

[1] siehe spieltheoretische Modelle

Im Folgenden werden grundsätzliche Möglichkeiten und gefährliche Fallsticke der Multiagentensimulation dargestellt.

4.2.2 Möglichkeiten der Multiagentensimulation

Der große Vorteil der Multiagentensimulation besteht durch die abstrakten Einheiten darin, daß die zugrunde liegende Konzeptionalisierung bereits relativ nahe an einem zu modellierenden Originalsystem ist. Selbstverständlich kann man die Methodik in vielen Domänen benutzen. Gerade bei der Nachbildung von Gesellschaften, die sich aus individuell agierenden, relativ autonomen Einheiten zusammensetzen, ist der Modellierungsprozess direkter zugänglich als beispielsweise bei einem äquivalenten Makromodell. Der Fokus der Modellierung liegt auf dem individuellen Verhalten, das eine natürliche Beobachtungsbasis für die Datenakquisition bis hin zur Validierung bildet. Man markiert beispielsweise ein einzelnes Tier und verfolgt dessen Aktionen in seiner natürlichen Umwelt oder seine Reaktionen auf kontrollierte Reize.

Der Einsatz der Multiagentensimulation bietet über diesen Aspekt der direkteren Konzeptionalisierung hinaus weitere Vorteile: Inhomogenitäten im Raum sind ebenso darstellbar, wie heterogene Gesellschaften. Es besteht die Möglichkeit emergentes Verhalten zu untersuchen und dabei Feedback-Mechanismen über verschiedene Systemebenen abzubilden. Mehr methodischer Natur sind die Möglichkeiten, die sich aus der inhärenten Modularität eines Gesamtmodells ergeben. Diese Aspekte werden im folgenden näher betrachtet.

Ein wichtiger Kritikpunkt an Makromodellen ist, daß diese ein System als homogen voraussetzen müssen. In Gleichungen für Räuber-Beute-Systeme wird beispielsweise angenommen, daß Räuber und Beutetiere gleichmäßig über den Raum verteilt sind. Derartige Einschränkungen gibt es in Multiagentenmodellen mit expliziter Repräsentation des Raums nicht. Weil Agenten in ihrer Umwelt an einer bestimmten Position oder Relation zu ihren Nachbarn stehen, kann lokales Verhalten auf der Basis von lokalen Informationen modelliert werden. Auf diese Weise können Inhomogenitäten im Raum dargestellt und auch in das Verhaltensmodell der Agenten beachtet werden.

Grundsätzlich ist das Verhaltensmodell eines individuellen Agenten in seiner Formulierung nicht beschränkt. Man kann sowohl quantitative Information, als auch Entscheidungen auf der Basis qualitativer Werte verwenden. Hypothesen über individuelle Informationsverarbeitung und Gedächtnismechanismen sind ebenso abbildbar, wie individuelles Lernen und Verhaltensadaption. Aufgrund der Möglichkeit, jeden einzelnen Agenten individuell gestalten zu können, sind echt heterogene Gesellschaften möglich. Verschiedene Formen von Interaktion zwischen verschiedenen Einheiten sind wegen dieser modellierbaren Heterogenität darstellbar. Auch für eine Nachbildung von Co-Evolution, also eine gegenseitige Anpassung verschiedener Agenten bzw. Agentenpopulationen, ist eine Konzeptionalisierung als Multiagentensystem hilfreich. Multiagentenmodelle ermöglichen allein durch die mögliche Heterogenität vielfältige Formen von Selektionsexperimenten. Man kann beispielsweise Agenten mit unterschiedlichen Verhaltensstrategien in der selben Umwelt um benötigte Ressourcen konkurrieren lassen und beobachten, welche Strategie sich mit der Zeit durchsetzt.

Die Dynamik des Gesamtsystems wird während eines Simulationsexperiments aus der Interaktion der individuellen Verhaltensweisen generiert. Es müssen also keine Annahmen über das globale Verhalten explizit im Modell verwendet werden. Damit eignet sich diese Methode hervorragend zur Nachbildung und Untersuchung emergenter Phänomene. Diese sind dadurch charakterisiert, daß die verschiedenen Beschreibungsebenen – lokal und glo-

bal – getrennt voneinander betrachtet werden müssen. Das Vokabular, mit dem das lokale Verhalten beschrieben werden kann, reicht für eine Beschreibung des aggregierten Verhaltens nicht aus [Wavish 1992, Steels 1994, Gilbert 1995, Cariani 1991], oder wie J. Holland es ausdrückt: “Das Ganze ist mehr als die Summe seiner Teile” [Holland 1998]. Emergentes Verhalten und Strukturen werden also am besten dadurch analysiert, daß man das Gesamtsystem auf der Basis einer Simulation seiner Bestandteile darstellt [Darley 1994].

Besonders interessante Untersuchungsmöglichkeiten bieten sich, wenn zusätzlich zum Verhalten der Agenten das des Gesamtsystems beschrieben wird. Man kann das lokale Verhalten der Agenten solange modifizieren, bis es das beschriebene globale Muster reproduziert und danach nochmals die Eigenschaften und Faktoren des lokalen Verhaltens genauer analysieren. Auf diese Weise kann die Simulation direkte Hinweise auf weitere Experimente z.B. an Tieren liefern oder darüber, ob Bausteine, die ein derartiges Gesamtsystem ergeben, wirklich existieren können. Andererseits kann man untersuchen, welches globale Muster durch ein gegebenes lokales Verhalten erzeugt wird und dieses Gesamtphänomen mit der Realität vergleichen. Analoge Aussagen kann man auch für mehrere Zwischenstufen von aggregiertem Verhalten treffen.

Multiagentenmodelle bieten darüber hinaus weitere methodische Vorteile. Durch die inhärente Modularität derartiger Modelle ist eine schrittweise Konstruktion möglich. Man beginnt damit, das Verhalten einer Agentenklasse zu spezifizieren und dieses valide zu gestalten. Danach fügt man weitere Agentenklassen hinzu. Änderungen im Verhaltensmodell haben meist nur lokale Auswirkungen, das Verhalten anderer Agentenklassen muß davon nicht betroffen sein, denn alle Interaktionen sind per definitionem lokal. Daraus ergibt sich auch die Möglichkeit, daß mehrere Modellierer zusammen am selben Gesamtmodell arbeiten. Verhaltensbeschreibungen von unterschiedlichen, auch räumlich verteilten Forschern können aufgrund der Modularität von Multiagentenmodellen zusammengefügt werden [Mentges 1999]. Das gilt im Besonderen auch für Modelle, die unterschiedliche Ebenen desselben Modells beschreiben.

4.2.3 “Fallstricke” und ihre Vermeidung

Agentenbasierte Simulation ist eine relativ neue Methodik. Demzufolge gibt es darüber kaum Kurse und Lehrbücher. Fachbereichsexperten, die diese Modellierungskonzepte benutzen wollen, sind quasi auf sich alleine gestellt [Axelrod 1997a] und mußten bisher mit den im Folgenden angesprochenen, möglichen Schwierigkeiten ohne weitere Unterstützung umgehen. Diese potentiellen Fallstricke gliedern sich in zwei Phasen: der Entwurf des Modells und dessen Umsetzung.

4.2.3.1 Schwierigkeiten bei Entwurf und Erstellung eines Multiagentenmodells

Weil man beim Entwurf eines Multiagentenmodells weniger Einschränkungen beachten muß als bei anderen Formen der individuenbasierten Simulation, ist die Wahl des Detaillierungsgrads, der lokalen Verhaltensweisen und der richtigen Parameterwerte schwieriger.

Findet man nicht den richtigen Abstraktionsgrad bei der Modellierung, kann das Modell nur noch mit großem Aufwand kalibriert und validiert werden oder liefert triviale Ergebnisse. Viele Details der Realität bei der Modellierung zu beachten und in das Modell einzubringen, kann gefährlich sein, wenn man dadurch die Übersicht über die Annahmen im Modell

verliert. Der Detaillierungsgrad des Modells sollte kein Selbstzweck, sondern durch die Fragestellung erforderlich und durch die im Modell gemachten Annahmen kontrollierbar sein. Abstrahiert man zu stark oder beschränkt sich zu weit, kann es sein, daß nur triviales Globalverhalten produziert wird, das auch ohne Simulation vorhersagbar gewesen wäre.

Die Erstellung eines Modells kann sehr aufwendig sein. Kennt man die lokalen Verhaltensweisen nicht genau genug, hat aber das Ziel, ein bestimmtes globales Muster zu reproduzieren, kann der Prozess, die dafür verantwortlichen Basisverhalten zu finden und richtig einzustellen, mühevoll sein. Bei einem derartigen, rigorosen "Bottom Up" –Ansatz bleibt unter Umständen dazu nur eine "Try and Error"-Strategie [Wavish 1992]. Eine weitere Schwierigkeit liegt demzufolge in der Beurteilbarkeit eines bestimmten Verhaltens: Kann man durch dieses lokale Verhalten das globale Muster generell nicht erzeugen oder sind nur die verwendeten Parametereinstellungen ungeeignet? Deshalb sollte man das Verhalten eines Agenten gut genug kennen, um es relativ genau bestimmen zu können. Üblicherweise kann man wohl davon ausgehen, daß ein Agent eine bessere Beobachtungsbasis bietet als das Gesamtsystem und dementsprechend fehlende Daten im Vergleich zu Makrosimulationen einfacher erhoben werden könnnen.

Im Vergleich zu anderen Simulationsmethodiken ist auch die Einstellung der Parameter sehr diffizil. Wird das Verhaltensmodell zur Steuerung vieler Agenten verwendet, potenzieren sich die Effekte eines Parameters, wie beispielsweise einer Reizschwelle. Dementsprechend ist es nicht überraschend, daß Multiagentensimulationen schon auf kleine Parameteränderungen sehr sensibel reagieren. Auch deshalb ist es sinnvoll, das Verhalten der einzelnen Agenten genau zu erfassen und dabei jeden Hinweis auf festzulegende Verhaltensparameter zu verfolgen.

Bei der Einstellung der Parameter kann die Lokalität des Verhaltens "unschöne" Effekte zur Folge haben, insbesondere wenn man erwartet, daß das individuenbasierte Modell die Ergebnisse eines Makromodells reproduziert. W. Wilson [Wilson 1998] bezeichnet die Beobachtung, daß diese Entsprechung nicht ohne beträchtlichen Kalibrierungsaufwand zu realisieren ist, als das Phänomen der "lokalen Populationen". Während man in einfachen Räuber-Beute-Modellen auf Makroebene eine versetzte Dynamik bei den Populationszahlen beobachten kann — bei der also keine der beiden Populationen ausstirbt —, kann dies bei individuenbasierten Simulation bei einer räumlich inhomogenen Verteilung der simulierten Individuen und der daraus resultierenden Interaktionen durchaus geschehen. Eine weitere Schwierigkeit bei der Einstellung von Parametern ergibt sich, wenn Verhaltensbeschreibungen mit stochastischen Elementen beteiligt sind, oder eigentlich kontinuierliche Parameter für die Simulation diskretisiert werden müssen. Die Kalibrierung eines Multiagentenmodells ist aus diesen Gründen ein aufwendiges und schwieriges Unterfangen, für das geeignete Vorgehensweisen untersucht werden (siehe dazu [Oechslein et al. 1999, Müller 1999]).

4.2.3.2 Probleme bei der Simulation eines Multiagentenmodells

Der wichtigste Nachteil jeder individuenbasierten Simulation besteht darin, daß sie deutlich mehr Hardware-Ressourcen erfordert, als eine abstrakte Nachbildung des System auf Makroebene. Es werden hohe Anforderungen an die Rechen- und Speicherkapazität der verwendeten Rechner gestellt: Jedes Individuum muß eine Entsprechung im Rechner besitzen und separat aktualisiert werden. Der Speicherbedarf und Rechenaufwand kann sogar die eigentlich notwendige Komplexität des Individuums bzw. eine Mindestanzahl von einzelnen Einheiten in Frage stellen. Dann kann man versucht sein, das darzustellende Verhalten zu

vereinfachen und weitere Abstraktionsmittel zu verwenden, wie beispielsweise die Formulierung von nicht weiter spezifiziertem Verhalten durch Wahrscheinlichkeitsverteilungen. Allerdings kommen dabei die oben beschriebenen Hinweise auf mögliche Fallstricke bei der Entwicklung eines Modells zum Tragen. Um tragfähige Aussagen über Modelle mit stochastisch gesteuerten Teilen zu erhalten, muß man deutlich mehr Simulationsexperimente mit der selben Parameterkonfiguration durchführen, als mit einem aufwendigerem aber dafür deterministischen Modell.

Diese Probleme bei der tatsächlichen Verwendung von Multiagentensimulationen kann man durch geeignete Tools mindern. Wie dies für die einzelnen Phasen des Modellierungsprozesses geschehen kann, wird im Folgenden beschrieben.

4.3 Unterstützung der Konstruktion eines Multiagentenmodells

4.3.1 Möglichkeiten ausgehend von den Phasen der Modellbildung

Für die Darstellung, wie die Entwicklung eines Multiagentenmodells unterstützt werden kann, möchte ich auf die im Abschnitt 3.1.2 eingeführten Phasen der Modellbildung und Simulation zurückgreifen. Während diese für jede Art von Simulationsstudie gelten, möchte ich mich im Folgenden auf die konkrete Form der Multiagentenmodelle und insbesondere auf die Möglichkeiten zur Unterstützung der jeweiligen Phase konzentrieren.

4.3.1.1 Konzeptionelle Hilfen

Eine Unterstützung beim Aufbau des Konzeptmodells ist eine wichtige und lohnende Hilfe. Ideal wäre eine etablierte Methodik, mit der ein Modellierer, ähnlich wie *system dynamics* bei Makromodellen (siehe Seite 3.1.3.1), von einer relativ vagen, informellen Systemvorstellung zu einem exakten Modell geführt würde. Die Hoffnung dabei ist, es auch unerfahrenen Modellierern durch methodische Unterstützung zu ermöglichen, ein Modell mit gutem Design zu konstruieren. Darüber hinaus sind auch Hilfen bei der Fehlersuche möglich. Eine weitere Idee dabei ist, daß unter Verwendung derselben Methodik besser vergleichbare Modelle erstellt werden können. “Gute” Modellierungsmethodiken erleichtern zudem die Kommunikation zwischen Modellieren [Monsef 1997, Balci 1997].

Design-Methodiken findet man vor allem im Software-Engineering. Mittlerweile wird auch beim Entwurf von Multiagentensimulationen auf dabei verwendete Diagramme und Formalismen zurückgegriffen, die mit UML[2][Fowler & Scott 1998] festgelegt wurden. So werden im MASSIF-Projekt [Mentges 1999] UML- Sequenz- und Interaktionsdiagramme zur Beschreibung der Kommunikationsprotokolle zwischen den Agenten benutzt. Wenn verschiedene Modellierer unterschiedliche Agenten konstruieren, dann sind derartige Spezifikationen besonders zur Dokumentation und Kommunikationsunterstützung geeignet. Für das Entwickeln der Agenten und des Agentensystems gibt es mittlerweile spezielle agentenorientierte Methodiken (siehe dazu Abschnitt 2.3), die gerade in dieser Phase auch für Multiagentenmodelle sinnvoll verwendbar sind.

[2] “Unified Modelling Language”, eine wohl mittlerweile als etabliert bezeichenbare, auf Diagramm-Formen basierende objekt-orientierte Methodik, die Konzepte aus verschiedenen Methodiken integriert.

4.3.1.2 Unterstützung bei der konkreten Modellkonstruktion

Die Umsetzung eines Konzeptmodells kann auf vielfältige Weise unterstützt werden. So setzen Entwicklungsumgebungen genau an dieser Stelle an. Sie bieten verschiedenartige Hilfen, um den Aufwand, mit dem das Konzeptmodell in ein implementiertes Modell übersetzt wird, zu verringern. Darauf liegt auch der Schwerpunkt dieser Arbeit, deshalb sollen im Folgenden die Ansatzpunkte für eine Unterstützung der konkreten Modellkonstruktion in Form einer Übersicht zusammengestellt und dabei jeweils auf entsprechende andere Abschnitte verwiesen werden:

Eine Unterstützung, die noch relativ nahe am Konzeptmodell ist, kann durch Referenzmodelle oder vorgegebene Agentenarchitekturen geschehen. In Abschnitt 2.2.1.2 findet man einen umfassenden Überblick über mögliche Architekturen auf der Agentenebene. Desweiteren werden im Abschnitt 4.4.1 bei der Behandlung formaler Spezifikationen Referenzmodelle aus dem Bereich der agentenbasierten Simulation vorgestellt.

Eine andere Möglichkeit, den Modellkonstruktionsprozess zu unterstützen, liegt in der Bereitstellung und Verwendung von deklarativen Repräsentationsschemata, die von einer separaten Simulationsroutine interpretiert werden. Eine an einen speziellen Modellformalismus angepasste Sprache kann aber Konstrukte auf einer Abstraktionsebene bereitstellen, die direkter an der Grundkonzeptionalisierung des Modells liegt, als dies bei einer der höheren Standardprogrammiersprachen der Fall ist. Eine Übersicht über derartige explizite Rahmen für Multiagentensysteme werden in Abschnitt 2.3.3 sowie im Abschnitt 4.4.1 für simulierte Agentensysteme gegeben.

Eine der naheliegenden Formen der Unterstützung ist die Bereitstellung von graphischen Tools, die ein Bindeglied zwischen Konzept und Programmcode darstellen. Diagramme werden nicht nur in CASE-Tools und Entwicklungsumgebungen für Expertensysteme verwendet, sondern können auch in Modellierumgebungen einen wichtigen Bestandteil bilden. Kapitel 7 zeigt, wie auf der Basis eines deklarativen Repräsentationsrahmens eine graphische Entwicklungs- und Experimentierumgebung realisiert wurde.

Eine andere Art, wie man die Kosten des Übersetzens eines Konzeptmodells in ein implementiertes Modell verringern kann, ist die Benutzung eines an die Anwendungsdomäne angepassten Tools. So lassen sich Bausteinbibliotheken oder Konstrukte vorgeben und über spezifische visuelle Tools zusammenfügen. Schon alleine eine domänenabhängige Terminologie zur Bedienung der Entwicklungsumgebung kann dabei eine Implementierung erleichtern, da der Modellbauer keine Abstraktion seiner Modellteile vornehmen muß, um z.B. den passenden Befehl zu finden. Domänenspezifische Simulationssysteme werden in dieser Arbeit nicht direkt besprochen.

4.3.1.3 Animation und Auswertungshilfen

Untersuchungen mittels Modellbildung und Simulation enden nicht mit der Erstellung des Modells. Die Konstruktion desselben kann noch so gut unterstützt werden, wenn Simulationsexperimente und das Extrahieren von Daten zu aufwendig ist, dann kann man trotz eines gut konzipierten Modells zu keinen brauchbaren Ergebnissen kommen. Einen wichtigen Aspekt bei der Unterstützung der Modellbildung und Simulation bilden dabei Animationsumgebungen und Statistik- oder sonstige Auswertungswerkzeuge. Grundsätzlich sind die mit der Bereitstellung derartiger Funktionalität verbundenen Probleme eher technischer Natur. Verwendet man eine Standardprogrammiersprache für die Entwicklung und Simula-

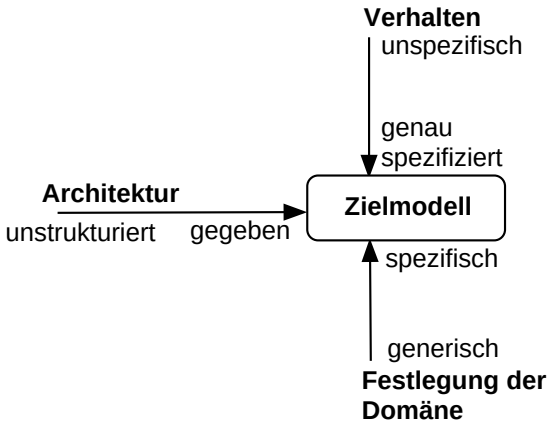
tion eines Modells, ist die Programmierung von Animationen, Datensammel- und Auswertungsfunktionen eine Arbeit, die relativ weit von der Entwicklung des eigentlichen Modells entfernt ist und deshalb von Modellierern wohl als besonders behindernd betrachtet wird.

Deshalb verfügt fast jede Simulationsumgebung nicht nur für Multiagentensimulationen über Hilfen zur Darstellung der Dynamik eines Experiments, Funktionen zur Aufzeichnung der dabei generierten Daten und, darauf aufbauend, zumindest einfache Möglichkeiten zu deren statistischer Aufbereitung. Auch Schnittstellen zu Standard-Statistikprogrammen sind weit verbreitet. Bei der Besprechung von Beispielen für Entwicklungsumgebungen für Multiagentensimulationen in Abschnitt 4.4.2 werden auch die jeweils bereitgestellten Hilfen für Animation und Auswertung dargestellt.

4.3.2 Hilfen bei der Konzipierung der einzelnen Bestandteile

Während im letzten Abschnitt der Prozess der Modellbildung im Vordergrund der Untersuchungen zur Unterstützung steht, sollen im Folgenden Möglichkeiten entsprechend der einzelnen Bestandteile eines Multiagentenmodells analysiert werden. Grundsätzlich gibt es dabei mehrere Dimensionen, die in Abbildung 4.3 dargestellt werden.

Abbildung 4.3 Dimensionen bei der Festlegung eines Multiagentenmodells



Diese Dimensionen sind nicht orthogonal, sondern abhängig von einander. Eine Architektur kann beispielsweise domänenspezifisch sein. Die Spezifikation des Verhaltens kann ohne eine Domänen- und Architekturfestlegung nur auf sehr abstrakte Weise geschehen. Diese Dimensionen kann man sowohl für das Gesamtmodell als auch für die Bestandteile des Modells, wie sie in Abschnitt 4.1 besprochen wurden, identifizieren. Das Zielmodell kann dabei ein einzelner Agent, dessen Verhalten und Architektur modelliert werden sollen, ein Agentensystem mit Interaktions- und Organisationsstrukturen oder auch das Modell der Umwelt sein. Für diese Bestandteile eines Multiagentenmodells werden im Folgenden jeweils konkrete Unterstützungsformen betrachtet.

4.3.2.1 Konstruktion der Prozesse eines Agenten

Ein Schwerpunkt bei der Entwicklung eines Agentensystems ist die Behandlung der Prozesse, die innerhalb eines Agenten ablaufen. Dabei ist die zentrale Frage, wie das Verhalten

eines Agenten auf eine operationalisierbare Art ausgehend von seinen Wahrnehmungen beschrieben werden kann. Die Formulierung der Verhaltensbeschreibung kann auf vielfältige Weise unterstützt werden. Grundsätzlich existieren dafür zwei Kategorien: Man kann einerseits generische Bausteine vorgeben, die zum Agenten zusammengesetzt werden. Als zweite Möglichkeit, die eine andere Ebene anspricht, kann man Sprachen und Konzepte zur Beschreibung und Struktur des Agentenverhaltens bereitstellen.

Auf der Ebene der Bausteine kann man im Grunde für alle Komponenten, die zu einem Agenten zusammengesetzt werden, generische Module entwickeln. Das können spezielle Sensoren und Effektoren in Form von mächtigen Schnittstellen zu Umwelt sein, die abstrahierte Daten liefern oder abstrakte Befehle ausführen können. Andererseits kann man Sensoren und Effektoren explizit als Module auffassen und dafür Bibliotheken bereitstellen, aus denen zum Beispiel ein "Insektenauge" als Sensor oder ein über einen Motor betriebenes Rad als Effektor ausgewählt und bei dem zu modellierenden Agenten eingesetzt werden.

Auch für die Module, die interne Prozesse des Agenten darstellen und steuern, kann man generische Komponenten zur Verfügung stellen. Eine vorgegebene Agentenarchitektur (siehe dazu Abschnitt 2.2.1.2) bildet dabei eine wichtige Hilfe zur Strukturierung der Reasoning-Methoden und die Wissensrepräsentationsformen, die ein Agent benötigt. Meist sind mit einer Architektur spezielle Plan- oder Aktivitätsrepräsentationen verbunden oder es werden methodische Vorgehensweisen impliziert. In diese Kategorie gehören ebenfalls die in Abschnitt 4.4.1.2 beschriebenen Referenzmodelle für simulierte Agenten, obwohl über die grundsätzlichen Strukturen hinaus keine Vorgaben gemacht werden.

Auf einer detaillierteren Ebene kann man zudem die Entwicklung einzelner Bestandteile einer Agentenarchitektur durch vorgegebene Komponenten oder Schemata unterstützen. Dazu gehören allgemeine Wissensrepräsenationsformalismen, wie KIF [Genesereth 1995] oder Truth-Maintenance- bzw. Belief-Revision-Systeme (siehe dazu [Kraetzschmar & Reinema 1993]). Aber auch bestehende generische Planer, Scheduler oder Diagnostik-Komponenten können in einen simulierten Agenten integriert werden, wenn es die Komplexität des zu modellierenden Verhaltens erfordert. Für das Zusammenspiel der internen Prozesse eines Agenten sind in gewissem Ausmaß vorhandene Standardtools zum Design und zur Analyse dieser Komponenten sinnvoll.

Bei der Benutzung derartig vorgegebener Strukturen, ob nun für die gesamte Aktionsauswahl oder nur für einen Teilaspekt seiner Aktivitäten, sollte der Benutzer darauf achten, daß die verwendete Architektur dem Problem angemessen ist. So benötigt man nicht unbedingt mächtige Planstrukturen, wenn ein einfaches reaktives Verhalten formuliert werden soll. Wozu nützt eine komplexe Struktur mit expliziter Repräsentation von Emotion oder sozialen Einstellungen, wenn eine einfachere Formulierung in Form eines zusätzlichen "Beliefs" ausreicht?

Bei noch relativ einfachem zu modellierenden Agentenverhalten bieten Spezifikationssprachen, wie zum Beispiel *DESIRE* (siehe 2.3.3.3) oder *Concurrent*METATEM (siehe 2.3.3.1) ausreichend abstrakte Konstrukte für die Beschreibung des Agentenverhaltens. Zudem kann mit ihnen die Interaktion eines Agenten mit anderen Agenten formuliert werden. Wegen ihrer direkten Ausführbarkeit oder Kompilierbarkeit in ausführbare Sprachen ist die Formulierung eines gesamten Modells in diesen abstrakten Sprachen gut denkbar. Allerdings sollte man beachten, daß dabei alle "first principles" des Modells formuliert werden müssen, da die verwendeten Symbole allein im Modell definiert sind und nicht vom Simulationssystem vorgegeben werden. Sie besitzen keinerlei Bezug zu einer zugrundeliegenden, konkreten Programmiersprache. So erhält ein Symbol wie "+" nur durch eine von Grund auf spezifi-

zierte Verarbeitung im Agenten eine Bedeutung, die sich von einer intuitiv vorgegebenen durchaus unterscheiden kann. Dabei werden automatisch alle dem Modell zugrundeliegenden Annahmen über Funktionsweisen, Vorgehensmodelle usw. explizit dargestellt. Das kann ein Modell sehr umfangreich und komplex machen. Allerdings besitzt der Modellierer eine vollständige Kontrolle über das so deklarierte Modell. Dennoch bleibt die Frage, ob eine ausreichende Offenlegung und Kenntnis der unsichtbaren Annahmen nicht ausreichen würde und auf diese Weise das Modell nicht möglicherweise unnötig aufgeblasen würde.

4.3.2.2 Konstruktion der Interaktionen zwischen den Agenten

Eine entscheidende Frage für die Entwicklung einer Multiagentensimulation ist, wie die Agenten zu einem Gesamtsystem zusammengesetzt werden. Welche Formen von Interaktionen oder Organisationsmustern dabei angenommen werden können, kann nicht völlig unabhängig von den internen Strukturen der Agenten geklärt werden. So ist es z.B. komplex und aufwendig, Verhandlungen zwischen sub-kognitiven Agenten zu modellieren. Agenten, die keine expliziten Ziele verfolgen, können kein Reasoning über eine mögliche Kooperation zu deren Erfüllung durchführen. Allerdings sollten die Strukturen eines Agenten möglichst einfach gehalten werden. Deshalb ist es auch wichtig, auf der Seite des Gesamtsystems die Konstruktion zu unterstützen.

Für die Konstruktion eines Gesamtsystems aus einzelnen Agenten sind Hilfsmittel auf unterschiedlichen Ebenen vorstellbar. Abhängig von dem zu modellierenden System kann man vorgegebene Strukturen auf der Ebene der Organisationsstruktur und Funktionsverteilung, ebenso wie auf der Ebene der direkten Interaktionen – dabei sowohl abstrakte, als auch technische Vorleistungen – benutzen. Diese Möglichkeiten sollen im Folgenden detailliert betrachtet werden. Allerdings sollte man im Auge behalten, daß hier Multiagentenmodelle erstellt werden, also ein gegebenes Originalsystem in ein Modell abgebildet wird.

Konstruktion einer Organisation Ein “natürliches” System besitzt meist eine gegebene Organisationsstruktur und Koordinationsmechanismen, die das Modellsystem widerspiegeln sollte und auf denen beim Zusammensetzen der Agenten zu einer simulierten Gesellschaft aufgebaut werden kann. Daher ergibt sich das Problem der Konstruktion einer Gesellschaft mit einem gewünschten kohärenten Verhalten bei Multiagentenmodellen nicht in dem Ausmaß wie in direkten Anwendungen eines Multiagentensystems.

Bei der Nachbildung einer stark strukturierten Gesellschaft mit gut identifizierbaren Rollen und Funktionen für die einzelnen Agenten können die unter 2.3.1 beschriebenen Methoden den Modellierprozeß gut leiten. Denn für derartige Systeme ist auch ein Top-Down-Vorgehen bei der Modellierung geeignet. Ein Beispiel ist eine hierarchisch organisierte Firma mit ritualisierten Verhandlungen und klarer Verteilung von Kompetenzen. In derartigen Szenarien ist es sogar denkbar, “Musterarchitekturen” für das Agentensystem als Schablone für das Gesamtsystem zu verwenden, wie es E. Horn [Horn & Reinke 1999] für Software-Agentensysteme vorschlägt. Dabei wird vor allem eine “gute” Aufgaben- und Kompetenzverteilung für eine bestimmte Anwendungsdomäne und ein gegebenes Gesamtproblem definiert. Zur konkreten Problemlösung muß man dieses generische Agentensystem instantiieren und konfigurieren. Gerade in der “Computational Organization Theory” beschäftigt man sich mit derartigen Organisationsmodellen menschlicher Gesellschaften und ökonomischer Vereinigungen.

Soziale Gesetze Mit Multiagentensimulation verbindet man allerdings meist eine Bottom-up Vorgehensweise: Man beobachtet und modelliert einzelne Agenten und deren Verhalten in ihrer Umwelt und erzeugt durch die Interaktion der Agenten ein Gesamtsystem. Das Problem dabei ist, daß es bei einem derartigen Vorgehen sehr schwierig sein kann, ohne direkt beobachtbare oder anderweitig bekannte Interaktionen der Agenten ein gewünschtes Globalverhalten zu erreichen. Eine interessante Möglichkeit, die Kohärenz einer Agentengesellschaft ausgehend von den Einzelagenten zu kontrollieren, bilden "soziale Gesetze". Ein bekanntes Beispiel sind die Regeln im Straßenverkehr, "rechts vor links" oder das Constraint, daß im (deutschen) Straßenverkehr per default auf der rechten Seite der Straße gefahren wird. Wenn sich alle Agenten an diese sozialen Regeln halten, gibt es im Prinzip keine Konflikte. Diese werden verringert bzw. vermieden, da diese Constraints die Plangenerierung der einzelnen Agenten beschränken [Shoham & Tennenholtz 1995]. Schon allein, weil weniger explizite Kontrollmechanismen und vor allem keine hierarchische Steuerung notwendig sind und deshalb das System robuster wird, werden derartige Regeln gerne in Multiagentensystemen eingesetzt. Allerdings benötigt man bei der Formulierung der sozialen Gesetze eine zentrale Sicht auf das Gesamtsystem. Ein derartiges Contstraint entspricht einer "von oben" gegebenen Anweisung. Es ist ein nicht-triviales Problem, die passenden Regeln zu finden und sie auf einer derartig lokalen Ebene darzustellen. Deshalb versuchen diverse Forscher, wie z.B. C. Castelfranchi [Castelfranchi 1998], sie formal zu fassen oder wie Y. Shoham und M. Tennenholtz [Shoham & Tennenholtz 1995], die Konstruktion derartiger Gesetze anhand von Verkehrsregeln für Multi-Robot-Szenarien und insbesondere die Komplexität einer automatischen Generierung derartiger Constraints zu untersuchen. Eine Methode, mit der man systematisch auf derartige Regeln kommt, gibt es wohl nicht.

Auch hier ist dies bei der Abbildung eines realen Systems in ein Modell einfacher, da der Modellbauer von möglicherweise beobachtbaren sozialen Gesetzen ausgehen kann. Auf der anderen Seite ist es eine interessante Frage, welche sozialen Reglements zu welchen globalen Mustern in der Agentengesellschaft führen.

Soziale Pläne In bestimmten Anwendungsszenarien kann kooperatives Verhalten mittels mehr oder weniger fest verknüpfter sozialer Pläne bzw. Planschablonen vorgegeben werden. Koordinationsbedarf und die konkreten Möglichkeiten zur Koordination können dabei über Konstrukte wie verteilte Zielsuch-Bäume geschehen, die z.B. in Und-Oder-Graphen formuliert werden können, die über gemeinsame oder konfliktionäre Ziele verknüpft sind [Jennings 1996]. Man könnte ähnliche Strukturen für Agenten in Multiagentenmodellen vorgeben. Mittels dieser kann beschrieben werden, unter welchen Bedingungen mehrere Agenten zusammen bestimmte Aktivitäten ausführen. Dabei werden die Teilaufgaben der einzelnen Agenten spezifiziert und den einzelnen Teilnehmer zugewiesen. Dabei können nicht nur Schablonen für gemeinsam zu lösende Aufgaben sinnvoll sein, sondern auch Konstrukte, die auf einer höheren Abstraktionsebene für Koordination sorgen. Ein Beispiel ist eine Aktivität einer Gruppe von Agenten, nach deren Ausführung diese Agenten in einem bestimmten räumlichen Areal gleichmäßig verteilt sind. Eine Schablone für derartige gemeinsame Pläne aufzustellen und diese zu interpretieren ist dabei wohl ein weniger schwieriges Problem, da der Mensch, der einen sozialen Plan aufstellt, eine globale Sichtweise besitzt. Allerdings ist es alles andere als trivial, diesen Rahmen dann in einem Modell zu füllen, indem man in einer Situation die globalen Ziele auf einzelne Aktionen abbildet, dabei allerdings möglicherweise nicht einmal eine konstante Menge von Agenten zur Verfügung hat.

Man könnte derartige Aktivitäten auch als Aktionen eines "Super-Organismus" verstehen. Die Idee des hierarchischen Modellierens ist in Simulationssystemen und -sprachen weit verbreitet (siehe dazu Abschnitt 4.4.1). Bei Multiagentenmodellen könnte man sich eine derartige Struktur wie folgt vorstellen: Eine Gruppe wird explizit als (Quasi-)Agent repräsentiert. Die Wahrnehmungen dieser Einheit setzt sich aus den Wahrnehmungen der primitiven Agenten zusammen, die Aktionen werden als soziale Pläne analog auf die einzelnen Agenten aufgelöst. Aber auch hier muß zwischen der Beschreibung des Superorganismus und den einzelnen primitiven Agenten ein Perspektivenwechsel des Modellierers stattfinden.

Interaktionsprotokolle Auch auf einer abstrakteren Ebene kann die Konzeptionalisierung und Implementierung von Interaktionen unterstützt werden. Es gibt mittlerweile generische Interaktionsmuster für bestimmte Verhandlungsprotokolle. Das am weitesten verbreitete ist das *Contract Net Protocol* [Smith 1980], mit dem einfache Vertragsverhandlungen zur Verteilung von Aufgaben auf Agenten über Anbieter - Nachfrager - Rollen nachgebildet werden können: Ein "Manager" zerlegt ein Problem in Teilprobleme und schreibt die Erfüllung dieser Teilprobleme aus. Agenten, die diese Aufgaben erfüllen können, bewerben sich entsprechend ihrer individuellen Ressourcensituation. Der Manager vergibt die Aufgabe an den besten Bewerber. Als Erweiterungen dieses einfachen Schemas sind beispielsweise mehrstufige Contract Nets möglich. Ähnliche Protokollmuster gibt es für andere Verhandlungssituationen. Auf der Basis von Modellierungssprachen für Koordinationsprozesse, wie beispielsweise *COOL* [Barbuceanu & Fox 1997], kann man derartige Verhandlungsmuster erzeugen und untersuchen. *COOL* basiert, wie andere derartige Modellierungshilfen, auf Formalismen wie Zustandsautomaten, mit dem der Zustand eines Agenten und seine Übergänge basierend auf Nachrichten von anderen Agenten repräsentiert und analysiert werden.

Kommunikation Auch auf der Kommunikationsebene kann man vielseitige Hilfen bereitstellen. Dies kann auf zwei Ebenen geschehen, einerseits auf einem relativ abstrakten Niveau mit Interaktionsprotokollen, andererseits auf der technischen Seite durch Kommunikationsmittel, die ein transparentes Schicken und Empfangen von Nachrichten ermöglichen.

Gerade eine standardisierte Kommunikation zwischen den Agenten ist ein Schlüsselkonzept für ein offenes System. Arbeiten mehrere Modellbauer jeweils an verschiedenen Aspekten eines gemeinsamen Agentensystems, dann ermöglicht es nur ein konsequentes Benutzen der definierten Schnittstellen, daß ein korrekt funktionierendes Gesamtsystem entstehen kann. Wenn ein Agent keine Information aus seiner Umwelt beziehen und diese nicht verändern kann, wird eine Interaktion unmöglich. Werkzeuge, mit denen eine Kommunikation überhaupt ermöglicht werden kann oder die Unterstützung in Form von Konzepten, (Quasi-)Standards und darauf aufbauenden Tools bieten, kann man auf drei Ebenen finden: die der Infrastruktur, der Übertragungsprotokolle und der Kommunikationssprachen. Diese drei Ebenen werden im folgenden kurz besprochen.

- Auf der Ebene der "High Level" Infrastruktur werden Dienstleistungen, wie beispielsweise Adressen-Server, angeboten. Diese spielen eher die Rolle einer Hilfsstruktur, um ein Ansprechen anderer Agenten zu erleichtern oder, wie beispielsweise ein "yellow page server", öffentliche Kompetenzinformation zu verwalten. Zu derartigen Tools gehören auch Systeme, die eine Blackboard-Architektur unterstützen. Auf einem der-

artigen gemeinsamen Speicherbereich kann ungerichtet Information geschrieben werden. Gerade im kooperativen Problemlösen sind Agentensystemarchitekturen mit einer Blackboard weit verbreitet (für grundlegende Papiere siehe [Bond & Gasser 1988b]).

- Die eher technische Ebene der Übertragungsprotokolle sichert eine vollständige und korrekte Übertragung der Nachrichten. Derartige Protokolle werden als vorhanden und funktionierend vorausgesetzt. Dies ist ein wichtiger Punkt, denn alleine der Modellierer sollte in seinem Umweltmodell bestimmen, ob und wann Nachrichten nicht korrekt übertragen werden.

- Die Ebene, auf der zur Zeit die meisten Standardisierungsbestrebungen zu beobachten sind, ist die der "Agent Communication Languages" (ACLs), also der Formate und Inhalte der Nachrichten, die zwischen Agenten ausgetauscht werden. Mit *KQML* ("Knowlegde Query and Manipulation Language") [Finin et al. 1997] etablierte sich mittlerweile ein fast schon standardisiertes Format für Nachrichten zwischen Agenten. So wurden auch Systeme entwickeln, die eine für *KQML*-Kommunikation notwendige Infrastruktur bereitstellen, wie zum Beispiel *Jackal* (Java-based tool for communicating with the KQML ACL) [Cost et al. 1999][3].

  Die in KQML möglichen Nachrichtentypen basieren auf Sprechakten und bilden ursprünglich den Rahmen für den Austausch von Wissen. Über den Typ der Nachricht wird die Intention, die hinter der Nachricht steckt, formuliert, z.B. `ASK-ONE` als direkte, adressierte Anfrage. An dieser Agenten-Kommunikationssprache ist besonders interessant, daß mit jeder Nachricht sowohl Information über die Sprache, in der der Inhalt formuliert wurde, z.B. daß es sich um ein Prolog-Statement handelt, als auch über die den verwendeten Begriffen zugrundeliegende Ontologie mitgeschickt werden.

Eine gemeinsame Ontologie ist eine der wichtigsten Voraussetzungen für eine erfolgreiche Kommunikation zwischen mehreren Agenten. Nur so besitzen die verschickten Begriffe für alle Beteiligten die gleiche Bedeutung. Bei Multiagentensimulationen könnte man allerdings meinen, daß mit der Umwelt ein Modellierer auch die Kommunikationsinhalte kontrolliert. Allerdings kann ein "Verstehen" bei Multiagentenmodellen auch problematisch sein, insbesondere wenn mehrere Modellierer unterschiedliche Agenten konstruieren ohne sich vorher auf eine gemeinsame Ontologie geeinigt zu haben. Eine andere Problemquelle kann eine langwierige, iterative Modellentwicklung sein, wenn sich die modellierte Bedeutung der Begriffe im Lauf der Entwicklungszeit ändert, ohne daß ältere Agenten angepasst werden. Aber schon alleine dadurch, daß eine simulierte Umwelt meist deutlich einfacher als die zu modellierende ist, dürfte auch die von den Agenten benutzte Ontologie weniger komplex sein.

4.3.2.3 Konstruktion des Umweltverhaltens

Die Repräsentation der Bestandteile und Dynamik der nicht in Form von Agenten dargestellten Umwelt ist an sich nicht an die Entwicklungsmethodik für Agentensysteme gebunden. Eine wichtige Voraussetzung ist aber, daß die Schnittstellen zwischen Agenten und Umwelt klar sind. Allerdings kann ein Modellierer, wenn er die Repräsentationsformen für

[3] Für weitere Tools siehe [UMBC 2000]

die Umweltdynamik völlig von denen für die Agenten und ihren Interaktionen trennt, nur wenig durch ein Rahmenwerk unterstützt werden. Hier kann die Strategie zum Tragen kommen, daß die Dynamik und Konfiguration der Umwelt in einem speziellen Umwelt-Agenten realisiert und in das Agentensystem integriert wird. Auf diese Weise kann man die Modellierhilfen und Interaktionsmechanismen, die für die eigentlichen Agenten bereitgestellt wurden, wiederverwenden. Dies kann aber bedeuten, daß die Agenten mit ihrer Umwelt mittels *KQML*-Nachrichten oder anderweitigen Standardsprachen kommunizieren müssen.

Das andere Extrem findet man, wie in Abschnitt 4.1.5 angedeutet, wenn die Umwelt mit hohem Aufwand sehr detailliert repräsentiert wird. Dabei ist meist die Umwelt vollständig in den Simulator integriert (siehe z.B. [Atkin et al. 1998]). Der generische Aspekt des Multiagentenmodells liegt dann vor allem beim Agentensystem. Demzufolge gibt es bei der Konstruktion und Implementierung der Umwelt keine explizite Unterstützung.

4.4 Existierende Softwaresysteme

Bevor das Kapitel mit einem vollständigen Überblick über existierende Systeme zur Simulation von Multiagenten endet, werden zunächst Methoden, Sprachen und Systeme vorgestellt, die speziell für Multiagentenmodelle entwickelt wurden.

4.4.1 Spezifikations- und Modelliersprachen aus der Simulationstechnik

Die Möglichkeiten, die sich aus der Konzeptionalisierung und Modellierung eines System als Multiagentensystem ergeben, wurden auch von Forschern aus der Simulationstechnik erkannt. Bei dem dabei verwendeten Konzept der Multiagentensimulation wird im Unterschied zu Techniken des agenten-orientierten Software-Engineering seltener einem Top-Down Entwurfsmuster gefolgt. Demzufolge ist die generelle Vorgehensweise mit dem Identifizieren einer Aufgabenhierarchie und Zuweisen der einzelnen Aufgaben zu Agenten nicht bei jeder Simulationsanwendung einsetzbar. Die Modellierung von Kooperationsmustern bildet aber auch hier eine wichtige Teilaufgabe bei der Entwicklung einer Multiagentensimulation. Wenn allerdings ein existierendes System, bzw. ein System mit entsprechend beobachtbaren oder hypothetischen Vorgaben für die Interaktionmuster nachgebildet werden soll, ist deren Formulierung deutlich leichter. Die Aufgabe diese Muster darstellen ist also hier weniger ein Konstruktionsproblem, als ein Abstraktionsproblem. Ein weiterer Unterschied zwischen Entwicklungsmethoden für agentenbasierte Problemlösungen und Multiagentensimulationen liegt in einer stärkeren Fokussierung auf die Modellierung einer Umwelt für das simulierte System, denn insbesondere mittels Variationen dieser Umwelt können systematische Experimente durchgeführt werden

Wie bei der allgemeinen Agenten-orienten Software-Entwicklung gibt es auch hier unterschiedliche Ansatzpunkte, die vor allem von der Basis des jeweiligen Modellierungsrahmens geprägt sind. Im Folgenden sollen Beispiele vorgestellt werden, die auf formalen Grundlagen aus der Simulationstechnik beruhen: Petri-Netze und Spezifikationen, die auf einer Erweitertung der Definitionen dynamischer Systeme beruhen und deren Implementierung objekt-orientiert vorgenommen wird.

4.4.1.1 Simulation interagierender Prozesse

Die Modellierung von Prozessen mittels Agenten ermöglicht auch bei der prozeß-orientierten Simulation, komplexe Vorgänge darzustellen. So können nicht nur simulierte Maschinen, sondern auch modellierte Menschen miteinander interagieren, um so Simulationen mit detailierten Modellen durchzuführen. Obwohl prozeß-orientierte Modellierungsparadigmen bis zu einem gewissen Grad als Grundlage für komplexere Verhaltensprozesse – durch die Agenten dargestellt werden können – dienen könnten, findet man nur wenig Informationen über Ansätze, die in Richtung einer speziellen agentenbasierten Simulation zeigen würden. Das kann auch daran liegen, daß Multiagentensimulation auf der Basis seiner Ursprünge in der Künstlichen Intelligenz ausschließlich mit regel- oder logikbasierten, rein qualitativen Repräsentationsformen identifiziert wird (wie beispielsweise in [Möhring 1994]).

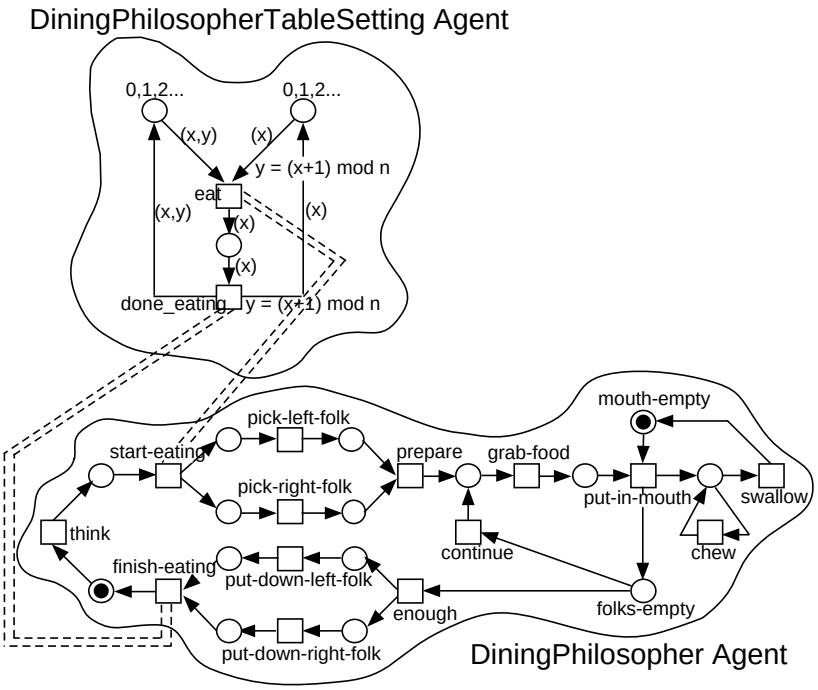
Petri-Netze bieten eine einfache, anschauliche, aber zugleich formale Methode, komplexe Prozesse zu spezifizieren [Baumgarten 1990]. Gerade bei nebenläufigen, asynchronen Prozessen werden sie gerne als Modellierungsbasis verwendet [Bernardinello & Cindio 1992, Monsef 1997]. Ein Petri-Netz in seiner ursprünglichen Form ist ein bi-partiter, gerichteter und (schwach) zusammenhängender Graph mit zwei unterschiedlichen Knotentypen, die passive und aktive Systemkomponenten repräsentieren. Die passiven “Stellen” (durch Kreise dargestellt) repräsentieren (lokale) Zustände des Systems, die während des Ablaufs eines Systems erfüllt sein können. “Transitionen” (durch Kästchen oder vertikale Linien visualisiert) stellen Aktionen oder Ereignisse dar, für deren Ausführung bzw. Eintreten die Vorgänger-Zustände Vorraussetzung sind und nach deren Auftreten ihre Nachfolger-Zustände erreicht werden. Petri-Netze gibt es in verschiedenen Variationen und Erweiterungen. Die prominenteste ist das “Condition / Event”-System, das die obige Definition um eine Token-Menge erweitert[4]. Ein Token in einer Stelle repräsentiert das Erfülltsein einer Bedingung für das “Feuern” der Transition. Eine andere Bedingung ist, daß an den Nachfolger-Stellen keine Token liegen. Beim “Feuern” werden ein Token von den Vorgängerstellen zu den Nachfolgerstellen transferiert. Es gibt Ein-Token-Netze – also Netze, in denen die Zahl der Token konstant und beschränkt ist, aber auch Netze mit beliebig vielen Tokens. Als weitere Variation gibt es “Coloured Petri Nets” mit unterschiedlichen Token-Farben für jeweils andere Kategorien an Bedingungen, usw. (siehe [Bernardinello & Cindio 1992] für umfassende formale Definitionen)

T. Holvoet [Holvoet 1995] versucht, ein Agentensystem mittels Petri-Netzen zu modellieren. In Abbildung 4.4 wird das als Beispiel verwendete “Dining-Philosopher-Problem” dargestellt. Transitionen (Kästchen) repräsentieren die Aktionen, Stellen (Kreise) die Situation des jeweiligen Agenten. Über ein weiteres Netz (spezielle Kanten) werden die einzelnen Agenten koordiniert. Andere Modellierungen dieses Standard-Problems auf der Basis von Petri-Netzen sind auch ohne Agentensicht möglich und dabei deutlich einfacher [Fishwick 1995].

Trotz diverser Vorteile, wie z.B. graph-theoretischer Analysemöglichkeiten, bietet eine derartige Modellierung nicht die Flexibilität, die eigentlich für die Formalisierung eines Agentensystems notwendig wäre. Das liegt nur zum Teil an den fixen Verbindungen zwischen den Transitionen, die eine variable Struktur nicht unterstützen. Schwerwiegender sind die Defizite bei der internen Beschreibung des Agenten. Es lassen sich weder Attribute noch Sensoren oder Effektoren eines Agenten explizit repräsentieren.

[4] Von manchen Autoren, z.B. in [Fishwick 1995] werden Condition / Event-Systeme ohne weiteren Zwischenschritt als Petri-Netze eingeführt.

Abbildung 4.4 Darstellung des "Dining Philosopher" Szenarios auf der Basis von verknüpften Petri-Netzen (nach [Holvoet 1995]). Das untere Netz spezifiziert das Verhalten eines Philosophen – es können mehrere davon im selben Agentensystem existieren. Das obere Netz steuert über die gestrichelten Linien Aktionen, die mit anderen Agenten (angedeutet durch 0,1,2 ... an Kästchen des Koordinationsnetzes) kollidieren würden.



Es sind andere Möglichkeiten vorstellbar, ein Agentensystem als Petri-Netz darzustellen, die die Nebenläufigkeiten und die regelbasierte Beschreibungform mittels Tokens besser ausnutzen. Allerdings kranken diese Ansätze ebenfalls an der fixen, rein verhaltensbeschreibenden Struktur.

Die Modellierung von Prozessen, die jeweils das Verhalten eines Agenten beschreiben, bildet einen Schwerpunkt in der sozialwissenschaftlicher Simulation. Dabei liegt der Fokus auf Kooperation, Synchronisation und Interaktion in Organisationen. Insbesondere in der Arbeitsgruppe von Prof. K. Troitzsch werden dabei auch individuenbasierte Simulationsmodelle sozialer Prozesse entwickelt. Dazu soll mit der *Massif*-Umgebung [Mentges 1999] ein besonderer Rahmen geschaffen werden, in dem Modellierung und Simulation unterstützt werden, In dieser Umgebung werden verschiedene Sprachen, z.B. UML, für jeweils unterschiedliche Aspekte verwendet. Als Agentenarchiktur wird, wie auf Seite 22 angedeutet, *InteRRaP* verwendet.

4.4.1.2 Agenten- und objekt-orientierte Simulation

Ebenso wie agenten-orientierte Software-Entwicklung als Erweiterung von objekt-orientierten Methoden aufgefasst werden kann, werden auch von Experten der Simulationtechiken Formen der agenten-orientierte Simulation einwickelt. Die Agentensicht soll eine Forma-

lisierung komplexerer Modelle erleichtern, indem durch die höhere Abstraktionsebene bei der Konzeptionalisierung die Beschreibung und Formalsierung natürlicher Agentensysteme auf einer detaillierten Ebene miteinbezogen werden kann. kann.

Im Folgenden werden die zwei bekanntesten Versuche, objekt-orientierte Simulation dahingehend zu erweitern, daß Multiagentenmodelle in jeweils existierenden Simulationsrahmen dargestellt werden können, vorgestellt. Auf diese Weise kann man das beträchtliche simulationstechnische Know-How und die umfangreichen Werkzeuge, mit dem die generellen Umgebungen ausgestattet sind, weiter nutzen. Im folgenden werden als Beispiele die Referenzarchitektur *PECS* und die Erweiterung der DEVS-Spezifikation *AgedDEVS* erläutert.

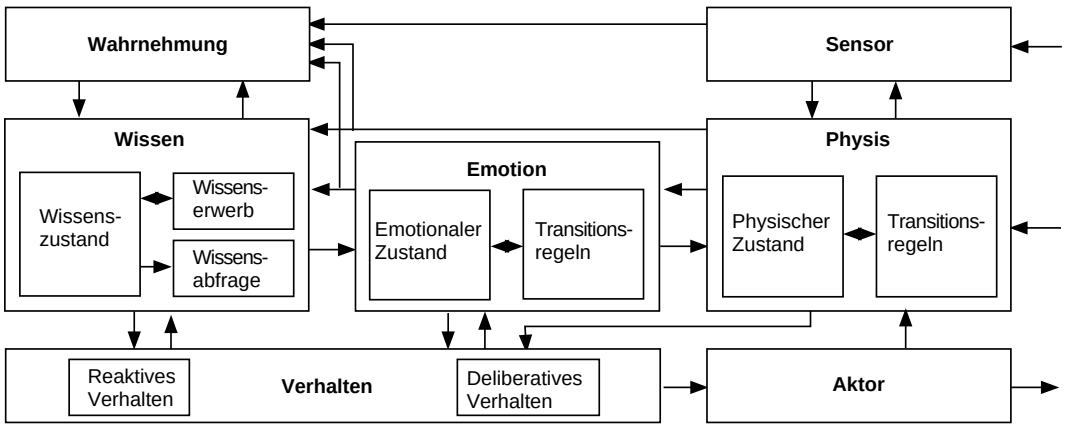
Ausgangspunkt des Referenzmodells für simulierte Agenten von C. Urban [Urban 2000] ist die Überlegung, daß mit der Bereitstellung einer Standard-Architektur für einen Agenten die Einbindung von simulierten Agentensystemen in vorhandene Simulationsumgebungen ermöglicht wird. Da bevorzugt menschliche Akteure mit dieser Referenzarchitektur dargestellt werden sollen, orientiert sich diese an den psychologischen Modellen zur menschlichen Verhaltenssteuerung von D. Dörner [Dörner 1999].

Eine *PECS*-Agentenwelt (Physis, Emotion, Cognition, Social status) besteht aus drei Bestandteilen:

- Den Agenten, die auf der Basis dieser Architektur modelliert werden,
- einer “Konnektor”-Komponente, über die der Informationaustausch der Agenten organisiert wird und
- eine Umwelt, mittels derer alle externen Faktoren modelliert werden.

Die Struktur der *PECS*-Agentenarchitektur wird in Abbildung 4.5 dargestellt.

Abbildung 4.5 Schema eines *PECS*-Agenten [Urban 1997]



Auf diese Weise wird ein Agent als ein System, wie es in der Systemtheorie formal definiert ist (siehe Seite 39), aufgefaßt und dabei die Mengen der Eingabe-, Zustands- und Ausgabevariablen detallierter strukturiert. Die Eingabevariablen werden in Sensor- und Wahrnehmungsmenge, die Menge der interne Variablen in die Physis, Emotion, Kognition und Status und die Ausgabemenge in Verhaltens- und Aktormenge unterteilt. Die Pfeile stehen für Abbildungen zwischen den jeweiligen Mengen. Welcher Art diese Abbildungen sind,

hängt vom Wertebereich der einzelnen Komponente ab. Für ein konkretes Modell müssen neben den Abbildungen auch diese Wertebereiche spezifiziert werden. Vorgaben über die internen Strukturen — die konkreten Wissensrepräsentationen und Verarbeitungsschemata — werden dafür eigentlich nur für die Komponente "Verhalten" gemacht, indem für reaktive Verhaltensmuster eine Skript-Sprache unterstützt wird. Für Kommunikationsaktionen wird eine (proprietäre) Nachrichtensyntax vorgebenen, die vom Konnektor interpretiert und weitergegeben werden kann. Diese Referenzarchitektur wurde auf der Basis von *SIMPLEX II*, einer allgemeinen, objekt-orientierten Simulationsumgebung implementiert und bei verschiedenen, einfachen Fallstudien eingesetzt [Urban 1997, Urban 2000].

Ausgangspunkt der Entwicklung von *AgedDEVS*[5] [Uhrmacher 1996] war das Problem, Modelle mit variablen Strukturen elegant zu formulieren. Darunter sind Modelle mit einer dynamischen Anzahl von Bestandteilen und variablen Wechselwirkungen zu verstehen. Gerade in biologischen Anwendungen ist eine derartige Variabilität unerläßlich.

Dieses Problem wurde mit agenten-orientierter Simulation gelöst, indem ein Agent zusätzlich zu den bei *DEVS* vorhandenen Schnittstellen und Sub-Komponenten Repräsentationen seiner eigenen Einbindung in die simulierte Welt in Form eines internen Modells erhält. Auf diese Weise kann der Agent die Beziehungen zu anderen Agenten manipulieren, bzw. bei seinem Tod erfahren dies die mit ihm verbundenen Agenten und können darauf reagieren.

Wie bei *PECS* werden hier nur formale Mengenstrukturen vorgegeben, keine deklarative Beschreibungssprachen oder Inferenzmechanismen. Auch für *AgedDEVS* gibt es eine Implementierung einer Modellierumgebung, die die Umsetzung und das Testen von derartig spezifizierten Modellen erleichtert [Uhrmacher & Schattenberg 1998].

4.4.1.3 Deklarative und regelbasierte Simulationsansätze

Ausgehend von der Idee, kognitive Agenten und ihre Interaktionen zu simulieren, wurden verschiedene Simulationsrahmen für Untersuchungen in der Psychologie, Soziologie und der Organisationstheorie entwickelt. Die möglichen Ausprägungen bewegen sich dabei von "Testbed"-ähnlichen Systemen, bei denen verschiedene Formen eines Agentensystems nur über unterschiedliche Konfigurationen gesteuert werden können, bis hin zu regelbasierten Programmiersprachen. Letzere werden zum einen wegen ihrer Beziehung zu kognitionswissenschaftlichen Modellen (*Soar*), aber auch wegen guter Verhaltensbeschreibungsmöglichkeiten benutzt.

Um den traditionellen regelbasierten Modellierungsrahmen so zu erweitern, daß das Verhalten eines Multiagentensystems beschrieben werden kann, benötigt man die zwei folgenden Ingredenzien: Man muß die einzelnen Agenten separat beschreiben können, d.h. ihnen werden zumindest getrennte Fakten- und Regelbanken zugewiesen; dazu kommen Schnittstellen, mit denen ein Agent mit anderen und seiner Umwelt kommuniziert.

SDML ("Strictly Declarative Modelling Language"), die am "Centre for Policy Modelling" der Universität Manchester speziell für die Multiagentenmodellierung von Organisationen entwickelt [Moss et al. 1997] wurde, behandelt diese Aspekte wie folgt: Ein Agent ist ein Objekt mit einer Regelbasis und einer Datenbank mit Fakten, die für ihn wahr sind.

[5] Agent-oriented DEVS: *DEVS*[Zeigler 1984] steht für Discrete Event Specification. Das ist eine von B. Zeigler entwickelte, ebenfalls stark an der formalen Systemdefinition angelehnte Spezifikation ereignisbasierter, hierarchisch strukturierter Modelle

Durch das Feuern der Regeln, die sowohl vorwärtsverkettet als auch rückwärtsverkettet behandelt werden können, werden neue Fakten in die Datenbanken geschrieben oder "Prozeduren nachgebildet". Die Datenbank eines Agenten kann privat, öffentlich oder intern sein. Die Kommunikation zwischen den Agenten erfolgt über das Schreiben in und Lesen aus den Datenbanken anderer Agenten. Durch spezielle Mechanismen mit Regelbehandlungen auf verschiedenen Zeitebenen und unterschiedliche Kombinationen von Regelmengen können Agenten zu Organisationen zusammengefasst werden. Für eben die Modellierung von Organisationen sind diese Typen von zusammengesetzten Agenten interessant.

S. Moss et al. konnten mit ihrer regelbasierten Modellierungssprache die *Soar*-Architektur und darauf aufbauend das *Plural-Soar*-Modell nachbauen [Moss et al. 1997]. Also sollte man beachten, daß eine deklarative Modellierung nicht in jedem Fall automatisch mit höheren Sprachkonzepten gleichgesetzt werden kann.

Auch bei anderen soziologisch-psychologisch motivierten Modellen, wie dem EOS-Projekt [Doran & Palmer 1995], wurden regelbasierte Multiagentensystem-Testbeds und parallele Prolog-Varianten entwickelt und eingesetzt. Die kognitiven Verhaltensgrundlagen, die Dynamik lokaler interner Modelle relativ zu global zu beobachteten Makro-Strukturen bildet den Schwerpunkt derartiger Simulationen und dementsprechend auch den Fokus der Modellierumgebungen.

Gerade für die Nachbildung von Organisationen, von Teamwork usw. sind mittlerweile *Soar*-basierte Simulationen relativ weit verbreitet [Prietula et al. 1998]. Da *Soar* auf einer etablierten kognitionswissensschaftlichen Theorie beruht, geht man davon aus, daß so zumindest die verwendete kognitive Architekturgrundlage valide ist [Carley & Newell 1994]. Eine kurze Beschreibung der *Soar*-Architektur ist auf Seite 24 dargestellt. Beschreibungen von Beispielen findet man im Abschnitt 4.1.1.3.

4.4.2 Übersicht über existierende Systeme zur Multiagentensimulation

Bisher wurden in der vorliegenden Arbeit über diverse Abschnitte verteilt, Systeme besprochen, die für die Entwicklung von Multiagentensimulation konzipiert oder verwendbar sind. Ein zusammenfassender Überblick soll in diesem Abschnitt gegeben werden. Die möglichen Systeme sind dabei vielfältig und in hoher Anzahl verfügbar, weshalb ich mich hier auf eine Auswahl von für diesen Anwendungsbereich wirklich relevanter Werkzeuge konzentrieren möchte

Für Darstellungen allgemeiner Simulationssprachen oder Spezialsimulatoren, die nicht explizit dem Multiagentenparadigma verbunden sind, sei auf die einschlägige Literatur verwiesen[6]. Ebenso wenig sollen (kommerzielle) Werkzeuge für die Anwendungsbereiche intelligenter, agentenbasierter Benutzerschnittstellen, oder für mobile Agenten behandelt werden[7]. Auch Infrastruktur-Module zur Kommunikation oder Verhandlung, wie beispielweise die sogenannte "Middleware" passen meiner Ansicht nach nicht in die folgende Auflistung. Sie können zwar durch Bereitstellung von höherer Kommunikationsinfrastruktur durchaus die Grundlage für verteilte Modelle mit wenigen, aber komplexen Agenten bilden. Allerdings ist der damit verbundene "Overhead" der reinen Infrastruktur-Verwaltung für Modelle mit mehreren, einfacheren Agenten zu groß, als daß der Einsatz sinnvoll wäre.

[6] Beispielsweise in den "Simulation News Europe", mit umfangreichen Vergleichen von Simulationswerkzeugen oder den Proceedings zu den großen Simulationskonferenzen der SCS

[7] Siehe "Agent Contruction Tools" `http://www.agentbuilder.com/AgentTools/` für eine Auflistung gerade kommerziell und über Forschungseinrichtungen verfügbarer Werkzeuge.

4.4.2.1 Ansatzpunkte einer Übersicht

Es gibt diverse Ansätze für Klassifikationen vorhandener Softwaresysteme für die Modellierung von Multiagentensystemen und deren Simulation. J. Kulijs [Kuljis 1994] als Simulationstechnikerin unterscheidet grundsätzlich zwischen Simulationssprachen und Simulatoren: Während man bei Simulatoren für eine beschränkte Klasse von Systemen graphische Benuteroberflächen zur Modellierung, Animation usw. zur Verfügung haben kann, muß man bei Simulationssprachen diese oftmals selbst programmieren. Letzere sind Aufsätze auf traditionelle Programmiersprachen und dementsprechend flexibel bei den modellierbaren Strukturen.

In der Verteilten Künstlichen Intelligenz findet man weitergehende Klassifikationen als Rahmen für Übersichten. K. Decker [Decker 1996] subsummiert alle Systeme unter “Testbeds”. In [Kraetzschmar & Reinema 1993] wird dagegen von G. Kraetzschmar und R. Reinema nach Experimentier- und Entwicklungsumgebungen, Softwarekomponenten, sowie Programmiersprachen und -umgebungen differentiert, abhängig davon, welche Teile des Entwicklungsprozesses durch das jeweilige Werkzeug unterstützt werden. Diese Einteilung ist an sich sehr hilfreich, leider kann sie nicht immer konsequent angewendet werden, da viele Systeme in mehrere Kategorien eingeordnet werden müßten. Gerade bei Entwicklungssystemen, Test- und Programmierumgebungen kann man viele Überscheidungen feststellen.

Obwohl also eine konsequente Klassifikation von Simulationssystemen für Multiagentenmodelle nicht identifizierbar ist, soll im folgenden Abschnitt dennoch wenigstens ansatzweise versucht werden, eine Ordnung zu vermitteln. Diese orientiert sich an der obigen Einteilung nach G. Kraetzschmar und R. Reinema orientiert.

4.4.2.2 Stand der Forschung

Selbst mit den oben gemachten Einschränkungen gibt es wohl Hunderte von Werkzeugen, die eigentlich in dieser Aufstellung auftauchen sollten. Es scheint fast so, als würde jedes Projekt für die Entwicklung eines agentenbasierten Systems dadurch gekrönt, daß die dabei entwickelten Klassenbibliotheken als “Toolkit” bezeichnet und allgemein verfügbar gemacht würden. Software-Komponenten sind in diesem Zusammenhang uninteressant, da sie nur die Entwicklung eines Teilsystems durch Vorgabe von Methoden, beispielsweise zur verteilten Begründungsverwaltung, unterstützen. Andere Werkzeuge, die eindeutig als weitgehend vorgegebene Testumgebungen identifizierbar sind, sollen im nächsten Abschnitt behandelt werden. Danach folgt ein kurzer Abschnitt zu relativ niedrigen Programmiersprachen der Verteilten Künstlichen Intelligenz. Danach folgt eine umfassende tabellarische Aufstellung der hier relevanten Entwicklungs-Tools.

Testumgebungen Eine vollständige Aufstellung müßte konfigurierbare oder generische Testumgebungen, wie *Phoenix* [Cohen et al. 1989] oder *Tileworld* [Pollack & Ringuette 1990] umfassen. Sie sind mit Spezialsimulatoren vergleichbar und bieten eine weitgehende Unterstützung ihrer Benutzer, die allerdings meist wenig mehr als die Modellkonfiguration manipulieren dürfen. Demzufolge besitzen diese Testumgebungen durch die vorgegebenen Strukturen von Umweltverhalten oder Agentenformen nur wenig Flexibilität. Bei dem ersten der Beispiele, *Phoenix*, können Multiagentenplaner im detailliert nachgebildeten Yellowstone Nationalpark Pläne zum Einsatz von Feuerlöscheinheiten bei Waldbränden testen. Das Waldbrandmodell ist ein Zellulärer Automat und kann auch auf andere Katastrophens-

zenarien, wie Havarien von Öltanker übertragen werden [Cohen et al. 1989]. Ein ähnliches Ziel verfolgt *Tileword* [Pollack & Ringuette 1990], mit dem ein abstraktes Testszenario als dynamischer Blocksworld-Ersatz geschaffen wurde. Aktuellere Testbeds sind *Echo* [Hraber et al. 1997], bei dem fast vollständig vorgegebene, über Genetische Algorithmen optimierbare Agenten auf einer zweidimensionalen Karte interagieren, indem sie bestimmte Ressourcen austauschen, Nachkommen produzieren, usw. Auch für organisationstheoretische Simulationen gibt es Systeme, wie das *Virtual Design Team VDT* [Levitt et al. 1994]. Das ist ein Testbed, bei dem in vorgegebenen Organisationsstrukturen Kommunikationsmittel und Aufgaben konfiguriert und über Experimente Ergebnisse zu Bearbeitungsdauer usw. erhalten werden können.

Programmiersprachen Das andere Extrem einer völligen Flexibilität bieten Programmiersprachen, die keinerlei strukturelle Vorgaben, weder für die Repräsentation der Umwelt noch bei der Agentenarchitektur machen. Eigentlich müßte man in der folgenden Aufstellung auch Aktor-Sprachen aufnehmen [Agha & Jamali 1999]. Sprachen für verteilte objektorientierte Systeme, parallele Prolog-Varianten, u. s. w. könnten eine Basis für die Implementierung von Multiagentensimulationen bieten. Da der Schwerpunkt dieser Arbeit allerdings auf der Unterstützung der Modellierung und Simulation für nicht programmierwillige Domänenexperten liegt, sollen auch derartige Programmiersprachen weitestgehend ignoriert werden, solange sie nicht mit einer passenden Entwicklungsumgebung verbunden sind oder ein ausreichend hohes Sprachniveau besitzen.

Übersicht über Entwicklungsumgebungen In der folgenden Tabelle werden Entwicklungs- und Simulationssysteme aus verschiedenen Bereichen untersucht, die für Multiagentensimulation benutzbar wären oder dafür entwickelt wurden. Dabei findet sich in der ersten Spalte der Name und die Referenz eines repräsentativen Systems, das in der zweiten Spalte kurz charakterisiert wird. Dabei werden neben den verfügbaren Werkzeugen auch vergleichbare Systeme aufgelistet. In den beiden anderen Spalten werden die beiden Ebenen, in denen Vorgaben bei der Repräsentation eines Multiagentenmodells besonders wichtig sind, behandelt: Bietet das System eine vorgegebenen Agentenarchitektur? Wie muß die Umwelt repräsentiert werden? Dabei bedeutet kein Eintrag, daß die Umwelt als weiterer, “normaler” Agent modelliert werden muß.

Die Systeme sind dabei nicht alphabetisch, sondern nach der Art des Systems sortiert. Eine klare Unterteilung ist dabei, wie oben angedeutet, nicht möglich. Dennoch wurde versucht, die Vorgehensweise nachvollziehbar zu gestalten: Die Tabelle beginnt mit visuellen Modellierumgebungen, also Systemen, bei denen der Prozess der Beschreibung des Verhaltens ein wichtiges Ziel ist, z.B. wegen der Ausrichtung der edukativen Simulation. Auch bei formalen Spezifikationsrahmen steht die Beschreibung von Verhalten im Vordergrund, deswegen folgen darauf derartige Systeme. Danach folgt ein umfangreicher Block von Entwicklungsumgebungen für Multiagentensysteme und Simulatoren, die von einer mehr oder weniger formalen Spezifikation ausgehen. Bei diesen soll die “professionelle” Erstellung eines komplexen Softwaresystems unterstützt werden. Danach werden Systeme mit mehr und mehr vorgegebenen Agentenarchitekturen beschrieben, von regelbasierten bis zu aktivitätsbasierten Systemen. Die Tabelle endet mit Systemen, die in der Dimension der vorgegebenen Umweltstrukturen immer weiter festgelegt sind, also eher generischen Testbeds entsprechen.

In den nächsten Kapiteln wird die Multiagentensimulationsumgebung SESAM vorgestellt, die in dieser Aufstellung nicht aufgelistet ist. Man könnte sie aber entsprechend wie folgt charakterisieren: SESAM ist eine Entwicklungsumgebung für Multiagentenssimulationen, die basierend auf regelbasierter Verhaltensbeschreibung, mit graphischer Modellieroberfläche und Animations- und Auswertungstools ausgerüstet wurde. Die Agentenarchitektur ist regelbasiert und hochgradig strukturierbar. Die Umweltrepräsentation ist explizit zwei-dimensional (diskret oder kontinuierlich), durch Kopplung dieser Karten kann eine dritte Dimension angedeutet werden.

System	Beschreibung	Agentenarchitektur	Umwelt
Visuelle Systeme			
AgentSheets [Repenning & Summner 1994] [Repenning 2000]	visuelle Programmierumgebung für räumlich explizite Systeme, Automatische Generierung von JAVA-Applets möglich, keine Auswertungstools. siehe auch *Cocoa*[Cypher 1998]	unstrukturierte Mengen erweiteter graphischer Ersetzungregeln	2D-Grid
LiveWorld [Travers 1996]	regelbasierte "Agenten" werden in schachtelbaren Editoren zu Szenen zusammengesetzt. Kindgerecht, Konstrukte auf rel. niedrigem Abstraktionsniveau siehe *Playground*[Fenton & Beck 1989]	gegebene Agentenbausteine, Sensoren . . . können graphisch zu Netz verknüpft werden.	2D-kont.
Spezifikations-Tools			
DESIRE [Gavrila & Treur 1994]	Formales Rahmenwerk für zusammengesetzte, hierarchisch organisierte Systeme. Deklarative Spezifikation mit Codegenerator. Formale Rahmenwerke, wie *Concurrent*METATEM werden in Abschnitt 2.3.3 beschrieben	Allgemeine Organisation als "compositional system" mit Verknüpfun-, gen. Kein vorgegebener Inhalt	-
MAS-Entwicklungsumgebungen			
ZEUS [Nwana et al. 1999]	Umfangreiche Modellierumgebung für komplexe, heterogene Agenten im VPL-Bereich, Visualisierungstools Ähnliche Systeme sind *KAoS* [Bradshaw et al. 1997] *ARCHON* [Jennings 1994], *MAST* [Iglesias et al. 1996] *DECAF* [DECAF 2000], *DAISY* [Poggi 1995] . . .	vorgegeben, konfigurierbare Schablonen, offen teilweise mit vorgebenen Architekturen für Teilaspekte	-
dMars [AAII 1996]	C++ - Entwicklungsumgebung mit Hilfen zur Kommunikation mit anderen Prozessen, graphische Werkzeuge zum Debugging, Plan-Editoren, usw. Viele erfolgreiche Anwendungen	PRS siehe 2.2.1.2	-
JAFMAS [Chauhan 1997]	Java-Klassenbibliothek mit Schwerpunkt der Unterstützung von Kommunikation und Verhandlungen; Werkzeuge zur formalen Analyse der Automaten vergleichbar: *COOL* [Barbuceanu & Fox 1997]	Pläne als Konversations-Automaten	-

System	Beschreibung	Agentenarchitektur	Umwelt
MACE [Gasser et al. 1988]	Eine der ersten Entwicklungsumgebungen für MAS mit Systemagenten, z. B. für Benutzerinterface, Agenten mit verschieder Granularität in Beschreibungsdatenbank Tools zur Performanzevaluation, Kommunikation . . . . Status nach 1990 unklar Andere ältere Entwicklungsumgebungen mit Werkzeugen, Bausteinen, generischen Architekturen und unklarem Status *RATMAN* [Buerckert & Müller 1991], *MAGES* [Bouron et al. 1991]	Agentenattribute und "Engine" als Bausteine	-
MASSIF [Mentges 1999]	Rahmenwerk für sozialwissenschaftliche Simulation, keine neue Sprache, sondern Integration vorhandener, wie UML-Diagramme, KQML-Nachrichten, . . .	Generisches Agent-Modell basiert auf InteRRaP siehe 2.2.1.2	-
MA-Simulationssysteme	**(Klassenbibliotheken)**		
SWARM [Minar et al. 1996]	Objective-C Klassenbibliothek, aufwendige Ereignisverwaltung, hierarchisch organisierter Agenten MAML als Skriptsprache [Gulyas & Kozsik 1999], Animation und Auswertungstools, sehr bekannt für Artifical Life Anwendungen (SFI) ähnliches System in Java: *RePast* [RePast 2000])	Aktualisierungsmethode ist zu programmieren	-
ECOSIM [ECOSIM 1999]	C++ Klassenbibliothek für individuen-orientierte Modelle in der Ökologie. Werkzeuge für Datensammeln und Animation via "trace objects"	-	2D-Grid Jede Zelle aktiv
MA-Simulationssysteme	**(Regelinterpreter)**		
MAGSY [Fischer 1993]	Entwicklungssprache, ursprünglich für autonome Systeme in flexiblen Fabriksystemen. Jeder Agent verwaltet lokale Fakten, Regelinterpreter, . . . C++ in Unix-Umgebungen, Kommunikation via TCP/IP	vorwärtsverketteter Regelinterpreter ohne spezielle Semantik	-
SMDL [Moss et al. 1997]	Ähnlich wie *MAGSY*, Kommunikation über Schreiben und Lesen in öffentlichen Teil der Agenten-Datenbank. Implementiert in Smalltalk mit Ansätzen graphischer unterstützter Modellierung	vorwärtsverketteter Regelinterpreter ohne spezielle Semantik	-

System	Beschreibung	Agentenarchitektur	Umwelt
Soar [Soar 2000]	Soar wurde mit Tcl/Tk versehen und kann nun an beliebige Umwelten gekoppelt werden. Keine Entwicklungumgebung i. e. S., aber bedeutend Basis für verschiedene Modelle, inkl. Rahmen für Teamarbeit in STEAM [Tambe 1997]	Soar (Abschnitt 2.2.1.2)	-
AGENT-0 [Shoham 1993]	abstrakte Programmiersprache für rationale Agenten mit BDI-Konzepten, Kommunikation mit Sprechakten. Vergleichbare Sprache ist *PLACA*[Thomas 1995]	unstrukturierte Verpflichtungsregeln	-
GEAMAS [Marcenac & Giroux 1998]	MA-Simulationsumgebung mit hierarchisch organisierten Agenten in *ReAcTalk* (V.2 in Java) implementiert, Geophysikalische Anwendungen Andere Umgebung mit vorgegebener Organisation *MadKit*[MadKit 2000]	Agentensystem auf mehreren Ebenen	2D-Grid, Netz, abh. von Ebene
Sim_Agent [Sloman & Logan 1999]	Poplog-Klassenbibliothek mit Regelinterpreter nutzt umfangreiche Poplog-Graphikbibliotheken	Mehr-Schichtenarchitektur: Regelinterpreter, verknüpft Neuronalen Netzen Datenbanken	
MA-Simulationssysteme	**(aus der Simulationstechnik)**		
SIF [Funk et al. 1998] [DFKI 2000]	"Social Interaction Framework" als Java-Bibliothek mit "Welt-Server" als Manager der Sensoren und Aktionen und Animationstool als weiterer Agent	-	explizit als "Medium"
PECS-Simplex II [Urban 2000]	Aufsatz auf etabliertes Simulationssystem mit Animation und Auswertungstools. Interaktion über Umwelt-Objekt mit strukturierten Nachrichen	PECS siehe 4.4.1.2	explizit, nicht vorgegeben
JAMES [Uhrmacher 1996]	Java-basierte Agenten-Modellierumgebung, basiert auf einer parallelen *DEVS*-Version. Variable Strukturmodelle mit BDI-ähnlicher Unterteilung Einbinden anderer Softwaresysteme zur Agenten steuerung wie Planer zur Testbed-Entwicklung	AgeDEVS, BDI-ähnliche Unterteilung des internen Zustandsraums	-

System	Beschreibung	Agentenarchitektur	Umwelt
Generische Testbeds			
StarLogo [Resnick 1991]	Turtlegraphik, über Logo-Befehle gesteuert. sehr schnelle Animation, Edukative Simulation	Logo-Programm für individuelle Turtle	2D-Grid
XRaptor [Mössinger et al. 1995]	C++-Klassenbibliothek als Basis für diverse, fest integrierte Domänensimulationen. Flexible Sensor- und Effektormodelle, freies Aktionsselektion-Modul 3D-Animation	Aktionsselektion frei definierbar	3D-Kont. beliebig, aber explizit
GenSim [Anderson & Evans 1995]	Als Testbed-Entwicklungsumgebung konzipiert, klare Trennung zwischen Agent und Umwelt, in Lisp für den Eigengebrauch entwickelt (nur textuelle Ausgabe . . .) Domänenabhängige Simulationsteuerung.	-	-
MICE [Durfee & Montgomery 1989]	"klassisches" generisches Testbed für Koordinationsexperimente. Alt, wird noch verbreitet Konfigurierbar verrauschte Kommunikationskanäle Animationstool	Aktionsselektion als Lisp-Funktion	2D-Grid
EMF [Drogoul & Ferber 1994]	eigentlich generisches Testbed, *Acttalk* Reiz-Propagation über Umweltgitter steuert Aktionsselektion der Agenten. Animation, Auswertung für konkrete Domäne andere Architekturen mit Programmierumgebung "Real Time *ABLE*" (RTA) [Wavish & Graham 1995]	reiz- und bewertungsgesteuerte Aktivitätsauswahl	2D-Grid

5

Ein strukturiertes Schema zur Verhaltensbeschreibung in einem Multiagentenmodell

Die Entwicklung eines Multiagentenmodells betrifft zwei Ebenen: Zum einen muss das Verhalten der einzelnen Individuen festgelegt werden, zum anderen bildet die Organisation, die durch die Interaktionen der Agenten untereinander und mit ihrer Umwelt entsteht, einen Schwerpunkt bei der Modellbildung. Die Hypothese, auf der die vorliegende Arbeit basiert, ist, daß die Repräsentation eines Multiagentenmodells durch einen vorgegebenen Rahmen strukturiert und somit die Erstellung erleichtert werden kann. Wichtig ist dabei, nichttriviale Schablonen und Paradigmen für die Repräsentation eines Multiagentenmodells zu finden, die einerseits ausreichend klar und einfach sind, andererseits allgemein genug, um eine Vielzahl von Anwendungen zuzulassen.

Für die Strukturierung gibt es dabei zwei Bereiche als Ausgangspunkte für eine sinnvolle Unterteilung: die Beschreibung der Struktur und die des Verhaltens der einzelnen Bestandteile. Im folgenden werden die möglichen Formen zunächst allgemein vorgestellt und vielversprechende Ansätze näher verfolgt.

5.1 Beschreibung der Architekturstrukturen

Der erste Ansatzpunkt bei der Repräsentation eines Multiagentenmodells liegt bei der Vorgabe von Strukturen für die zwei Ebenen der Beschreibung eines Agentensystems: Wie in Abschnitt 2.2 eingeführt, sind dies die Architektur des Gesamtsystems und die der einzelnen Agenten. Dabei wurden auch vorhandene Ansätze zur Unterstützung der Entwicklung einer konkreten Spezifikation auf der jeweiligen Ebene besprochen, wie zum Beispiel domänenabhängige Musterarchitekturen [Horn & Reinke 1999], oder verschiedene Agentenarchitekturen [Müller 1999] mit mehr oder weniger abstrakten Strukturvorgaben.

Darauf aufbauend stellt sich letztendlich die Frage, in welchem Ausmaß eine Strukturierung – sowohl auf der Ebene des Gesamtsystems, als auch auf der der Agenten – vorgegeben werden kann. Dies ist insbesondere dann fraglich, wenn dem Modellierer ein einfach zugängliches, dennoch domänenunabhängiges Konzept nahegebracht werden soll. Im folgenden soll nun ausgehend von einem bekannten Schema für einzelne Agenten – dem “Reflex-Agenten mit internem Zustand” nach S. Russell und P. Norvig [Russell & Nor-

vig 1995] – eine einfache Möglichkeit vorgestellt werden, sowohl die Repräsentation eines Agentensystems, als auch des internen Zustand der einzelnen Agenten zu strukturieren.

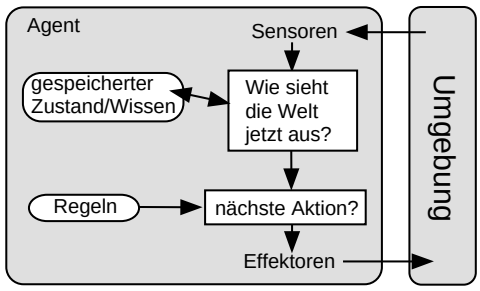
5.1.1 Reflex-Agent mit internem Zustand

5.1.1.1 Definition

S. Russell und P. Norvig entwickelten in [Russell & Norvig 1995] eine Klassifikation von unterschiedlich komplexen Agentenarchitekturen. Ein Überblick über sie wurde auf Seite 16 gegeben.

Das im folgenden vorgestellte Konzept der Strukturierung eines Multiagentenmodells basiert auf dem Schema eines einfachen Reflex-Agenten mit internem Zustand. In Abbildung 5.1 wird eine leicht vereinfachte Darstellung der grundlegenden Architektur eines derartigen Agenten gezeigt.

Abbildung 5.1 Vereinfachte Darstellung eines Reflex-Agenten mit internem Zustand nach S. Russell und P. Norvig [Russell & Norvig 1995]



Genau besehen kann die Aktualisierung eines derartigen Agenten in folgende Phasen eines Zyklus unterteilt werden. Dabei sind die Eingaben Daten aus der Wahrnehmung des Agenten. Als Ausgabe liefert dieser Aktualisierungszyklus die Aktion des Agenten an seine Umgebung, bzw. den Simulator :

1. $zustand \leftarrow UPDATE - ZUSTAND[zustand, wahrnehmung]$
2. $regel \leftarrow REGEL - AUSWAHL[zustand, regelmenge]$
3. $aktion \leftarrow REGEL - AKTION[regel]$
4. $zustand \leftarrow UPDATE - ZUSTAND[zustand, aktion]$

In der Abbildung 5.1 wurde – im Gegensatz zur Originalgraphik – die Darstellung der zwei konkreten Formen von Wissen weggelassen, die der Agent zum Aktualisieren seines Zustands, im besonderen seines Weltmodells, benötigt.

Zum einen ist dies das Wissen, wie die Welt sich entwickelt. Mittels diesem kann der Agent sein internes Weltmodell auf der Basis seiner Wahrnehmung mit der “Realität” abgleichen. Zum anderen sind es Kenntnisse, wie die Welt sich bei der Ausführung der ausgewählten Aktion verändert. Dieses Wissen benötigt der Agent, wenn er die Effekte seiner Aktionen nicht direkt wahrnehmen kann. Wenn der Agent nicht in jedem Zyklus vollen Zugriff auf die benötigte Information über seine Umwelt besitzt, dann ist derartiges Wissen notwendig, um das in seinem internen Zustand gespeicherte “Weltmodell” so aktuell wie nötig zu halten.

Diese Form eines Agenten bildet die Basis für das im folgenden verwendete Paradigma, da das Konzept der "vorwärtsverketteten" Vorgehensweise dem in Abschnitt 3.3.3.2 dargestellten einfachen regelbasierten Mechanismus zur Verhaltensbeschreibung entspricht. Die zielbasierten oder nützlichkeitsbasierten Schemata erlauben zwar eine mehr vorausschauende Formulierung rationalen Verhaltens mittels explizit repräsentierten Zielen, sind allerdings deutlich komplizierter als das einfache Abbilden von Sensor- und Zustandsinformationen auf Aktionen, das für eine Beschreibung des Verhaltens ausreicht. Andererseits ist die Bereitstellung von Schablonen und Verarbeitungsmechanismen für einen internen Zustand wichtig, da auch in einer simulierten Welt nicht jede für die Aktionsauswahl notwendigen Informationen zum richtigen Zeitpunkt über die Wahrnehmung zugänglich sein kann.

5.1.1.2 Vergleich mit "hysteretic agent" nach Genesereth/Nilson

Einen ebenfalls etablierten Rahmen zur Kategorisierung von Agentenformen findet man bei M. Genesereth und N. Nilson [Genesereth & Nilson 1987]. Sie unterscheiden vier Formen von Agenten: "tropistic agents", d.h. reine Reflex-Agenten ohne eine Form eines internen Zustands und "hysteretic agents", die zusätzlich über einen internen Zustand verfügen. Abhängig von Form und Inhalt dieses Zustands werden zwei weitere Agententypen angegeben: "knowledge based agents", deren Zustand auf einer abstrakten Ebene aus Aussagen besteht, die das Wissen des Agenten darstellen und "stepped knowledge based agents", bei denen diese Aussagen mit einem Zeitstempel versehen wurden.

Ein System mit einem "hysteretic agent" kann nach M. Genesereth und N. Nilson durch die Angabe folgender Mengen und Funktionen charakterisiert werden:

$$\langle I, S, T, A, see, do, internal, action \rangle$$

mit I: Menge von internen Zuständen
S: Menge von Zuständen der externen Welt
T: Menge der für den Agenten unterscheidbaren Partitionen von S
A: Menge der für den Agenten möglichen Aktionen
$see : S \rightarrow T$ (Sensor-Funktion)
$do : A \times S \rightarrow S$ (Effektor-Funktion)
$internal : I \times T \rightarrow I$ (Update des internen Zustands)
$action : I \times T \rightarrow A$ (Aktionsselektion)
Dabei ist zu beachten, daß diese Definition ein Gesamtsystem bestehend aus Umwelt (S) und einem Agenten (T, I und A) beschreibt. Ein Aktualisierungszyklus für beide Teile besteht dabei aus

1. $t_t := see(s_t)$
2. $i_t := internal(i_{t-1}, t_t)$
   $a_t := action(i_t, t_t)$
3. $s_{t+1} := do(a_t, s_t)$

Nach M. Genesereth und N. Nilson können durch einen "hysteretic agent" Agenten auf beliebigen Detaillierungsebenen beschrieben werden. Die Mechanismen, durch die das interne Modell aktualisiert und die nächste durchzuführende Aktion ausgewählt wird, sind nicht näher festgelegt. Die Unterteilung fokussiert dabei – im Gegensatz zu der von S. Russell und P. Norvig – nicht auf unterschiedlich komplexen Mechanismen zur Aktionsauswahl,

sondern auf das Vorhandensein eines internen Zustands und dessen Inhalt bzw. Abstraktionsgrad. Der Rahmen eines "hysteretic agent" ist allerdings zunächst nur ein Ein-Agentensystem: Die Menge S ändert sich nur durch die Aktionen des Agenten, auch wenn der Agent diese Änderungen möglicherweise nicht vollständig wahrnehmen kann.

Eine Verwendung dieses Schemas als Basis für die nachfolgenden Überlegungen wäre durchaus attraktiv, vor allem wegen der Ähnlichkeit zum in der Simulationstechnik etablierten Systembegriff (siehe Abschnitt 3.1.1) und der expliziten Repräsentation von Umweltzuständen. Auch die Erweiterung auf ein System mit mehreren "hysteretic agents" durch eine Kopplung über den Agenten gemeinsame Mengen S, T, I oder A, wie sie von R. Fulbright und L. M. Stephens [Fulbright & Stephens 1994] für ein Zwei-Agentensystem vorgeschlagen wird, ist möglich.

5.1.2 Struktur des internen Zustands eines Agenten

Der Sinn des internen Zustands bei einem einfachen Reflex-Agenten nach S. Russell und P. Norvig liegt hauptsächlich darin, die Wahrnehmung des Agenten so zu speichern, daß in Situationen, in denen die Umwelt für den Agenten nicht vollständig einsehbar ist, Informationen über die Situation, in der sich der Agent befindet, in Form eines "Weltmodells" zur Verfügung stehen.

Eine ähnlich Vorstellung steht auch hinter dem Konzept eines internen Zustands eines Systems im Sinne der Simulationstechnik. In den Zustandsvariablen sollen – wie in Abschnitt 3.1.1 ausgeführt – die Eingaben der "Inputvariablen" so gespeichert werden, daß die Ausgaben denen des realen Systems entsprechen können. In bestimmten Anwendungen besitzen die internen Zustandsvariablen zugleich die Funktion von Ausgabevariablen. Beispiele findet man in Makrosimulationen von Gesellschaften: Eine übliche Zustandsvariable stellt dabei die Populationsgröße dar. Diese ist gleichzeitig die wichtigste Ausgabegröße des Systems bei Simulationsexperimenten.

5.1.2.1 Interner Zustand als Vektor

In Anlehnung an den Systembegriff soll der interne Zustand eines Agenten in Form einer Menge von Zustandsvariablen repräsentiert werden. Dabei können mehr oder weniger getrennt von einander zu betrachtende Aspekte eines internen Modells auch explizit separat voneinander gespeichert werden: Jede Zustandsvariable kann eine andere Facette des internen Zustands widerspiegeln. Alternativen wären dabei z.B. Mengen von logischen Aussagen, die als Faktenbasis das repräsentieren, was der Agent über seine Umwelt weiss. Im Gegensatz zu einer derartigen unsortierten Menge von Aussagen, bildet die Darstellung über einen Zustandsvariablenvektor eine grundsätzlich andere, von vorne herein strenger organisierte Form. Allerdings verschließen sich dabei Abfragen des internen Zustands eher den klassischen Mechanismen, wie Unifikation oder Pattern Matching, da dazu sowohl Abfragen des internen Zustands, als auch der Zustand selbst im selben Formalismus dargestellt werden müsste, – z.B. in Form von prädikatenlogischen Termen, die mittels Unifikation "gleichgemacht" werden. Auf diese Weise nimmt der interne Zustand die Form eines Zustandsvariablen-Vektors an:

$$I_i = \{(z_0, z_1, \ldots, z_m) | z_j \in Z_j \text{ Wert der ZVariable } j \text{ mit Wertebereich } Z_j \text{ bei Agent } i\}$$

Durch diese Form der Repräsentation ist noch keine Beschränkung der jeweiligen Wertebereiche oder Repräsentationsformen impliziert. Dadurch wird vielmehr eine Grundlage für eine mögliche Strukturierung des internen Zustands eines Agenten gelegt, indem mögliche Kategorien von Zustandsvariablen identifiziert werden. Allerdings sind die für die einzelnen Agenten benutzten Variablen ein Bestandteil des jeweiligen Modells und werden vom Modellbauer festgelegt. Dennoch kann man zusätzlich zu Kategorien, die ein Modellierer vorgibt oder in einer Domäne sinnvoll sind, bestimmte Hypothesen zu grundlegenden Kategorien ausnutzen. Ein prominentes Beispiel für eine mögliche Unterteilung des internen Zustands sind die Kategorien des BDI-Paradigmas (siehe Abschnitt 2.2.1.2) Die Kategorien “Beliefs”, “Desires” und “Intentions” besitzen dabei relativ festgelegte Funktionen im Aktionsselektions- bzw. Planungsprozess des Agenten: “Beliefs” für das, was der Agent über seine Situation zu wissen glaubt, “Desires” für die (groben) Ziele, “Intentions” für das auszuführende Zielverhalten, bzw. seine Pläne.

5.1.2.2 Kategorien von Zustandsvariablen

Gesucht wird also eine Zerlegung der Menge der Zustandsvariablen als Basis für die Strukturierung des Vektors, mit dem der interne Zustand eines Agenten beschrieben wird. Ideal wäre dabei eine Kategorisierung, die inhaltlich zusammengehörende Variablen zu Klassen zusammenfaßt. Diese Zerlegung der Variablenmenge ist allerdings eine Frage der Interpretation, d.h. es gibt keine absolut richtige Zuordnung einer Zustandsvariable zu einer Kategorie: Dies liegt vor allem daran, daß es keine für alle Multiagentenmodelle gleichermaßen passende, vorgegebene Menge an Kategorien gibt, da diese nicht nur von der Modelldomäne, sondern auch direkt von dem Abstraktionsgrad der Beschreibung abhängen.

Formal definiert, sieht eine derartige Zerlegung K in Teilmengen der Menge aller Zustandsvariablen Z wie folgt aus:

$$K \subseteq \mathbf{P}Z \text{ mit}$$
$$X \in K \wedge Y \in K \Rightarrow X \cap Y = \emptyset \text{ (disjunkte Mengen)}$$
$$\bigcup K = Z$$

Jedes Element der Kategorienmenge K – eine Teilmenge von Zustandsvariablen – kann mit einem bedeutsamen Namen charakterisiert werden, unter dem die der jeweiligen Teilmenge zugeordneten Zustandsvariablen aussagekräftig subsumiert werden können. Dabei soll jede Variable eingeordnet werden können – idealerweise sollte jede Form der Kategorisierung eindeutig sein (disjunkte Teilmengen). Dabei können mehrere dieser Klassifikationsrahmen (jeder einzelne wie oben definiert) kombiniert werden, um eine mehrdimensionale Einteilung realisieren zu können.

Die Vorgabe von derartigen Kategorien zur Unterteilung der Menge der Zustandsvariablen eines Systems wird auch in anderen Rahmenwerken verwendet: Die Referenzarchitektur *PECS* (siehe 4.4.1.2) bietet eine Menge von vorgegebenen Kategorien, die an psychologische Verhaltensmodelle angelehnt sind. Für die Beschreibung des Verhaltens wird so eine Schablone bereitgestellt, in die einzelne Zustandsvariablen eingeordnet werden können. Dieses Referenzmodell wurde zur Nachbildung menschlicher Agenten entworfen.

In einem domänenunabhängigen Rahmenwerk kann allerdings auf einer modell-inhaltlichen Ebene “nur” ein Rahmen für die Repräsentation von Zustandsvariablenkategorien

gegeben werden. Zusätzliche domänenunabhängige Kategorien von Zustandsvariablen findet man allerdings bei einer Betrachtung von Funktionen und anderen syntaktischen Eigenschaften

5.1.2.3 Möglichkeiten für domänenunabhängige Kategorien

Domänenunabhängige Informationen zu möglichen Kategorien, die einer sinnvollen Zerlegung der Menge der Zustandsvariablen zugrunde liegen können, findet man auf implementierungsnahen Ebenen der Beschreibung des internen Zustands einer Einheit: Dies sind eher syntaktische Kriterien, wie Wertebereich oder Veränderlichkeit.

$$
\begin{aligned}
K_{wertebereich} \quad = \quad & (\{Z|Z \subseteq \mathbb{N}\}, \\
& \{Z|Z \subseteq \mathbb{R}\}, \\
& \{Z|Z = \{a, b, c \ldots\} \text{ mit } a, b, c, \ldots \in \text{ Menge von Symbolen } \}, \\
& \ldots)
\end{aligned}
$$

$$
\begin{aligned}
K_{dynamik} \quad = \quad & (\{Z| \text{ Zustandsvariable Z mit konstantem Wert } \}, \\
& \{Z| \text{ mit Veränderung nur durch Aktionen des Agenten } \}, \\
& \{Z| \text{ mit } \exists f : f : \mathbb{N} \times I \rightarrow Z \\
& \quad \text{d.h. Wert von } Z \text{ ist zeitabhängig,} \\
& \quad I \text{ ist dabei der (gesamte) interne Zustand des Agenten } \})
\end{aligned}
$$

Durch Einordnung einer Zustandsvariablen in ein oder mehrere diese Klassifikationsschemata wird vor allem festgelegt, mit welchen Mittel und welcher Steuerung diese Zustandsvariable behandelt werden kann. Die Einteilung nach Wertebereich ist dabei sehr naheliegend, wenn man von einer Typisierung der Zustandsvariablen ausgeht. Diese eher syntaktische Information ist im Gegensatz zu den inhaltlichen Kategorien nahe an der Implementierung des Modells und insbesondere in Kombination mit einem konkreten Simulationswerkzeug sinnvoll, da so eine effiziente Interpretation der Zustandsbeschreibung und der zugehörigen Dynamik erreicht werden kann. Inhaltliche Kategorien sind dagegen bei der Beschreibung, bzw. Spezifikation des Konzeptmodells sinnvoll.

5.1.3 Agentensystemstrukturen

5.1.3.1 Kopplung über eine gemeinsame Umwelt

Ein naheliegender Weg, aus der Betrachtung eines einzelnen Agenten zu einem Agentensystem überzuleiten, besteht darin, mehrere Agenten in einer gemeinsamen Umgebung zusammenzustellen. Dabei handeln diese Agenten im Idealfall vollständig parallel und interagieren auf eine nicht zentral vorgegebene und durch Selbstorganisation synchronisierte Art und Weise. Dabei stellt sich die Frage, welche strukturellen Vorgaben ein Rahmenwerk machen sollte, bzw. machen kann:

Einerseits sind dabei Vorgaben zur technischen Umsetzung möglich: Dabei muß die Methode der Aktualisierung des Gesamtsystems festgelegt werden – gerade bei der Simulation paralleler Prozesse auf sequentiell arbeitenden Rechnern hat dies Auswirkungen auf die

Umsetzung des Modells, und ist somit auch grundlegend für die Formulierung des Verhaltens. Das Problem ist dabei die Abbildung eigentlich gleichzeitig geschehender Interaktionen zwischen den Agenten durch eine Sequenz von Aktionen. An sich können dabei entstehende Dilemmas am besten auf der Ebene des konkreten Modells gelöst werden, da die zeitlichen Aspekte zentraler Bestandteil eines Verhaltensmodells sind. Allerdings ist dazu die Kenntnis von Details der Umsetzung während der Simulation notwendig. Diese werden in Abschnitt 6.3.1 im nächsten Kapitel besprochen. Für die im folgenden dargestellten Strukturierungskonzepte genügt es, davon auszugehen, daß das gesamte Agentensystem und seine Umwelt auf synchrone Weise aktualisiert werden: In jedem Zeittakt wird jede Entität genau einmal betrachtet; daran anschließend wird die globale Uhr um eine Einheit hochgezählt. Die Reihenfolge der Aktualisierung der einzelnen Einheiten ist dabei nicht festgelegt.

Unabhängig von diesen technischen Aspekten der Umsetzung parallel ablaufender Interaktionen während einer Simulation kann man die übersichtliche Repräsentation des Gesamtsystems durch strukturelle Vorgaben forcieren und die Agentensystemarchitektur und ihre Interaktionen ordnen. Üblicherweise werden bei Vorgaben durch Entwicklungswerkzeuge alle Entitäten gleich behandelt, wie in Abschnitt 4.4 anhand von Beispielsystemen erläutert wurde. Dies hat den Nachteil, daß Informationen über hypothetische Strukturen nicht benutzt werden, um das Gesamtsystem übersichtlicher zu repräsentieren.

Grundsätzlich sind Vorgaben dabei einfacher für domänenabhängige Modellierumgebungen zu machen, wie dies bei dementsprechenden Simulationsumgebungen üblich ist. So geht man z.B. bei der Repräsentation einer Produktionsanlage vom Vorhandensein stationärer Maschinen mit mehr oder weniger flexiblen Steuerungen und Transportverbindungen für bewegliche Materialien, die bearbeitet und transportiert werden, aus und bietet dem Modellierer dementsprechende Grundtypen von Einheiten an. Die Frage, die im folgenden beantwortet werden soll, ist, ob man entsprechende Vorgaben unabhängig von einer konkreten Domäne, aber für die speziellen Strukturen machen kann, die man bei einem Multiagentenmodell identifizieren kann. Dabei wird im folgenden von einer Unterscheidung zwischen Umwelt, passiven “Ressourcen” und aktiven Agenten ausgegangen.

5.1.3.2 Umwelt-Agent

Die Identifikation einer expliziten Einheit, die die Umwelt für die eigentlichen Agenten darstellt, bildet eine gute Basis für die Strukturierung der Gesamtarchitektur. Wie in Kapitel 4 hervorgehoben, werden in den Multiagentenmodellen, auf die sich die vorliegende Arbeit beschränkt, geschlossene Systeme betrachtet: d.h. mit dem Agentensystem wird auch deren Umwelt abgebildet. Die Einzige nicht von einem Modellbestandteil kontrollierte Variable ist die Zeit, deren Fortschritt vom Simulator quasi von außen vorgegeben wird. Die Folge davon ist, daß die Umwelt des Agentensystems vollständig und explizit mit-modelliert werden muß. Dieses Modell einer Umwelteinheit kann im vorgestellten Rahmen folgende Aspekte umfassen:

1. Eine Menge von globalen Zustandsvariablen, mit denen der Zustand der Umwelt repräsentiert wird.

2. Eine Menge von globalen Ereignissen, mit denen neue Agenten erzeugt, Agenten gelöscht, bzw. der Zustand von Agenten geändert werden können. Auf diese Weise kön-

nen Struktur- und sonstige Änderungen, die nicht von einem der Agenten kontrolliert werden, abgebildet werden.

3. Repräsentation des Raums, über den die Lokalität der Agenten und ihrer Aktionsradien dargestellt werden kann.

Besondere Einheiten, die diese Vorgaben erfüllen, können auch in einer domänenunabhängigen Modellierumgebung vorgegeben werden. Die beiden ersten Punkte deuten auf eine naheliegende Interpretation als weiterer "normaler" Agent hin, der über eine ausgezeichnete Funktion verfügt. Der dritte Punkt impliziert allerdings eine besondere, zentrale Rolle, die für eine explizite Behandlung durch den Simulator spricht.

Fraglich bleibt dementsprechend die Form der Integration in das "restliche" Agentensystem. Wird die Umwelt als weiterer, vom Simulationssystem als den anderen gleichwertiger Agent betrachtet, muß der Modellbauer selbst sicherstellen, daß die notwendigen Interaktionen über diesen Umweltagenten stattfinden. Behandelt der Simulator die Umwelt getrennt von den Agenten, kann der Modellbauer die Analogie als Umwelt-Agent nicht ausnutzen. Der Modellierer hat dann meist nicht die Möglichkeit, auch die globale Dynamik der Umwelt zu formulieren, sondern muß auf Hilfsmittel, wie zusätzliche Agenten mit globalem Einfluß zurückgreifen. Wie in Abschnitt 4.4 dargestellt, existieren sowohl Spezifikationsrahmen und Simulationsumgebungen, die keine explizite Umweltrepräsentation vorgeben, als auch Modellierumgebungen, bei denen die explizit dargestellte Umwelt auf eine Raumrepräsentation beschränkt ist.

Die Lösung dieses Dilemmas besteht hier darin, besondere Einheiten als Umwelt-Systeme auszuzeichnen. Zu den "normalen" Zustandsvariablen kommt eine Raumdarstellung, in der eine Karte repräsentiert wird. Jeder Agent muß dazu seine Position in einem Umwelt-System ("Bezugssystem") kennen. Aktionen der Agenten, die zu einer Veränderung auf der Karte führen – wie zum Beispiel Bewegung – werden analog zu Interaktionen zwischen den Agenten behandelt. Das bedeutet, daß spezielle Relationen zwischen "normalen" Agenten und Umweltagent vorgegeben sind. Bei Modellen, in denen keine explizite Repräsentation des Raums notwendig ist, können diese Raum-Bezug-Relationen nicht benutzt werden. Es bleiben aber die globalen Variablen und das global wirksame Verhalten der expliziten Umwelteinheit bestehen.

Die Anzahl der Umwelt-Systeme in einem Modell ist grundsätzlich nicht vorgegeben und auf ein einziges beschränkt. Dies läßt sich dadurch begründen, daß man so jeweils zweidimensionale Karten verknüpfen kann, um eine dritte Dimension an den notwendigen Stellen nachzubilden. Jedes Umwelt-System verfügt dabei – wie ein normaler Agent – über eine Position in einem Bezugssystem (ungleich sich selbst). Agenten an dieser Position können sich zum Beispiel zwischen Bezugssystemen bewegen.

Auch Umwelt-Systeme als Quasi-Agenten können ihre Zustandsvariablen aktualisieren, und Aktionen durchführen. Da diese Aktionen auf einer konzeptionell anderen Ebene zu sehen sind, wie die Aktionen eines Agenten, sollen sie anders bezeichnet werden: Als externe Ereignisse[1], im Sinne von "Eingriffen" von außen in das eigentliche Agentensystem (siehe dazu auch Abbildung 5.2).

[1] Dieser Ereignisbegriff als "externer Eingriff" entspricht nicht dem in der Simulationstechnik üblichen. Bei der ereignis-orientierten Simulation ist jede Veränderung des Systemzustands ein Ereignis. Demnach kann auch jede Aktion eines Agenten als "Ereignis" gesehen werden. Bei der expliziten Unterscheidung zwischen Umwelt und Agenten ist auch eine sprachliche Differenzierung zwischen Agentaktion und Umweltereignis sinnvoll.

5.1.3.3 Repräsentation nicht-aktiver Bestandteile

Zusätzlich zu den explizit als solche ausgezeichneten Umwelt-Systemen können auch andere Entitäten des Agentensystems deutlich unterschiedliche Funktionen oder Verhaltenscharakteristika aufweisen: Sei es als aktiver Agent oder eher passive Ressource. Hindernisse, Futterobjekte, usw. bereichern die Umwelt eines Agentensystems, verändern diese aber nicht aktiv[2].

Ressource-Objekte als solche explizit zu klassifizieren, hat zudem nicht nur den Vorteil eines bis in die Umsetzung durchgängigen Konzepts, sondern auch der Simulator kann diese Information benutzen, um die Aktualisierung effizienter durchzuführen. So muß bei einem explizit als Ressource gekennzeichneten Objekt kein "Reasoning" angestoßen werden, das dann quasi ein Nullverhalten zurückliefert, sondern es ist klar, daß diese Einheit nicht auf ihre Umwelt wirken kann.

Allerdings kann ein Ressource-Objekt auch über Zustandsvariablen verfügen, die einen veränderlichen Zustand der Einheit repräsentieren. Dieser Zustand kann dabei nicht nur von Zeit, sondern auch von Aktionen und Ereignissen abhängen. Die Konsequenz daraus ist, daß ein Ressource-Objekt in der hier dargestellten Form über Sensoren und Aktualisierungsmechanismen für seinen internen Zustand verfügen sollte. Es sind aber keine verhaltensbestimmenden Regeln oder Effektoren notwendig. Abbildung 5.2 fasst die verschiedenen Typen von Einheiten zusätzlich zu den Reflex-Agenten mit internem Zustand zusammen.

5.1.3.4 Gruppen und Kategorisierung von Agenten

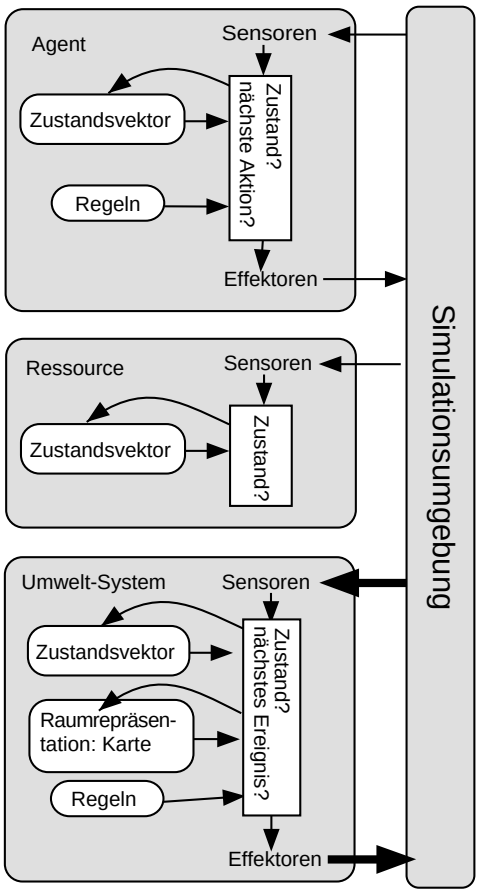
Das wichtigste Mittel zur Strukturierung des Agentensystems selbst sind Organisationsstrukturen und Gruppenbildung. Diese Charakterisierungen werden üblicherweise dazu eingesetzt, um die Interaktionsmöglichkeiten der Agenten zu strukturieren, möglicherweise auch zu beschränken, bzw. überschaubarer zu gestalten. In Simulationsanwendungen dienen derartige Gruppen vor allem dazu, vorhandene soziale Strukturen direkt nachzubilden. Über Zuweisung von Rollen in einer Gruppe kann auch das Verhalten eines Agenten strukturiert werden, wie im zweiten Teil dieses Kapitels näher ausgeführt wird.

In diesem Zusammenhang stellt sich die Frage, wie die Zusammengehörigkeit von Agenten in Agentengruppen oder anderen Arten von Organisationen repräsentiert werden kann. Dazu gibt es im bisher vorgestellten Rahmen zwei Möglichkeiten: Implizit über eine gleichzeitige Lokalisierung in einem gemeinsamen Bereich, z.B. auf der Karte eines Bezugssystems. Zum anderen ist dies explizit über einen entsprechenden Wert einer als "Zugehörigkeitszustand" ausgezeichneten Zustandsvariable möglich. Inhaltliche Gruppen sind nur auf einer domänenspezifischen Ebene möglich.

Eine Ausbaustufe dieser Ansätze – in objekt-orientierten Simulationsschemata findet man diese Form häufig – besteht in einer hierarchischen Gruppierung: Eine Gruppe aus Entitäten (Agenten, Ressourcen und Umwelt-Systemen) ist eine spezielle Form von Entität, die wiederum Mitglied in einer Gruppe sein kann. Jede Gruppe verfügt als explizite Entität über einen zusätzlichen Zustandsvariablenvektor. Das Verhalten kann auf das Verhalten der untergeordneten Agenten zurückgeführt werden. Dabei kann jeder Agent seine durch seine Regeln und seinen internen Zustand bestimmte Aktionen ausführen. Dieser Ansatz würde

[2] Ihr Entstehen oder Verschwinden verändert zwar die Umwelt, wird aber durch externe Ereignisse des Umwelt-Systems oder durch Aktionen von Agenten bewirkt.

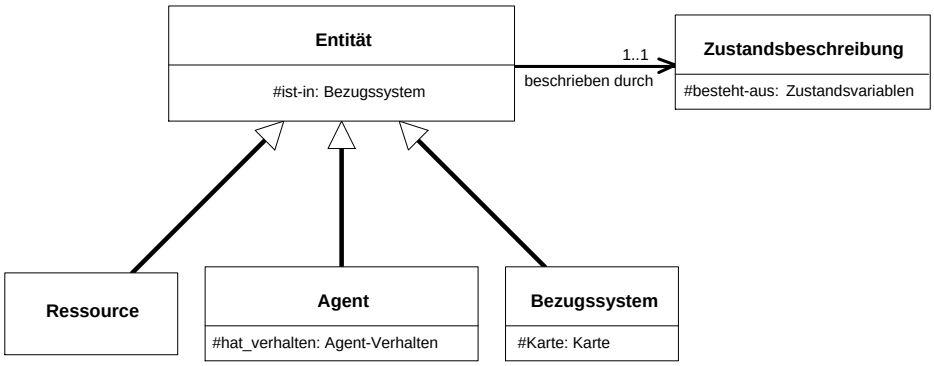
Abbildung 5.2 Verschiedene, strukturell unterschiedliche Typen von Einheiten in einem Multiagentenmodell: Agenten, Ressourcen und Umweltsysteme.



der oben angedeuteten expliziten Repräsentation der Zugehörigkeit zu Gruppen über eine Zustandsvariable entsprechen. Eine andere Möglichkeit besteht darin, das Gruppenverhalten explizit zu formulieren und damit das Verhalten der einzelnen Agenten zu unterdrücken. So entspricht die Gruppeneinheit einem Agenten. Das Zusammenfinden bzw. Auflösen dieses Gruppenagenten geschieht dabei durch Aktionen von untergeordneten Agenten in Verbindung mit Erzeugen und Löschen vorhandener Einheiten. Auf diese Weise lassen sich derartige Gruppenstrukturen ohne Veränderung in den vorgestellten Rahmen integrieren.

Abbildung 5.3 fasst die bisher vorgestellten Kategorien an möglichen Einheiten in einem Multiagentenmodell zusammen. Diese Skizze ist in Notation eines *UML*-Klassendiagramms gegeben. Sie zeigt, wie in diesem Rahmen Gemeinsamkeiten, z.B. die Beschreibung des internen Zustands durch Zustandsvariablen für die Realisierung in einem objekt-orientierten Rahmen genutzt werden können.

Abbildung 5.3 Überblick über einen Rahmen für ein Gesamtsystem aus Agenten, Ressourcen und expliziten "Bezugssystemen", d.h. Umwelt-Systemen. Das Diagramm folgt der Notation eines UML-Klassendiagramms.



5.2 Repräsentation von Interaktionen

Einer der wichtigsten Aspekte bei der Darstellung eines Multiagentenmodells ist die Repräsentation der Interaktionen der Agenten untereinander und mit ihrem Umweltsystem. In dem vorgestellten Rahmen existieren zwei Schnittstellen, mittels derer sich Interaktionen realisieren lassen: Über Sensoren und Effektoren, mit denen der Agent die Daten aus seiner Umwelt über die Wahrnehmung empfängt und seine Aktionen ausführt, bzw. an seine Umwelt weitergibt.

5.2.1 Interaktion mittels Sensoren und Effektoren

Interaktionen, die sich durch eine Kombination von ineinandergreifenden zur Verfügung stehenden Wahrnehmungen und Effektor-Kommandos formulieren lassen, können problemlos im bisher erklärten Rahmen dargestellt werden. Das heißt, die Veränderungen, die ein Agent in seiner Umwelt verursacht, können von anderen Agenten wahrgenommen und die daraus gewonnene Information kann weiter verarbeitet werden. Dieses Konzept ist beispielsweise gut geeignet, um die Kommunikation zwischen Ameisen über in der Umwelt "absichtlich" platzierte Pheromonspuren nachzubilden.

5.2.2 Interaktion mit Hilfe von Kommunikation und Sensor-Aktionen

Andere Formen und Ebenen der Kommunikation bedürfen dagegen weiterer Erläuterung: Das Senden einer Nachricht kann ohne weiteres als Aktion interpretiert werden. Wenn das Empfangen einer Nachricht als Wahrnehmung, z.B. über einen "Nachrichtensensor" dargestellt werden kann, dann ist auch diese Form nachrichtenbasierter Kommunikation formulierbar. Spezielle Anforderungen an diesen Sensor ergeben sich, wenn die Aktualisierung der anliegenden Nachrichten nicht mit der des Agenten synchron ist. Dann muß der Sensor Nachrichten solange zwischenspeichern können, bis der Agent in der Lage ist, diese abzufragen. Allerdings gehören auch diese Fragen eher zur konkreten Modellformulierung und können weniger unabhängig vom darzustellenden Verhalten beantwortet werden.

Fraglich könnte dagegen die Integration in diesen Rahmen sein, wenn zum Erhalt von Information aus der Umwelt bewußte Aktionen des Agenten notwendig wären, dei im Folgenden "Sensor-Aktionen" genannt werden. Dies ist beispielsweise der Fall, wenn Daten von einem Blackboard gelesen werden sollen oder die Ausführung der Aktion eigentlich einen relevanten "Rückgabewert" liefern sollte. Bei der Aktualisierung eines Reflex-Agenten mit internem Zustand nach Russell / Norvig (siehe oben) steht am Ende folgender Aufruf:

4. $zustand \leftarrow UPDATE - ZUSTAND[zustand, aktion]$

Dazu ist nach S. Russell und P. Norvig "nur" Wissen darüber notwendig, wie die Aktion auf die Umwelt wirkt. Sensor-Aktionen dienen aber gerade dazu, Wissen über den aktuellen Zustand der Welt zu sammeln. Dieses Wissen steht allerdings über die "unbewußte" Wahrnehmung zur Verfügung.

1. $zustand \leftarrow UPDATE - ZUSTAND[zustand, wahrnehmung]$

Auch Sensor-Aktionen lassen sich somit auf einer technischen Ebene auf nachrichtenbasierte Kommunikation reduzieren: Der Agent schickt an einen anderen Agenten, bzw. das Umweltsystem eine Nachricht mit einer Anforderung um Mitteilung bestimmter Sachverhalte. Im nächsten Zyklus – oder später – liegt am "Kommunikationssensor" die Antwort vor.

Als Fazit für die Unterstützbarkeit der Interaktionsformulierung muß man feststellen, daß sich auf dieser domänenunabhängigen Ebene kaum mehr als triviale Vorgaben machen lassen. Wichtiger ist es dabei, über vorgegebene Grundaktionen, wie z.B. "schicke REQUEST für . . . " bzw. Sensoren, es dem Modellierer zu erlauben, auch höhere Protokolle oder anderes Interaktionverhalten einfach darzustellen.

5.3 Strukturierte Beschreibung des Verhaltens

Die Beschreibung von Verhalten auf der Basis einer Regelmenge ist, wie dargestellt, vor allem wegen des deklarativen Charakters der Regeln (siehe dazu Abschnitt 3.3.3) sehr attraktiv und dementsprechend weit verbreitet. Allerdings hat die Verwendung einer nicht weiter unterteilten Regelmenge einige Nachteile: Die fehlende Effizienz beim Bestimmen der Regeln, die im aktuellen Kontext feuern können, verursacht bei großen Regelmengen Probleme. Zudem sind große, unstrukturierte Regelmengen kaum mehr wartbar, wie man beim Aufbau und Experimenten mit frühen regelbasierten Expertensystemen feststellen mußte [Puppe 1991]. Durch eine Strukturierung der Regelmenge kann man dagegen hoffen, eine Grundlage für die effiziente Bestimmung der Teilmenge von Regeln, deren Vorbedingung im aktuellen Kontext wahr ist, zu schaffen. Zudem kann auf dieser Basis die Regelmenge übersichtlich präsentiert und somit leichter erstellbar und wartbar werden. Dies gilt insbesondere, wenn die Unterteilung für den Domänenexperten nachvollziehbaren Merkmalen folgt und dabei konsequent vorgenommen wird.

Wie bei der Beschreibung der Architekturstrukturen kann man auch bei der Repräsentation der verhaltensbeschreibenden Regelmenge zwei Ebenen unterscheiden: Eine Unterteilung auf Agentensystem-Ebene und eine innerhalb eines Agenten bzw. hier auch Umwelt-System.

Die erstere ist die übliche Zuordnung einer Regelmenge zu einem Individuum, dessen Verhalten durch diese Regelmenge beschrieben wird. In einer objekt-orientierten Implementierung kann man diese Regelmenge auch zu einer Einheitenklasse angeben. Bei vielen Agenten, die ein strukturell gleichartiges Verhalten aufweisen, ist es sinnvoll, Individualität als Antwort auf unterschiedliche Werte in den internen Zustandsvariablen zu reduzieren. Diese Form der Strukturierung ist diejenige, die in vorhandenen Modellier- und Simulationsrahmen für Multiagentensystemen als einzige verwendet wird. Beispiele reichen von *Concurrent* METATEM als Spezifikationsrahmen auf Basis von Ausdrücke in temporaler Logik bis hin zu regelbasierten Simulationssprachen, wie *SDML*. Für eine Beschreibung dieser Ansätze sei auf die Abschnitte 2.3 und 4.4 verwiesen.

Um allerdings bei der Beschreibung des Verhaltens von Agenten – und gleichermaßen des Verhaltens ihres Umweltsystems – auch eine größere Zahl von Regeln verwenden und diese trotzdem noch übersichtlich präsentieren und repräsentieren zu können, sollte man auch die Teil-Regelmenge weiter strukturieren. Welche Mechanismen und Ansätze dabei möglich sind, wird in den folgenden Abschnitten näher dargelegt. Dazu soll zunächst betrachtet werden, auf welcher Basis eine Regelmenge grundsätzlich strukturiert werden kann, um im Folgenden dies als Ausgangspunkt für eine übersichtliche Beschreibung von Agentenverhalten zu verwenden.

5.3.1 Ansatzpunkte zur Aufteilung einer Regelmenge in einem Multiagentenszenario

Für die Strukturierung einer Regelmenge zur Verhaltensbeschreibung eines Agenten bieten sich zwei Dimensionen, an denen sich eine derartige Unterteilung der gesamten Regelmenge orientieren kann: An Aspekten, die bei der Behandlung von Regeln relevant sind oder an Kategorien, die man bei der Betrachtung von Agentenspezifika finden kann.

5.3.1.1 Ansatzpunkt Regelschema

Ausgehend vom Regelschema aus Kondition und Aktion (siehe dazu Abschnitt 3.3.3.2) kann man folgende Dimensionen für die Bestimmung von idealerweise disjunkten Teilmengen und demzufolge Indizierungsmöglichkeiten für Regeln identifizieren. Grundsätzlich ist dabei auch eine mehrdimensionale und somit weiter verfeinerte Kategorisierung der Regeln möglich.

1. Gemeinsame Aspekte in der Vorbedingung:
   Die Regeln werden dabei ausgehend von einer Situation, bzw. einem Situationsaspekt behandelt. Diese werden in der Regelkondition behandelt: Dabei werden – durch die einzelnen Prädikate – Sensoren oder Zustandsvariablen abgefragt. Demzufolge kann man eine Regel ausgehend von Teilen der Kondition an einen Sensor des Agenten oder einen Teil seines internen Zustands, also eine Zustandsvariable, knüpfen. Ändert sich der Wert dieser Zustandsvariable während der Aktualisierung oder die Wahrnehmung am Sensor zwischen zwei Aktualisierungszyklen des Agenten, kann so direkt festgestellt werden, welche Regeln davon betroffen sind, ohne die gesamte Regelmenge abzufragen. Bei – z.B. aussagenlogisch – verknüpften Aussagen in der Vorbedingung können auf diese Weise Regeln in einer netzartigen Struktur organisiert werden.

Durch diese Art der Indizierung können auch größere Regelmengen effizient behandelt werden. Das wurde schon früh erkannt (siehe dazu [Puppe 1991, Barr & Feigenbaum 1989]).

2. Regel-Nachbedingung wirkt auf gleiches Objekt:
   Eine weitere Möglichkeit besteht darin, die Regeln danach zu gruppieren, welche Aktion sie zur Ausführung auswählen, also im Grunde auf welches Objekt sie wirken. Allerdings hat dies bei einem vorwärtsverketteten Regelinterpreter, der ausgehend von der Wahrnehmung, also von den Regelvorbedingungen, die Menge der möglichen Regeln bestimmt, während der Simulation selbst nur eine beschränkte Wirkung zur Effizienzsteigerung. Andererseits ist die Bestimmung der Regeln, die zur Auswahl einer bestimmten Aktion führen, wohl die zentrale Fragestellung bei der Fehlersuche im Verhaltensmodell: Welche Regeln können für die fehlerhafte Auswahl "verantwortlich" sein? Dafür muß genau die Regelmenge bestimmt werden, die zur "falschen" Aktion führte, bzw. die Regelmenge, die die "richtige" Aktion nicht auswählte.

3. Regeln gehören zur selben "Funktionskategorie":
   Das heißt, die Regelaktionen haben einen vergleichbaren Effekt bzw. die Vorbedingungen eine ähnliche Struktur. Dies ist nur scheinbar ein relativ uninteressanter Ansatz, da die beiden obigen Gruppierungsparadigmen zu einer deutlich feineren Strukturierung führen können. Kann man aber die Auswahl der nächsten Aktion in mehrere Phasen unterteilen und für jede dieser Phasen eine separate Regelmenge und eine spezialisierte Form der Regelbehandlung identifizieren, macht eine derartige Strukturierung durchaus Sinn. Auf diese Weise kann die Übersichtlichkeit der Regelmenge deutlich erhöht werden. So wäre es zum Beispiel überraschend, wenn in *Soar* (siehe Seite 24) der Regelspeicher nicht nach den einzelnen Regelfunktionen – Wähle Problemraum; Suche Aktion, Situation oder Präferenz – organisiert wäre, auch wenn diese Unterteilung in keinem wissenschaftlichen Papier dokumentiert ist.

Auf der Basis dieser grundsätzlichen Überlegungen werden im Folgenden nun Konzepte zur Strukturierung einer Regelmenge für die Verhaltensbeschreibung von Agenten entwickelt. Aspekte der konkreten Umsetzung werden dabei im nächsten Kapitel erläutert.

5.3.1.2 Ansatzpunkt Agentenaktualisierung

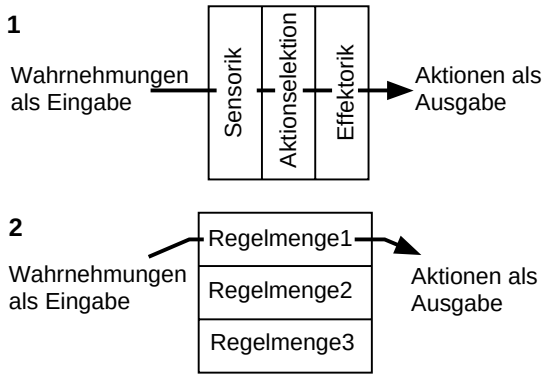
Für die Beschreibung von Agentenverhalten gibt es innerhalb einer Regelmenge zwei Vorgehensweisen bei deren Unterteilung:

- "horizontal"
  Alle betrachteten Regeln haben die selbe Funktion, die Unterteilung geschieht entsprechend betroffener Modellbestandteile, wie "Rollen" oder "Aktivitäten" (Abschnitt 5.3.2).

- "vertikal"
  Der Aktionsauswahlprozeß wird in Phasen unterteilt, die nacheinander behandelt werden: Sensorik, Aktionsauswahl und Effektorik, (Abschnitt 5.3.3).

Abbildung 5.4 verdeutlicht diese unterschiedlichen Ansätze zur Unterteilung nochmals.

Abbildung 5.4 Verdeutlichung "vertikaler" (1) und "horizontaler" (2) Strukturierung der verhaltensbeschreibenden Regelmengen.



5.3.2 Horizontale Aufteilung: Aktivitäten und Verhaltenskategorien

5.3.2.1 Rollen

Aus Überlegungen, die bei der Spezifikation von Organisationen in Agentensystemen mit Arbeitsteilung angestellt werden, ergibt sich direkt ein naheliegender Ansatzpunkt zur Gruppierung von verhaltensbeschreibenden Regeln: Es werden dabei jene Regeln zusammengefaßt, die zu einer bestimmten Rolle des Agenten gehören. In einer derartigen Teilmenge finden sich also Regeln, deren Situations-Aktions-Verknüpfung charakteristisch für einen Agenten gesehen werden kann, der diese Rolle einnimmt. Eine derartige "Verhaltenskategorie" muß dabei nicht nur einem bestimmtem Aufgabentypus entsprechen, wie sie z.B. in einer Firma mit einer bestimmten Stellenbeschreibung assoziiert werden kann, sondern kann auch zur Repräsentation eines Entwicklungsstadiums des Agenten in einem biologischen Modell benutzt werden. Ein Agent soll dabei in seinem "Leben" unterschiedliche Rollen annehmen können, bzw. sein Verhalten durch verschiedene Verhaltenskategorien kontrolliert werden können. Eine Rolle sollte demnach gewechselt werden können, um unterschiedlichen Handlungskontexten des Agenten Rechnung zu tragen.

Dabei ist fraglich, ob ein Agent gleichzeitig über mehrere aktive Verhaltenskategorien verfügen soll oder nur eine eindeutige Kategorie für das aktuelle Verhalten zuständig sein soll. Um die Notwendigkeit komplexer Mechanismen für die Konfliktauflösung zu vermeiden, soll im folgenden ein Agent durch genau eine Rolle mit entsprechender Regelmenge beschrieben werden können. Eine hierarchische Organisation der Rollen, Spezialisierung und Defaultverhaltensregeln ist dabei sinnvoll, um die Strukturierung weiter zu unterstützen: Abstraktere Rollen definieren das Defaultverhalten und "vererben" dieses an spezialisiertere Kategorien. Ein Konfliktauflösungsmechanismus für gleichzeitig aktivierbare Regeln aus unterschiedlich abstrakten, gleichzeitig "aktiven" Rollen ist dabei nicht notwendig. Das speziellere Rollenverhalten überschreibt das allgemeinere.

Diese Unterteilung der Regelmenge kann man, wenn man Rollen explizit repräsentiert, auch zur Indizierung der entsprechenden Regeln verwenden. Solange der Agent entsprechend einer Rolle handelt, werden nur die entsprechenden Regeln abgetestet. Speichert man

die aktuelle Rolle eines Agenten als Wert einer Zustandsvariablen seines internen Zustands, kann man diese Vorgehensweise auf den oben skizzierten Mechanismus der Indizierung über gemeinsame Vorbedingungsaspekte (1) zurückführen: Jede Regel der entsprechenden Teilmenge enthält dabei in der Vorbedingung folgende zusätzliche Einzelbedingung: WENN "der Agent entsprechend Rolle xy handelt". Auf diese Weise kann die Regelmenge, die aktuell das Verhalten des Agenten beschreibt, durch den Wert einer Zustandsvariable charakterisiert werden – bzw. den Wert einer Zustandsvariable und zusätzlichem Wissen über Vererbungsstrukturen von Rollen.

Derartige Verhaltenskategorien können insbesondere auf einer konzeptionellen Ebene als ein wichtiges Hilfsmittel betrachtet werden. So wird die Identifikation von Rollen in einer arbeitsteiligen Gemeinschaft oft als wichtige Phase im agenten-orientierten Software-Design verwendet (siehe dazu Abschnitt 2.3).

Über das Konzept der Rollen könnte sich die verhaltensbeschreibende Regelmenge im Prinzip schon bis zu einer sehr feinen Struktur organisieren lassen. Mit aussagekräftigen Namen – also modellspezifischen Bezeichnungen – versehen, kann so die Regelmenge transparent dargestellt werden und das Navigieren in der Gesamtregelmenge unterstützt werden. Dennoch sollte man beachten, daß eine "Rolle" eher mit einem abstrakten Verhaltenstypus bzw. groben Aufgabenbeschreibung assoziiert wird.

5.3.2.2 Aktivitäten

In Kombination mit der vertikalen Einteilung der Regelmenge auf der Basis der Phasen des Aktualisierungszyklus ist ein weiterer Strukturierungsansatz sinnvoll. Ausgangspunkt für die Gruppierung und demzufolge Indizierung der Regeln sind dabei die Aktionen, die der Agent auswählen bzw. ausführen kann.

Die Grundlage bilden dabei folgende drei Annahmen oder Beobachtungen: Zum einen ist dies die Festlegung, daß eine vom Agenten ausgewählte Aktion quasi ohne Zeitdauer ausgeführt werden kann – also genau bzw. maximal einen Zeittakt bis zum nächsten Anstoß einer Aktualisierung dauert. Andererseits ändert sich bei der Ausführung nur ein Teil des Situationskontextes, in dem sich der Agent befindet. Demzufolge ist eine erneute Auswahl der gerade ausgeführten Aktion – bereits bei der nächsten Aktualisierung – sehr wahrscheinlich. Ein gutes Beispiel ist eine schrittweise "Bewegung" eines Agenten während des "Absuchens" eines Areals. In jedem Takt würde immer wieder "Mache x Schritte" gewählt, bis die Situation andere Aktionen erfordern würde. Als dritte Beobachtung kann man folgende typische Situation anführen: Der Agent soll in einer bestimmten Reihenfolge Aktionen auswählen. Beispielsweise kann nach einer Bewegung eine zweite Aktion "Markiere die aktuelle Position" notwendig sein. Ein Verhalten mit diesen Eigenschaften kann durch eine entsprechend formulierte Regelkette dargestellt werden, allerdings muß der Modellierer dabei Reihenfolgeabhängigkeiten, mögliche Interferenzen, usw. im Blick behalten. Das ist bei dem einfachen Beispiel unproblematisch, kann aber bei komplexeren "Tätigkeiten" der Agenten durchaus schwierig werden.

Eine naheliegende, attraktive Erweiterung der rein regelbasierten Verhaltensbeschreibung liegt demnach in der Einführung von expliziten "Aktivitäten"[3]. Eine derartige Struktur

[3]Der Name "Aktivität" für einen Aktions-Container wurde hier vor allem wegen der umgangssprachlichen Bedeutung gewählt – man verbindet damit üblicherweise ein "sich in einer bestimmten Weise betätigen". In der ereignisbasierten Simulation werden "Aktivitäten" dagegen verwendet, um ein abstrakteres Bild bei der Erzeugung einer Folge von Ereignissen verwenden zu können [Monsef 1997]. Ein Ereignis entspricht dabei generell

fasst alle Aktionen zusammen, die ein Agent gleichzeitig ausführen sollte. Diese Aktionen werden dabei in einer Sequenz geordnet und immer wieder ausgeführt, solange der Agent sich in dieser Aktivität befindet. Dieses Konzept hat nicht nur den Vorteil, daß Verhalten, das oben charakterisiert wurde, auf eine naheliegende Weise dargestellt werden kann. Wichtig ist zudem die Möglichkeit, eine detaillierte Zeitsteuerung formulieren zu können: Die Ausführung einer Aktion ist nicht mehr auf genau einen Takt festgelegt, sondern eine Menge von Aktionen wird einem Takt zugeordnet. Der Modellbauer erhält so mehr Flexibilität bei der Beschreibung von Verhalten mit zeitlichen Unterschieden. Dadurch, daß mehrere Aktionen in einem Takt ausgeführt werden können, ist es möglich, je nachdem wieviele und welche Aktionen in einer Aktivität zusammengefasst werden, Verhaltensweisen mit feineren Unterschieden bei der Ausführungsdauer zu modellieren.

Solange sich der Agent also in einer Aktivität befindet, führt er immer wieder – in jedem Takt – alle mit dieser Aktivität assoziierten Aktionen aus. Aktivitäten werden dabei über Regeln ausgewählt und terminieren, wenn eine Regel feuert, die eine andere Aktivität selektiert. Man kann so z.B. die Aktivität “Essen” durch eine ständige Wiederholung der primitiven Aktionen “Nahrung auf Gabel spießen”, “Gabel zum Mund führen”, usw. darstellen und die für die Auswahl jeder einzelnen Aktion notwendigen Regeln weglassen. Die einzelnen Aktionen können auch bei anderen Aktivitäten sinnvoll sein: Die Aktion “Nahrung auf Gabel spießen” kann dabei auch in einer Aktivität “Kochen” verwendet werden.

Durch die Festlegung, daß Aktivitäten persistent sein sollen, d. h. länger als einen Aktualisierungszyklus angenommen werden, kann man die aktuelle Aktivität auch als eine Art Zustandsvariable mit besonderer Bedeutung betrachten. Diese hat dabei nicht nur die Funktion einer Ausgabevariable – über sie werden direkt die auszuführenden Aktionen bestimmt – sondern kann auch in Prädikaten der Regelvorbedingung abgefragt werden.

Auf diese Weise wird eine Unterteilung der Menge einzelner Aktionen über eine potentielle Zusammengehörigkeit erreicht, die gleichzeitig eine abstraktere Sicht auf das Agentenverhalten zuläßt. Jede Aktivität enthält einen definierten Satz von konkreten Aktionen. Eine solche Zusammenfassung von möglichen Agentenaktionen ist hilfreich, wenn man ein komplexes Verhaltensmodell konstruiert. Dies gilt insbesondere, wenn viele annähernd gleiche Aktionen in unterschiedlichem Kontext durchgeführt werden sollen. In diesem Fall wird ein Überblick über die Situationen, in der eine konkrete Aktion ausgeführt wird, immer schwieriger und die Formulierung der Konditionen undurchsichtiger. Aktivitäten stellen also eine echte Erweiterung des bisher vorgestellten Rahmenwerks zur Verhaltensbeschreibung dar. Dazu ist die Erläuterung einiger Details, wie Struktur und Terminierung einer Aktivität notwendig. Wegen der Persistenz von Aktivitäten ist zudem der Begriff der “Unterbrechung” notwendig. Diese Aspekte werden im folgenden näher dargestellt.

Terminierung von Aktivitäten und Regelindizierung. Wie oben bereits eingeführt, sind Aktivitäten persistent, d. h. sie müssen nicht in jedem Aktualisierungszyklus neu bestimmt werden. Stattdessen legen sie solange die Aktionen des Agenten fest, bis eine Situation eintritt, in der eine andere Aktivität besser ist. Wenn also eine Regel feuern kann, wird eine andere Aktivität ausgewählt. Die Aufgabe einer Regel besteht nun darin, eine Aktivität auszuwählen, bzw. eine Aktivität zu beenden. Demzufolge hat man zwei Alternativen, diese

einer Zustandsänderung. Betrachtet man das System auf der Basis derartiger Aktivitäten, dann wird ein Ereignis erzeugt, wenn eine bestimmte Aktivität beginnt und ein weiteres, wenn diese wieder endet. Dieses Ziel und das Konzept einer Aktivität ist durchaus mit dem hier verwendeten ähnlich. Die Differenzen gründen hauptsächlich auf der unterschiedlichen Behandlung von Zeit.

Selektionsregeln mit Aktivitäten zu verknüpfen und auf diese Weise einen Mechanismus zur Regelindizierung zu nutzen: An die Aktivität in der Regelnachbedingung oder an eine Aktivität in der Regelvorbedingung.

Speichert man eine Regel bei der Aktivität, die durch diese ausgewählt wird, erhält man eine eindeutige Indizierung, da jede Regel genau eine Aktivität selektiert. Abhängig von der Interpretation der Regeln ermöglicht dies aber nur eine beschränkte Verkleinerung der in jedem Zyklus zu testenden Regeln: Bei der oben vorgestellten Terminierung, wenn eine andere Aktivität ausgewählt wird, ist dies der Fall, da alle Regeln, die andere Aktivitäten auswählen, betrachtet werden müssen. Bei einer alternativen Interpretation für die Terminierung einer Aktivität – eine Aktivität terminiert, wenn sie durch ihre Auswahlregeln nicht mehr selektiert wird – ist so die Regelmenge zur Bestimmung, ob eine Aktivität endet, so klein wie möglich, allerdings müssen im nächsten Schritt wiederum alle anderen Regeln getestet werden, um eine andere Aktivität auszuwählen.

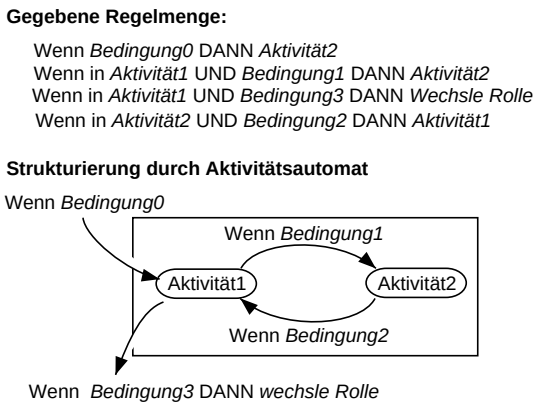
Demzufolge ist eine Indizierung auf der Basis einer (eindeutigen) Aktivitätsabfrage in der Vorbedingung einer Auswahlregel sinnvoll. Das hat allerdings den Nachteil, daß nicht alle Regeln indiziert werden können. Kommen in der Vorbedingung Aussagen, wie “Agent ist in Aktivität X oder in Aktivität Y?” vor oder ist die Regelvorbedingung gänzlich unabhängig von der aktuellen Aktivität, kann die Regel keiner Aktivität zugeordnet werden. Wenn aber ein mit anderen Situationsaspekten mittels Konjunktion verknüpfte Aktivitätsvergleich (“Agent ist in Aktivität X?” und...) verwendet wird, ist eine Indizierung auf dieser Basis besonders sinnvoll, da diese Regel generell nur dann feuern kann, wenn der Agent in der entsprechenden Aktivität ist. Geht man davon aus, daß sehr viele Regeln diese Struktur besitzen, und nur wenige Regeln unabhängig von einer einzigen Aktivität sind, kann man annehmen, daß die abzutestende Regelmenge relativ klein ist, wenn man auf diese Weise die Auswahlregeln anordnet.

Durch die Zuordnung von Regeln zu Aktivitäten in ihren Vorbedingungen ergibt sich eine besondere Metapher für die Darstellung des Agentenverhaltens innerhalb einer Verhaltenskategorie. Weil diese Regeln wiederum eine Aktivität auswählen, kann man durch eine Regel quasi den Übergang von einer Aktivität zu ihren Folgeaktivitäten darstellen. Derartige Strukturen lassen sich in einem Aktivitätsautomat abbilden – obwohl nicht alle Regeln daran beteiligt sind und der Graph nicht zusammenhängend sein muß. Aktivitäten entsprechen dabei den Knoten und Regeln zwischen den Aktivitäten den Kanten. Abbildung 5.5 erläutert diese Metapher.

Zudem ist die Verknüpfung mit Vorbedingungsobjekten die Form der Indizierung, die bei einer vorwärtsverketteten Behandlung der Regeln sinnvoll ist (siehe Seite 109), um eine Effizienzsteigerung bei der Regelinterpretation zu erreichen.

Durch diese Form der eingeschränkten Indizierung – es werden Regeln an Aktivitäten geknüpft, die ausschließlich dann feuern, wenn der Agent sich in dieser Aktivität befindet – kann man zwei Arten von Regeln identifizieren: Indizierbare und nicht-indizierbare Regeln. Durch diese Strukturen können nur die Regeln betrachtet werden, die wirklich für die Terminierung der aktuellen Aktivität relevant sind: Das sind zum einen Regeln, die bei der aktuellen Aktivität indiziert sind – im folgenden “Terminierungsregeln” genannt, da sie beim Feuern genau eine bestimmte Aktivität beenden. Zum anderen handelt es sich um Regeln, die keiner Aktivität zuzuordnen sind. Deren Zahl ist abhängig vom Verhaltensmodell oft sehr klein. Durch eine entsprechende Formulierung der Regeln kann der Modellierer somit entscheiden, ob eine Regel eine Aktivität “terminiert” oder in jedem Zyklus abgetestet werden soll. So kann eine möglicherweise beim Modellierer vorhandene konzeptionel-

Abbildung 5.5 Aus einer gegebenen Menge von Aktivitätsauswahlregeln und Aktivitäten wird ein "Aktivitätsautomat" erzeugt: Das Testen von Regeln, die nur wahr sein können, wenn der Agent in einer bestimmten Aktivität ist, können mit dieser Aktivität verknüpft werden.



le Vorstellung des Verhaltens als Automat direkt umgesetzt werden. Die Frage nach den nicht-indizierbaren Regeln wird weiter unten bei den Betrachtungen zur Reaktivität des so beschriebenen Agentenverhaltens nochmals aufgegriffen. Um dem Konzept der Aktivität als Container für Agentenaktionen besser zu entsprechen, wird im folgenden diese Tätigkeitsrepräsentation um Aspekte erweitert, die zum einen die Sequenzierung der Aktionen vereinfacht, zum anderen durch Integration von mehr Wissen über die Anwendbarkeit einer Aktivität deren Modellierung, insbesondere die Validation, bzw. das Debugging bei Fehlern erleichtert.

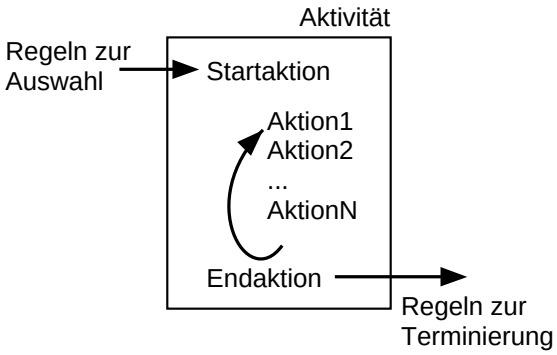
Interne Struktur einer Aktivität. Um dem Konzept einer abstrakteren Tätigkeit besser zu genügen, das mit "Aktivität" verbunden werden kann, kann man zu der in jedem Aktualisierungszyklus ausgewählten Aktionenmenge zwei weitere Aktionen hinzufügen: Eine der beiden, die Startaktion, dient dazu die eigentliche Aktivität vorzubereiten, indem sie ausgeführt wird, wenn die Aktivität durch eine Regel als aktuelle Aktivität selektiert wird. Die "Endaktion" besitzt eine analoge Funktion: Wenn eine Regel feuert, mit der eine neue Aktivität ausgewählt wird, soll durch das Ausführen dieser Aktion ein Aufräumen geschehen. Abbildung 5.6 visualisiert diese einfache interne Struktur.

Trotz ihrer Einfachheit hat der Modellierer dadurch die Möglichkeit, Aktivitäten auch wirklich als abstrakte verhaltensbeschreibende Einheiten zu betrachten und kann "gleichwertigere" Aktivitäten formulieren, statt für Vorbereitungs- und Aufräumarbeiten wiederum Aktivitäten formulieren zu müssen.

Dabei betrachtet man eine Aktivität nicht mehr als einfache sortierte Menge von Aktionen, sondern weist ihr mehr die Bedeutung einer explizit repräsentierten Datenstruktur zu. Dabei können und sollen weitere sinnvolle Aspekte hinzugefügt werden. Aktivitäten können benannt und Bedingungen, die für die generelle Auswählbarkeit[4] erfüllt sein müssen,

[4]Damit sind abstrakte Bedingungen für die gesamte Aktivität gemeint. Für die Ausführbarkeit einzelner

Abbildung 5.6 Schema der internen Struktur einer Aktivität



zur Aktivität formuliert werden. Dieses Wissen kann die Validation eines durch Aktivitäten strukturierten Verhaltensmodells erheblich erleichtern. Auch bei der Fehlersuche können Informationen, in welchem Kontext während der Simulation eine Aktivität ausgewählt wurde, obwohl ihre Nebenbedingungen nicht erfüllt waren, sehr hilfreich sein. So kann man feststellen, welche impliziten Annahmen über den Kontext der Auswahl und Ausführung der Aktivität nicht zutrafen.

Aktivitäten und Reaktivität: zur Bedeutung nicht-indizierbarer Regeln. Die Persistenz einer Aktivität des Agenten über mehrere Aktualisierungszyklen hinweg widerspricht der Forderung nach Reaktivität des Agentenverhaltens: Ein Agent soll dabei rechtzeitig auf Veränderungen seiner Umwelt reagieren können, also Dynamik in der Umwelt wahrnehmen und sein Verhalten, d.h. seine aktuelle Aktivität dementsprechend ausrichten können. Es steht zu befürchten, daß – während der Agent in einer Aktivität verharrt – eigentlich relevante Regeln nicht in das Reasoning zur Auswahl der nächsten einbezogen werden könnten.

Regeln, die die Reaktivität des Verhaltens sichern, bzw. die einfache Formulierung von Reaktionen auf Umweltveränderungen – aber auch Reaktionen auf unerwartete Effekte von Aktionen – ermöglichen, sind diejenigen, deren Vorbedingung nicht an eine eindeutige Aktivität geknüpft sind. Man kann also diese nicht-indizierbaren Regeln als “Notfall-Regeln” interpretieren. Mit diesen Regeln werden also Verhaltensänderungen in Situationen beschrieben, die für den Agenten in jeder Aktivität relevant sind. Ein prominentes Beispiel in biologischen Szenarien wäre folgendes: Wenn ein simuliertes Tier bedroht wird, sollte es in eine “Flucht”-Aktivität oder ähnliches wechseln, unabhängig davon, welche Aktivität gerade aktiv ist.

Durch diese Interpretation von nicht-indizierten Regeln im Sinne der notwendigen Reaktivität des Agentenverhalten wird dem Modellierer ein weiteres Paradigma zum Aktivitätsautomaten zur Verfügung gestellt. Dieses Schema entspricht im Grunde der üblichen Gleichbehandlung aller Regeln in der Regelmenge, also einer klassischen Stimulus-Response-Architektur. Dieses Bild läßt sich auch mit dem Aktivitätsautomaten kombinieren: Eine Notfallregel würde jeden Aktivitätsknoten im Automaten mit der durch diese Regel auswählbaren Aktivität verbinden.

Aktionen sei auf die Betrachtungen zur vertikalen Unterteilung der Regelmenge in Abschnitt 5.3.3 verwiesen.

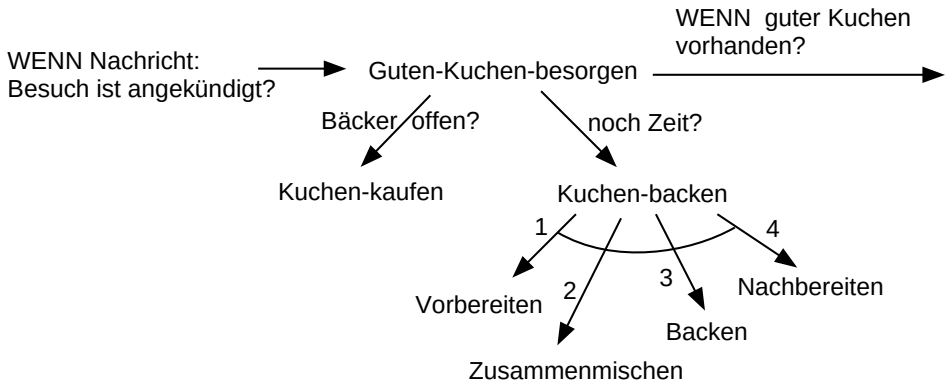
5.3.2.3 Strukturierung von Aktivitäten

Auch wenn die Einführung von Aktivitäten die zur Verhaltensbeschreibung notwendige Regelmenge – nicht die Menge der möglichen Aktionen des Agenten – verringert, kann das Verhaltensmodell eines Agenten insgesamt trotz des Bildes eines Aktivitätsautomaten unübersichtlich werden.

In Abschnitt 5.3.2.1 wurde das Konzept einer "Rolle" eingeführt, um die gesamte Regelmenge nach Verhaltenskategorien zu unterteilen. Kombiniert man diese Struktur im Sinne der Aktivitäten, kann man durch eine Verhaltenskategorie zweierlei erreichen: Eine Unterteilung der Aktivitätenmenge und eine Indizierung der Notfallregeln. Das bedeutet, solange ein Agent sich entsprechend einer Rolle verhält, sind für ihn die bei dieser Verhaltenskategorie eingetragenen Notfallregeln relevant. Das ist – im Vergleich zur Relation zwischen Aktivitäten und Notfallregeln – schon eine deutlich Einschränkung bei der Formulierung von reaktivem Verhalten, allerdings ist so die Indizierung aller Regeln möglich. Darauf aufbauend ist es sinnvoll, Aktivitäten entsprechend einer Rolle zu gruppieren. Abhängig vom Modell kann diese Assoziation von Aktivitäten zu Rollen eindeutig sein. Unabhängig davon wählen auf diese Weise Notfallregeln nur Aktivitäten innerhalb der Verhaltenskategorie aus, mit der sie verknüpft sind.

Analog zur Gruppierung und Sequenzialisierung von Aktionen können auch bei der Formulierung der Aktivitäten vom Modellierer eigentlich intendierte Zusammenhänge zwischen Aktivitäten explizit gemacht und so für die weitere Strukturierung der Verhaltensbeschreibung innerhalb einer Rolle genutzt werden: Durch abstrakte Aktivitäten werden konkretere Tätigkeiten in einer Sequenz oder als direkt "konkurrierende" Alternativen organisiert. Abbildung 5.7 illustriert dies anhand einer "Kuchen-Besorgen"-Aktivität. Während der Agent diese Aktivitäten verfolgt, werden in jedem Aktualisierungszyklus zudem die nichtindizierten Regeln, in denen z.B. auf die Situationsänderung "Besuch wird abgesagt" durch die Auswahl neuer Aktivitäten reagiert werden kann. Die aktuelle Aktivität des Agenten besteht dabei aus dem Pfad zu dem gerade ausgeführten Blatt-Tätigkeit.

Abbildung 5.7 Beispiel für die Organisation von Aktivitäten durch konkurrierende und sequenzierte Aktivitäten. Die beiden konkurrierenden "Kuchen kaufen" und "Kuchen backen" werden dabei solange ausgewählt, bis die Terminierungsregel für "Guten-Kuchen-Besorgen" feuern kann.



Aktivitätssequenzen. In direkter Analogie zu der Zusammenfassung von Aktionen zu Aktivitäten kann man die Kombination von Aktivitäten zu Aktivitätssequenzen sehen. Das bedeutet, wenn die Aktivitätssequenz als ganzes als aktuelle (abstrakte) Aktivität gewählt wird, wird zugleich die erste Aktivität aktiviert – usw. bis eine Aktivität mit "echten" Aktionen gefunden wird. Diese Aktivität bleibt solange aktiv, bis die ihr zugeordneten Terminierungsregeln feuern. Die Aktion dieser Regeln besteht dabei per default aus der nächsten Aktivität in der Sequenz. Mit der letzten Aktivität terminiert auch die Sequenz – außer die abstrakte Aktivität wird zuvor durch die bei ihr indizierten Regeln oder Notfallregeln beendet. Auf diese Weise wird zwar die Menge der Regeln nicht verkleinert, allerdings ihre Organisation durch das Bild einer Sequenz erleichtert.

Aktivitäten als konkurrierende Alternativen. Eine abstrakte Aktivität kann nicht nur aus nacheinander aktiven Tätigkeiten bestehen, sondern auch durch mehrere Alternativen umgesetzt werden, wie dies im obigen Beispiel beim Besorgen von gutem Kuchen der Fall ist. Während dabei die abstrakte Aktivität das Verhalten des Agenten bestimmt, werden – über eine dadurch beschränkte Regelmenge – Aktivitäten ausgewählt, die danach wiederum verfeinert werden können. Diese Tätigkeiten sind solange aktiv, bis ihre Terminierungsregeln feuern oder die übergeordnete Aktivität beendet werden sollte. Endet eine Alternative, die übergeordnete Aktivität aber nicht, wird eine andere Alternative ausgewählt – solange bis auch die übergeordnete Tätigkeit terminieren kann, also die in ihren Terminierungsregeln formulierte Situation eingetreten ist.

Skelettplan-Metapher in Kombination mit Aktivitätsautomaten. Durch Zusammenfassung dieser beiden Möglichkeiten der Aktivitätssequenzen und Aktivitätsalternativen entstehen Und-Oder-Bäume, – vergleichbar mit Skelettplänen – an deren Blättern sich Aktivitäten mit ausführbaren Aktionen befinden. Wie oben erwähnt, besteht der aktuelle Wert der Zustandsvariable, die die Aktivität speichert, aus dem Pfad in diesem Baum. Allerdings gibt es zwei wichtige Unterschiede zu Standard-Skelettplänen (siehe dazu [Puppe 1990]): Die wiederholte Auswahl von Alternativen bei "Oder"-Knoten und die Festlegung der Reihenfolge bei "Und"-Knoten.

Die Organisation von Aktivitäten in Und-Oder-Bäumen bildet eine Metapher für die Modellierung von Verhaltensweisen, die gut durch das Verfolgen eines vorgegebenen Plans dargestellt werden können. Man könnte sogar die Vorbedingungen von Regeln, die derartige Aktivitäten terminieren als Beschreibung von Zielsituationen interpretieren, wenn man diese Aktivitäten mit Zielverhalten assoziiert.

Allerdings bleibt die Behandlung dieser Aktivitätsbäume streng vorwärts verkettet. Dennoch bildet diese Aktivitätshierarchie-Struktur ein Mittel, Verhalten auch "top-down" zu beschreiben und direkt in vorgegebene Strukturen umzusetzen. Diese hierarchische Aktivitätenstruktur – gerade in Kombination mit ihrer regelbasierten Auswahl und Terminierung – besitzt Ähnlichkeiten mit verschiedenen Beschreibungsformalismen für vorgegebene Pläne reaktiver Planer. Besonders zu erwähnen ist die Ähnlichkeit mit den RAP ("Reactive Action Packages" siehe Seite 21) und den "Knowlegde Areas", den Planbeschreibungen der PRS-Architektur (siehe Seite 22). In Abschnitt 5.3.5 wird die Beziehung zu vorhandenen Methoden und Strukturen näher erläutert.

Zunächst sollen jedoch – um die Vorstellung der Möglichkeiten für die Strukturierung einer regelbasierten bzw. aktivitätsbasierten Verhaltensbeschreibung vollständig zu machen,

auch die Ansätze, die sich aus einer vertikalen Aufteilung ergeben, vorgestellt werden.

5.3.3 Vertikale Aufteilung: Phasen der Aktualisierung

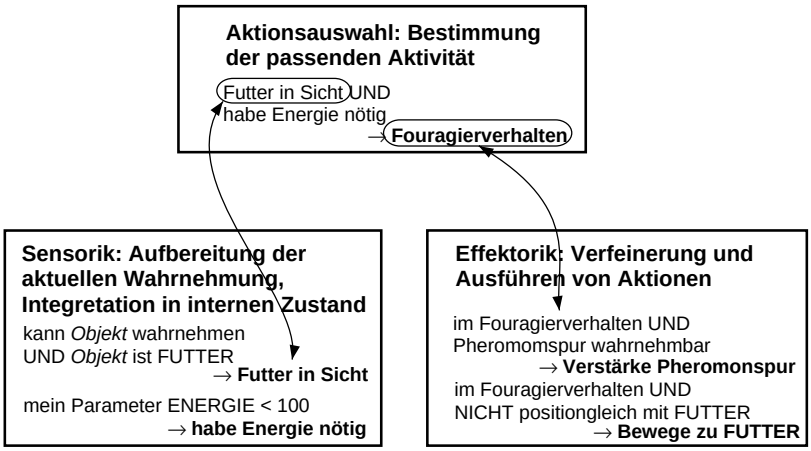
Ein weiterer Ansatz zur Strukturierung der verhaltensbeschreibenden Regelmenge für einen Agenten ergibt sich aus der genaueren Betrachtung der Phasen eines Aktualisierungszyklus.

5.3.3.1 Sensorik-Auswahl-Effektorik

Wie auf Seite 98 beschrieben, lassen sich bei der Aktualisierung eines Agenten drei Phasen unterscheiden: Wahrnehmung und deren Aufbereitung in $UPDATE-ZUSTAND$, Auswahl der nächsten Aktion, bzw. passenden Regel in $REGEL-AUSWAHL$ und als drittes Bestimmung und Ausführung dieser Aktion durch $REGEL-AKTION$. Beim zweiten Aufruf der Funktion $UPDATE-ZUSTAND$ mit der gewählten Aktion wird bestimmt, wie der interne Zustand nach der Ausführung der Aktion aussieht. Diesen letzten Funktionsaufruf kann man in die Ausführung der Aktion integrieren. Auf diese Weise ergibt sich eine "klassische" Dreiteilung für die Modellierung von Agentenverhalten in Wahrnehmung – Aktionsauswahl – Aktionsausführung.

Anstatt diese drei Phasen in einer einzigen Ebene von Regeln abzubilden, die ausgehend von Daten aus der Sensorik auf die zu tätigenden Aktionen zu schließen, kann man für jede dieser Teilphasen eine getrennte Regelmenge formulieren und diese kaskadenförmig anordnen. Gerade das oben vorgestellte Konzept einer Aktivität bildet eine wichtige Grundlage dafür, diese Unterteilung konsequent vornehmen zu können. Durch die Kombination von horizontaler und vertikaler Strukturierung ergibt sich ein interessantes Gesamtkonzept zur Verhaltensbeschreibung bei Agenten. Abbildung 5.8 liefert einen ersten Eindruck.

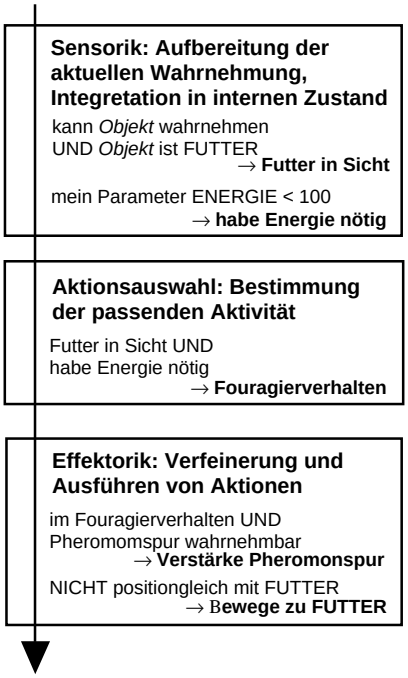
Abbildung 5.8 Die drei Phasen eines Aktualisierungszyklus für einen Agenten können jeweils durch unterschiedliche Regelmengen formuliert werden.



Allerdings müssen diese Schritte nicht unbedingt nacheinander, bzw. synchronisiert ablaufen. In bestimmten Anwendungsszenarien – gerade bei reaktiven Systemen – kann es durchaus sinnvoll sein, gleichzeitig und permanent die Umwelt zu beobachten und "Pläne" ausarbeiten, während die Effektoren noch vorhandene Kommandos abarbeiten.

Dies führte zur Entwicklung von sogenannten hybriden Agentenarchitekturen, die ab Seite 21 erläutert werden. Dadurch daß in einem Multiagentenmodell die Umwelt vollständig vom Modellierer spezifiziert wird, können hier vereinfachte Forderungen an die Reaktivität des Agentenverhaltens gestellt werden und im folgenden eine sequentielle Abarbeitung angenommen werden. Dadurch wird die komplizierte Synchronisierung asynchron arbeitender Sensorik-, Auswahl- und Effektorikprozesse vermieden. Der Aktualisierungsprozess insgesamt bleibt so einfach wie möglich.

Abbildung 5.9 Die drei Phasen eines Aktualisierungszyklus für einen Agenten können auch bei der Verwendung unterschiedlicher Regelmengen nacheinander ablaufen.



Auf diese Weise wird die Einteilung in Aktualisierungsphasen nur benutzt, um die Regelmenge zur Verhaltensbeschreibung eines Agenten besser handhabbar zu machen. Im folgenden wird dies bei der genaueren Behandlung der einzelnen Teil-Regelmengen und der dabei resultierenden Anforderungen an die Struktur der einzelnen Regeln deutlich:

5.3.3.2 Aktivitätsauswahl

Die zentrale Rolle spielen dabei die Regeln zur Auswahl der nächsten Aktivität. Durch die so geschaffene Verbindung zwischen Sensorik und Effektorik wird das eigentliche Verhalten bestimmt. Demzufolge liegt der Sinn der Unterteilung in drei Regelmengen genau darin, diese Regelmenge so einfach und übersichtlich wie möglich zu halten.

Dies ist wegen folgender Ansatzpunkte problemlos möglich: Im vorgeschalteten Aufbereitungsschritt der Wahrnehmung werden Fakten und Abstraktionen über Abfragen und Vergleiche von Werten interner Zustandsvariablen zugänglich gemacht. So kann die Struktur der Konditionen der Auswahlregeln fast ausschließlich auf einfache aussagenlogische Verknüpfung von Prädikaten beschränkt werden, die auf ein “Nachschauen” bei internen

Zustandsvariablen reduzierbar sind. Durch die Möglichkeit, Aussagen mit Vergleichsoperatoren zwischen Zustandsvariablen zu formulieren, können dadurch viele relevante Situationen beschreiben werden. Da allerdings auch die Vorverarbeitung in der "Sensorik-Phase" nicht unverhältnismäßig kompliziert sein sollte, muß man doch zumindest eine beschränkte Erweiterung in Richtung Prädikatenlogik vornehmen, indem man Terme mit Aussagen über die Werte aller Zustandsvariablen des internen Zustands des Agenten und Existenzaussagen über Werte von Zustandsvariablen formulierbar macht. Dadurch daß die Menge der Zustandsvariablen, die für die Darstellung des internen Zustands zur Verfügung stehen, im Modell beschränkt und explizit ist, ist dies in einem Simulator mit einfachen Mitteln wie Abfrage-Mechanismen realisierbar und auf Aussagenlogik abbildbar. Eine andere Möglichkeit, die Sensorik-Phase auf Kosten der Aktivitätsauswahlregeln einfach zu halten, besteht darin, Werte von Zustandsvariablen mit Zeitstempeln zu versehen und auf dieser Basis Prädikate mit der Semantik punkt-basierter temporaler Logik zu integrieren. Eine Umsetzung wäre wegen der Form der Zustandsrepräsentation relativ einfach in dieses Konzept integrierbar, aber technisch sehr aufwendig und ineffizient. Deshalb wird die Möglichkeit der Formulierung derartiger Ausdrücke nicht weiter betrachtet. Die Darstellung der relativ umfangreichen bisherigen Anwendungen (siehe Abschnitt 8), bei denen die Formulierung derartiger Ausdrücke nicht notwendig war, muß als Begründung reichen, um diese Unterlassung zu rechtfertigen.

Wegen der auf die Aktivitätsauswahl folgenden expliziten Phase der Aktions- bzw. Aktivitätsausführung können auch die Regelaktionen einfach strukturiert werden: Die Effekte des Feuerns einer Regel können darauf reduziert werden, daß eine Verhaltensweise aus einer Menge vorgegebener Aktivitäten ausgewählt wird[5]. Durch den nachfolgenden Verfeinerungsschritt kann auf aufwendige Parametrisierungen und Variablenbindungen verzichtet werden. Die Struktur dieser Aktivitätsauswahlregeln ist auch aus diesem Grunde relativ einfach. Allerdings spielt dabei die Konfliktauflösung eine wichtige Rolle, die die Aktivitätsauswahl komplizierter gestalten kann:

Können auf der Basis ihrer Vorbedingungen – trotz der schon durch die diversen Indizierungsschemata verkleinerten Regelmenge – mehrere Aktivitätsauswahlregeln gleichzeitig feuern, gibt es grundsätzlich zwei Möglichkeiten: Alle Regeln feuern zu lassen oder eine passende Regel auszuwählen. Es wäre durchaus möglich, alle selektierbaren Verhaltensweisen gleichzeitig zu aktivieren. Kann man allerdings Interferenzen zwischen den verschiedenen, gleichzeitig aktiven Verhaltensweisen nicht durch eine geschickte Formulierung der Regeln vermeiden, verlagert man die Lösung dieser Konflikte auf die nächste Phase, die der Verfeinerung und Ausführung dieser Aktivitäten. Dadurch wird die Spezifikation eines Modells in der Effektorikphase allerdings unverhältnismäßig kompliziert – der Modellierer muß immer alle Aktivitäten betrachten, wenn er ein gleichzeitiges Aktiv-sein nicht ausschließen kann –, als daß ein mehr oder aufwendige Erklären eines alternativen Konfliktauslösungsmechanismus dies nicht wett machen könnte.

Soll genau eine Aktivität das Verhalten des Agenten bestimmen, darf also nur eine einzige Aktivitätsauswahlregel feuern. Der dabei notwendige Auswahlmechanismus muß dabei so einfach wie möglich sein und sollte daher eher nicht-technische Charakteristika aufweisen – wie dies z.B. bei "wähle die Aktion der speziellsten Regel" der Fall wäre. Dadurch daß die Auswahl der Aktivität zudem nicht davon abhängig sein soll, in welcher Reihenfolge

[5] Die Auswahl einer nicht weiter zu konfigurierenden Aktivität entsprechend einer gegebenen Situation ist eine im Vergleich zu Planungsproblemen einfache Aufgabe (siehe dazu [Puppe 1990, Puppe et al. 1996]).

die Regeln getestet werden, fällt auch die Selektion der ersten "gefundenen" Aktivität aus. Demnach bleiben zwei weitere einfache Möglichkeiten: Eine zufällige oder eine prioritätsgesteuerte Auswahl unter möglichen Regeln. Bei ersterem verliert der Benutzer im Prinzip die Kontrolle über die letztendliche Auswahl. Das Modell erhält so an einer Stelle einen stochastischen Faktor, der vom Modellierer möglicherweise nicht intendiert ist. Das heißt, wenn Wissen über die Präferenz des Agenten für die Auswahl einer Aktivität vorhanden ist, sollte man dieses Wissen bei der Aktivitätsselektion nutzen und nur bei absichtlich gleicher Priorität der Aktivitäten eine zufällige Auswahl verwenden. Durch die explizite Darstellung von Aktivitäten (siehe Seite 115) kann eine Erweiterung des Schemas um numerische Prioritäten – wichtig ist dabei nur die Ordnung der Zahlen – vorgenommen werden. Durch die Angabe dieser Gewichte hat der Modellbauer mehr Einflußmöglichkeiten auf die Auswahl, als durch eine einfache Bestimmung der Testreihenfolge mit Auswahl der ersten möglichen Aktivität. Die Normierung der Prioritäten kann wegen der einfachen Verrechnung ohne Probleme dem Modellierer überlassen werden.

5.3.3.3 Sensorik

In dieser Phase werden die direkten Wahrnehmungen bzw. die in den Variablen des internen Agentenzustands gespeicherten Informationen aufbereitet. Dieser Schritt ist an sich nicht unbedingt notwendig, da auf der Basis von Kombinationen von Situationsabfragen auch direkt die nächste Aktion ausgewählt werden könnte. Das Formulieren von derartigen Zwischenschritten trägt aber unmittelbar dazu bei, das Verhalten eines Agenten übersichtlicher und nachvollziehbarer, und somit besser wartbar darzustellen.

Betrachtet man diese Phase getrennt von der Aktivitätsauswahl, dann kann man folgende sinnvolle Möglichkeiten für die Vorgehensweise identifizieren:

1. "Sensor-getriebene" Vorgehensweise, mit der asynchron zur eigentlichen Verhaltensbeschreibung die eingehenden Sensordaten aufbereitet werden. Die Aktivitätsselektion greift auf ein unabhängig aktualisiertes und abstrahiertes Weltmodell zu.

   Diese Vorgehensweise ist notwendig, wenn ein internes Weltmodell in einer dynamischen Umwelt aktuell gehalten werden muß. Eine im Zyklus synchrone Abfrage der Sensoren mit anschließender vorwärtsverketteter Abstraktion reicht in folgenden Situationen:

   (a) wenn sich die Welt nicht ändert, während der Agent "nachdenkt", also seine Aktion selektiert,

   (b) wenn alle relevanten Daten während dieser Phase des Aktualisierungszyklus vorhanden sind und

   (c) wenn diese nicht nur kurzfristig anliegen, während der Agent auf seine nächste (synchronisierte) Aktualisierung wartet

   Die kritische Frage ist dabei allerdings, wieviel redundante Information so erhoben bzw. vorverarbeitet wird. Benötigt der Agent im aktuellen Zyklus nur einen Bruchteil der möglichen Information oder generell kein ständig aktualisiertes Modell seiner Umwelt, dann ist eine aufwendige, komplette Sensor-getriebene Vor-Abstraktion nicht nötig.

2. "Aktivität-getriebene" Vorgehensweise: Die Sensoren werden dann "abgefragt", wenn diese Daten für die Aktivitätsselektion benötigt werden. Dabei wird ausgehend von den Regeln zur Aktivitätsauswahl quasi rückwärtsverkettet durch die Abstraktion, bzw. die angegebenen Aussagen zurückgerechnet, bis Daten für die Feststellung des Wahrheitsgehalts der Kondition einer Auswahlregel vorhanden sind[6].

   Auf diese Weise werden nur die Informationen verarbeitet, die wirklich im aktuellen Zyklus benötigt werden. Allerdings kann es zu weniger reaktivem Verhalten führen, wenn der Agent Änderungen in der Umwelt nicht registriert, was aber auch vom Modellbauer intendiert sein kann – ansonsten wäre es eher ein Fehler im Modell.

Beide Vorgehensweisen besitzen Vor- und Nachteile und spielen ihre Stärken in unterschiedlichen Domänen, bzw. unterschiedlichen Modellen aus. Eine Entscheidung für einen der beiden Mechanismen könnte die Verwendbarkeit eines generellen Beschreibungsrahmens stark behindern. Auf der anderen Seite lassen sich beide in den hier dargestellten Rahmen integrieren:

Eingabevariablen als Sensoren. Wie in Abschnitt 5.1.2 dargestellt, besteht der interne Zustand eines Agenten aus Zustandsvariablen. Dementsprechend muß auch ein internes Weltmodell in diesen gespeichert werden. Auf der anderen Seite entsprechen in diesem "System"-nahen[7] Konzept Sensoren den Eingabevariablen des Systems, die hier als Zustandsvariablen mit spezieller – modelltypischer – Bedeutung realisiert werden können. Die Werte dieser Eingabevariablen werden dabei in der Sensorik-Phase gesetzt, indem über Schnittstellen Umweltinformation eingeholt wird. Während der Aktualisierung der (dynamischen) Zustandsvariablen zu Beginn der Agentenaktualisierung ("$UPDATE-ZUSTAND''$) können die Werte dieser Eingabevariablen nicht nur gesetzt, sondern auch Änderungen in den eigentlichen Zustandsvariablen des Agenten vorgenommen werden. Allerdings sollte der Modellierer bei der Angabe der Dynamik der Zustandsvariablen quasi freie Hand haben. Deshalb werden bis jetzt keine weiteren Vorgaben gemacht.

Rückgekoppelte Abstraktionsregeln. Als zusätzlicher Mechanismus existiert folgender, der in dem hier vorgestellten Rahmen attraktiverer ist. Von den Regeln zur Aktivitätsauswahl ausgehend können die Prädikate, die in Vorbedingungen verwendet werden, schrittweise verfeinert werden. In Abbildung 5.8 wird dies angedeutet. Für ein derartiges abstraktes Prädikat wird überprüft, ob die angegebenen Vorbedingungen, die wiederum abstrakte Prädikate enthalten können, wahr sind. Diese Verfeinerung geht so weit, bis die Aussagen auf primitive Abfragen von Werten von Eingabe- oder Zustandsvariablen bzw. Schnittstellen zur Umwelt zurückgeführt sind. Für jedes abstrakte Prädikat, das der Modellierer in Form einer Regel "zusammenbaut", existiert dabei genau eine Verfeinerung.

5.3.3.4 Effektorik

In der letzten Phase des Aktualisierungszyklus eines Agenten wird die Aktivität des Agenten in konkrete Aktionen umgesetzt und diese ausgeführt, bzw. an den Simulator weitergegeben. In der Agentenfunktion von S. Russell und P. Norvig (siehe Seite 98) besteht diese

[6] Diese Vorgehensweise könnte man auch als einfaches schrittweises Verfeinern von Abstraktionen sehen, und dabei die Regelmetapher nicht strapazieren.

[7] siehe Systembegriff in Abschnitt 3.1.1

Phase aus zwei Teilen: Der Bestimmung der Aktion aus der ausgewählten Regel und dem Aktualisierung des internen Zustands mit dieser Aktion. Im Folgenden sollen diese beiden Teile gleichbehandelt werden, indem sowohl nach außen als auch auf interne Zustandsvariablen wirkende Aktionen zugelassen werden sollen. Dadurch muß der Modellbauer mit nur einem Mechanismus zurecht kommen und kann analog zu den Aktionen, die in der Umwelt wirken sollen, auch das Verändern des internen Zustands beschreiben.

In Abschnitt 5.3.2.2 wurde die Aktivität eines Agenten als ein Container für die im aktuellen Zyklus ausgewählten Aktionen des Agenten eingeführt. Dementsprechend ist das Ergebnis der Verfeinerung einer Aktivität eine Menge von vollständig spezifizierten, quasi ausführbaren Aktionen, also Effektorkommandos, Funktionsaufrufe oder Methoden-Nachrichten. Die oben beschriebenen Aktivitäten sind dabei nicht parametrisierbar – was sich durch die geforderte, möglichst einfache Struktur der Regeln zur Aktivitätsselektion und der dabei zu erfüllenden Klassifikationsaufgabe begründen läßt. Damit Aktivitäten sinnvoll eingesetzt werden können, muß man allerdings eine kontextabhängige Umsetzung in Aktionen zulassen und somit eine dritte Schicht mit einem regelbasierten Schema.

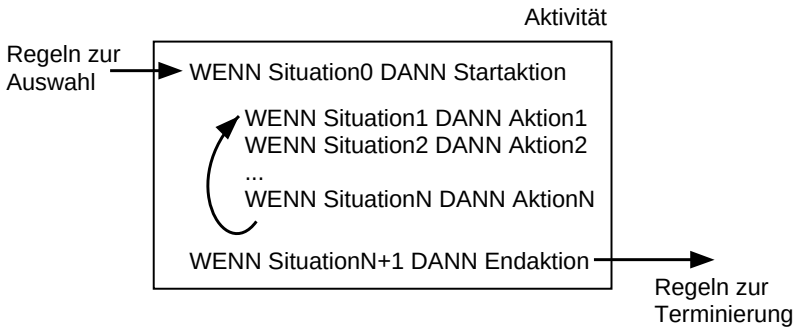
Bei der Umsetzung einer Aktivität in Aktionen – für eine Verfeinerung der strukturierten Aktivitäten in konkretere siehe Abschnitt 5.3.2.3 – ergeben sich folgende Anforderungen.

- Die Reihenfolge der pro Zyklus einmaligen Betrachtung der Regeln soll vollständig vom Modellierer kontrollierbar sein. Auf diese Weise könnte eine derartige Umsetzung der Aktivität auch als einen skript-ähnlicher Ansatz aus bedingten Aktionen bezeichnet werden.

- Objekte der Vorbedingung sollen für die Parametrisierung der Aktion verwendbar sein, im Sinne von: “Wenn ein Objekt xy wahrnehmbar ist, dann laufe in Richtung *dieses* Objekts . . .

Diese Anforderungen werden dadurch umgesetzt, daß – wie in Abbildung 5.6 bereits angedeutet und in Abbildung 5.10 weitergeführt – Aktionsregeln in einer genau vorgegebenen Reihenfolge behandelt werden.

Abbildung 5.10 Schema der internen Struktur einer Aktivität mit Effektorschicht-Regeln in Form von bedingten Aktionen.



Die Anforderung der Parametrisierbarkeit von Aktionsregeln hätte an sich einen Unifikationsmechanimus erfordert, der allerdings auf der Basis der Zustandsrepräsentation nicht

einfach umzusetzen gewesen wäre. Statt einer für den Modellierer manchmal undurchsichtige Definition von Variablen und deren Verwendung[8] während des Abtestens einer Vorbedingung wurden deiktische Referenzierungen auf wahrnehmbare Aspekte verwendet. Das heißt, anstatt eine neue Variable zu definieren, deren Wert zu setzen und diese dann als Parameter einer Aktion einzusetzen, muß als Parameter einer Aktion ein direkter "Zeiger" auf einen Aspekt, Wert oder ein Objekt verwendet werden. Die Formulierung einer Regel wie oben wird dadurch zwar umständlicher, kann aber vom Modellierer so weit wie möglich kontrolliert werden: "Wenn ein Objekt xy wahrnehmbar ist, dann laufe in Richtung des wahrnehmbaren Objektes xy". Durch die Kombination der nicht parametrisierbaren Aktivitäten mit einer kontextabhängigen Umsetzung dieser mittels Aktionen, wurde eine relativ transparente und zudem modulare Form der Verhaltensbeschreibung eines Agenten geschaffen.

Bevor im nächsten Kapitel Details der Umsetzung dieses Schemas in einer implementierten Entwicklungsumgebung für Multiagentensimulationen erläutert werden, sollen noch einige kurze Anmerkungen zu Regeln mit stochastischen Elementen und abschließend die Konzepte der Verhaltensbeschreibung in den Kontext vorhandener Ansätze gestellt werden.

5.3.4 Wahrscheinlichkeiten als Mittel zur Verkleinerung der Regelmenge

Die wohl weitreichendste Form der Situationsabstraktion und darauf aufbauend der Verhaltensbeschreibung ist eine Darstellung basierend auf einer Wahrscheinlichkeitsverteilung: Anstatt kausale oder anderweitig motivierte Zusammenhänge zwischen Situationensaspekten und Aktivitäten bzw. Aktionen festzulegen, kann man Teile eines Verhaltensmodells vereinfachen, indem man (situationsabhängige) Häufigkeiten für bestimmte Aktionen eines Agenten angibt. Dies ist gegebenenfalls sogar notwendig, wenn kein Modell, das interne Strukturen berücksichtigt, formuliert werden kann oder soll, weil die Hypothesen dazu fehlen, bzw. für die Fragestellung nicht wichtig sind.

Durch Einführung von zusätzlichen Wahrscheinlichkeiten für das Feuern einer Regel, wenn die Vorbedingung erfüllt ist, wird es dem Modellierer möglich, derartige Abstraktionen darzustellen. Durch die Kombination "grober" Situationsbeschreibungen in den Vorbedingungen mit – auch aus internen Zustandsvariablen berechenbaren – Wahrscheinlichkeiten, die das Feuern der Regel beeinflussen, kann die zu formulierende Regelmenge verkleinert werden, da so abstraktere Faktoren als "Abkürzungen" miteinbezogen werden können. Dies gilt für jede Regelmenge, unabhängig ob sie zur Auswahl oder kontextabhängigen Verfeinerung oder zur Umsetzung einer Aktivität verwendet wird.

Allerdings folgt aus dieser Erweiterung eine Form des Nicht-Determinismus für die Simulationsexperimente, die gerade bei komplexeren Modellen eher nachteilig ist. Um das Verhalten des Modells für dessen Validation und weiterführende Tests ausreichend zu untersuchen, muß für jeden Parametersatz eine entsprechend große Stichprobe an Experimenten produziert werden. Bei Modellen ohne zufallsgesteuerte Anteile reicht im Gegensatz dazu pro Parametersatz ein Simulationslauf aus. Wie viele Simulationsläufe im Fall eines Modells mit Wahrscheinlichkeiten notwendig sind, kann nur im konkreten Beispiel anhand der Simulationsergebnisse entschieden werden.

[8] Dabei lag das Problem allerdings weniger bei der formalen Erklärung der Variablen und ihrer Belegung, sondern daran, eine angemessene graphische Notation dafür zu finden.

5.3.5 Betrachtung dieser Strukturen im Kontext vorhandener Methoden

5.3.5.1 Agenten-orientiertes Software-Engineering

Die Identifikation von Rollen, die Agenten einnehmen können, bildet einen zentralen Punkt der in Abschnitt 2.3 beschriebenen Methoden des agenten-orientierten Software-Engineerings. Ausgehend von den Rollen werden dabei Ziele, die ein Agent mit dieser Rolle verfolgen soll, oder auch Dienste, die mit dieser Rolle assoziiert werden, festgelegt. Eine derartige – relativ zu dem verteilten Paradigma – "top-down"-Vorgehensweise bei der Erstellung eines Softwaresystems ist auch bei der Entwicklung eines Multiagentenmodells interessant. Dies gilt vor allem in der Phase der Spezifikation des Konzeptmodells.

Im Gegensatz zu expliziten Rollen oder Verhaltenskategorien werden in dem hier dargestellten Konzept eines strukturierten Beschreibungsschema keine Ziele explizit behandelt. Die Methoden aus Abschnitt 2.3 entwickeln – aufbauend auf diesen Spezifikationen der Ziele, bzw. Dienste – (Teil-) Pläne, mit denen der jeweilige Agent diese erfüllen kann. Die dabei verwendeten Planschablonen werden auf der anderen Seite auch in Repräsentationen dargestellt, denen das hier vorgestellte Konzept einer Aktivität wiederum ähnlich ist.

Grundsätzlich muß man aber bemerken, daß derartige Methoden für andere Typen von Systemen und Vorgehensweisen konzipiert sind. Wie in Abschnitt 2.3 näher charakterisiert, werden bei der Erstellung eines derartigen Softwaresystems (auch bei der Integration mit menschlichen Benutzern) Ziele des verteilten Problemlösens verfolgt. Das bedeutet, daß das Multiagentensystem konstruiert werden soll, das in einer "realen" Umwelt vorgegebene Probleme lösen muß. Bei einer Multiagentensimulation wird dagegen ein näher zu untersuchendes Multiagentensystem in einer simulierten Umwelt nachgebildet. Die Analysephase hat hier also weniger mit Konstruktion einer Gesellschaft, als mit Abstraktion und Reproduktion von Verhaltensweisen zu tun. Gerade bei der Modellierung von Agentensystemen mit komplexeren, d.h. kognitiven bzw. deliberativen Agenten ist die Identifikation von Zielen der einzelnen Agenten und ihrer daraus folgenden Aktivitäten die Vorgehensweise der Wahl. Der Schritt der Identifikation der Agenten und der Zuordnung von Zielen zu Agenten entfällt in jedem Fall, da die Art bzw. die Grenzen der einzelnen Einheiten durch das Originalsystem vorgegeben sind. Demzufolge wird die Entwicklung eines Multiagentenmodells einen deutlichen "bottom-up"-Charakter besitzen.

5.3.5.2 Aktivitäten, *RAP*s und *KA*s

Aktivitäten und ihr Kontext in Form von Auswahl- oder Terminierungsregeln sind vollständig spezifizierten Planbausteinen sehr ähnlich, wie sie von einigen reaktiven Planern verwendet werden. In Abbildung 5.11 werden die Strukturen von "reactive action packages" des RAP-Planers skizziert – die Interpretation dieser Planbausteine wird auf Seite 21 erläutert.

Als ein weiteres prominentes Beispiel für Teilpläne, die von BDI-Agenten verwendet werden können, kann man "knowledge areas" angeben, die die Grundlage für die Vorgehensweise eines durch PRS[9] gesteuerten Agenten bilden. Eine erweiterte Form einer derartigen Planschablone wird in Abbildung 2.7 dargestellt. Beide Planschablonen sind an sich schon relativ alt, werden aber heute beide noch verwendet, bzw. weiterentwickelt. So bilden

[9] "procedural reasoning system" – siehe auch Seite 22

Abbildung 5.11 Struktur eines *RAP* [Firby 1989]: Anstelle von `SUCCEED` ist auch `(MONITOR-STATE formula)` oder `(MONITOR-TIME zeitangabe)` möglich, um bestimmte Tasks regelmäßig ausführen zu können, z.B. Aktualisierung des Weltmodells, Reaktionen in Notfallsituationen, usw. Bei der Definition eines TASK-NETS besteht die Spezifikation eines Schrittes wiederum aus einem Index, der auf ein RAP deutet. Die `annotation` besteht dabei aus einer Zuordnung, welche Situation für welchen folgenden Schrittes des Tasknetzes vorhanden sein muß.

```
(DEFINE-RAP
    (INDEX index)
    (SUCCEED formula)
    (METHOD
        (CONTEXT formula)
        (TASK-NET task-network-description))
    (METHOD
        (CONTEXT formula)
        (PRIMITIVE primitiv-action-request))...)

mit Definition eines TASK-NETs:

(TASK-NET
    (step1-tag priority step-specification annotation)
    (step2-tag priority step-specification annotation)
...)
```

"knowledge areas" die Basis für Plandarstellungen der Modellierungsmethodik für BDI-Agenten nach D. Kinny et al. (siehe Abschnitt 2.3.1.2).

Auf der anderen Seite kann man Planbausteine, die mehr als eine einzelne primitive Aktion umfassen, kaum anders gestalten. Eine derartige aktivitätsbeschreibende Struktur muß folgende Bausteine umfassen:

- Name des Teilplans, bzw. der Aktivität, um diese adressieren zu können – bei Verfeinerungen, usw.
- Beschreibung der Situation, in der der "Plan" anwendbar ist.
- Beschreibung der Zielsituation, die durch Ausführung dieser Tätigkeiten erreicht werden soll.
- Beschreibung der Situation, in der die Aktivität enden soll, bzw. Plan terminiert - abhängig von der Ausdrucksstärke der Zielrepräsentation kann hier zwischen "erfolgreich" und "gescheitert" unterschieden werden.
- Beschreibung der Aktionen oder der Verfeinerung des Plans, also der Tätigkeiten, die der Agent, wenn er dieser Repräsentation folgt, ausführt.

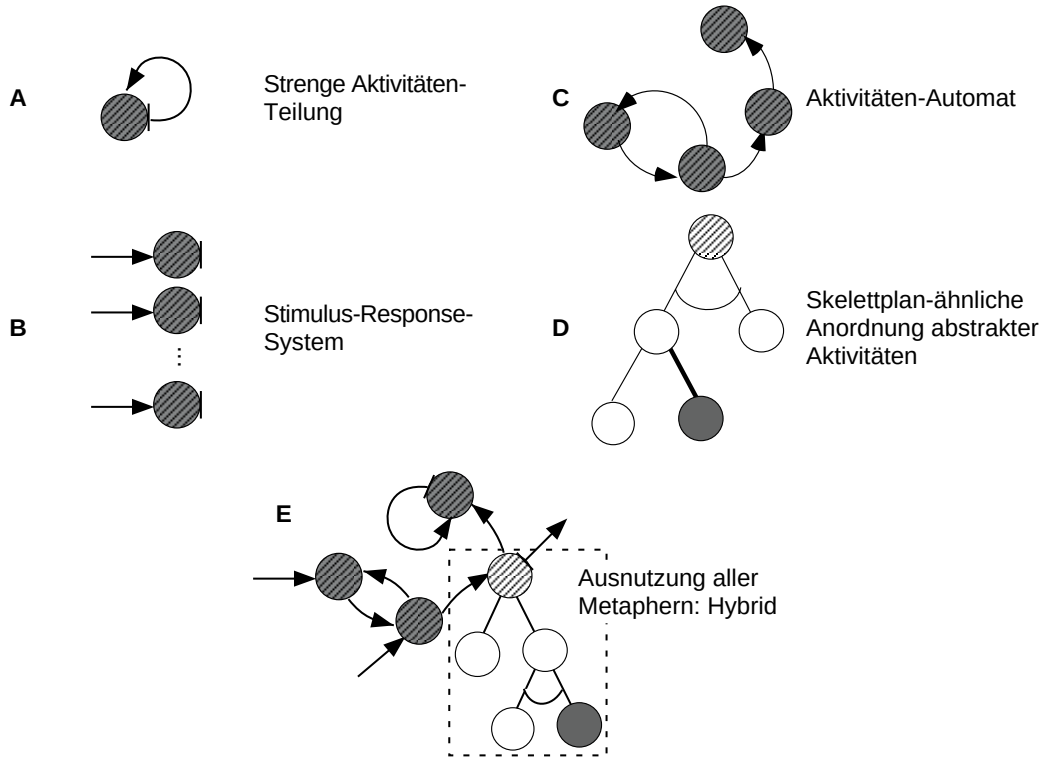
Der jeweils damit verbundene Mechanismus für die Auswahl eines derartigen Planbausteins ist meist deutlich anders als der hier dargestellte. Das Ziel der in dem vorliegenden Konzept verwendeten Aktivitäten ist es, Verhalten zu beschreiben. Dennoch könnte man die Vorbedingungen der Terminierungsregeln einer Aktivität so interpretieren, daß sie die Situation beschreiben, die durch Ausführung dieser Aktivität erreicht werden soll. Aber dies ist eine Möglichkeit der Interpretation durch den Modellierer und nicht durch den Rahmen

der Verhaltensbeschreibung vorgegeben. Weil also Aktivitäten nicht in jedem Fall eine explizite Repräsentation ihres Endzustandes besitzen, sind sie als Grundlage eines planenden Agenten ungeeignet. Sie kodieren Verhalten auf eine andere Art als dies die Planschablonen machen. Dennoch sind sie für die Beschreibung von Verhalten ausreichend, wie in den nächsten Abschnitten dargestellt wird.

5.3.5.3 Aktivitätenbasierte Beschreibung als generische Agentenarchitektur

Das vorgestellte Schema zur Beschreibung von Agentenverhaltens auf der Basis von Aktivitäten, einfachen Regeln zu deren Auswahl, Vorverarbeitung und Verfeinerung bildet eine generische Agentenarchitektur. Sie erlaubt in einer transparenten Art und Weise die Implementierung von Verhaltensbeschreibungen auf der Basis von ganz unterschiedlichen Sichtweisen auf die Organisation dieses Verhaltens. Dabei kann man vier grundlegende Möglichkeiten von Metaphern für ein in diesem Rahmen formulierbares Verhalten identifizieren, die in Abbildung 5.12 skizziert werden.

Abbildung 5.12 Formulierbare generische Agentenarchitekturen



- Strenge Aktivitäten-Teilung (A): Jedem Agent wird eine Rolle mit genau einer Aktivität zugewiesen, die der Agent nicht ändern kann, ohne die Rolle zu wechseln. Es gibt keinerlei Terminierungsregeln. So lassen sich starre Arbeitsteilungssysteme gut darstellen. Reaktionen auf Umweltänderungen usw. müssen in der Verfeinerung dieser Aktivität durch kontextabhängige Aktionen dargestellt werden (siehe Abbildung 5.10).

- Pure Stimulus-Response-Systeme (B) kann man beispielsweise über eine Sammlung von nicht-indizierten Notfallregeln darstellen, die die aktuelle Aktivität in jedem Zyklus erneut bestimmen. Mittels Regelwahrscheinlichkeiten und Aktivitätsprioritäten kann man eine Wettbewerbsauswahl wie bei "klassischen" reflexbasierten Architekturen nachbilden.

- Aktivitäts-Automaten (C) wurden in Abschnitt 5.3.2.2 besprochen. Eine Regel, die für die Terminierung einer Aktivität sorgt, wählt zugleich die nachfolgende Aktivität aus. Dem Agenten wird beim Erzeugen eine Startaktivität zugewiesen, danach durchläuft er eine Struktur, die einem endlichen Automaten ähnlich ist.

- Eine Skelettplan-ähnliche Organisation (D) erhält man durch Kombination verschiedener abstrakter Aktivitäten, wie in Abschnitt 5.3.2.3 dargestellt.

Ein Modellierer kann das Strukturierungsspektrum zur Formulierung kombinieren, oder sich auf ein Muster beschränken. In Abbildung 5.12 (E) wird skizziert, wie diese vier Metaphern gleichzeitig in einer Verhaltensbeschreibung genutzt werden können. Auf diese Weise können für unterschiedliche Agenten jeweils angepaßte Beschreibungsstrukturen verwendet werden, die die konkrete Organisation des Gesamtverhaltens bzw. bestimmter Teilverhaltensweisen erleichtern.

5.3.5.4 Zur Mächtigkeit der Aktivitätsbasierten Verhaltensbeschreibung

Die zentrale Frage bei der Beschreibung dieses Schemas wurde bisher nur angedeutet: Wie mächtig ist dieses Schema, bzw. welche Art von Verhalten kann so dargestellt werden, welche dagegen nicht?

Wie oben mehrmals angedeutet, kann dieses Schema nicht für die Beschreibung von Verhalten auf der Ebene von planenden Agenten verwendet werden. Aktivitäten beschreiben das Verhalten eines Agenten "flach", d.h. ohne daß der Agent "weiß", welches Ziel er damit verfolgt. Weder Ziele noch eine Situationsbeschreibung nach Ausführung der Aktionen werden explizit repräsentiert. Der Fokus liegt hier immer auf dem "von außen" beobachtbaren Verhalten eines Agenten.

Die Ausdrucksmächtigkeit einer Verhaltensbeschreibung läßt sich in einer grundsätzlichen Weise darin messen, ob Schleifen und Verzweigungen formulierbar sind: Schleifen lassen sich dabei nicht nur durch Zyklen im Aktivitätsautomaten formulieren. Während sich ein Agent in einer Aktivität befindet, wird die dabei spezifizierte Folge der Aktionen wiederholt ausgeführt. Auf diese Weise lassen sich Schleifen sowohl auf der Aktivitäten- als auch auf der Aktionsebene beschreiben. Analoges gilt für bedingtes Verhalten, bzw. Verzweigungen, die durch Regeln zur Aktivitätsselektion und durch regelbasierte Verfeinerung der Aktivität ausgedrückt werden können.

Problematisch ist in diesem Schema allerdings die Behandlung von Variablen. Im Prinzip steht dabei einer Verwendung einer Zustandsvariablen als Variable bei der Programmierung von Verhalten nichts im Wege. Allerdings entsprechen diese dabei globalen Variablen, die vom Modellierer ein einigermaßen diszipliniertes Vorgehen verlangen. Allerdings bietet die gesamte Strukturierung des Formulierungsrahmen dabei gute Anhaltspunkte zur Gestaltung einer übersichtlichen Verhaltensbeschreibung.

Bevor die Anwendbarkeit dieses Beschreibungsschemas in einer Entwicklungsumgebung für Multiagentensimulationen dargestellt wird, sollen im folgenden Kapitel wichtige Details der Umsetzung in einem Simulator erläutert werden.

6

Umsetzung des Entwurfs einer Verhaltensbeschreibung durch den SESAM-Simulator

6.1 SESAM: "Shell für Simulierte Agentensysteme"

Für das im letzten Kapitel entworfene Rahmenwerk für die Repräsentation eines Multiagentenmodells wurde ein Simulator entwickelt und implementiert. Dieser wurde durch eine graphische Modellier-, Experimentier- und Auswertungsumgebung angereichert und mit dem Namen SESAM bezeichnet: "Shell für Simulierte Agentensysteme". Die wichtigste Designentscheidung war dabei, das Verhaltensmodell explizit repräsentieren: Während eines Simulationsexperiments werden demnach die getrennt von den eigentlichen Instanzen gespeicherten Verhaltensbeschreibungen durch die Simulationsroutine interpretiert – dies gilt sowohl für ihre (dynamischen) internen Zustandsvariablen, als auch für die eigentliche Repräsentation ihres Verhaltens durch Regeln und Aktivitäten. Das heißt, alles Wissen über das Verhalten der Agenten, bzw. auch der Umweltbestandteile ist explizit. Dies hat gegebenenfalls den Nachteil, daß die Dauer eines Simulationsexperiments im Vergleich zu einer direkt auf der Ebene einer Programmiersprache implementierten Umsetzung länger ist. Allerdings gewinnt man durch diese Trennung von Modell und Simulator einige Vorteile bei der Flexibilität und dem Abstraktionsniveau bei der Modellentwicklung, der Manipulierbarkeit und Zugänglichkeit des Modells, und bei der Wiederverwendbarkeit des so anwendungsunabhängigen Simulators.

Ein derartig explizites Schema zur Verhaltensrepräsentation verringert die Distanz zwischen einer auf Beschreibung von Agentenaktivitäten beruhenden Modellspezifikation und deren Umsetzung in einem ausführbaren Modell. Ebenso kann eine graphische Modellieroberfläche ohne aufwendige Mechanismen zur Übersetzung zwischen graphischer und programmiersprachlicher Darstellung repräsentiert werden. Die Konfigurierung der Simulationsexperimente auf der Basis systematischer Modellvariation ist ebenso leichter zugänglich, da Modifikationen in einer aussagekräftigen Repräsentation vorgenommen werden, und keine Variablen auf programmiersprachlicher Ebene manipuliert werden müssen.[1].

[1] Diese Punkte beschränken sich immer auf domänen- und anwendungsunabhängige Simulationsumgebungen – bei Spezialsimulatoren können angepaßte Schnittstellen die Arbeit eines Modellierers deutlich mehr erleichtern als eine explizite Modellschablonen. Diese wird immer relativ abstrakt sein, daß ein unter Umständen

Bevor in den folgenden Abschnitten die Umsetzung dieses Schemas und seines Interpreters, bzw. im nächsten Kapitel ein Überblick über die auf diesen Simulator aufbauende Entwicklungsumgebung gegeben werden wird, soll nochmals die dabei verfolgte Zielsetzung vergegenwärtigt werden: Es sollte weniger ein Simulator für wenige, dafür aber mächtig im Sinne von kognitiv-planenden Agenten erstellt werden, sondern eine Simulationsumgebung für viele (> 500) Agenten, bei denen jeder einzelne dennoch ein umfangreiches Verhaltensrepertoire aufweisen kann. Der Simulator und das Repräsentationsschema sind daher eher für die schnelle Erstellung einfacher, stark miteinander und ihrer Umwelt interagierende Einheiten geeignet. Das Verhalten kann durchaus eine beträchtliche Komplexität aufweisen, wie die in Abschnitt 8 dargestellten Anwendungsbeispiele zeigen. Das Modell kann anspruchsvoll sein. Die Mechanismen, die man bei der Umsetzung benötigt, sollten dagegen einfach sein, aber auch bei aufwendigeren Verhaltensrepräsentationen skalieren.

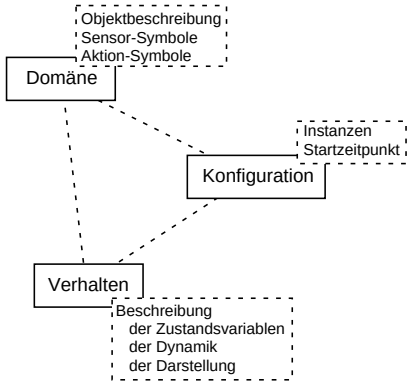
6.2 Umsetzung im objekt-orientierten Rahmen

6.2.1 Vorgegebene Klassen und das Konzept der Klassenbeschreibung

6.2.1.1 Allgemeine Organisation der Modellteile

Für die Umsetzung des im letzten Kapitel beschriebenen Schemas zur Repräsentation eines Multiagentenmodells kann man drei Teilaspekte bei der Beschreibung identifizieren. Diese umfassen grundlegendes anwendungs- bzw. domänenspezifisches Wissen über die Bestandteile des Modells. Dazu kommt das eigentliche Verhaltensmodell mit einer Beschreibung, welche Attribute, bzw. Zustandsvariablen eine Einheit besitzen kann und wie diese sich verändern. Auf einer dritten Ebene kann man die Repräsentation der Konfiguration im Modell darlegen. Dabei wird beschrieben, welche konkreten Instanzen der Agentenklassen in welcher Relation zueinander, bzw. wo lokalisiert sind. Abbildung 6.1 liefert einen Überblick über diese Teile.

Abbildung 6.1 Überblick über die Wissensebenen, entsprechend derer die Bestandteile eines Multiagentenmodells angeordnet werden können.



Diese Dreiteilung ist vor allem deshalb sinnvoll, weil dabei im Endeffekt unterschiedliche Ebenen im Modell behandelt werden. Bei der Beschreibung der Domäne sind es vor

schwieriger Transfer, zumindest der Terminologie, zur Anwendungsdomäne notwendig ist.

allem terminologische Fragen, die geklärt werden müssen – im Grunde wird dadurch eine Entwicklungs- und Simulationsumgebung an die Domäne angepaßt. Das Verhaltensmodell liefert die Strukturen, die der Simulator interpretiert, während die so "gefundenen" Veränderungen an der Konfiguration des Modells durchgeführt werden.

Man könnte etwas pauschal sagen: Die Domänenbeschreibung konkretisiert die Sprache, die Verhaltensrepräsentation die Algorithmen und die Konfiguration liefert die Daten, bzw. den Kontext für die Algorithmen. Bei einer Trennung von Modell und Simulator sind explizite Repräsentationen aller drei Bestandteile notwendig. Konkret bestehen die Beschreibungen für die einzelnen Teile aus folgenden Bausteinen, die in den nächsten Abschnitten sukzessive näher erläutert werden:

- Die Domäne – als dem Rahmen für die Verhaltens- und Konfigurationsbeschreibung – umfaßt die anwendungsspezifischen Klassendefinitionen und Beschreibung der Methoden dieser Klassen in Form von "Primitiven", mit denen diese Schnittstellen für Aktionen, Prädikate und (Umwelt-)Referenzierungen "deklariert" werden.
- Die Verhaltensbeschreibung enthält die Teile des Modells, die direkt vom Simulator zur Erzeugung des Verhaltens einer Einheit interpretiert werden. Somit gehören alle Regeln, Aktivitäts- und Rollenbeschreibungen, aber auch Schablonen für die Zustandsvariablen einer Einheit zum Modell. Dabei besitzen letztere, die man auch als Attribut-Beschreibungen betrachten kann, eine Affinität zur Domäne.
- Die Konfiguration beschreibt die Modellsituation, die alle zu einem Simulationszeitpunkt aktuellen Instanzen der Domänenklassen mit Werten für die bei ihnen relevanten Zustandsvariablen und ihre Position auf einer Karte enthält.

In Abbildung 6.2 werden die Bestandteile der unterschiedlichen Ebenen nochmals dargestellt. Im folgenden werden diese entsprechend der Darstellung von oben nach unten näher erläutert. Dabei wird auf den bekannten objekt-orientierten Konzepten "Klassen", "Objekten" und "Methoden" aufgesetzt[2].

6.2.1.2 Grundklassen der SESAM-Modelleinheiten

Das Grundgerüst für den Rahmen der Domänenklassen und somit für die Einheiten, deren Verhalten das Modell beschreibt und deren Instanzen die Bestandteile der Konfiguration sein können, bilden die im folgenden charakterisierten Klassendefinitionen (siehe dazu Abbildung 6.3).

Die Klasse "System" beinhaltet das übergeordnete Konzept aller Einheiten, indem bei allen Agenten, Ressourcen, usw. gleichermaßen vorhandene Attribute zusammengefaßt werden. Dies sind vor allem die Strukturen, in die die interne Zustandsbeschreibung – bestehend aus Zustandsvariablen und ihren Werten – eingetragen werden. Wegen der Abstraktheit dieser Klasse, von der jede Klasse für konkrete Einheiten der Konfiguration erbt – auch das Umweltsystem – sei die Benutzung des Begriffs "System" erlaubt. Diese Klasse besitzt eine Methode "Aktualisieren", die vom Simulator für jedes im aktuellen Takt existierende Objekt aufgerufen wird. Diese Methode wird für die Klassen, die von "System" erben, spezialisiert, bzw. implementiert (siehe auch dazu Abschnitt 6.3.2).

[2] Der SESAM-Simulator wurde in der objekt-orientierten Programmiersprache CLOS ("Common Lisp Object System"), einer objekt-orientierten Erweiterung von Lisp [Graham 1996, Steele 1990]) realisiert.

Abbildung 6.2 Überblick über die Bestandteile eines Modells: Auf der obersten Ebene finden sich anwendungsspezifische Klassen- und Primitivbeschreibungen. Schablonen für Zustandsvariablen können als "Attributrepräsentationen" zwischen Domäne und Modell eingeordnet werden. Aktivitäts- und Rollenobjekte bilden mit den zugehörigen Regeln die Basis für eine konkrete Situationen: Instanzen der Modellklassen verfügen über aktuelle Zustandsvariablen, Rollen und Aktivitäten, die die Konfiguration einer Einheit im Modell darstellen.

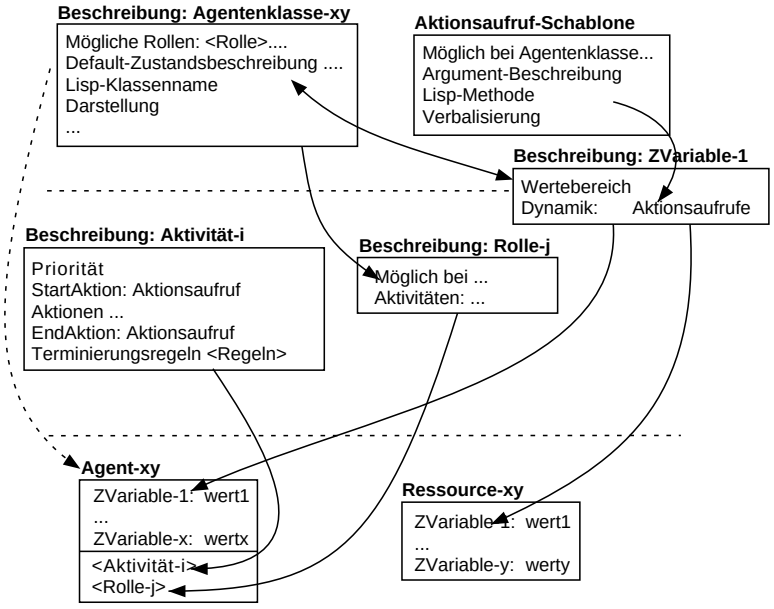
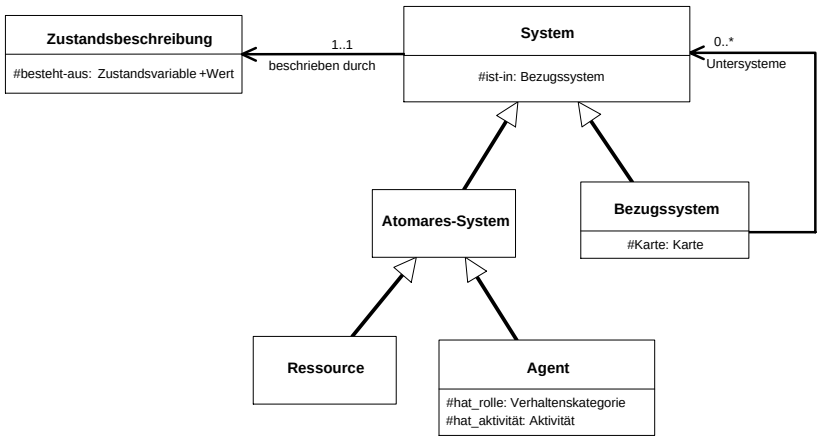


Abbildung 6.3 Überblick über die domänenunabhängige Klassenhierarchie in UML–Notation.



Von "System" erben zwei speziellere Klassen: "Atomares-System" und "Bezugssystem". Die Klasse "Atomares-System" faßt alle Einheiten zusammen, die nicht weiter verfeinert werden können, bzw. keine anderen Objekte "enthalten", also Ressourcen- und Agenten-

klassen. Dabei zeichnen sich letztere durch Attribute aus, in denen Referenzen auf Bestandteile der expliziten Verhaltensbeschreibung, Rollen- und Aktivitäten, stehen. Die Aktualisierungsmethoden für derartige Objekte werden bei der Darstellung des Interpreters – der mittels dieser Methoden realisiert wird – näher erläutert.

Eine andere Spezialisierung eines "Systems" findet man in der Klasse "Bezugssystem". Darüber werden unterschiedliche Formen von Containern für andere Systeme realisiert: Ein Bezugssystem beinhaltet über das Attribut "Karte" eine Repräsentation des Raumes. In Abbildung 6.3 sind davon erbende speziellere Klassen "Bezugssystem-mit-diskreter-Karte", bei der dieses Attribut eine Referenz auf ein zweidimensionales Array aus "Gebietseinheiten" beinhaltet, oder die Klasse "Bezugssystem-mit-kontinuierlicher-Karte", bei der eine entsprechende Repräsentation realisiert wurde, nicht dargestellt. Als weitere Form der Gruppierung besitzt eine Instanz dieser "Bezugssystem"-Klasse ein Attribut, das eine Liste aller "System"-Instanzen speichert, die sich in diesem Bezugssystem befinden, bzw. diesem zugehörig sind. Dadurch daß Bezugssysteme selbst auch als Systeme modelliert sind, kann durch diese Verzeigerung eine Hierarchie von zusammengefaßten Einheiten aufgebaut werden – als "Blätter" werden Instanzen von Unterklassen zu "Atomares-System" verwendet. Der hier umgesetzte Gruppenbegriff entspricht dabei der auf Seite 106 vorgestellten Repräsentation über eine gemeinsame (hier auch übertragen zu betrachtende) "Lokalisierung". Die anderen beiden, dort beschriebenen Formen – über einen entsprechenden Wert einer bestimmten Zustandsvariable, bzw. einen Gruppenagenten sind – ohne weiteres mit den bisher beschriebenen Mitteln umsetzbar.

"Agent", "Ressource" und "Bezugssystem" bilden die Oberklassen für sämtliche modellspezifische Klassen, deren Instanzen für die Konfiguration des Modells verwendet werden. Diese domänenunabhängigen Klassen sind abstrakte Klassen, d. h. es sollte keine Instanz direkt von diesen erzeugt werden. Modellspezifische Klassen bilden nicht nur eine zusätzliche Schicht für eine mehr anwendungsbezogene Terminologie. Über zusätzliche, explizite Beschreibungsobjekte, die von der Entwicklungsumgebung automatisch beim Anlegen derartiger Modellklassen erzeugt werden, können weitere Informationen zu diesen Klassen gespeichert werden. Diese werden im folgenden behandelt.

6.2.1.3 Modellspezifische Klassen und ihre Beschreibung

Anwendungsspezifische Klassen werden angelegt, indem der Modellierer eine Klasse von einer der abstrakten Klassen "Agent", "Ressource" oder "Bezugssystem"[3] ableitet. Dazu muß ein Beschreibungsobjekt generiert und mit Informationen gefüllt werden. Diese explizite Trennung zwischen Modellrepräsentation und den vom Simulator benutzten Strukturen wird bei der Erstellung anwendungsspezifischer Klassen von der gewählten Programmiersprache *CLOS* unterstützt, denn sie ermöglicht ein Anlegen von Klassen zur Laufzeit der Modellierumgebung. Information über diese Klasse wird dabei allerdings nicht direkt in dieser kodiert, sondern durch spezielle Beschreibungsobjekte für die Einheiten-Klassen und die bei ihnen verwendbaren Zustandsvariablen realisiert.

Metaklassen. Wie oben erwähnt wird zu jeder anwendungsspezifischen Klasse ein Beschreibungsobjekt generiert, das weitere Informationen über diese Klasse beinhaltet. Dazu

[3] Um genau zu sein: Der Modellierer wählt zwischen "Bezugssystem-mit-diskreter-Karte" und "Bezugssystem-mit-kontinuierlicher-Karte".

gehören insbesondere Angaben, die ein Modellierer für das Erzeugen einer Instanz dieser Klasse bereitstellen kann. Ein derartiges Metaobjekt enthält zudem Information zur Visualisierung und zur Spezialisierung von Editoren, mit denen ein Objekt der anwendungsspezifischen Klasse manipuliert werden kann[4]:

Tabelle 6.1 Auflistung der Angaben, die in einem Beschreibungsobjekt zu einer anwendungsspezifischen Klasse gemacht werden sollen.

Name der anwendungsspezifischen CLOS-Klasse
Name der direkten Überklasse
Konfiguration und Defaultwerte der Zustandsvariablen für den internen Zustand des Objekts beim Erzeugen
Manipulierbare Attribute, also welche vorgegebenen Zustandsvariablen der Benutzer während eines Simulationslaufs manipulieren darf
Statusdarstellung = Spezifikation von Information über den internen Zustand des Objekts, die während der Simulation angezeigt wird
Bild, mit dem ein Objekt bei der Animation des Simulationslaufs dargestellt wird
Erklärender Text zur anwendungsspezifischen Klasse

Bei Agentenklassen kommen Angaben dazu, ob ein Benutzer während der Simulation in die Aktivitätsselektion eingreifen darf, also ob und welche zusätzlichen Schnittstellen für einen Beobachter der Simulation möglich sind, um das Verhalten des Agenten zu steuern.

Bei einer Bezugsystemklasse ist dies eine Beschreibung der Palette eines Editors für die Inhalte der jeweiligen Karte und demzufolge auch eine Festlegung, von welcher Klasse Instanzen auf der Karte positioniert werden dürfen. Für alle Einheiten wichtig sind dabei der Name, die Überklasse und die Default-Konfiguration des internen Zustands. Alle anderen Angaben sind sinnvoll mit Defaultwerten vorbelegt. Weitere Informationen sind vor allem für eine Anpassung der Entwicklungs- und Simulationsumgebung relevant. Ein ähnliches Konzept wurde für die Integration von Aktionen, Prädikaten und anderen Basisprimitiven verwendet – siehe dazu Abschnitt 6.2.2.1.

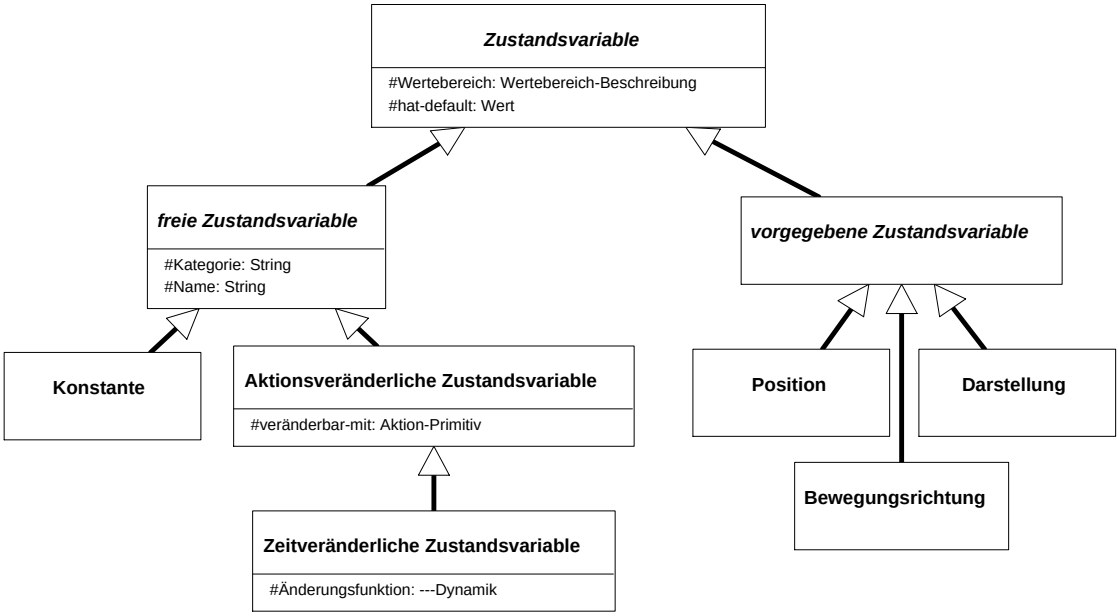
Zustandsvariablen statt Attribute. Die Information zum internen Zustand eines Agenten, Ressource oder Bezugssystems würde man in einer direkten Implementierung als Attribute der jeweiligen Klasse kodieren: Ein Attribut würde einer Zustandsvariable entsprechen. Attribut-Wert-Kombinationen würden den aktuellen Status des internen Zustands codieren. Mittels entsprechender anwendungsspezifischer Klassenhierachien könnte man diese Attribute auf eine sinnvoll strukturierte Weise in diesen Klassen organisieren.

Analog zu den oben erläuterten Metabeschreibungen für anwendungsspezifische Simulationsobjekt-Klassen kann man durch die Trennung der Beschreibung explizite und zusätzliche Information zu den Zustandsvariablen speichern. Wichtig ist dabei vor allem, daß die Veränderungen des Wertes einer Zustandsvariablen nicht in einer Lisp-Methode versteckt werden, sondern auf Aktionsprimitive abgebildet werden können. Auf diese Weise ist zumindest die Abbildung der Dynamik einer Zustandsvariable auf vorgegebene Primitivato-

[4] Die genaue Syntax für diese Angaben wird im Handbuch [Klügl & Oechslein 1999] erläutert.

me gegeben. Das Diagramm 6.4 zeigt die Organisation der Zustandsvariablen-Klassen, deren Instanzen mit einem aktuellen Wert kombiniert die Bestandteile einer Zustandsbeschreibung eines Simulationsobjektes darstellen.

Abbildung 6.4 Zustandsvariablen als Schablonen der Bestandteile der Zustandsbeschreibung eines Simulationsobjektes



Interessant ist dabei die Trennung zwischen “freier Zustandsvariable” und “vorgegebener Zustandsvariable”. Während die erstere vom Modellierer speziell für sein Modell angelegt wird und danach in beliebigen domänenspezifischen, verhaltensbeschreibenden Objekten und Instanzen von Zustandsbeschreibungen verwendet wird, entsprechen die vorgegebenen Variablen quasi hart kodierten Attributen:

Die Repräsentation der Position und der Bewegungsrichtung (nur bei Agentenklassen) kann nicht frei wählbar sein, da sie nur in Verbindung mit der Form der Kartendarstellung beim jeweiligen Bezugssystem Sinn macht. Ebenso wird das veränderbare Darstellungsbild einer Einheit von der Animationskomponente interpretiert. Deshalb mußten an dieser Stelle die Syntax und Verarbeitung dieser Zustandsvariablen vorgegeben werden. Um dem Simulator einen schnelleren Zugriff zu ermöglichen, wurden diese als Attribute der jeweiligen domänenunabhängigen Klassen (Position bei “System”, Darstellung bei “Atomares System” und Bewegungsrichtung bei “Agent”) repräsentiert.

Die Organisation der vom Modellierer zu erstellenden Zustandsvariablen entspricht der in Abschnitt 5.1.2.3 vorgebbaren Charakterisierung entsprechend der Dynamik der Variable. Eine Zustandsvariable kann so als Instanz der als “Konstante”, “Aktionsveränderliche Zustandsvariable” und “Zeitveränderliche Zustandsvariable” bezeichneten Klassen erzeugt werden. Diese Instanz einer Zustandsvariable kann über den Namen und eine weitere domänenabhängige, also frei formulierbare, bzw. aus bisher eingegeben auswählbare Katego-

rie adressiert und näher charakterisiert werden[5]. Die Spezialisierung der Klassen auf der Basis der Dynamik-Kategorie erleichtert die Aktualisierung des internen Zustands – wie im Abschnitt 6.3 beschrieben.

6.2.2 Klassen und Objekte zur Verhaltensbeschreibung

Wie die Klassen und Zustandsvariablen der einzelnen Einheiten soll auch die Beschreibung ihres Verhaltens explizit repräsentiert und vom Simulator interpretiert werden. Demzufolge soll ein Modellierer das Verhalten auf der Ebene der im letzten Kapitel eingeführten Strukturen, also Aktivitäten, Verhaltenskategorien, usw. darstellen können. Der Modellierer erzeugt also eine Verhaltensbeschreibung und weist diese – analog zu einer Schablone für eine interne Zustandsvariable – einer Agentenklasse zu. In den folgenden Abschnitten werden die konkreten Strukturen vorgestellt, die der Interpreter – der ab Seite 144 erläutert wird – verarbeiten kann. Dabei sollen zunächst die primitivsten Bausteine erläutert werden, gefolgt von den eigentlichen, höheren Elementen der generischen Agentenarchitektur.

6.2.2.1 Prädikate, Aktionen und sonstige "Primitive"

Ein Repräsentationsschema hängt in nicht zu unterschätzendem Maße vom Abstraktionsniveau seiner atomaren Bausteine ab. Diese bestehen hier aus folgenden "Primitiven":

- Prädikate, die zu Vorbedingungen von Regeln verknüpft werden. Sie leisten die Abfrage von Werten der internen Zustandsvariablen der Agenten, aber auch anderer Einheiten, sowie Positions- oder andere Vergleiche, usw..
- Aktionsprimitive implementieren Effektorkommandos, die Agenten – wenn sie vollständig mit Argumenten spezifiziert sind – ausführen. Ein derartiges Kommando entspricht dabei einem Aktionsaufruf zur Bewegung, Änderung von Werten des internen Zustands, Versenden von Nachrichten, usw..
- Referenzierungen stellen "Zeiger" auf Objekte und Werte aus der Umwelt des Agenten dar und realisieren dadurch eine deiktische Repräsentation von Situationsaspekten: Bei einem Kommando "Bewege in Richtung <Objekt xy>" liefert ein Referenzierungsprimitiv den Zeiger auf das <Objekt xy>, das von einem Aktionsprimitiv "Bewege in Richtung <?>" verwendet werden kann, um die Bewegungsrichtung zu bestimmen.
- Basisoperatoren realisieren Filterbedingungen für Objektabfragen, (numerische) Verküpfungen, usw. In dem obigen Beispiel könnte für die Bestimmung von "Richtung von ... " auch ein entsprechender Basisoperator verwendet werden. Naheliegende Beispiele sind auch Primitive zur Addition und Multiplikation von Werten von Zustandsvariablen.

Hinter jedem dieser Primitive verbirgt sich eine Methode, durch die die beabsichtigten Abfragen, Vergleiche oder auch Veränderungen implementiert werden. Allerdings muß, um diese Methoden in den Rahmen integrieren zu können, die Methode "explizit" gemacht

[5]Eine Erzeugung durch Vererbung analog zu den domänenspezifischen Klassen hätte die Zahl der Objekte während der Simulation – zu jedem Simulationsobjekt käme jeweils noch ein Satz Zustandsvariablen-Objekte – potenziert. Deswegen wurde darauf verzichtet und anstelle dessen eine einzige Instanz für jede Zustandsvariable angelegt, die über einen Token (Name) mit dem aktuellen Wert im Simulationsobjekt referenziert wird.

werden. Dies geschieht auf die gleiche Weise, wie bei den anwendungsspezifischen Klassen: durch Beschreibungsobjekte, die auf einer Meta-Ebene diese Primitive und potentielle Aufrufe repräsentieren.

Metabeschreibungen für die Primitiv-Methoden. Für jede Methode, mit der ein Primitiv für den vorgestellten Modellierungsrahmen realisiert werden soll, muß ein explizites Beschreibungsobjekt angegeben werden. Dieses Beschreibungsobjekt beinhaltet die in Tabelle 6.2 aufgelisteten Aspekte. Deren Organisation wird in Abbildung 6.5 verdeutlicht.

Tabelle 6.2 Auflistung der Angaben, die in einem Beschreibungsobjekt eines Primitivs gemacht werden sollen.

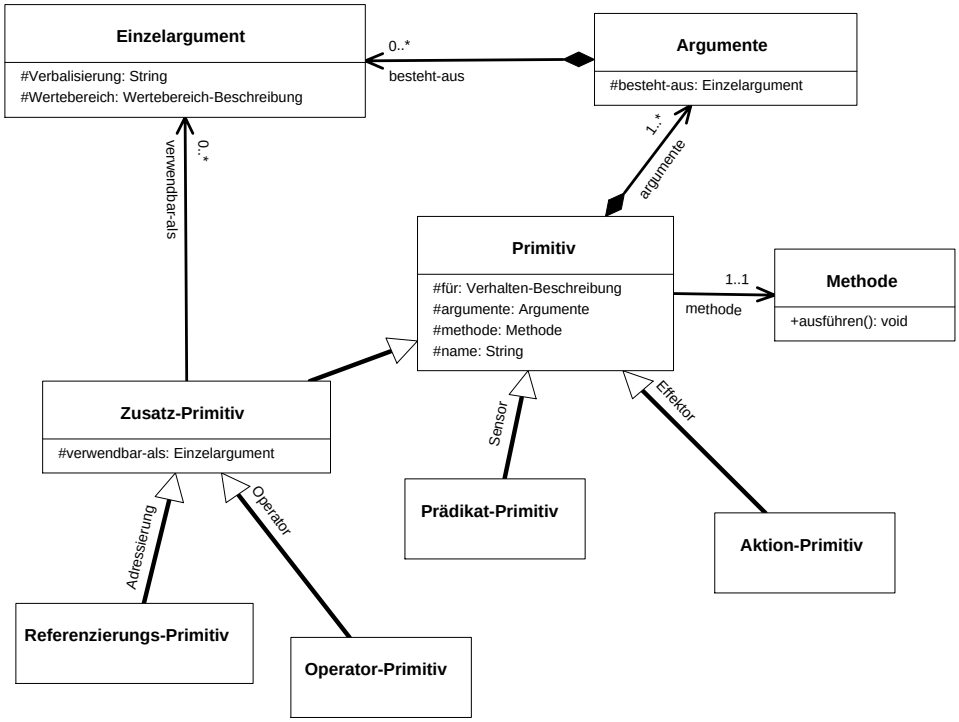
Name des Primitivs
Name der zugrundeliegenden CLOS-Methode
Beschreibung der Argumente: Namen (bzw. Schlüsselörter), sinnvolle Kombinationen und – der wichtigste Aspekt – Charakterisierung des Wertebereichs für jedes Argument. Die Möglichkeiten dafür reichen von Spezifikation von Primitiven, deren Aufruf eingesetzt werden kann, bis hin zur Aufzählung absoluter Alternativen.
Charakterisierung des Rückgabewerts
Verbalisierung möglicher Aufrufe
Erklärender Text zum Primitiv

Diese Objekte zur Primitivbeschreibung sind nicht nur bei der Modellrepräsentation wichtig, sondern die Information – insbesondere zur Verbalisierung von Kommandos, Aussagen, usw. – bildet die Basis für die graphische Modellierumgebung. Durch Interpretation dieser Beschreibungen kann man es möglich machen, Primitivaufrufe durch graphische Operationen (“Point and Click”) zusammenzustellen. Die Umsetzung einer Primitivbeschreibung wird im Abbildung 7.4 anhand eines Beispiels visualisiert.

Nur Primitive, die mit einem derartigen Metaobjekt beschrieben wurden, sind für Regeln, Aktionskommandos, usw. verwendbar. Das gilt auch für alle Vorgaben, die in der Simulationsumgebung gemacht werden. Problematisch an diesem Konzept ist allerdings, daß die eigentliche Semantik dieser Primitive durch den CLOS-Code der entsprechenden Methode definiert wird. Das bedeutet, daß keine Logik oder anderweitige vollständig deklarative Repräsentation als Basis dieser Verhaltensbeschreibung zur Verfügung steht, um die Effekte von Aktionen oder der Wahrheitsgehalt von Prädikaten darzustellen. Die an dieser Stelle fehlende Explizitheit der Effekte einer atomaren Aktion oder Prädikats ist verantwortlich dafür, daß die so repräsentierten Verhaltensbausteine nicht zur Verwendung in einem klassischen Planungsmechanismus taugen. Für die Beschreibung von Verhalten ist diese Form durchaus ausreichend. Dennoch ist es wichtig, die Beschreibung der Primitive ausführlich zu halten, um die Effekte eines Primitivs für alle Modellierer in ausreichender Genauigkeit einsehbar zu machen.

Vorgaben für einfache Primitiv-Methoden mit Beschreibungsobjekten. Wie oben erwähnt, ist die Frage nach Primitiven eines Modellierrahmens für die Einsetzbarkeit dieses Schemas entscheidend, denn diese atomaren Bausteine für Regeln, Aktivitäten, usw. bestimmen das

Abbildung 6.5 Schema der Bestandteile von Beschreibungsobjekten für Primitive



Abstraktionsniveau auf dem das Modell dargestellt werden kann, bzw. welche “Gräben” zwischen der Spezifikation und der lauffähigen Simulation überwunden werden müssen.

Eine domänenunabhängige, vorgegebene Primitivmenge kann die Ausgangsbasis für jedes Verhaltensmodell bilden. Folgende inhaltliche Aspekte müssen bei einer solchen Minimalmenge an primitiver Funktionalität betrachtet werden, die dennoch ausreichend viele mögliche Anwendungsszenarien abdeckt:

- Primitive, die sich auf die Zustandsvariablen des Agenten beziehen: Möglichkeiten zur Abfrage und Manipulation von Werten des internen Zustands. Das beinhaltet auch Änderungen und Abfrage der Darstellung und Bewegungsrichtung.
- Primitive zur Behandlung und Betrachtung von Raumrepräsentation und Raum-Relationen: Bewegen, Distanzvergleiche, usw.
- Primitive zum Existenztest, zum Erzeugen und Löschen des Agenten selbst, aber auch anderer Simulationsobjekte.
- Primitive zur Kommunikation könnten – wenn Agentennachrichten über Ressource-Objekte dargestellt werden – durch Erzeugen und Löschen von Objekten realisiert werden. In einer aktuellen Realisierung des Simulators werden allerdings mit Zeitstempel versehene Symbole verschickt und vom Empfänger interpretiert.
- Abfrage-Primitive zur Wahrnehmung von Aspekten an anderen Objekten: Typ, Darstellung, Werte von Zustandsvariablen . . .

- Vergleichsoperatoren, numerische Operatoren wie "+", usw.
- Generelle Operatoren zur Manipulation von Werten – Addition und andere numerische Operationen, Primitive zum "Parsen" von Strings, usw..
- Besondere "Meta-Primitive", die andere Primitive auf eine Menge von Zustandswerten, Objekten, o. ä. anwenden oder die bedingte Primitivaufrufe ermöglichen.

Für eine vollständige Auflistung der einzelnen vom SESAM-Simulator bereitgestellten Prädikate, Aktionen, Referenzierungen und Operatoren sei auf das Handbuch verwiesen [Klügl & Oechslein 1999]. Interessant ist zu bemerken, daß durch die Explizitheit der Verhaltensrepräsentation auch Primitive realisiert werden können, mit denen das Verhaltensmodell selbst abgefragt oder sogar modifiziert werden kann. Auf diese Weise könnte man sogar Agenten simulieren, die das Verhaltensmodell anderer Agenten interpretieren und darauf hin ihr Eigenes modifizieren.

Realisierung von Wahrnehmungsabstraktion und Aktionsskripten über abstrakte Primitive. Mit den oben aufgelisteten Primitiven lassen sich alle Aspekte in der Konfiguration eines Modells manipulieren bzw. abfragen. Dennoch muß für diese Menge an Primitiven eine Erweiterungsmöglichkeit bestehen, um domänenspezifische Aspekte, meist in Form von Abstraktionen oder Abkürzungen zu repräsentieren. Für die einzelnen Typen an Primitiven sieht eine Abstraktion der atomaren Bestandteile wie folgt aus:

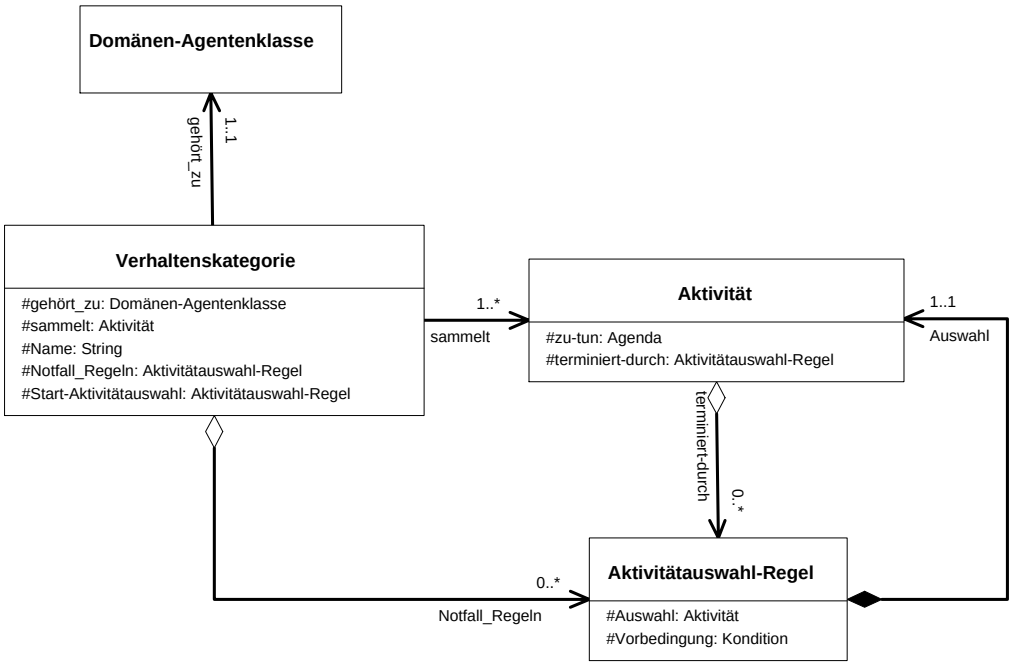
- Aktions-Primitive: Eine abstrakte Aktion besteht aus einer Sequenz von Aktionskommandos, also vollständigen Aktionsaufrufen. Da – wie oben in der Auflistung verschiedener Primitivkategorien angegeben – auch Primitive zur Kombination von Aufrufen existieren, können so beliebige Aktionsskripte, quasi "Unterprogramme", formuliert werden und mit einem speziellen Namen versehen werden. Derartige abstrakte Aktionen können verschachtelt und so als modellspezifische primitive Aktionskommandos in Aktivitäten verwendet werden.
- Prädikats-Primitive fragen Situationsaspekte ab. Demzufolge bildet eine Kombination derartiger Primitive, wie in einer Regel-Vorbedingung über logische Konnektoren, wiederum einer Beschreibung eine Situation, die modellspezifisch benannt und in verschiedenen Kontexten verwendet werden kann. Ein derartiges abstraktes Prädikatsprimitiv kann man auch als Regel interpretieren: Wenn die Situation erfüllt ist, dann ist auch die abstraktere Aussage wahr, die bei der Abfrage des modellspezifischen Primitives quasi rückwärtsverkettet interpretiert wird. Allerdings kann man eine derartige Funktionalität auch als Makro bezeichnen.
- Sonstige Primitive, also Referenzierungs- und Operationsprimitive, können zusammengebaut bzw. verschachtelt werden. So werden modellspezifische "Abkürzungen" bereitgestellt: Für die Adressierung bestimmter Situations- oder Umweltaspekte durch Referenzierungsprimitive oder Abstraktionen ausgehend von Operatoraufrufen. So kann beispielsweise ein "Energie-Potential" eines bestimmten Agenten der Umwelt durch die Addition mehrerer Werte relevanter und abfragbarer Zustandsvariablen eines durch ein Referenzierungsprimitiv beschriebenen Agenten formuliert werden. Nur durch derartig abstrakte Referenzierung wird eine deiktische Adressierung sinnvoll einsetzbar.

Durch solche Abstraktions- und Abkürzungsmechanismen ist der Modellierer in der Lage, die Ebene der Aktionskommandos und Situationsbeschreibungen selbst zu bestimmen. So wird also die Sprache für die Bestandteile von Regeln und Aktivitäten definiert. Durch die modellspezifische Prädikatsprimitive werden zudem die auf Seite 123 vorgestellten Abstraktionsregeln der Wahrnehmungs-Vorverarbeitung realisiert.

6.2.2.2 Regeln, Aktivitäts- und Rollenobjekte

Analog zu den explizit dargestellten Zustandsvariablen, die nur einmal als Zustandsvariablenobjekt repräsentiert, aber über eine Referenz in jeder Instanz einer entsprechenden internen Zustandsbeschreibung adressiert werden, können auch die Strukturen zur Beschreibung des Agentenverhaltens dargestellt werden. Abbildung 6.6 zeigt einen Überblick über die Beziehungen zwischen den einzelnen Klassen. Bei einer Agenteninstanz selbst findet man einen Zeiger auf seine aktuelle Verhaltenskategorie, sowie einen weiteren auf die gerade von ihm verfolgte Aktivität, bzw. auf den entsprechenden Pfad im Und-Oder-Baum einer abstrakten Aktivität.

Abbildung 6.6 Verhaltenskategorien, Aktivitäten und Auswahlregeln



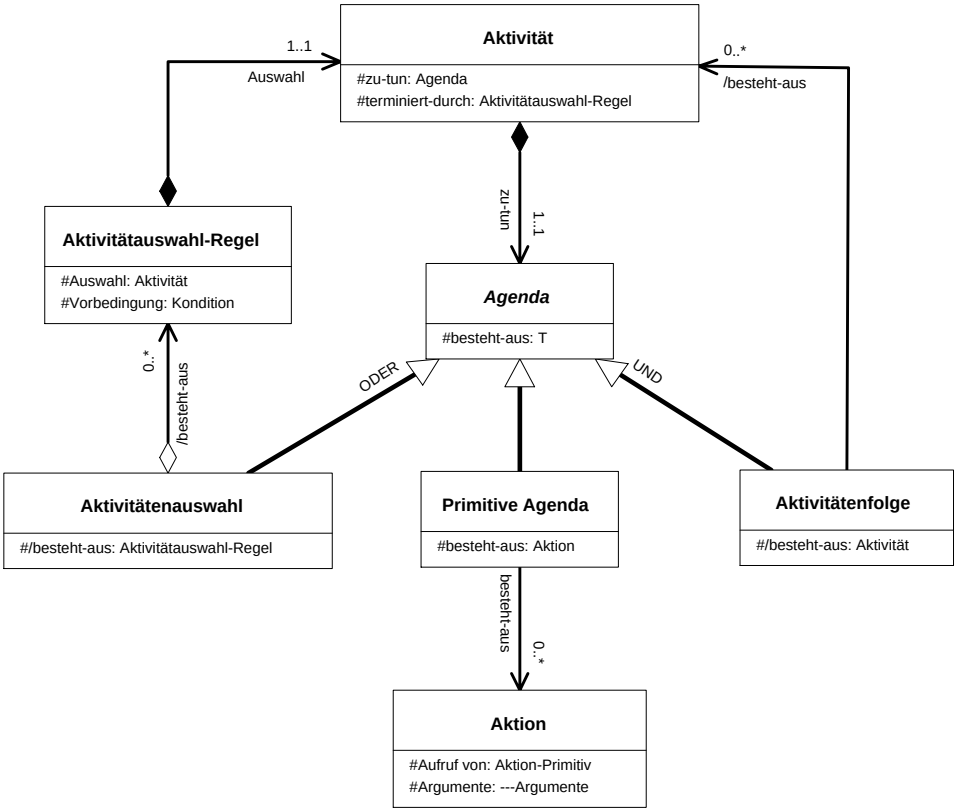
Die einzelnen Klassen sind dabei direkt als Umsetzung der im letzten Kapitel vorgestellten Strukturelemente zu sehen.

Verhaltenskategorien. Eine Verhaltenskategorie besitzt neben einem Namen zwei Attribute, die das zu modellierende Verhalten enthalten: Ein Attribut, im dem alle zu dieser Kategorie gehörenden Aktivitätsobjekte verzeichnet sind und ein Attribut, das die Notfallregeln,

die unabhängig von einer konkreten Aktivität feuern können, verzeichnet. Als weiteres Attribut ist in Abbildung 6.6 das für die Regeln zur Auswahl einer Startaktivität eingeführt: Darin sind Aktivitätsauswahlregeln enthalten, die eine situationsabhängige Default-Auswahl von Aktivitäten ermöglichen. Dies ist sinnvoll, um den Agenten immer in einem definierten Zustand zu halten – siehe dazu auch Abschnitt 6.3.2.

Aktivitäten. Besondere Aufmerksamkeit verdient die Umsetzung des Konzepts der Aktivitäten, insbesondere die Integration der abstrakten Aktivitäten (siehe Abschnitt 5.3.2.3). Abbildung 6.7 verdeutlicht die Repräsentation dieser unterschiedlichen Aktivitätenklassen.

Abbildung 6.7 Aktivitäten mit primitiven Aktionen und abstrakte Aktivitäten – Spezialisierung über die Agenda-Repräsentation.



Aktivitäten mit primitiven Aktionen und abstrakte Aktivitäten werden auf die gleiche Weise ausgewählt – mittels Aktivitätsauswahlregeln. Der große Unterschied besteht dabei in der Behandlung ihrer Agenda, also des Teils, der die Ausführung der Aktivität beschreibt. So besteht nur die Agenda einer primitiven Aktivität aus Aktionskommandos, also Aufrufen von Aktionsprimitiven (siehe oben). Die Agenda einer Aktivitätssequenz besteht dagegen aus einer sortierten Folge von Aktivitäten, während die einer Aktivität mit konkurrierenden Alternativen aus Aktivitätsauswahlregeln besteht. Die Interpretation dieser so repräsentierten Strukturen wird bei der Beschreibung der Aktualisierung, bzw. der genauen Aktivitätsauswahl eines Agenten auf Seite 149 ausführlich erläutert.

6.2.2.3 Spezialfall des Umweltverhaltens

Wie in Abschnitt 5.1.3.2 dargestellt wurde, soll das Umwelt-System die Charakteristik eines Agenten aufweisen, allerdings mit spezieller Bedeutung für die Interaktionen mit dem eigentlichen Agentensystem. Dort wurde auch der Begriff “Ereignis” für die Aktivität des Umweltsystems eingeführt. Um der Konnotation eines Ereignisses als besondere Zustandsänderung ohne längere Dauer Rechnung zu tragen, wird dieses Verhalten besonders behandelt: Ein Umwelt-System befindet sich per default in einer “Null-Aktivität”, in der mit keinerlei Aktionen die Einheiten in dieser Umwelt manipuliert werden. Nur wenn ein Ereignis über die entsprechende Regeln ausgewählt werden kann, wirken die Aktionen, die die Agenda dieses Ereignisses bilden. Nach einer einmaligen Ausführung terminiert das Ereignis wieder – wie aufgrund einer Default-Terminierungsregel, die wiederum die Null-Aktivität auswählen würde. Wegen dieser Kurzfristigkeit der Ereignisse ist eine Strukturierung über abstrakte Ereignisse nicht sinnvoll.

6.3 Funktionsweise des Interpreters

Die oben dargestellten Schemata werden als solche explizit repräsentiert und müssen vom Simulator interpretiert werden, um so die Änderungen am Modell durchzuführen, die während eines Simulationsexperiments vorgenommen werden müssen. Die Beschreibung dieser Interpretation des Multiagentenmodells muß wie das Modell selbst auf zwei Ebenen geschehen: Die Aktualisierung des Gesamtsystems und die Betrachtung auf der Ebene seiner – vor allem aktiven – Bestandteile.

6.3.1 Vorgehensweise auf der Ebene des Gesamtsystems

Auf der Ebene des Gesamtsystems ist vor allem die Frage zu klären, wie und wann die Aktualisierung der einzelnen Komponenten organisiert wird. Ein Multiagentensystem besteht an sich aus einem System gleichzeitig agierenden Einheiten. Demzufolge müßte auch bei der Simulation jeder Agent als paralleler Prozess simuliert werden. Meist sind aber nur sequentiell arbeitende Rechner verfügbar.

6.3.1.1 Aktualisierungsmethoden, verteilte Prozesse und sequentielle Rechner

Es gibt einige Möglichkeiten, auch auf sequentiell arbeitenden Rechnern parallele Prozesse zu simulieren. Im folgenden sollen Fragen der Verteilung auf mehrere Rechner und allgemeine Betrachtungen zur Sequentialisierung paralleler Prozesse behandelt werden.

Verteilung der Multiagentensimulation auf mehrere Rechner. Hat man genügend Rechner zur Verfügung, kann man jedem Agenten einen Rechner zuordnen und so auf einem Rechnernetz das Multiagentenmodell abarbeiten. Dies dient zwar in erster Linie der Beschleunigung der Simulationsexperimente. Man kann aber so wirkliche Parallelität – zumindest bei der internen Aktualisierung von Agenten – erreichen. Allerdings gibt es dabei mehrere Schwierigkeiten: Eine derartige Verteilung macht nur Sinn, wenn jeder Agent für sich relativ aufwendige Berechnungen durchführt und im Verhältnis wenig mit anderen – und vor

allem mit dem gemeinsamen Umweltsystem – interagiert. Ansonsten ist der Kommunikationsaufwand relativ zum Agentenprozess zu groß, um auf diese Weise langwierige Simulationsexperimente in sinnvoller Zeit durchzuführen. In [Belli & Hain 1998] wurde bereits testweise eine verteilte Version des SESAM-Simulators implementiert. Allerdings wurden dabei nicht einzelne Agenten auf verschiedene Rechner verteilt, sondern Umweltsysteme zusammen mit allen auf ihnen lokalisierten Agenten. Dabei waren Interaktionen zwischen Agenten auf das Umweltsystem, bzw. auf Einheiten im selben Umweltsystem beschränkt. Agenten konnten nur an ganz bestimmten Stellen zwischen Umweltsystem – und somit auch zwischen Rechnern – wechseln. Dabei stieß man auf ein auch in der verteilten Simulationstechnik bekanntes Synchronisationsproblem: Beim Wechseln eines Agenten von einem System in das andere, muß in beiden Systemen der gleiche Zeitpunkt simuliert werden. Optimistische Simulation mit Rollbacks oder ähnliche Ansätze aus der verteilten Simulation[6] sind bei taktweise wechselnden Agenten nicht sinnvoll. Demzufolge wurde das Gesamtsystem durch einen globalen Takt synchronisiert. Durch eine Balancierung – die Systeme mit den meisten Agenten wurde dem schnellsten Rechner zugeordnet – wurde zumindest versucht, das Gesamtsystem so gut wie möglich zu verteilen.

Insgesamt wäre dennoch eine Parallelisierung auf Agentenebene deutlich sinnvoller, denn konzeptionell sind es genau diese Einheiten, die gleichzeitig handeln. Im folgenden werden nun Ansätze betrachtet, wie ein Multiagentenmodell auf einem Ein-Prozessor-System simuliert werden kann.

Betrachtungen zur Simulationszeit. Die kritische Frage gilt dabei zunächst der Organisation der Simulationszeit: Dabei existieren verschiedene Modi: Zeitfortschreitung unabhängig von der Aktualisierung der Agenten oder durch diese ausgelöst.

Eine asynchrone Aktualisierung der Agenten entspricht einer Umsetzung einer Parallelverarbeitung, wie sie meist auf Betriebssystemebene nachgebildet wird: Jeder Agent entspricht einem Prozess, der virtuell ohne Unterbrechung rechnet. Sensordaten werden – wie schon in Abbildung 5.8 angedeutet – unabhängig vom Reasoning zu Zeitpunkten erhoben, wenn die Daten verfügbar sind. Für den Agenten sinnvolle Effektorkommandos müssen für den Simulator bereitstehen, damit – wenn der Agent seine Aktionen ausführen darf – diese vom Simulator abgeholt werden können und dafür rechtzeitig zur Verfügung stehen. Der Agent kennt den Zeitpunkt für neue Sensordaten bzw. die Effektorabfragen nicht ohne weiteres[7]. Derartige Vorgehensweisen können bei der Ausführung eines Multiagentenmodells sinnvoll sein, wenn die Zeit, die ein Agenten im Vergleich zu anderen beim "Nachdenken" verbringt, mit zum Untersuchungsgegenstand gehört. Andererseits zwingt ein derartiges asynchrones Aktualisieren einen Modellierer dazu, bei der Umsetzung seines Modells die Dauer von Berechnungen oder andere implementierungstechnische Details mitzuberücksichtigen. Dabei werden für den Modellierer Fragen relevant, wie z.B. die Prädikatsprimitive in Regelvorbedingungen angeordnet oder sogar implementiert sind, in welcher Reihenfolge Positionen abgefragt, oder welche Tests verwendet werden.

Bei einer synchronen Aktualisierung werden die Zyklen der Agenten so ausgelöst, daß die Zeit, die ein Agent mit der Auswahl seiner aktuellen Aktivität verbringt, unwichtig wird.

[6] Das "schnellere" System wird optimistisch voraus-simuliert und notfalls in einen früheren Zustand zurückversetzt.

[7] Außer wie beispielsweise beim RoboCup-Simulator [RoboCup 2000], bei dem in festgelegten Zeitabständen bzw. Zeitpunkten die Sensordaten zur Verfügung stehen und Effektorkommandos "abgefragt" werden.

Alle Agenten werden aktualisiert. Danach wird die Simulationszeit um eine Einheit erhöht. Diese Vorgehensweise kann "zyklus-gesteuert" genannt werden, da sie nur sehr bedingt der üblichen kontinuierlichen, bzw. "time-stepped" Methode entspricht. Bei letzterer werden zwar auch alle Gleichungen "durchgerechnet", aber die dabei verwendeten Zeitintervalle können unabhängig von den Gleichungen erhöht werden. Bei der hier verwendeten Simulationsroutine ist dies nicht der Fall: Der Zeitfortschritt wird zwar extern realisiert, aber indirekt von den Agenten selbst vorgegeben. Abhängig davon, wie genau der Aktualisierungszyklus der einzelnen Einheiten miteinander verzahnt wird, kann man mehr oder weniger Parallelität nachbilden. Das bedeutet allerdings für die Simulationszeit, daß sie nicht direkt an die reale Zeit, z.B. über die Rechneruhr, gebunden werden kann[8].

Verzahnte Aktualisierung. S. Russell und P. Norvig schlagen für ein Agentensystem eine einfache Basis-Version eines Simulators vor [Russell & Norvig 1995], der die verschiedenen Phasen bei der Aktualisierung eines Agenten jeweils für das gesamte System trennt. Dabei läßt der Simulator zyklenweise jedem Agenten einen Ausschnitt des Umweltzustands als Wahrnehmung zukommen, stößt anschließend die Aktionsselektion an, "sammelt" die einzelnen Aktionen und aktualisiert in einem letzten Schritt die Modellkonfiguration ausgehend von diesen Aktionen. Folgender Algorithmus verdeutlicht die Vorgehensweise:

Verzahnte Aktualisierung eines Agentensystems

```
Wiederhole
  1. für jeden Agenten:
     Generiere Wahrnehmung für jeden Agenten aus gesamter
     Umwelt und schicke an Agenten
  2. für jeden Agenten:
     Aktualisiere Agenten mit Wahrnehmung und speichere
     die vom Agenten zurückgelieferte Aktion.
  3. Aktualisiere globalen Zustand auf Basis der
     Agentenaktionen
```

Dabei werden spezielle Formen von Einheiten, wie Umwelt-Systeme oder Ressource-Objekte, nicht betrachtet. Dieser Simulationsalgorithmus ließe sich aber dementsprechend erweitern.

Charakteristisch ist dagegen, daß alle Agenten so zwar nicht gleichzeitig handeln, aber auf die gleiche Situation reagieren. Also geschieht zumindest die Wahrnehmung parallel. Während der Aktionsselektion im zweiten Schritt passiert keinerlei Interaktion: Jeder Agent wählt für sich die nächste Aktion. Echt sequentiell ist also nur der dritte Schritt: Die Ausführung der Agentenaktionen. Dabei können genau die Probleme auftauchen, die auch bei einer asynchronen Aktualisierung zu finden sind: Bei der Ausführung der Aktion eines Agenten verändert sich die Umwelt so, daß die Aktion eines anderen Agenten nicht mehr möglich, bzw. sinnvoll ist. Allerdings hat der Agent zu diesem Zeitpunkt keine Kontrolle mehr über die Ausführung – die Aktion "klappt" einfach nicht. Erst im nächsten Wahrnehmungszyklus erhält der Agent die Rückmeldung, ob seine Aktion erfolgreich war. Bei einer festen Reihenfolge bei der Aktionsausführung können dabei immer die selben Agenten benachteiligt werden. Der Modellierer muß sich dabei im besonderen um die Synchronisierung des Verhaltens der einzelnen Agenten kümmern, z.B. um Deadlocks zu vermeiden.

[8] Allerhöchstens um unterschiedliche Aktualisierungsdauer von Agenten vergleichen zu können.

6.3.1.2 Vorgehensweise in SESAM

Obwohl der oben beschriebene Simulationsalgorithmus am besten dem entspricht, was der Simulation eines parallel arbeitenden Agentensystems auf einem sequentiell arbeitenden Rechner nahe kommt, wurde dennoch eine einfachere Simulationsroutine verwendet. Dies liegt daran, daß komplizierte Konfliktauflösungsmechanismen bei der Feststellung, welche Aktion – ganz oder möglicherweise nur teilweise – ausgeführt werden darf, vermieden werden sollten. Jeder der verwendeten Mechanismen sollte ja einem Modellierer transparent gemacht werden können. Auch steht zu befürchten, daß durch die Unsicherheit, daß eine Aktion des Agenten überhaupt nicht ausgeführt wird, das Verhaltensmodell unnötig kompliziert werden kann. Für die Betrachtung eines Agenten sollte also innerhalb eines Aktualisierungszyklus keine Repräsentation von reaktivem Verhalten notwendig sein, bzw. sich durch eine kontextabhängige Formulierung der Aktionen in einer Aktivität behandeln lassen.

Die verwendete – demzufolge sehr einfache – Simulationsroutine aktualisiert jeden Agenten nacheinander jeweils vollständig. Dies ist einerseits gut, da so das Verhaltensmodell der Agenten einfacher gestaltet werden kann – der Modellierer muß sich um keinerlei Konfliktauflösung kümmern. Allerdings ignoriert diese Art der Vorgehensweise die Parallelität in einem Agentensystem – kein Agent handelt gleichzeitig mit anderen.

So kann ein Agent die Effekte der Aktionen eines anderen Agenten bereits in der Umwelt wahrnehmen, während der Zyklus noch nicht abgeschlossen ist, da er bereits auf der modifizierten Umwelt operiert. Nach einem Zyklus wird die Aktualisierungsreihenfolge der Agenten zufällig geändert. Auf diese Weise hat der Modellierer keine Kontrolle über die Reihenfolge, was bei der Formulierung von Koordination zwischen Agenten und ihren Aktivitäten beachtet werden muß. Ein deterministisches Mischen wäre an sich auch möglich, denn es könnte fair gestaltet werden. Allerdings ist ein stochastischer “Shuffle” grundsätzlich fairer, da der Modellierer keinerlei Hinweis auf den Zyklus der Aktualisierungsreihenfolge hat.

So ergibt sich folgender Algorithmus für die Aktualisierung eines Agentensystems (auch mit mehreren Umweltsystemen)

Aktualisierungszyklus in SESAM

```
Wiederhole pro Zyklus
  1. für jedes Umweltsystem
     (a) Aktualisiere Umweltsystem
     (b) für jeden Agenten in diesem System
         Aktualisiere Agenten komplett, inklusive der
         Ausführung seiner Aktionen
     (c) Zufälliges Permutieren der Aktualisierungsrei-
         henfolge der Agenten innerhalb des Umweltsystems
```

Dazu kommen noch Routinen zur Animation, Protokollierung, usw., die auf die Aktualisierung des Gesamtsystems keinen Effekt haben.

6.3.2 Aktualisierung der einzelnen Bestandteile

Wie in Abschnitt 6.2.1.2 ausgeführt, werden in einem Zyklus die "Aktualisieren"-Methoden der einzelnen Grundklassen "Agent", "Ressource" und "Bezugssystem" angestoßen. In diesen Methoden wird der Interpreter des den einzelnen Objekten zugeordneten Verhaltensmodells realisiert. Dabei gibt es in allen drei Klassen gemeinsame Mechanismen zur Aktualisierung der internen Zustandsbeschreibung eines Objekts.

Die Aktualisierung der Werte der dynamischen Zustandsvariablen geschieht bei allen Instanzen eines "Systems" wie folgt: Es wird der Aktionsprimitiv-Aufruf, der bei der Zustandsvariable angegeben ist, mit dem aktuellen Wert des gerade betrachteten Objekts bei eben dieser Zustandsvariable aufgerufen und der neue Wert gespeichert. Analog ist das Vorgehen bei den Werten für die Eingabevariablen eines Agenten, die ebenso als Zustandsvariablen realisiert sind. Eine besondere Betrachtung und Behandlung der Reihenfolge bei der Aktualisierung wird nicht vorgenommen.

Im folgenden werden die dabei verwendeten, speziellen Aktualisierungsalgorithmen zusammenfassend erläutert.

6.3.2.1 Agent

Die Aktualisierung eines Agenten geschieht über den unten stehenden Algorithmus. Die einzelnen Schritte werden danach erläutert.

Aktualisierung eines Agenten

1. Update der Eingabevariablen
2. Wähle aktuelle Aktivität:
   (a) Teste Notfallregeln der zugeordneten Rolle.
   (b) **Neue-Aktivität** ← Konfliktauflösung der möglichen Notfall-Aktivitäten
   (c) Wenn keine **Neue-Aktivität**, dann teste Terminierungsregeln der aktuellen Aktivität[a]
   Neue-Aktivität ← Konfliktauflösung der möglichen Nachfolge-Aktivitäten
   (d) Wenn **Neue-Aktivität**, dann setzte sie als aktuelle Aktivität und terminiere alte (also führe End-Aktion aus, falls vorhanden).
   (e) Verfeinere aktuelle Aktivität. Falls dabei festgestellt wird, daß die aktuelle Aktivität terminiert, wähle über Start-Regeln eine Default-Aktivität.
3. Führe Aktionen der aktuellen primitiven Aktivität aus.
4. Update der dynamischen Zustandsvariablen

[a] Bei einer Struktur aus abstrakten Aktivitäten wird immer nur die Regelmenge der primitiven "Blatt"-Aktivität getestet.

Wie oben bemerkt, werden also nur die Regeln getestet, die als nicht über Aktivitäten in-

dizierte Notfallregeln bei der dem Agenten zugeordneten Verhaltenskategorie eingetragen sind, und diejenigen, die die aktuelle Aktivität terminieren. Die Notfallregeln besitzen dabei die höhere "Priorität" – falls eine dieser Regeln feuern kann, werden die sonstigen Terminierungsregeln nicht mehr betrachtet. Fraglich bleiben bei dem obigen Algorithmus noch die Punkte, wie die Konfliktauflösung vorgenommen wird, wie Aktivitäten "verfeinert" und wie die Aktionen ausgeführt werden.

Konfliktauflösung. Der verwendete Konfliktauflösungsmechanismus ist dabei unabhängig davon, welche Regelmenge gerade betrachtet wird, sondern wird immer, wenn Konflikte auftreten können, verwendet: Dabei wird für alle Trefferaktivitäten[9] deren Priorität (Default 0) und Anzahl in der Menge der aktuell möglichen Trefferaktivitäten multipliziert. Die nächste Aktivität wird dann dementsprechend gewichtet zufällig gewählt[10].

Verfeinerung der Aktivitäten. Aktivitäten müssen verfeinert werden, wenn sie direkt keine primitiven Aktionskommandos beinhalten, sondern als abstrakte Aktivitäten UND- oder ODER-Knoten entsprechend der Skelettplan-Metapher darstellen. Die Verfeinerung wird über ein Durchlaufen des assoziierten Baums realisiert:

Wird ein Knoten – also eine Aktivität – neu ausgewählt, geschieht die Verfeinerung dadurch, daß entweder auf der Basis der angegebenen Auswahlregeln bei einer Alternativen-Aktivität eine Aktivität eine "Stufe weiter unten" ausgewählt wird oder bei einer Aktivitätssequenz die erste Tätigkeit aus der angegeben Agenda selektiert wird.

Analog wird vorgegangen, wenn die aktuelle Aktivität als Blatt in einer derartigen Struktur terminiert: Zunächst wird auf der Ebene darüber untersucht, ob der Vaterknoten terminiert. Dies kann geschehen, wenn bei einem ODER-Knoten keine weitere Regel zur Verfeinerung feuern kann oder bei einer Sequenz gerade mit dem Blatt die letzte Aktivität der Agenda terminiert wurde. Falls dies der Fall ist, werden auf einer übergeordneten Ebene die selben Tests gemacht, bis hin zur Wurzelaktivität. Ansonsten wird der nächste Schritt einer Aktivitätssequenz oder regelbasiert eine neue Aktivität gewählt.

Aktionsausführung. Die Ausführung der Aktionsaufrufe in der Agenda einer primitiven Aktivität geschieht einfach dadurch, daß nacheinander die Methoden, die mit den einzelnen Aktionsprimitiven assoziiert sind, mit den angegebene Argumenten und dem gerade betrachteten Agenten aufgerufen werden. Ändert sich die Aktivität, wird vor der Ausführung der eigentlichen Agenda das Start-Aktionskommando der neuen Aktivität aufgerufen.

6.3.2.2 Ressource

Ressourcen zeichnen sich dadurch aus, daß sie ihre Umwelt nicht aktiv verändern können, also über kein Verhalten verfügen. Damit reduziert sich die Aktualisierung einer Ressource auf das Update ihrer Zustandsbeschreibung (siehe oben).

[9] Also diejenigen Aktivitäten, deren Auswahl aufgrund der Kondition und Feuerwahrscheinlichkeiten ihrer Regel möglich ist.

[10] Man trägt alle Gewichte in dem Intervall $[0, 1]$ auf, zieht eine Zufallszahl zwischen $[0, 1)$ und wählt die Aktivität deren Intervall getroffen wird.

6.3.2.3 Aktualisierung der Umwelt

Das Verhalten eines Bezugssystems ist im Vergleich zu den Agenten relativ einfach – siehe dazu 6.2.2.3. Im Folgenden wird die konkrete Vorgehensweise erläutert.

Aktualisierung eines Umweltsystems

```
1. Behandle Ereignisse

   (a) Teste Regeln zur Ereignisauswahl

   (b) Für jedes Ereignis, das ausgewählt werden konnte
       (in zufälliger Reihenfolge):
       Führe Aktionskommandos der Ereignisagenda in
       angegebener Reihenfolge aus.

2. Update der dynamischen Zustandsvariablen
```

Die Regeln zur Auswahl eines Ereignisses besitzen die selbe Struktur wie Aktivitätsauswahl-Regeln und werden über das Beschreibungsobjekt zur modellspezifischen Bezugssystem-Klasse verwaltet.

6.4 Betrachtungen zur Effizienz

Wie in Abschnitt 4.2.3.2 bereits erklärt, ist die Simulationsgeschwindigkeit bei Multiagentensimulationen von zentraler Bedeutung. Gerade bei dem hier verfolgten Aktualisierungsparadigma, bei dem in jedem Zyklus jeder Agent betrachtet und aktualisiert werden muß, ist die Untersuchung der maximalen Zahl von mit praktikablen Aufwand modellierbaren und simulierbaren Agenten wichtig. Nur wenn bei Experimenten eine ausreichend große Zahl an Einheiten mit interessantem Verhaltensrepertoire dargestellt werden kann, ist eine Modellier- und Experimentierumgebung für Fachexperten relevant. Können keine Experimente in einer relevanten Größenordnung durchgeführt werden, wird eine Modellierumgebung nicht wirklich zum Einsatz kommen.

Die Dauer eines Simulationsexperiments wird dabei durch folgende Bestandteile des Aktualisierungszyklus bestimmt:

$$\begin{gathered}(\text{Dauer der Aktualisierung der Umwelt}) + \\ \textstyle\sum_j (\text{Dauer der Aktualisierung des Agenten } j) + \\ \text{Simulationsinfrastruktur (Animation, Datenaufzeichnung)}\end{gathered}$$

Die Bestandteile der Aktualisierungsschritte kann man aus den oben dargestellten Algorithmen direkt herauslesen: Die Behandlung des Umweltsystems zerfällt in zwei Teile: in ein Testen und Ausführen von Ereignissen und eine Aktualisierung von Zustandsvariablen. Auch bei jedem Agenten werden Zustandsvariablen aktualisiert (Eingabevariablen und interner Zustand), die aktuelle Aktivität bestimmt und deren Aktionen ausgeführt. Zusätzlich zu dieser Interpretation des Verhaltensmodells kosten auch Methoden der Animation, Datenaufzeichnung usw. Rechenzeit.

Für die Bestimmung der Dauer eines derartigen Simulationsschritts gibt es drei Determinanten: der Modellierer, der Interpreter und die Hardware. Die dabei im einzelnen relevanten Einflußgrößen und möglichen Optimierungsschritte werden im Folgenden näher betrachtet.

Die wichtigste Determinante für die Dauer der Aktualisierung ist der Modellierer, der – wenn er die Ansätze, die das Repräsentationsschema und sein Interpreter bieten, ausnutzt – eine Beschleunigung um mehrere Größenordnungen im Vergleich zu einem naiven Ansatz erreichen kann. Der Modellierer kann bei folgenden Aspekten "kooperieren":

- Bei der Identifikation der wirklich aktiven Bestandteile – also der Teile des Modells, die über ein "echtes" Verhalten verfügen: Nur diese müssen vom Interpreter getestet und aktualisiert werden. Das Repräsentationsschema erlaubt dabei eine Kategorisierung auf folgenden zwei Ebenen.

  - Auf der Einheitenebene kann der Benutzer passive Bestandteile als solche explizit charakterisieren (siehe Abschnitt 5.1.3.3). Der Interpreter verwaltet und behandelt diese getrennt von den als aktiv gekennzeichneten Agenten. Ein Test, ob eine Einheit ihr Verhalten ändert, ist unnötig, wenn dieses Objekt nur über ein "Null-Verhalten" verfügt. Beispielsweise bei einer Blütenwiese mit mehreren Tausend passiven Blüten, die von Agent in bestimmten Mustern besucht werden, wäre es also ungeschickt, die Blüten als Agenten zu behandeln.
  - Auch auf der Ebene des internen Zustands kann der Modellierer Zustandsvariablen als "aktiv", also dynamisch oder als "passiv", d.h. nur über Aktionen veränderbar charakterisieren. Der Interpreter kann diese Kategorisierung ausnutzen und bei der Aktualisierung des internen Zustandsvariablenvektors alle nicht dynamischen Variablen ignorieren.

  Alleine eine konsequente Kategorisierung in passive und aktive Bestandteile des Modells kann ein Simulationsexperiment um mehrere Größenordnungen beschleunigen. Blütenwiesen mit mehreren zehntausend Blüten kosten nur Speicherplatz und beeinflussen die Experimentdauer nur indirekt[11], wenn eine einzelne Blüte als völlig passives Objekt ohne eigenständig dynamische Zustandsvariablen repräsentiert wird.

- Bei der Identifikation der relevanten Situationen und deren Behandlung: Diese besitzen hauptsächlich Einfluß auf die Dauer des internen Reasoning-Prozesses eines Agenten. Wie in Abschnitt 6.3.2.1 dargestellt, testet der Interpreter die Notfallregeln der Rolle, die dem Agenten zugewiesen wurde, und die Terminierungsregeln der aktuellen Aktivität des Agenten. Rolle und Aktivität sind dabei Behälter, über die diese Regelmengen direkt referenziert werden können. Nutzt der Modellierer also die gegebenen Strukturierungsmöglichkeiten, ist die in jedem Zyklus getestete Regelmenge nicht nur sehr klein, sondern auch schnell zugreifbar.

  Auf diese Weise beeinflußt der Modellierer, wieviele Regeln während des Reasoning des Agenten abgetestet werden. Auch bei der Wahrnehmung des Agenten ist es entscheidend, auf welche Teile der Umwelt ein Agent Zugriff hat und wie dieser gestaltet ist. Gerade der Radius der Wahrnehmung ist entscheidend. Ein einfaches Beispiel verdeutlicht dies: Ein Agent bewegt sich über die oben erwähnte Blütenwiese auf der Suche nach einer sehr seltenen Blütenform b. Verfügt der Agent über einen Wahrnehmungsradius von nur 20 Feldern, müssen in jedem Zyklus $20 \times 20 = 400$ Felder dahingehend betrachtet werden, ob auf dem Feld sich eine derartige Blüte befindet. Gibt es

[11] Der Einfluß tritt also nur bei Speichermangel auf, bzw. bei Operationen, die die gesamte Modellkonfiguration betreffen, z.B. Speichern der Situation, usw.

auf einem Feld gleichzeitig mehrere Objekte, vervielfachen sich die zur Bestimmung der wahrnehmbaren *b*-Blüten notwendigen Tests. Existieren dagegen auf der Blütenwiese insgesamt nur etwa 100 derartige Ressource-Objekte, ist es geschickter, diese Blütenwahrnehmung über eine Abstandsberechnung zwischen dem Agenten und den entsprechenden *b*-Blüten zu realisieren.

- Bei der Identifikation der Zeitpunkte, in denen sich wirklich etwas ändert: So kann ein Modellierer zu jeder Zustandsvariable und jeder Aktivität einen "Zeitrahmen" angeben. Dieser wurde in den bisherigen Ausführungen nicht erwähnt, da er für das Konzept der Verhaltensbeschreibung und ihrer Umsetzung nicht relevant ist, dagegen sehr wohl bei Betrachtungen zu den Kosten eines Aktualisierungszyklus. Ein Modellierer kann bestimmen, in welchen Zeitabständen ein Agent die Aktionen seiner (primitiven) Aktivität ausführt oder in welchen Intervallen der Wert einer dynamischen Zustandsvariablen neu berechnet wird. In den Aktualisierungszyklen zwischen diesen Zeitpunkten werden die Aktionen, mit denen der Agent auf die Umwelt oder seinen internen Zustand wirkt sowie Wert von Zustandsvariablen vom Interpreter nicht betrachtet.

  Eine ähnliche Wirkung kann man bei der Bestimmung von Ereignissen zur Umweltdynamik erzielen: Will der Modellierer Ereignisse in bestimmten Intervallen stattfinden lassen, kann dies mittels entsprechender Regeln formuliert werden, die in jedem Zyklus abgetestet werden müssen. Um dies zu verhindern verwaltet der Interpreter über eine "Ereignisliste": Kennzeichnet der Benutzer einen derartigen Mechanismus zur Ereignisauswahl und gibt das Intervall an, dann werden die Zyklen, in denen dieses Ereignis geschehen kann, vorausberechnet. Das Testen in jedem Zyklus entfällt.

Auch hier wirkt der deklarative Ansatz der Verhaltensrepräsentation unterstützend, da es relativ leicht ist, ein anfänglich ineffizientes Modell durch ein gleichwertiges effizienteres zu ersetzen. Die Analyse der Darstellung und das Formulieren von äquivalentem Verhalten ist in einem deklarativem Rahmen deutlich einfacher, da der Modellierer explizit formuliert, was er beobachtet hat oder über das System weiss.

Die zweite Determinante – der Interpreter – bestimmt nicht nur durch direkte Ausführung der Verhaltensbeschreibung die Dauer eines Aktualisierungszyklus. Innerhalb des vom Modellierer vorgegebenen Modells können weitere beschleunigende Maßnahmen realisiert werden. Die oben erwähnte Einführung einer Ereignisliste für die Vorausberechnung von Ereignissen gehört schon zu diesem Bereich. Weitere Verbesserungen betreffen vor allem die Wahrnehmung der Agenten. Während des Tests der Vorbedingungen von Regeln in einem Zyklus eines Agenten ist wegen der generellen Aktualisierungsmethode sichergestellt, daß sich die Umwelt des Agenten nicht ändert. Das bedeutet, dass Prädikate, die einmal als wahr befunden wurden, innerhalb dieses Reasoning-Prozesses weiterhin wahr sind. Durch ein Speichern dieser Prädikate kann man bei Regeln mit ähnlichen Vorbedingungen ein nochmaliges Berechnen des Wahrheitsgehalts dieses Prädikats vermeiden. Alternativ kann man diese Gemeinsamkeiten zwischen Regelvorbedingungen in einem Compilierungsschritt so in eine Datenstruktur fassen, dass derartige Prädikate nur einmal betrachtet werden (vergleiche *Rete*-Algorithmus, z.B. beschrieben in [Russell & Norvig 1995]). Durch derartige Maßnahmen konnte der Interpreter – zusammen mit Optimierungen im Kleinen, z.B. bei den primitiven Aktionen und Prädikaten – insbesondere in einem umfangreicheren Reimplementierungsschritt Anfang 1999 – so verbessert werden, daß Simulationsexperimente heu-

te etwa dreimal so schnell durchgeführt werden können, wie mit der einfacheren Version Anfang 1998. Damit einher ging eine Optimierung des Speicherbedarfs, so daß auch eine entsprechend größere Anzahl an Agenten und Ressourcen behandelt werden können.

Wie oben aufgeführt kostet auch die Simulationsinfrastruktur viel Zeit. Deswegen durchlief die Animationskomponente ebenfalls mehrere Iterationen, über die es letztendlich erreicht wurde, daß eine gleichzeitige Animation ein Simulationsexperiment nur noch wie folgt verlangsamt: z.B. beim Ameisenmodell (siehe 8.3.1) mit mehreren Tausend Ameisen und durchschnittlich etwa 10 Regeln, die pro Ameise und Takt abgetestet wurden, war das Verhältnis Interpretation zu Animation etwa vier zu eins. Bei den Experimenten zur Routenwahl im Nagel-Schreckenberg-Modell (siehe Abschnitt 8.2.3) lief ein Simulationsexperiment nach einem Ausschalten der Animation etwa doppelt so schnell. Dabei werden grundsätzlich nur Änderungen dargestellt: Während der Verhaltensinterpretation wird gespeichert, ob sich die Position oder Darstellung geändert hat, nur diese Einheiten werden bei der Animation betrachtet und ihre Änderungen präsentiert. An dieser Stelle ist ein weiteres Optimierungspotential vorhanden, da alle Darstellungsänderungen behandelt werden, unabhängig davon, ob sie auf dem aktuellen Bildschirmausschnitt sichtbar sind. Würde man nur diejenigen aktualisieren, die der Experimentator wirklich sehen kann, kann sich die Animation weiter beschleunigen.

Ein ähnliches Vorgehen wie bei der Animation wurde für die Datenaufzeichnung realisiert. Der Modellierer gibt an, welche Daten ihn bei welcher Art von Objekt interessieren. Die Einheiten, bei denen sich ein derartiges Datum verändert hat, schreiben ihre Änderung in eine Protokolldatei. Es ist vor allem abhängig davon, wieviele Daten in die Datei protokolliert werden müssen. Das Verhältnis der Dauer der Interpretation im Vergleich zur Datenaufzeichnung ist ähnlich wie bei der Animation. Auch hier besteht weiteres Potential zur Verbesserung der Kosten: In der aktuellen Version wird jede Änderung eines vom Experimentator gewünschten Attributs aufgezeichnet. Man kann stattdessen in gewissen Abständen, die ebenfalls der Experimentator angegeben müssen, jeweils alle gewünschten, aktuell vorhandenen Werte ausschreiben. Dies ist sinnvoll, wenn der Benutzer keine taktweise Auflösung der Wertänderungen benötigt, kann aber bei ausreichend großen Abständen die Experimentdauer deutlich verkürzen.

Die *Hardware*, auf der Simulationsexperimente durchgeführt werden, bildet den Rahmen für die generell mögliche Simulationsgeschwindigkeit. Auch hier hat das Ziel, Domänenexperten eine Modellierumgebung für selbständige Arbeiten an die Hand zu geben, Einfluß: Die Experimente sollten auf einem Rechner möglich sein, der diesen Fachexperten üblicherweise zur Verfügung steht. Meist findet man auf Schreibtischen von Biologen oder Psychologen, usw. Rechner in der Größenordnung von PCs.

Die Konsequenz daraus ist, daß Simulationsläufe auf derartigen – weitverbreiteten – Rechnern stattfinden müssen. Allerdings sind diese Computer mittlerweile sehr leistungsfähig und man kann wohl zu recht hoffen, daß die aktuelle Hardware-Entwicklung weiterhin dafür sorgt, daß Simulationsexperimente mit immer mehr Agenten, die jeweils über komplexere Verhaltensmodelle verfügen, durchführbar werden. Bereits 1996 waren Experimente mit etwa 2000 relativ komplexen Agenten (im Ameisenmodell - Abschnitt 8.3.1) in akzeptabler Zeit möglich, mit heutigen Rechner (im Jahr 2000) dürfte nach dem Moore'schen Gesetz (siehe *http://www.research.microsoft.com/ Gray/Moore_ Law.html*: Verdoppelung der Rechenleistung alle 18 Monate) die achtfache Zahl behandelbar sein, da im Modell die Aktualisierungszeit pro Agent unabhängig von der Gesamtzahl der Agenten ist. Doch Rechenleistung alleine ist nicht entscheidend. Die Speicherverfügbarkeit beschränkt ebenfalls die möglichen

Experimente, da für jede Einheit Informationen gespeichert werden müssen, nur wenn die Agenten im Arbeitsspeicher gehalten werden können, kann auf sie während der Aktualisierung schnell zugegriffen werden. Da Speicher sehr billig geworden ist, stellt dies aber kein ernsthaftes Problem mehr dar.

Auf der Basis einer Analyse dieser Einflußgrößen konnte der Interpreter so gestaltet werden, daß auf Experimente in relevanter Größenordnung – also mit mehreren Tausend Agenten mit jeweils interessanten Verhaltensrepertoire – durchführbar waren. Im Kapitel 8 werden diese Experimente aus dem Bereich der Simulation von sozialen Insekten, wie Ameisen und Bienen, näher erläutert. Dabei kommt auch die Größenordnung der Systeme zur Sprache, die tatsächlich simuliert werden konnten.

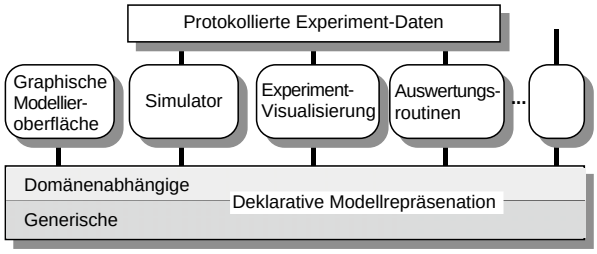
7

Eine Entwicklungs- und Experimentierumgebung für SESAM

7.1 Grundsätzlicher Aufbau

In den letzten beiden Kapiteln wurde das Konzept und die Umsetzung eines erweiterten regelbasierten Schemas vorgestellt, das durch seine Strukturierung und Anpaßbarkeit an eine konkrete Domäne das Formulieren umfangreicher Verhaltensmodelle für Multiagentensysteme ermöglicht. Ein Spezifikationsrahmen alleine reicht nicht aus, um Systemexperten, die nicht auf der Ebene einer Programmiersprache modellieren und simulieren wollen, zu deren Benutzung zu motivieren. Ein einfaches, auch für unerfahrene Modellierer zugängliches Entwickeln und Experimentieren ermöglicht man durch Bereitstellen weiterer Werkzeuge zusätzlich zu einem Simulator. Dadurch wird auf der anderen Seite grundsätzlich das "Rapid Prototyping" unterstützt. Abbildung 7.1 zeigt eine Übersicht der Tools, die aufbauend auf dem Repräsentationsschema bereits entwickelt wurden bzw. zusätzlich möglich sind. Das gesamte Simulationssystem wurde mit dem Namen SESAM bezeichnet, das Akronym steht dabei für "Shell für Simulierte Agentensysteme" [Klügl & Oechslein 1999].

Abbildung 7.1 Auf der Basis einer deklarativen Modellrepräsentation lassen sich verschiedene Module aufsetzen.



Bisher wurden folgende Module für SESAM entwickelt und implementiert:

- In einer *Modellieroberfläche* kann die Spezifikation von Agenten- und Umweltstrukturen und ihren Verhaltensweisen über Formulare, Graphen, u. a. erstellt und manipuliert werden.

- Der im letzten Kapitel ausführlich behandelte *Simulator* interpretiert schrittweise das Modell und zeichnet die Änderungen an der aktuelle Modellkonfiguration auf. Aus diesen Änderungen können Trace-Files erzeugt werden.

- Die *Animationskomponente* benutzt die Daten vom Simulator und Spezifikationen für die Darstellung der Modellbestandteile, um das Experiment während der Simulation zu visualisieren.

- Eine *Auswertungskomponente* ermöglicht es, auf der Basis der durch den Simulator erstellten Protokolle, die Verläufe von Parameterwerten für die Interpretation des Experimentes aufzubereiten.

Diese Werkzeuge werden im folgenden genauer vorgestellt. Denkbar wären weitere Hilfen zur Analyse der Modellspezifikation, beispielsweise zum Testen von Deadlocks in der Verhaltensspezifikation oder sonstiger möglicherweise fehlerhafter Strukturen. Auch wären Alternativen bei der Interpretation der Modellstruktur über die einfache Simulation hinaus möglich, zum Beispiel könnte man die Modellspezifikation in eine andere Repräsentationsform überführen und diese mit einem anderen Simulator interpretieren.

Grundsätzlich müssen derartige Werkzeuge, um ihre Benutzung für viele Benutzergruppen attraktiv, bzw. generell erst verwendbar zu machen, über graphische Oberflächen verfügen. Diese verringern die Distanz, die ein Modellierer bei der Überführung seines Konzeptmodells in ein implementiertes Modell zurücklegen muß [Kuljis 1994]. Allerdings gehören die Zielanwendungen von SESAM weniger in den Bereich edukativer Simulation – wie dies bei *Cocoa* [Cypher 1998] oder *AgentSheets* [Repenning & Summner 1994] der Fall ist – sondern Fachexperten oder Studenten mit wissenschaftlichem Interesse, die ein Modell direkt und schnell spezifizieren und damit experimentieren wollen. SESAM ist nicht nur für Modellier-Novizen gedacht, sondern auch als komfortables Werkzeug für erfahrene Benutzer. Die Konsequenz daraus ist, daß die graphischen Oberfläche konfigurierbar und die gesamte Modellierumgebung mehr auf das schnelle Erzeugen von Prototypen ausgelegt ist.

Bei der Entwicklung der Modellier- und Experimentierumgebung wurden dabei folgende Prinzipien zu Grunde gelegt [Bamberger et al. 1997, Klügl 2000]:

- Durchgängiges Editorkonzept: Jeder strukturell ähnliche Editor besitzt auch graphisch den selben Aufbau. Dies gilt insbesondere für die Formulare.

- Wiederverwendung von Graphikbausteinen: Der Benutzer muß sich nur an ein Konzept, z.B. zur Spezifikation eines Primitivaufrufs gewöhnen: Strukturell gleiche Eingaben müssen immer mit den gleichen graphischen Bausteinen unternommen werden.

- Relevante Information anzeigen, bzw. schnell zugänglich machen: Ist für die Spezifikation eines Sachverhalts weitere Kontextinformation notwendig, soll diese sichtbar oder zumindest schnell abrufbar sein, d.h. ohne aufwendiges Navigieren.

Folgende Abschnitte zeigen, wie diese Prinzipien in den entsprechenden Komponenten einer Modellier- und Simulationsumgebung umgesetzt wurden. Der Erfolg dieses Konzepts und des Repräsentationsrahmens wird im Anschluss daran durch eine Vorstellung ausgewählter Anwendungen von SESAM dargestellt.

7.2 Die Modellierumgebung

Für eine explizite Modellspezifikation kann man eine graphische Benutzeroberfläche entwickeln, die eine Schnittstelle zwischen Benutzer und Spezifikationssprache bildet.

Mit SESAM soll ein Werkzeug entwickelt werden, mit dem Fachexperten schnell und relativ einfach Modelle erstellen, diese validieren und mit ihnen experimentieren können. Dazu wurde eine visuelle Entwicklungsumgebung realisiert. Diese graphische Schnittstelle zur eigentlichen Modellierungssprache verfügt über Strukturierungs- und Abstraktionsmöglichkeiten zusätzlich zur zugrundeliegenden Repräsentationsschema, die die Erstellung komplexer Verhaltensmodelle weiter erleichtern.

Die meisten der dabei verwendeten Editoren, insbesondere Tabellen, Graphen und alle Formulare, beruhen auf in [Gappa 1996] besprochenen generischen Editorformen, die in einer Graphikbibliothek zur Verfügung standen. Im Prinzip kann man bei der Benutzung der Editoren folgende Methodik identifizieren. Auf der detailliertesten Ebene, welche die "direkte" Sicht auf die Objektstrukturen wiedergibt, finden sich die Formulare. Man könnte dabei jede Manipulation eines Bestandteils mit dem Spezifizieren des Inhalts eines Objektattributs gleichsetzen. Im einfachsten Fall kann man in sortierten Listen der Modellobjekte zu den entsprechenden Formulare navigieren. Schon auf dieser Basis könnte eine vollständige Modellieroberfläche geschaffen werden. Wird die Menge an Zustandsvariablen, Aktivitäten, Regeln, usw. aber größer, wird ein derartiges Navigieren aufwendig und die Übersicht über die Menge der Formulare schwierig. Abhilfe bieten Graphen und Tabellen. In Graphen lassen sich Zusammenhänge übersichtlich darstellen, indem Relationen durch Kanten und zusammengehörende Knoten nahe beieinander platziert werden. In Tabellen kann man Teilaspekte aus verschiedenen Objekten in zwei Dimensionen auflisten und auf diese Weise aktuell relevante Daten aggregieren.

Im folgenden sollen nun verschiedene Aspekte der Modellierungsumgebung von SESAM näher ausgeleuchtet werden. An dieser Stelle kann und soll keine vollständige Übersicht und "Bedienungsanleitung" gegeben werden, sondern auf die dabei verfolgten Konzepte und Ideen fokussiert werden. Für weitergehende Informationen über die Bedienung der Editoren und für eine vollständige Beschreibung der Modellieroberfläche sei auf das SESAM – Handbuch [Klügl & Oechslein 1999] verwiesen.

7.2.1 Modellierung der Systemstrukturen

Ausgangspunkt eines jeden SESAM – Modells ist das Anlegen der anwendungsspezifischen Agenten-, Ressourcen und Bezugssystemklassen und das zugehörige Erzeugen der Beschreibungsobjekte (Abbildung 7.2, Editor A).

In speziellen Editoren kann man den Instanzen einer domänenspezifischen Klasse eine Standard-Darstellung (Abbildung 7.2, Editor B) und Startwerte für die Zustandsvariablen (Abbildung 7.2, Editor C) zuordnen. In dieser Tabelle ordnet der Benutzer einer Domänenklasse die für sie relevanten Zustandsvariablen mit den jeweiligen Startwerten zu. Für jede Zustandsvariable, die in der Tabelle nur als Name auftaucht, existiert ein Formular, in dem diese spezifiziert werden kann. Dabei ist auch die domänenabhängige Kategorie editierbar, unter der der Parameter in die Tabelle eingetragen werden kann. Auf diese Weise kann der Benutzer selbst eine Strukturierung der möglicherweise sehr umfangreichen Zustandsvariablenmenge bestimmen. In einer auf die selbe Weise strukturierten Tabelle kann die Default-

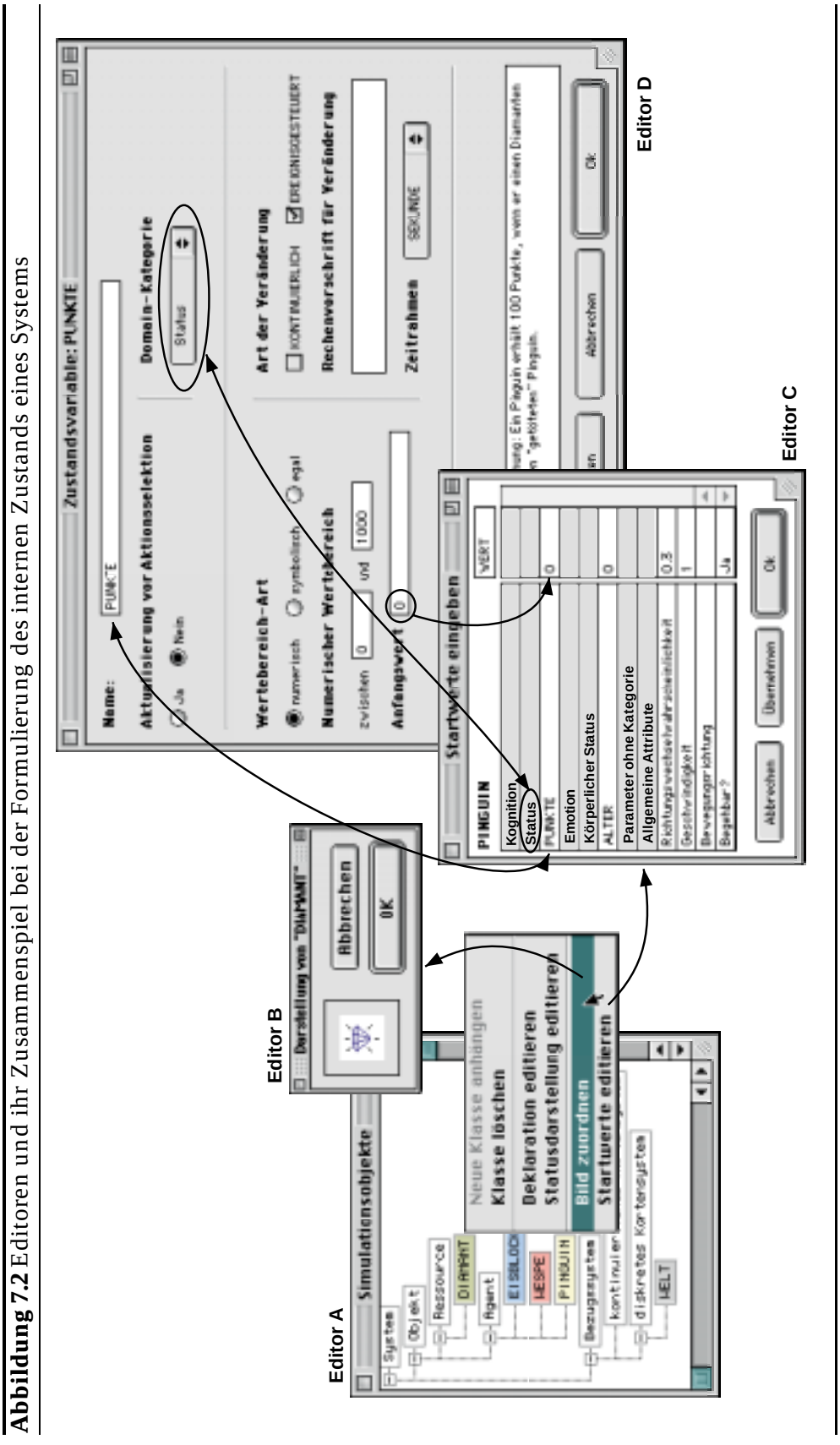


Abbildung 7.2 Editoren und ihr Zusammenspiel bei der Formulierung des internen Zustands eines Systems

Zustandsbeschreibung für Agenten in einer bestimmten Rolle, aber auch für einzelne, individuelle Agenten vorgenommen werden.

Am Formular für Zustandsvariablen (Abbildung 7.2, Editor D) kann man den grundsätzlichen Aufbau eines jeden Modellierungsformulars sehen. *Jeder* derartige Editor ist dreigeteilt. Im obersten Teil kann man allgemeine Information über das Objekt editieren. Dabei steht der Name des Objekts immer an erster Stelle. Es folgen Attribute zur konkreten Einordnung des Objekts. Beim Zustandsvariablenformular ist dies die zeitliche Relation zur Auswahl der aktuellen Aktivität, d. h. ob es sich um eine Eingabevariable handelt (vor Auswahl) oder eine "normale" dynamische Zustandsvariable (nach Auswahl) – siehe Algorithmus auf Seite 148. Das zweite Attribut ist die anwendungsspezifische Kategorie, die zur Kategorisierung der Zustandsvariable in der Tabelle verwendet wird. Im mittleren Block können die domänenunabhängigen Informationen, wie Wertebereichsinformation oder Festlegungen zur Veränderbarkeit eingegeben werden. Bei letzterem kommen die durchgängig für das Editieren von Primitiv-Spezifikationen verwendeten Fenster zum Einsatz, die im nächsten Abschnitt besprochen werden.

Das untere Drittel nimmt bei jedem Formular ein Editierfeld ein, in dem erklärende Bemerkungen zu dem hier editierten Modellobjekt gespeichert werden können. Auf diese Weise kann eine Forderung nach Dokumentierbarkeit erfüllt werden, die nach [Burnett et al. 1995] wichtig für ein Skalieren einer visuellen Programmiersprache ist. In der Buttonleiste werden der "Löschen" und der "Abschicken"-Button von der Tabelle zum Editieren der Startzustandsbeschreibung verdeckt.

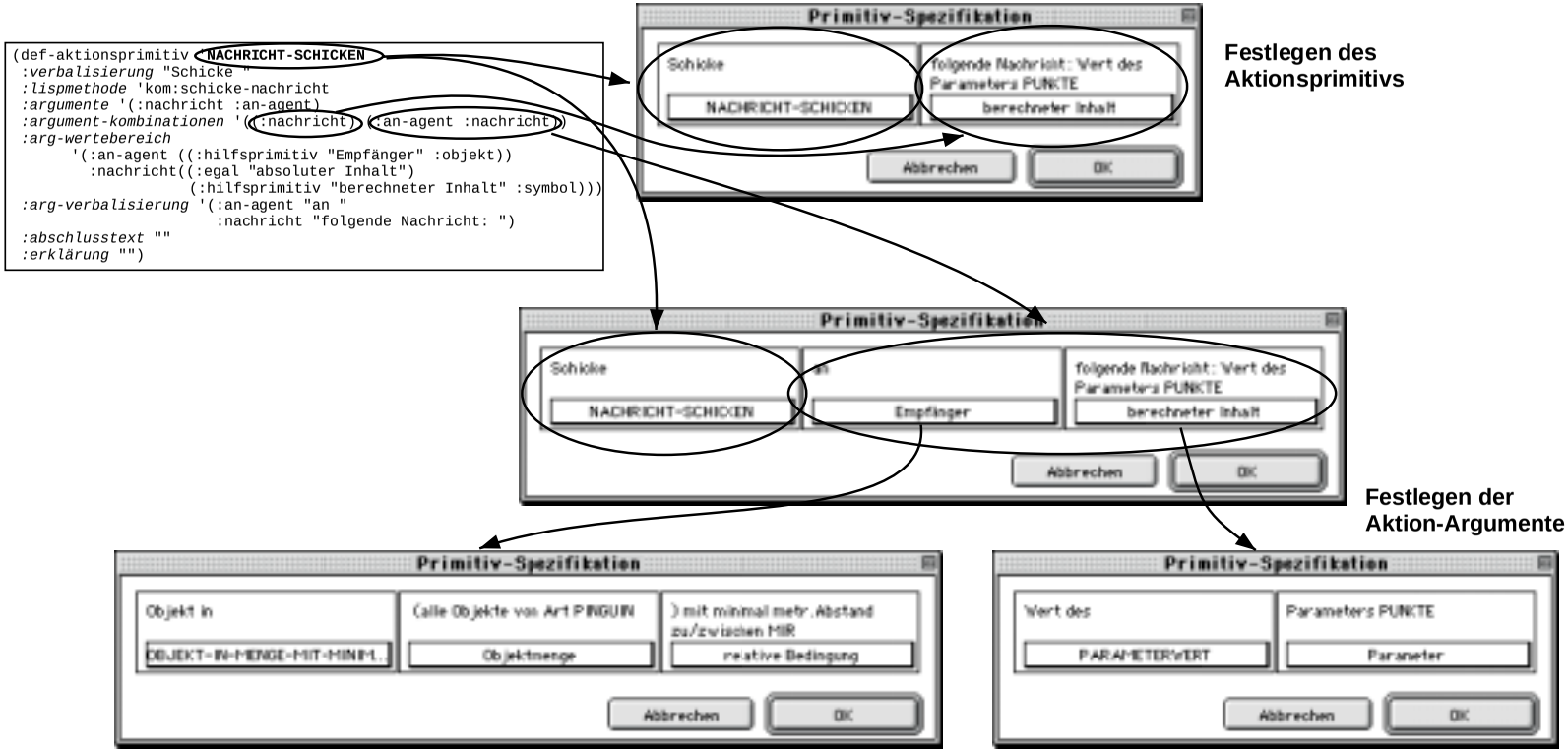
Grundsätzlich kann man in SESAM ein Objekt ausschließlich über Betätigung eines Button in einem Formular löschen. Dies dient zur Vermeidung von Fehlern, denn auf diese Weise muß der Benutzer zumindest nochmals einen kurzen Blick auf die gesamte Spezifikation des Objekts werfen, bevor es aus dem Modell entfernt wird. "Abschicken" hat eine ähnliche Funktion wie der "OK"-Knopf. Alle bisher formulierten Änderungen werden in das editierte Objekt übernommen, aber der Editor im Gegensatz zum "Ok" nicht geschlossen. Will man kurzfristig in einem anderen Editor Eingaben machen, die sich auf das Objekt beziehen, später aber im Formular weiterarbeiten, vermeidet man auf diese Weise das erneute Navigieren und Öffnen des Formulars. "Abbrechen" und "Ok" haben den üblichen Effekt.

7.2.2 Verhaltensspezifikation

7.2.2.1 Primitive Prädikate und Aktionen

In Abschnitt 6.2.2.1 wurde ausführlich das Konzept der Aktions-, Prädikats- und Referenzierungsprimitive beschrieben. Für die Spezifikation jeder Aktion einer Aktivität oder eines Ereignisses, jeder einzelnen Vorbedingung bis hin zu den oben erwähnten Aktualisierungsfunktionen bei Zustandsvariablen muß ein Primitiv und seine Argumente ausgewählt werden. Diese Argumente können wiederum aus Referenzierungsprimitiven und deren Argumenten bestehen. Da die verschiedenen Primitivmengen nur beschränkt vorgegeben sein können und zudem genau an dieser Stelle der beste Ansatzpunkt für eine domänen- und anwendungsspezifische Anpassung ist, wurde für die Eingabe und Manipulation von Primitivaufrufe ein besonderes Konzept entwickelt.

Abbildung 7.3 Die Beschreibung der Primitive wird interpretiert und daraus Editoren und Verbalisierung einer Spezifikation des Primitivaufrufs generiert



Die in Abbildung 6.5 skizzierten Strukturen der Beschreibung eines Primitives werden interpretiert und daraus die passenden Editoren ebenso wie die Verbalisierung eines Primitivaufrufs zusammengesetzt. Wie dies geschehen kann, wird in Abbildung 7.3 an Hand des Beispiels des Aktionsprimitivs zum "Verschicken von Nachrichten" angedeutet.

Jedes Fenster zur Spezifikation eines Primitivaufrufs ist in nebeneinander gestellte Blöcke strukturiert. Jeder dieser Blöcke ist zweigeteilt: Oben wird der aktuelle Wert des Blocks verbalisiert, darunter wird der Eintrag genauer beschrieben. Hinter letzterem verbirgt sich ein Pop-up mit dem alternative Werte aus- oder spezielle Editoren angewählt werden können.

Über den Block ganz links, den "Funktionsblock", wird das Primitiv selbst selektiert. In dem dabei verwendeten Pop-up-Menü tauchen nur Primitive derselben Kategorie auf. Bei Aktions- und Prädikatsprimitiven nur Primitive mit der selben Funktion und bei Referenzierungsprimitiven wird die Menge abhängig vom Wertebereich des zu spezifizierenden Teilelements weiter beschränkt.

Auf der Basis der Beschreibung des im Funktionsblock ausgewählten Primitives werden rechts davon die "Argumentblöcke" erzeugt und positioniert. Die Beschreibung der "Argumentkombination" spezifiziert die Kombinationen von möglichen Argumentblöcken, zwischen denen der Benutzer bei der Manipulation des Primitivaufrufs auswählen kann. Für jedes einzelne Argument muß dabei der passende Wertebereich beschrieben werden. Dies geschieht in Form einer Liste aus alternativen Möglichkeiten. Diese bestehen entweder aus absoluten Werten oder aus Angaben zum erwarteten Wert. Abbildung 7.4 verdeutlicht, wie aus absoluten Werten Einträge eines Pop-up-Menüs generiert werden oder wie durch bestimmte Token, beispielsweise `:klasse` ein hierarchisches Pop-up-Menü erzeugt wird, in dem alle domänenspezifischen Klassennamen als mögliche Einträge auftauchen. Der Text in der Wertebereichsbeschreibung wird dabei zur Beschriftung des Auswahl-Pop-up-Menüs verwendet.

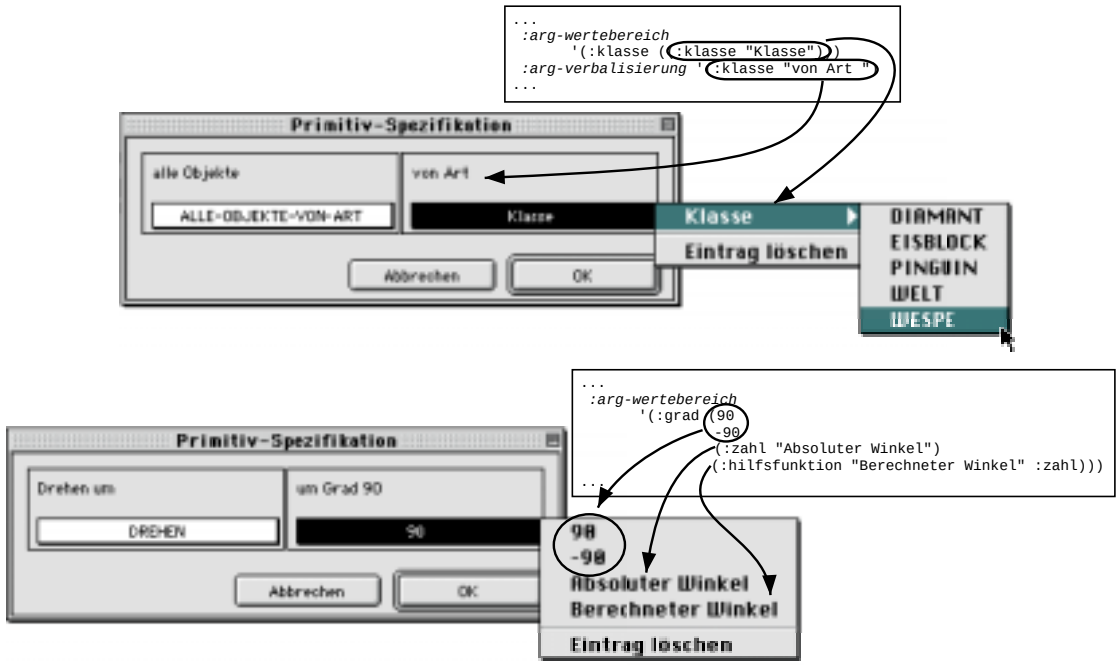
Eine besondere Rolle spielen die Token `:hilfsprimitiv`, `:sensorprimitiv` und `:aktionsprimitiv`. Dabei werden zur Spezifikation dieses Arguments wieder entsprechende Editoren geöffnet. Bei Hilfs- oder Referenzierungsprimitiven stehen dabei beim Funktionsblock nicht alle Referenzierungsprimitive zur Auswahl, sondern nur die, die für einen passenden Rückgabewert sorgen. So sind durch die Wertebereichsbeschreibung `(:hilfsprimitiv ... :objekt)` nur diejenigen Primitive zulässig, die ein Objekt zurückliefern können. Andere Token wie `:zahl` beschreiben, daß an dieser Stelle nur ein numerischer Wert zulässig ist. Beim Betätigen der entsprechenden Auswahl wird ein Editor geöffnet, in dem nur eine Zahl eingegeben werden kann.

Das Beschreibungsobjekt für ein Primitiv und seine möglichen Argumente werden zudem benutzt, um die Verbalisierung einer Primitiv-Konfigurierung zusammenzusetzen. Dazu wird rekursiv für den gesamten Aufruf jeweils die Verbalisierung des Primitives und seiner Argumente mit deren jeweiligen Wert-Verbalisierungen kombiniert. Durch ein geschicktes Wählen dieser Schablonen kann man fast natürlichsprachliche Aufruf-Verbalisierungen erzeugen. Eine Perspektive für eine Vereinfachung der Modellieroberfläche wäre es, auf dieser Basis natürlichsprachliche Eingaben zu parsen und daraus formale Primitivaufrufe zu generieren.

Über ein spezielles Verwaltungsfenster sind alle Beschreibungsobjekte für Primitive zugänglich. Dieses wird in Abbildung 7.5 dargestellt. Über den Dialog, mit dem die einzelnen Objekte editierbar sind (wird hier nicht abgebildet), können auch vom Modellierer bzw. dem Domänendesigner alle vorgegebenen Primitive adaptiert werden.

Für die in Abschnitt 6.2.2.1 beschriebene Abstraktion von Primitiven wurden entspre-

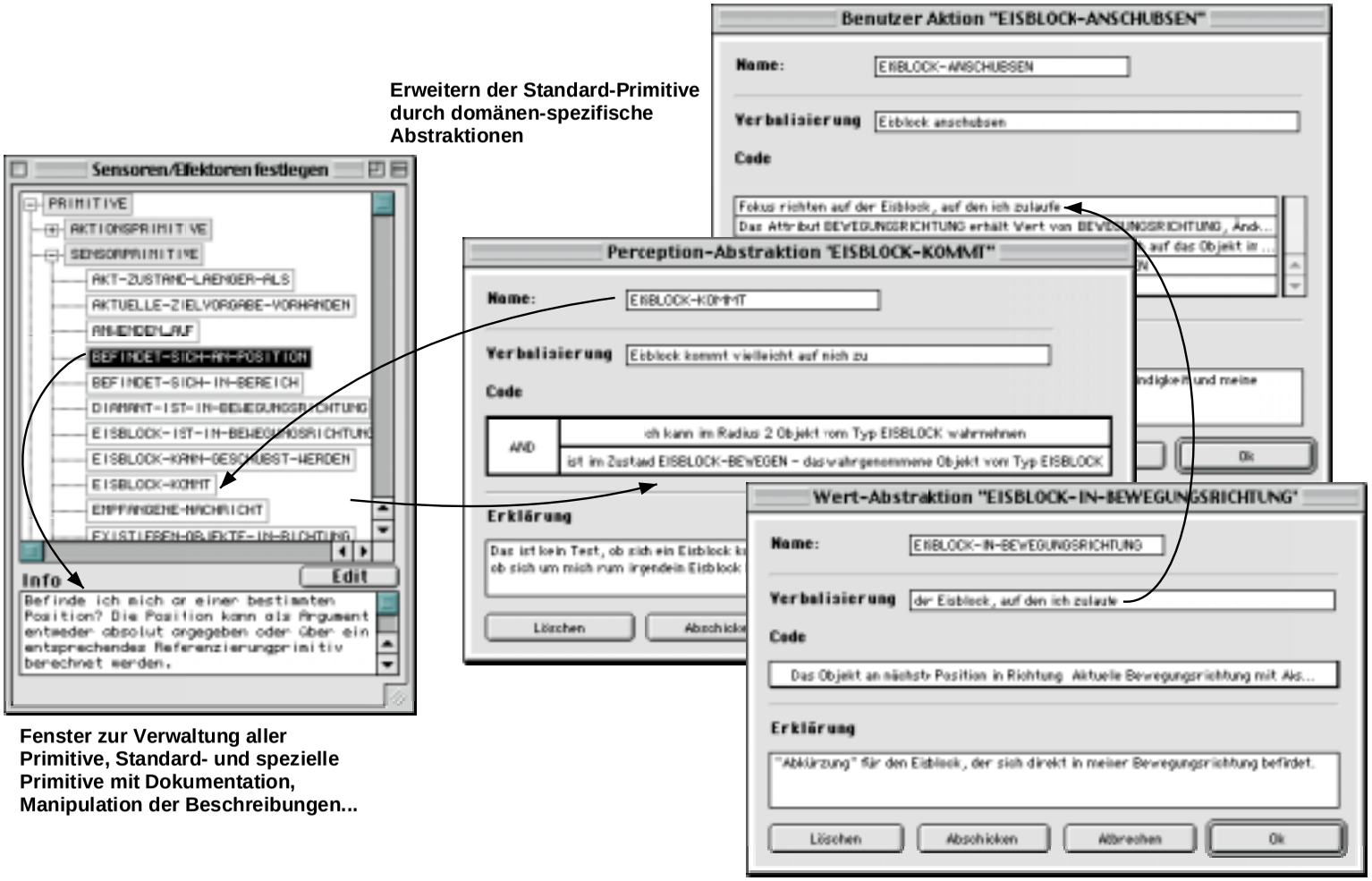
Abbildung 7.4 Für die Eingabe der Argumente einer Spezifikation eines Primitivaufrufs wird die Wertebereichsbeschreibung interpretiert.



chende Editoren bereitgestellt. In speziellen Formularen für “Benutzerprimitive” können vorgegebene Primitive oder bereits zuvor entworfene Abstraktionen zusammengefasst und benannt werden. Man bemerke bei diesen in Abbildung 7.5 dargestellten Formularen, daß auch sie wie die Zustandsvariablenformulare dreigeteilt sind: Name, spezielle Eingaben und Erklärung mit üblicher Buttonleiste. Bei diesen Formularen wurden Eingabefelder wiederverwendet, die auch an anderen Stellen für die Spezifikation des Primitivaufrufs benutzt werden. So ist das Sequenz-Objekt bei der Eingabe von abstrakten Aktionsprimitiven das selbe Graphikobjekt wie im Aktivitätsformular, die abstrahierende Prädikatsspezifikation wird auch in den Regelformularen für die Vorbedingungen verwendet. Referenzierungsprimitive werden an sich ausgehend von Editoren für andere Primitivarten verwendet. An dieser Stelle benötigt man allerdings nur ein Eingabefeld, das ein übliches Spezifikationsfenster aufruft und den Eintrag verbalisiert.

Eine Kombination von Referenzierungsprimitiven unabhängig von konkreten Aktions- oder Prädikatsprimitiven ist demnach nur durch Schachtelung von Aufrufen sinnvoll und nicht durch Aneinanderreihung oder logische Verknüpfung. Deshalb wurde an dieser Stelle ein Eingabefeld wiederverwendet, das in den Aktivitätsformularen für die Spezifikation einer einzelnen Aktion, die beim Start, bzw. bei der Terminierung der Aktivität ausgeführt werden soll, Verwendung findet, da dieses genau das Geforderte leistet. Die Folge davon ist, daß der Benutzer für die Konfiguration spezieller Primitive keine neuen graphischen Eingabekonzepte lernen muß, sondern die Bedienung aus anderen Editoren bekannt ist.

Abbildung 7.5 Über ein Verwaltungsfenster können Standard- und domänenabhängige Primitive manipuliert werden. Der Benutzer kann zudem über Editoren eigene Primitive aus den vorhandenen abstrahieren.



7.2.2.2 Regeln

Bei der Spezifikation eines Verhaltensmodells müssen an vielen Stellen Regeln formuliert werden. Auch hier wurde ein Formular mit einem einheitlichen Design für die verschiedenen Regelformen zur Verfügung gestellt. Abbildung 7.6 zeigt eine Übersicht der Formulare für Regeln mit unterschiedlich ausgewiesenen Zielsetzungen.

Man findet wieder die übliche Dreiteilung mit der allgemeinen Beschreibung der Zielsetzung. Regeln besitzen in SESAM keinen expliziten Namen, sondern werden nur im Zusammenhang mit anderen Modellobjekten angegeben. Deshalb steht hier anstelle des Namens ein nicht editierbarer Text, der das konkrete Ziel der Regel verbalisiert. Im Mittelteil werden Vorbedingung und Nachbedingung spezifiziert. Das Layout des Eingabe-Items für die Regelkondition ist intuitiv verständlich. Verknüpfungen umklammern auch graphisch die Prädikatsprimitivaufrufe, die sie betreffen. Der Benutzer muss explizit die zu verknüpfenden Blöcke auswählen und kann erst danach die logische Verknüpfung angeben. Die einzelnen Vorbedingungen werden über die im letzten Abschnitt beschriebenen Spezifikationsfenster für Prädikatsprimitive angegeben.

Für die Aktion wird ein erweitertes Pop-up-Menü zur Verfügung gestellt, das abhängig vom Regelkontext eine beschränkte Auswahl an Aktivitäten oder Ereignissen bereithält. Die Teilmenge der dabei möglichen Aktionsobjekte ergibt sich aus dem Objekt, an das die Regel vom Modellierer geknüpft wurde, bzw. von wo aus der Modellierer die Regel erzeugt hat. So kann bei einer Aktivitätsauswahlregel keine Aktivität eingegeben werden, die nicht für die zugehörige Rolle definiert ist. Als Teil der Aktion kann man die Wahrscheinlichkeit angeben, mit der die Regel grundsätzlich feuern kann. Auch hier werden für die Eingabe einer berechneten Wahrscheinlichkeit die Primitivaufruf-Spezifikationsfenster eingesetzt. Dabei ist die Menge der möglichen Primitive auf Standardoperationen und Referenzierungsprimitive beschränkt, deren Rückgabe als eine Zahl beschrieben wird.

Im untersten Teil der Regelformulare kann wiederum eine Erklärung für diese Regel angegeben werden. Die Buttonleiste verfügt bei Regelformularen über keinen “Abschicken”-Button, da dieser bei Regelformulare als Modaldialogen keinen Sinn macht. Regelformulare sind deswegen Modaldialoge, um Fehler wegen Unaufmerksamkeit des Modellierers zu vermeiden, da das Editieren von Regeln meist nur im Kontext anderer Modellobjekte sinnvoll ist.

7.2.2.3 Aktivitäten

Der zentrale Punkt bei der Spezifikation des Agentenverhaltens ist die Modellierung der Aktivitäten eines Agenten. Gerade an dieser Stelle ist es wichtig, dem Benutzer verschiedene Übersichten, aber auch ausreichend detaillierte Bearbeitungsebenen zur Verfügung zu stellen.

Dabei sind grundsätzlich folgende Aspekte zu beachten: Zusammenhängende Aktivitäten sollten auf einem Blick als solche erkennbar sein. Mögliche Folgen und Relationen zwischen Aktivitäten sollten als solche visualisiert werden. Abbildung 7.7 zeigt einen Überblick über die Editoren, die für die Spezifikation von Aktivitäten und ihrer Zusammenhänge relevant sind.

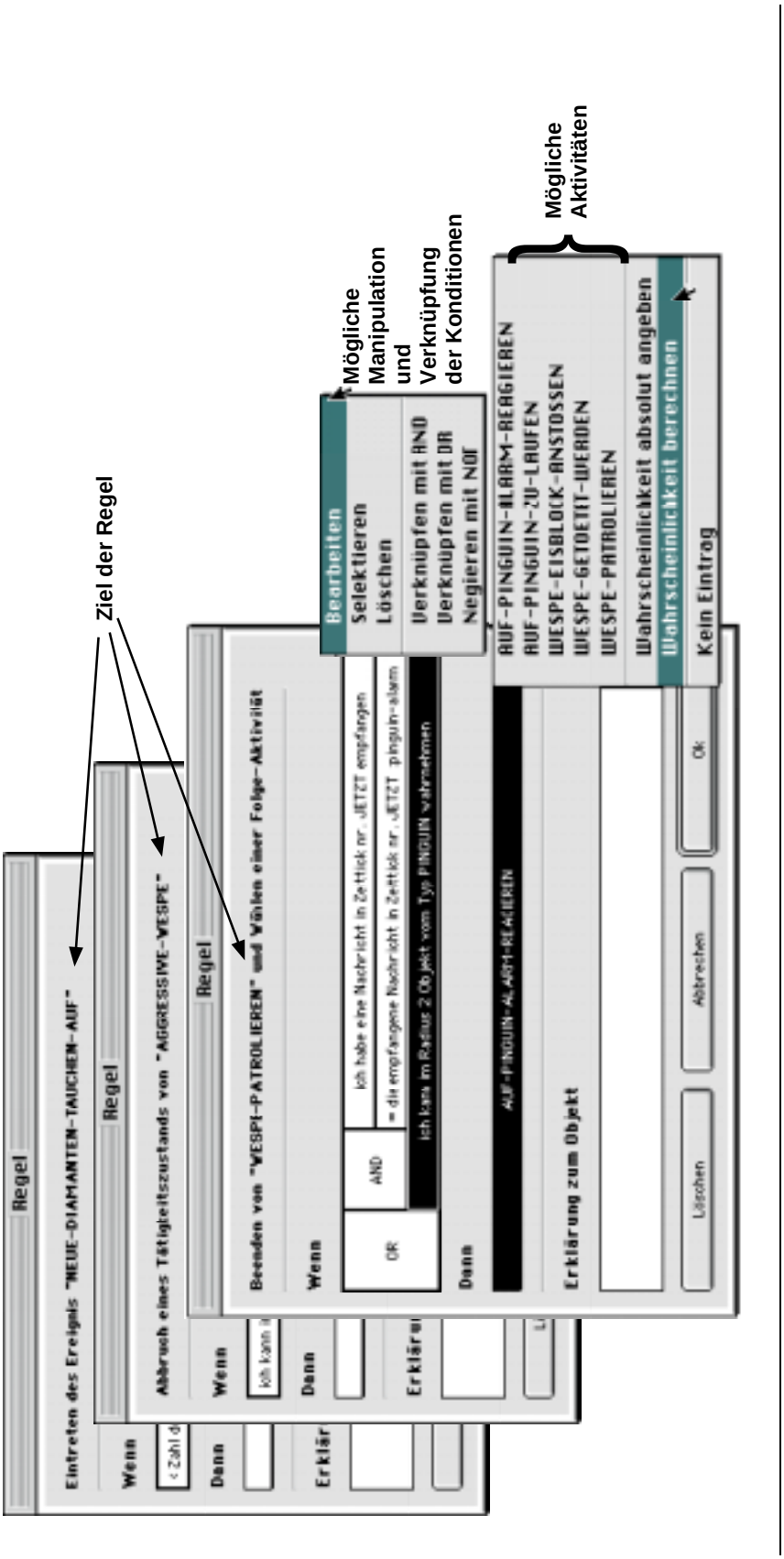
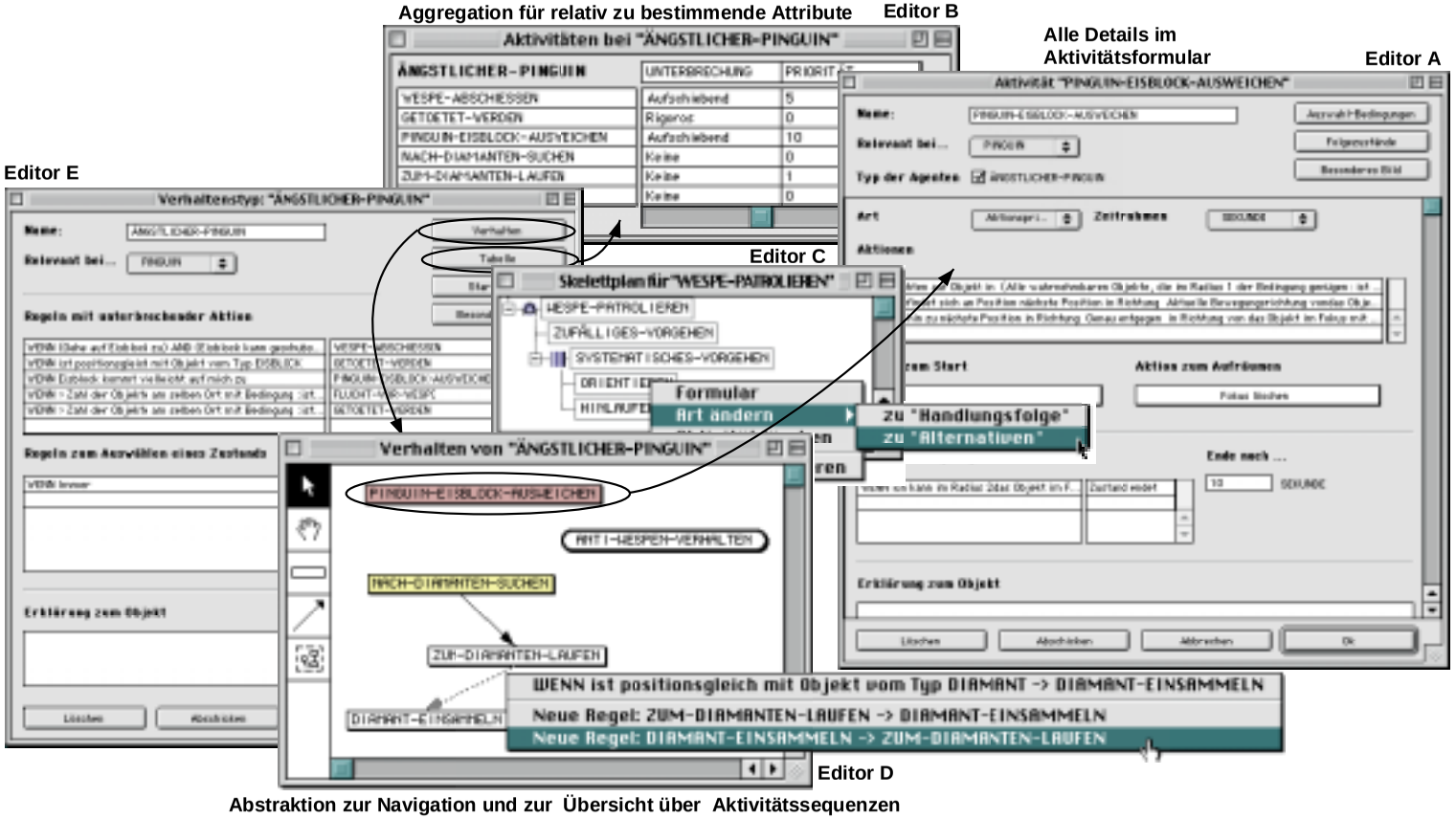


Abbildung 7.6 Regeleditoren unabhängig von der konkreten Zielsetzung der Regel.

Abbildung 7.7 Verhaltenspezifikation: Verschiedene Sichten auf Information in den Verhaltenstypen, Relationen zwischen Aktivitäten bis hin zur vollständigen Spezifikation einer Aktivität im entsprechenden Formular.



Aktivitätsformulare sind die wohl umfangreichsten Editoren in SESAM. Fast alle Attribute eines Aktivitätsobjekts können darin festgelegt werden. Auch dieses Formular, wie in Abbildung 7.7, Editor A dargestellt, folgt mit einem sehr umfangreichen Mittelteil, der bei kleinen Bildschirmen zum Einsetzen eines scroll-baren Bereichs führt, der Dreiteilung. Der oberste Teil dient zu Angabe kategorischer Information über die Agentenklasse und der passenden Rolle, der Mittelteil ist für die Information zur Funktion des Objekts und zu unterst findet sich Erklärungstextfeld und Buttonleiste. In diesem Formular ist der Mittelteil weiter gegliedert: Den zentralen Punkt im Mittelteil nehmen die Aktionen ein, die der Agent ausführt, wenn er sich in diesem Aktivitätszustand befindet, in diesen tritt, bzw. ihn verlässt. Der zweite Teil beinhaltet Wissen über die Terminierung des Aktivitätszustands mit einer Übersicht über die bei dieser Aktivität indizierten Regeln. Über diese Tabelle sind die im letzten Abschnitt beschriebene Regelformulare erreichbar. Eine Besonderheit ist der Eintrag “Ende nach ... ”. Damit wird eine Terminierungsregel, die als alleinige Vorbedingung die Zeit, die der Agent bereits in dieser Aktivität verbringt, besitzt generiert. Dadurch formuliert der Modellierer die Information, dass die Aktivität maximal x Takte dauert. Dementsprechend wird diese Regel besonders eingeben – jede kontextabhängige Terminierung muss als Regel formuliert werden.

Neben den kategorischen Informationen im obersten Teil des Formulars befinden sich zusätzliche Buttons, über die weitere Informationen zugänglich sind und deren Anzeige direkt im Formular nicht sinnvoll ist. Neben einem Editor zur Auswahl einer besonderen Darstellung des Agenten während des Aktivitätszustands (siehe Abbildung 7.2, Editor B), wird eine Auflistung von Aktivitäten, die über Terminierungsregeln für diese Aktivität ausgewählt werden können. Desweiteren sind in einer Regelliste alle Regeln, die für eine Auswahl der gerade editierten Aktivität verantwortlich sein können, in einer weiteren tabellarischen Übersicht zugänglich.

Aktivitätstabelle: Weitere Details einer Aktivität, wie ihre Priorität oder andere Attribute, die am besten relativ zu anderen Aktivitäten spezifiziert werden, konnten in früheren Versionen von SESAM auch im Aktivitätsformular eingegeben werden. Allerdings widerspricht dies einer Forderung bei visuellen Programmiersystemen, daß alle für die Eingabe eines bestimmten Werts relevante Information dabei sichtbar sein sollen.

Die Vorgehensweise, einzelne Formulare öffnen zu müssen, um beispielsweise die Prioritätswerte anderer Aktivitäten nachzuschauen, ist etwas umständlich. Deswegen wurde eine Tabelle erstellt, in der genau diese Werte für Aktivitäten untereinander aufgelistet werden und auch manipuliert werden können.

Eine derartige Aktivitätstabelle enthält per default alle Aktivitäten, die einer Rolle zugeordnet wurden (siehe dazu Abbildung 7.7, Editor B). So kann beim Editieren derartiger Werte ohne Aufwand auf die entsprechenden Werte anderer Aktivitätsobjekte Bezug genommen werden.

Skelettplan-Editor: Es ist zwar möglich, über die Einträge der Art einer Aktivität (“Primitivaktivität”, “Handlungfolge” oder “Alternativen”) und eine entsprechende Formulierung in der Aktionentabelle in den einzelnen Aktivitätsformularen eine Hierarchie abstrakter und primitiver Aktivitätsobjekte aufzubauen (siehe Abschnitt 5.3.2.3). Allerdings verliert man so schnell den Überblick über die beabsichtigten und bereits formulierten Zusammenhänge. Bei einem Skelettplan-ähnlichen Bild für die Organisation der Aktivitäten bietet eine Ta-

belle nur bedingt eine passende Übersicht. So ist es geschickter, die baumartigen Zusammenhänge über einen Editor in Form einer Hierarchie darzustellen. Genau das bietet der "Skelettplan-Editor", der zu einer bestimmten Aktivität geöffnet werden kann. Diese bildet die Wurzel der Hierarchie. Dieser Editor wird in Abbildung 7.7 als Editor C in der Mitte dargestellt. Als Basiseditortyp wurde eine "Klapphierarchie" benutzt. Dabei können Teilbäume "auf- und zugeklappt" werden, um auch in umfangreichen Hierarchien die Übersicht behalten zu können. Die verschiedenen Aktivitätsarten, wie "Handlungsfolgen" oder "Alternativen" werden durch kleine Icons vor den Knoten markiert. Auf diese Weise wird angedeutet, wie der darunter hängende Teilbaum behandelt wird. Über ein Pop-up an den Knoten kann man die Details der Aktivität im zugehörigen Formular editieren bzw. die Struktur des UND-ODER-Baum manipulieren.

Aktivitätsautomat: Auch für diese Metapher wurde ein spezieller Editor bereitgestellt. Wie im Skelettplan-Editor entspricht ein Knoten einem Aktivitätsobjekt. Die Kanten zwischen den Knoten repräsentieren Regeln, die von einer Aktivität zur anderen führen, in dem sie die Ursprungsaktivität terminieren und die Folgeaktivität auswählen (siehe Abbildung 7.7, Editor D). Dieses Fenster dient aber nicht nur zur Visualisierung von Zusammenhängen, sondern kann als Navigationsgrundlage und als Editor verwendet werden. So können über die Werkzeugpalette neue Aktivitäten und Verbindungen zwischen Aktivitätenknoten angelegt werden. Durch die Auswahl einer Kante werden die dahinter verborgenen Regeln angedeutet und können über eine Auswahl im entsprechenden Regelformular verändert werden, bzw. neue Regeln erzeugt werden. Über die Knoten ist auch ein Pop-up-Menü zugänglich, mit dem das entsprechende Aktivitätsformular, bzw. ein Skelettplan-Editor mit der Aktivität als Wurzel geöffnet werden kann. Verschiedene Knotenfarben deuten an, ob nicht-indizierte Regeln zu dieser Aktivität existieren. Die Position eines Knoten wird beim Anlegen oder Verschieben gespeichert. So kann der Graph mit dem Layout, das der Benutzer bestimmt hat, wieder geöffnet werden. Auf diese Weise bleibt der Knoten in dem räumlichen Kontext sichtbar, in dem er vom Benutzer angelegt wurde.

Bei einer großen Zahl von Aktivitäten und sie verbindenden Regeln können auch in einer derartigen Übersichtsdarstellung die Zusammenhänge nicht mehr richtig transparent visualisiert werden. Der unterste Eintrag in der Werkzeugpalette bietet dabei eine Strukturierungshilfe, um auch in einem großen Graph die Übersicht behalten zu können. Über diese Werkzeuge können mehrere Aktivitäten ausgewählt, "gruppiert" und als Gruppe benannt werden. Dabei werden diese zu einem neuen Knoten – mit runden Ecken – zusammengefasst. Der so entstandene Teilgraph kann durch Klick auf diesen Knoten in weiteren Verhaltensgraphen auf die selbe Weise wie im ursprünglichen Graph editiert, oder aber wieder aufgelöst werden. Auf diese Weise kann eine ganze Hierarchie von Aktivitätsautomaten erstellt werden und so beliebig abstrahierte Sichten auf die gesamte Verhaltensbeschreibung eines Typs erzeugt werden. Diese Struktur dient zur übersichtlichen Darstellung auf der Modellieroberfläche und wird vom Interpreter ignoriert.

Das Verhaltenskategorieformular ergänzt die Aktivitätsübersichten im Skelettplan-Editor und dem Aktivitätsautomaten um eine nicht direkt aktivitätsbezogene Form der Auflistung aller nicht-indizierten Regeln zur Auswahl von Aktivitäten. Dabei wird eine zusätzliche Unterscheidung zwischen den nicht-indizierten (unterbrechenden) Regeln, die in jedem Aktualisierungszyklus abgetestet werden und Regeln zur Bestimmung einer Default-Aktivität ge-

macht. An sich können die auch über einen Button im Aktivitätsformular angezeigt werden, allerdings hat man auf diese Weise eine Übersicht über die verschiedenen Regelkonditionen, die gleichzeitig abgetestet werden. Grundsätzlich sind Regeln aus einem jeden derartigen Graphikobjekt für Regellisten in jede derartige über "drag-and-drop" kopierbar: Also zwischen Listen in Rollen-Formularen, aber auch in Listen zwischen Aktivitätsformularen. Die Modellieroberfläche sorgt automatisch für die passende Verzeigerung in die jeweiligen Modellobjekte.

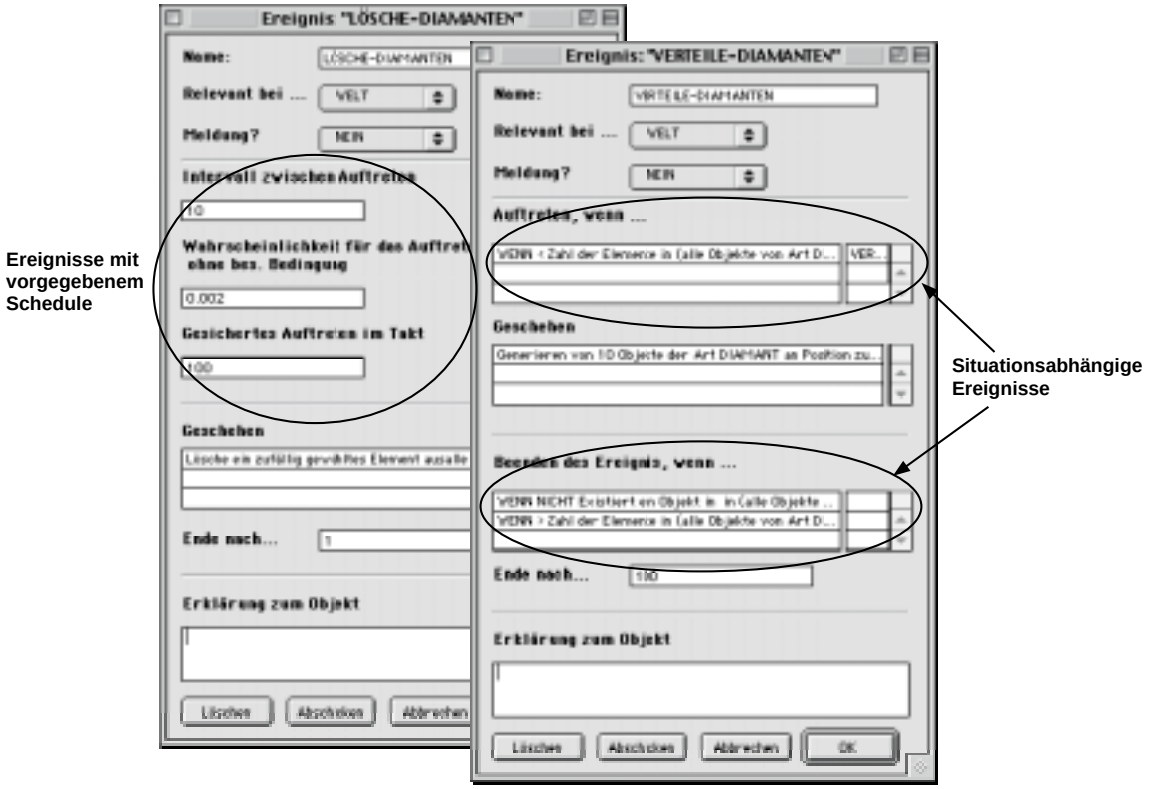
Auch dieses Formular, das in Abbildung 7.7 als Editor E dargestellt wird, ist dreigeteilt. Oben findet man die eher kategorische Information über den Namen des Verhaltenstyps und seine Zuordnung zu einer Agentenklasse. Im Mittelteil können die oben beschriebenen Regeln erzeugt, gelöscht und über Regelformulare bearbeitet werden. Im unteren Teil ist die übliche Möglichkeit, erklärenden Text einzugeben, und die standardmäßige Buttonleiste. Über die Knöpfe oben rechts kann man den zu diesem Typ gehörenden Aktivitätsautomat und die Aktivitätstabelle öffnen. Die in der Abbildung verdeckten Buttons führen zu einer für diese Rolle spezifischen Default-Zustandsbeschreibung, einem Fenster wie in Abbildung 7.2, Editor C und einem kleinen Darstellungsfenster (7.2, Editor B), in dem die Visualisierung eines Agenten mit dieser Rolle gesteuert wird. Auf diese Weise ist über dieses Formular jede Information zur Verhaltenskategorie zugänglich. Auch Verhaltenstypformulare sind nicht nur über die Menüzeile, sondern auch über die Systemklassen-Hierarchie (Abbildung 7.2, Editor A) zugänglich.

Die Navigation durch diese vielen Editoren, die jeweils verschiedene Sichten auf Teilaspekte des Verhaltensmodells bieten, ist nur auf den ersten Blick schwierig. Obwohl alle diese Fenster direkt über die Menüzeile bzw. einem weiteren Auswahlschritt, erreichbar sind, bieten viele Editoren "Abkürzungen". So kann über den "Verhalten"-Knopf im Verhaltenstyp-Formular die Darstellung der Aktivitätszusammenhänge im Graph geöffnet werden. Gerade letzterer eignet sich gut zur Navigation. Für jeden Knoten ist sowohl das zugehörige Aktivitätsformular, als auch der Skelettplan–Editor erreichbar. Über die Buttons im oberen Teil des Aktivitätsformulars sind die Formulare sowohl der Aktivitäten erreichbar, die über Regeln ausgehend vom vorliegenden ausgewählt werden können, als auch eine (nicht in der Abbildung dargestellte) strukturierte Liste von Regeln, die für die Auswahl der gerade editierten Aktivität verantwortlich sein können. Auf diese Weise ist es bei einem geschickten Vorgehen nur selten notwendig, über die Menüleiste oder in sortierten Listen nach einem konkreten Modellobjekt zu suchen, um es in einem Editor bearbeiten zu können. Wichtig ist dabei auch, dass zu dem gerade editierten Modellobjekt alle relevanten Informationen schnell eingesehen werden kann.

7.2.2.4 Ereignisse

Wie in Abschnitt 6.2.2.3 beschrieben, können als weiterer Aspekt des Umweltverhaltens zusätzlich zu dynamischen Zustandsvariablen Ereignisse spezifiziert werden, mit denen die Konfiguration der Umwelt verändert werden kann. Die Zustandsvariablen können dabei wie zu jeder Agenten- und Ressource-Klasse auch für anwendungsspezifische Bezugssystem-Klassen angegeben werden (siehe Abschnitt 7.2.1). Für die Eingabe und Bearbeitung von Ereignissen stehen spezielle Formulare zur Verfügung. Diese sind wie jedes Objektformular über die Menüleiste erreichbar, aber auch über das Pop-up-Menü beim jeweiligen Bezugssystemklassen-Knoten in der Systemklassenhierarchie (Abbildung 7.2, A).

Abbildung 7.8 Verschiedene Ereignisformen sind in ähnlichen Formularen spezifizierbar



Die Ereignisformulare sind wie alle anderen Formulare dreigeteilt. Im oberen Teil findet sich zusätzlich zu den üblichen Eingabefeldern für den Namen und die zugehörige Bezugsystemklasse eine Auswahl, ob das Auftreten des Ereignisses während der Simulation an den Benutzer gemeldet werden soll. Dies ist eine der wenigen Stellen, an denen Information zur Experimentiersteuerung in einem Bestandteil der Modellieroberfläche editiert werden kann bzw. muß. Im Mittelteil kann entweder eine regelbasierte Steuerung oder eine Zeitsteuerung spezifiziert werden. Diese Zeitsteuerung ist wie die maximale Zeitdauer nur eine "Abkürzung" für die Formulierung nur über zeitabhängige Aspekte gestuerte Ereignisse: Es werden Regeln erzeugt, die alle x Takte, mit einer bestimmten Wahrscheinlichkeit oder in einem absolut gegebenen Zyklus das Ereignis auslösen.

7.3 Die Experimentierumgebung

Eine Experimentierumgebung unterstützt einen Benutzer dabei, Simulationsläufe zu konfigurieren, durchzuführen und dabei Beobachtungen anzustellen. Sie sollte zudem Mittel bereitstellen, um diese Experimente auswerten zu können. Letztere werden allerdings nicht in diesem Abschnitt, sondern gesondert in 7.4 behandelt.

Neben Möglichkeiten, Startsituationen und Einstellungen für Experimente zu speichern, zu laden und zwischen verschiedenen Situationen hin und her zu schalten, besteht die Experimentierumgebung, die von SESAM zur Verfügung gestellt wird, aus drei Teilbereichen:

Zu einem Satz von Editoren, mit denen passende Ausgangskonfigurationen für Simulationsläufe erstellt werden kann, kommen verschiedene, auch zum Teil durch graphischer Editoren adaptierbare Fenster, in denen verschiedene Sichten auf den Fortschritt der aktuell durchgeführten Simulation möglich sind. Darüber hinaus gibt es mehrere Werkzeuge, mit denen in eine laufende Simulation eingegriffen werden kann. Diese drei Teilbereiche der Experimentierumgebung werden im folgenden einzeln behandelt.

7.3.1 Startkonfiguration

Die Editoren für die Startkonfiguration besitzen einen relativ engen Zusammenhang zur Modellieroberfläche, denn eigentlich wird hier dabei die Konfiguration des Modells erstellt. Abbildung 7.9 zeigt einen Überblick über dabei verwendbare Editoren.

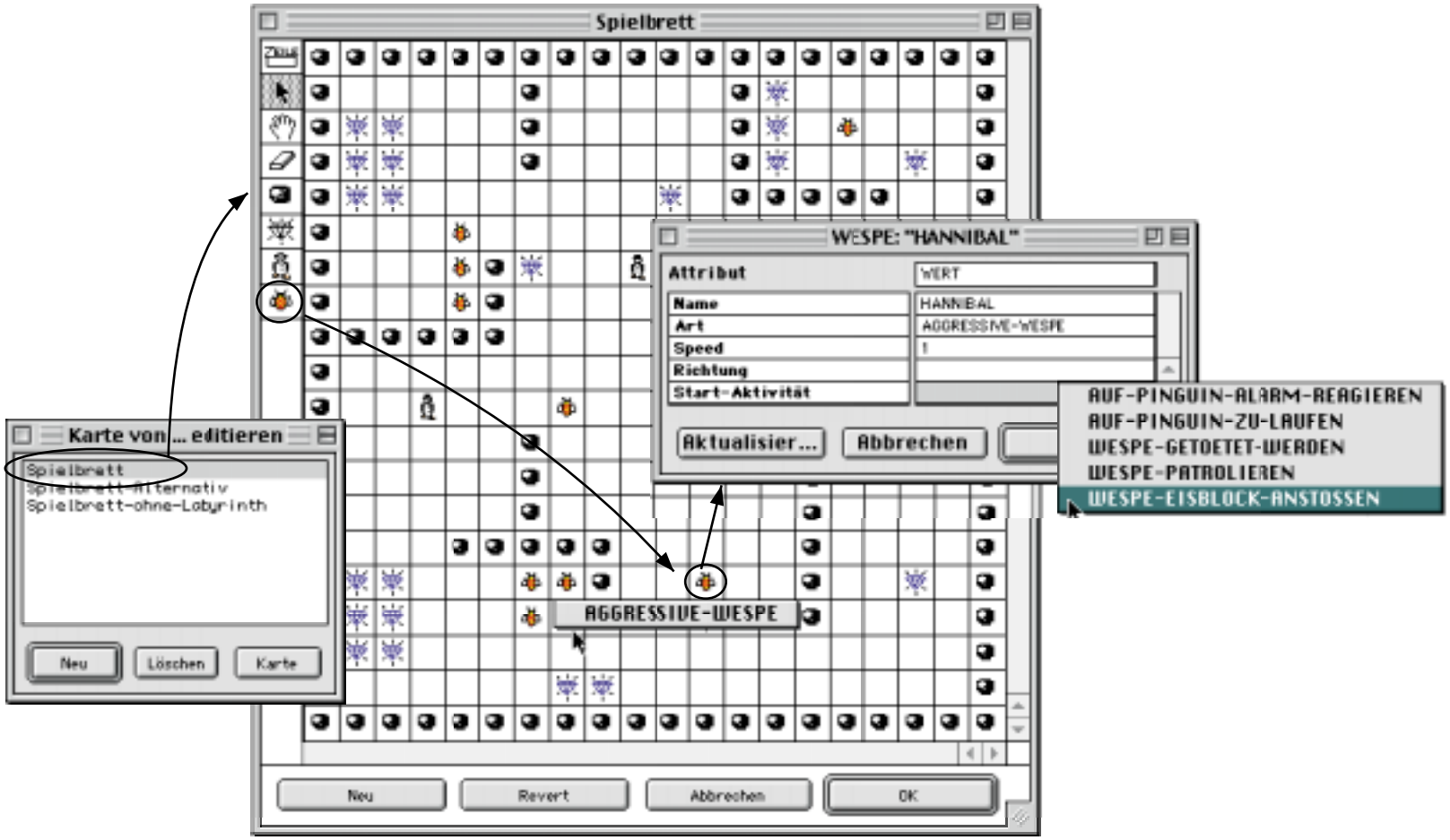
Ausgangspunkt für das Editieren einer Modellkonfiguration ist ein kleines Verwaltungsfenster, mit dem neue Bezugssysteme erzeugt, gelöscht bzw. über ein Formular ihre Größe und bestimmte vordefinierte Muster an Ressourcen- und Agentenverteilungen konfiguriert werden können. Ausgehend vom Verwaltungsfenster kann man auch Editoren zur direkten Manipulation von Karteninhalten öffnen. Ähnlich wie bei den Aktivitätsautomaten befindet sich dabei auch links eine Palette, durch die die Aktion, die durch Auswahl der Karte ausgelöst werden soll, eingestellt werden kann. Neben dem Löschen und Verschieben von Objekten auf der Karte kann man auch neue Elemente anlegen. Diese Paletteneinträge und die sich dahinter versteckenden Aktionen müssen im Beschreibungsobjekt der zugehörigen Bezugsystemklasse spezifiziert werden. Dazu gibt man an, Objekte welcher Klasse mit welcher Zustandsbeschreibung erzeugt und an der ausgewählten Stelle der Karte positioniert werden soll. Als Paletteneintrag kann man dazu eine beliebige PICT-Ressource angeben. In Abbildung 7.9 ist ein Karteneditor für diskrete Bezugssysteme abgebildet. Ein analoges Spezifikationsfenster existiert für kontinuierliche Karten.

Ebenso wie dieser Teil der Palette des Karteneditors können die Fenster zur Manipulation der Startwerte individueller Objekte im Beschreibungsobjekt der entsprechenden Klasse konfiguriert werden. Dazu können die Zeilen wie folgt angegeben werden. Der Modellierer erstellt dazu eine sortierte Liste aus vorgegebenen Token und ordnet jedem Token einen Text zu, mit dem die entsprechende Zeile in der Tabelle dargestellt wird. Aus dem Token ergibt sich eindeutig die Art und Bedeutung des Eintrags in die Tabelle.

Auf diese Weise können von einem Modellierer die Editoren für die Startsituation entsprechend einem bestimmten Multiagentenmodell so angepaßt werden, daß Experimentatoren ohne detaillierte Implementierungskenntnisse Konfigurationen einfach durch Auswahl zusammenstellen können.

Nachdem über einen Einstellungsdialog der Simulationslauf konfiguriert wurde, kann das Experiment gestartet werden. Diese Einstellungsdialoge dienen dazu, allgemeine Kontroll-Parameter, wie die maximale Dauer des Experiments in Takten oder Angaben, in welchen Abständen welche Daten gespeichert werden sollen, festzulegen. Derartige Einstellungen können zusammen mit Situationen und Verhaltensbeschreibungen, aber auch separat gespeichert und wieder geladen werden.

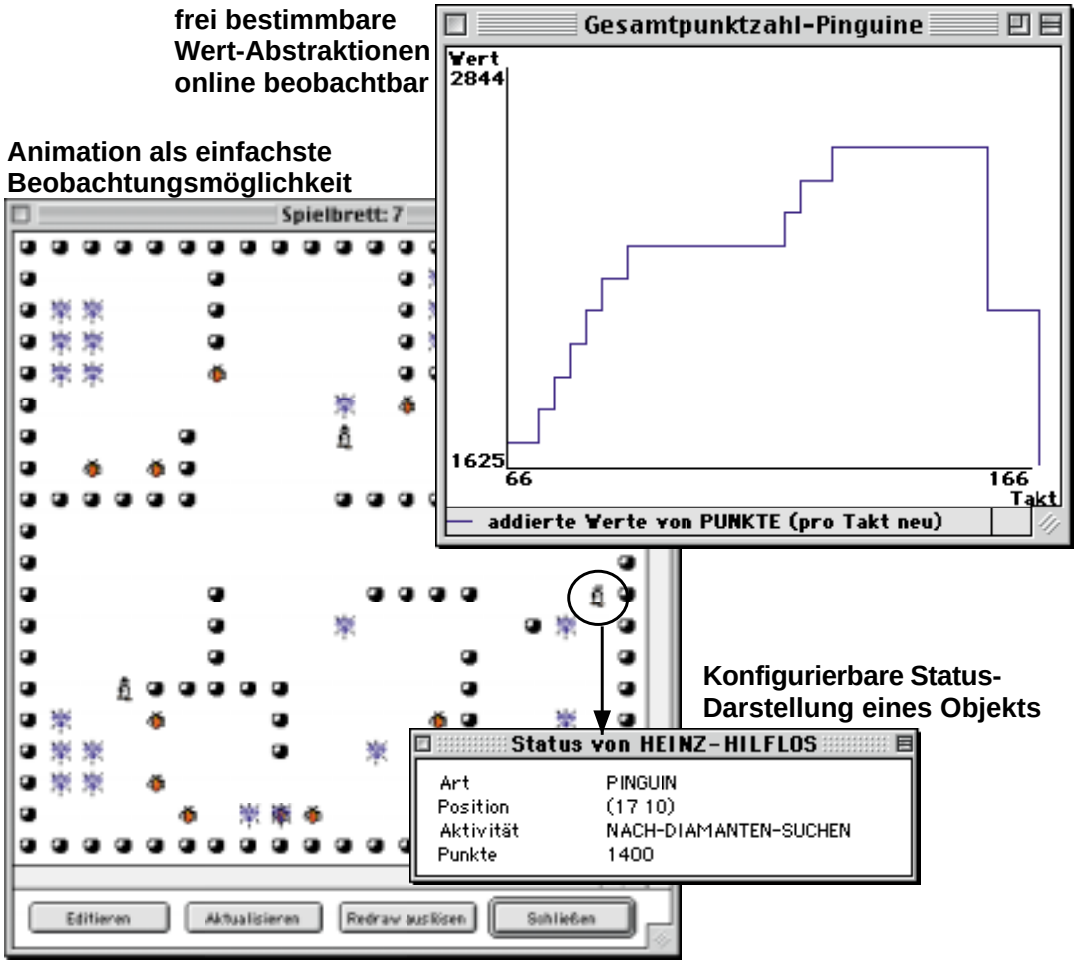
Abbildung 7.9 Umgebung zur Konfiguration der Startsituation eines Modells: Die Verwaltung mehrerer Konfigurationen ist gleichzeitig möglich. Auswahl zur konkreten Experimentierung ist über ein nicht dargestelltes Formular möglich.



7.3.2 Beobachten und Abstrahieren

Auch für die Beobachtung des Fortschritts eines Simulationsexperiments gibt es verschiedene Möglichkeiten, die jeweils unterschiedliche Schwerpunkte im Grad der Detaillierung der dargestellten Daten betonen: Fenster für die Information über einzelne Objekte, Animation des Gesamtzustands und Datenabstraktion durch "Dämonen". Ein Überblick wird in Abbildung 7.10 gegeben.

Abbildung 7.10 Umgebung zur Beobachtung des Simulationsexperiments mittels Animation, Statusdarstellungen und Online-Datenabstraktion.

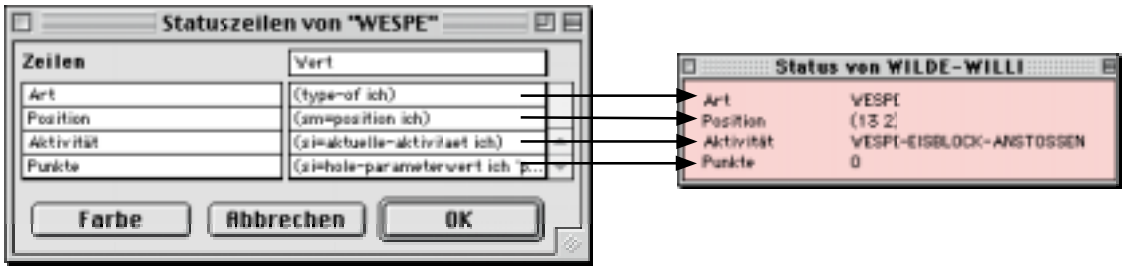


7.3.2.1 Statusinformation für einzelne Systeme

Da nicht von vorne herein klar sein kann, welche Aspekte des internen Zustands eines Systems für den Benutzer in einem Simulationsexperiment interessant sind, wird hier ein deklarativer Mechanismus bereitgestellt. Der Modellierer spezifiziert zu einer anwendungsspezifischen Klasse, welche Information er während der Simulation in einem "Statusfenster" zu einer Instanz dieser Klasse betrachten will. Abbildung 7.11 verdeutlicht dies. Auch hier

kann der Benutzer die Verbalisierung und Sortierung der Zeilen beeinflussen und für jede Klasse eine spezielle Hintergrundfarbe angeben.

Abbildung 7.11 Spezielle Tabelle zum Editieren der Statusdarstellung von Objekten während der Simulation



Auf diese Weise können zu jedem Objekt in der aktuellen Situation genau die Daten, die der Modellierer für notwendig erachtet, dargestellt und ihre Änderungen mitverfolgt werden. Diese “Statusfenster” öffnen sich nur für die vom Benutzer angewählten Objekte.

7.3.2.2 Animation

Eine übersichtlichere Visualisierung der Veränderungen auf einer Karte geschieht durch Animation der Bewegungen der Agenten bzw. der Zustandsänderungen auf der Basis der Darstellungen, die im Verhaltensmodell für Agenten in einem bestimmten Zustand angegeben wurden. Dazu wird ein Fenster bereitgestellt, in dem in einer beliebigen Auflösung ein Ausschnitt der Bezugsystemkarte oder das gesamte Areal visualisiert werden kann. Da das Mitführen der Änderungen des zugrundeliegenden Modell in einer Oberfläche nicht kostenlos ist, kann die Animation abgeschaltet, eine Aktualisierung per Hand oder in größeren Abständen angestoßen werden. Es gibt in vielen Simulationsumgebungen keine Online-Animation des Geschehens, sondern es werden während der Simulationsläufe Trace-Files erzeugt. In diesen werden alle für eine nachträgliche Animation relevanten Informationen aufgezeichnet. Nach der Simulation wird diese Datei an ein spezielles Animationsprogramm weitergeleitet, das daraus derartige Visualisierungen erzeugt. Ein derartiges Vorgehen ist in SESAM ebenso möglich. Eine Animation während der Simulation ist aber gerade in der Implementierungsphase eines Modells wichtig, um so direkt die Umsetzung des aktuell spezifizierten Geschehen beobachten zu können. Das Ausschreiben umfassender Protokolldaten kann außerdem die Simulation verlangsamen.

7.3.2.3 Datenabstraktion während eines Experiments

Als dritte Möglichkeit, das Geschehen während eines Simulationsexperiments zu visualisieren, gibt es automatische Datenabstraktionen. Ebenso wie Animation und Statusinformation einzelner Objekte können sich derartige Abstraktionen während Testläufen als wertvoll erweisen, da man so bequem die Entwicklung wichtiger Daten verfolgen und bei fehlerhafter Dynamik sofort eingreifen kann. Es gibt dazu zwei Formen: Die Visualisierung von abstrahierten Daten und das Testen einer bestimmten Situation mit Anzeige, bzw. automatischem Beenden der Simulation bei deren Auftreten. Beide werden über “Dämonen” realisiert, die das Experiment überwachen. Diese überwachen im Hintergrund das Simulationsgeschehen

und führen in spezifzierten Situationen die ihnen zugewiesenen Aktionen aus. Man könnte sie auch als “Beobachter” bezeichnen.

Die einfachste Form eines Dämons, ist der Melder, der in jedem Simulationstakt testet, ob eine bestimmte vom Experimentator angegebene Situation beobachtbar ist. Wenn dies der Fall ist, wird eine Nachricht auf dem Bildschirm angezeigt und gegebenenfalls das Simulationsexperiment angehalten. Auf diese Weise kann die Überwachung eines Simulationslaufs automatisiert werden.

In Abbildung 7.12 wird ein Formular für einen derartigen Dämonen gezeigt. Dabei wird spezifiziert, daß wenn kein Pinguin mehr existiert, die Simulation beendet und eine Meldung über den Sieg der Wespen angezeigt werden soll. Die Spezifikation dieser bemerkenswerten Situation kann über ein Eingabefeld zum Editieren einer Regelvorbedingung eingegeben werden.

Abbildung 7.12 Formular zur Spezifikation von Dämonen, mit denen Überwachung einer Simulation automatisiert werden kann.

Komplexere Formen von “Dämonen” können bestimmte, vorher konfigurierte Daten des Simulationslaufs aggregieren, abstrahieren und an ein Statistikfenster schicken, das den Verlauf dieser Daten visualisiert. In Abbildung 7.10 wird rechts oben ein derartiges Fenster dargestellt, in dem während der Simulation jeweils für die letzten 100 Takte die Summe aller Punkte, die Pinguin-Agenten durch das Aufsammeln von Diamanten-Ressourcen bisher erreicht haben, dargestellt. Derartige Online-Abstraktionen sind wie die in Abschnitt 7.4 beschriebenen Datenauswertungsmechanismen über sogenannte “Filter” möglich. Auch die Anzeige entspricht den Fenstern zur Funktionsvisualisierung.

7.3.3 Benutzereingriffe

Ausgehend von den Beobachtungen während des Simulationslaufs, könnte es sein, daß ein Benutzer direkt eingreifen will. Ein Experiment kann jederzeit abgebrochen, das Verhaltensmodell und die Modellkonfiguration über die entsprechenden Editoren verändert und mit diesem manipulierten Modell weitersimuliert werden. Eine Situation kann abgespeichert

und zu einem anderen Zeitpunkt wieder geladen und es kann an dieser Stelle weitersimuliert werden. SESAM bietet aber noch weitere Möglichkeiten, in den Verlauf eines Simulationsexperiments einzugreifen: Benutzergesteuerte Ereignisse und Aktivitätsauswahl.

Ereignisse, die wie in Abschnitt 7.2.2.4 beschrieben modelliert werden, können in jedem Takt über die Menüleiste vom Benutzer angestoßen werden. Die ausgewählten Ereignisse werden zu Beginn des nächsten Aktualisierungszyklus ausgeführt. Auf diese Weise können Systeme mit Benutzereingriffen wie Versuchskästen, bei denen ein Mensch jeden Morgen seine Tiere "füttert", als virtuelles Labor nachgebaut werden.

Die zweite Möglichkeit des Benutzereingriffs ist die Steuerung eines oder mehrerer Agenten "per Hand". Das bedeutet, daß der Experimentator für diese Einheiten die Aktivitätsauswahl übernimmt. Ob ein Experimentator diese Möglichkeit überhaupt hat, muß für jede Agentenklasse in der Beschreibung der jeweiligen Klasse angegeben werden. Dann öffnet sich bei der Auswahl des Objekts während der Animation nicht ein gegebenenfalls spezifiziertes Statusinformationsfenster, sondern ein "Kontroll-Tool" mit dem die aktuelle Aktivität des Agenten gesetzt, bzw. wieder zurück in einen automatischen Modus versetzt werden kann. Der Standarddialog für diese Aktivitätsauswahl per Mausklick wird in Abbildung 7.13 dargestellt. In der Beschreibung einer Agentenklasse sind Alternativen für dieses Kontrollfenster angebbar.

Abbildung 7.13 "Kontroll-Tool" zur manuellen Aktivitätsauswahl eines Agenten. Über einen Klick auf den "Info"-Button erhält man die Statusinformation des Agenten. Über die Buttons "Set ... " für "Setzen" und "Aut ... " für "Automatisch" kann die Aktivität gesetzt werden und wieder auf die automatische Aktionsselektion zurückgeschaltet werden. Der konkrete Modus, wie mit der vom Benutzer selektierten Aktivität umgegangen wird, ist wiederum in der Beschreibung der Agentenklasse konfigurierbar.

7.4 Die Auswertungsumgebung

Während eines Simulationsexperimentes können Daten aufgezeichnet und in Protokollen abgespeichert werden. Dabei werden aus Effizienzgründen nicht alle Daten, sondern nur die Änderungen von Werten in die Datei geschrieben, die der Experimentator als aufzeichnenswert festgelegt hat. Nach dem Ende des Simulationslaufs können diese Protokolle in SESAM ausgewertet werden. Eine derartige Aufbereitung des Simulationsexperiments geschieht in zwei Phasen: Zunächst werden die Informationen spezifiziert, die aus den Protokollen gelesen werden sollen, anschließend können diese Funktionen und Werte visualisiert und mittels diverser vorgegebender Operationen weiterverarbeitet werden oder in einem für Standard-Statistikprogrammen lesbarem Format exportiert werden.

7.4.1 Datenfilter

In den Protokollen werden während eines Simulationsexperiments für jeden Takt alle für den Benutzer interessante Parameter- und Attributänderungen ausgeschrieben, die zwischen den letzten Taktwechseln geschehen sind. Um daraus wiederverwendbare Datensätze in Form von Takt – Wert – Paaren zu erhalten, müssen diese ausgelesen werden. Dafür kann der Benutzer angeben, welche Daten für ihn relevant sind. Hinter diesen Ausleseoperationen verstecken sich LISP – Funktionen, die für jeden Eintrag in das Protokoll entscheiden, welche Information aus diesem Eintrag ausgelesen werden sollte. Wird ein Taktwechsel im Protokoll gelesen, wird der Datensatz erweitert. Diese Operationen können ähnlich wie die Referenzierungsprimitive (siehe Abschnitt 7.2.2.1) explizit beschrieben werden. Daran angelehnt werden sie “Filterprimitive” genannt. Die Spezifikation dieser Primitive kann für eine konkrete Domäne angepaßt und im Besonderen von den selben Mechanismen zum Erzeugen von graphischen Editoren interpretiert werden.

Auf diese Weise können für die Spezifikation, welche Datensätze aus den Protokollen ausgelesen werden sollen, die üblichen Editoren zur Konfiguration von Primitivaufrufen wiederverwendet werden. In Abbildung 7.14 wird die Organisation und Konfiguration dieser Ausleseaktionen dargestellt. Nach Betätigen des “Start”-Buttons und Auswählen eines Protokollsatzes, beginnt der Auslesevorgang.

7.4.2 Abstraktionsmechanismen

Die so aus Experimentprotokollen extrahierten Datensätze können in einem Textformat mit üblichen Trennzeichen exportiert und so in Visualisierungs-, Tabellenkalkulations- oder Statistikprogrammen eingelesen und weiterverarbeitet werden. Demnach könnte man an dieser Stelle eigentlich auf eine Bereitstellung von Möglichkeiten verzichten, die durch das Auslesen von Protokollen gewonnen Datensätze weiter auszuwerten. Allerdings kann ein derartiges Vergehen mit Wechseln des Programmsystems – ähnlich wie eine separate Animation – etwas umständlich sein. Beispielsweise können während der Test- und Validierungsphase des Modells viele kleinere Experimente stattfinden, bei deren Auswertung viele Datensätze schnell und ohne großen Aufwand betrachtet werden müssen. Für derartige Auswertungen, für die noch keine großen Ansprüche an Visualisierung der Datensätze oder statistische Auswertungsmechanismen gestellt werden, stellt SESAM eine einfache Aufbereitungsumgebung bereit. Abbildung 7.15 zeigt einen Überblick über die Organisation der Datensätze und dabei möglichen Anzeigeformen.

Abbildung 7.14 Die Konfiguration der Datensätze, die aus Protokollen zu Simulationsläufen ausgelesen werden sollen, geschieht über Primitivaufruf-Spezifikationen.

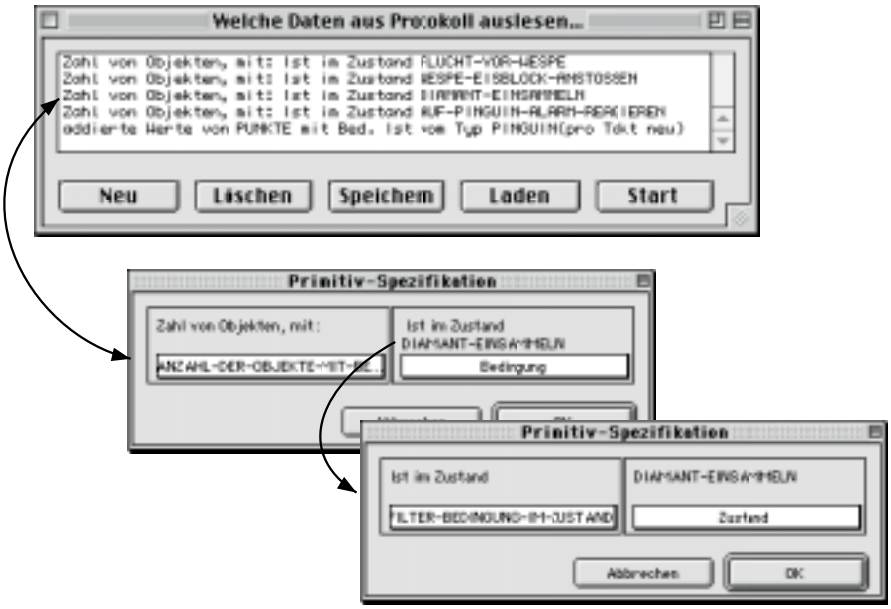
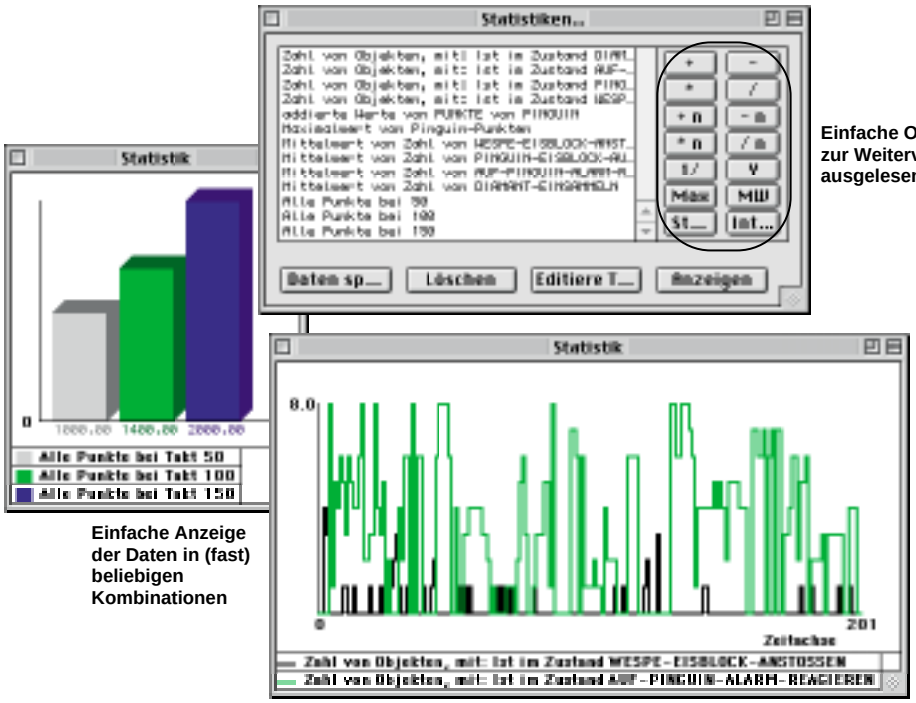


Abbildung 7.15 Elemente der einfachen Auswertungsumgebung zur Aufbereitung und Darstellung von Datensätzen.



Die Datensätze, deren Auslesen über Datenfilter gesteuert wurde, bilden die Basis für jede weitere Aufbereitungsaktion. Sie können umbenannt werden und in beliebigen Kombination angezeigt werden. Zusätzlich können grundlegende arithmetische Operationen mit diesen Datensätzen durchgeführt werden. Beispielsweise können mehrere Datensätze kombiniert werden, in dem die jeweiligen Datenwerte für jeden Zeittakt addiert werden. Darüber hinaus können Werte zu einem bestimmten Zeitpunkt abgegriffen oder Maximalwerte bestimmt werden. Die Berechnung von Mittelwerten, Standardabweichung oder Integralsummen runden die Menge möglicher Operatoren auf die Datensätze ab. Auch die so abstrahierten Datensätze können gespeichert bzw. exportiert werden.

Auf diese Weise entstehen zwei Formen von Datensätzen, die unterschiedlich visualisiert werden können: Einerseits können sie die Form von Funktionen (als Takt-Wert-Paare) annehmen, andererseits durch einen einzelnen Wert charakterisiert sein. Diese Datensätze können mit beliebig vielen anderen derselben Art zusammen visualisiert werden (siehe dazu auch Abbildung 7.15).

Auf diese Weise können in SESAM durchgeführte Experimente auch bis zu einem gewissen Grad aufbereitet werden, ohne die Umgebung zu verlassen. Durch Wiederverwendung des Konzepts der Primitivbeschreibung, folgt auch die Bedienung dieser kleinen Auswertungsumgebung den Vorgehensweisen, die ein Modellierer in SESAM gewohnt ist.

8

Anwendungsszenarien und Beispielmodelle

8.1 Anwendungsszenarien und Benutzergruppen

Bevor in den nächsten Abschnitten Anwendungsbeispiele besprochen werden sollen, die zeigen, wie und wozu man SESAM benutzen kann, sollen kurz einige grundsätzliche Anmerkungen zu Anwendungsszenarien und Benutzergruppen gemacht werden.

P. Mössinger et al. [Mössinger et al. 1995] unterscheiden für ihre Entwicklungsumgebung *XRaptor* drei Anwendertypen. Ein "dependent designer" entwickelt in einer gegebenen Umwelt die Kontrollarchitektur einer oder mehrere Agentenklassen. Der "independent designer" konstruiert dagegen die Umwelt, sowie die Sensoren und Effektoren der Agenten, also deren "Körper". Der dritte Typus von Benutzer ist der "supervisor", der verschiedene Modi der Simulation kontrolliert. Damit wird bestimmt, welche und wieviel Information den einzelnen Agenten zugänglich ist, also z.B. ob ein Agent seine absolute Position auf der simulierten Karte bestimmen kann oder nur auf die Daten aus seinen Ultraschall-Sensoren beschränkt ist. Der Anwendungsschwerpunkt für *XRaptor* liegt dabei in der Benutzung als Basis für Praktika und vorlesungsbegleitende Übungen. Dabei sind genau diese drei Anwendertypen wichtig, um einem Studenten als "dependent designer" eine passende Entwicklungsumgebung bereitzustellen. Der "independent designer" ist dabei der Tutor. Der "supervisor" bestimmt dann, ob der aktuelle Simulationslauf ein reiner Testlauf oder ein entscheidendes Turnierexperiment darstellt.

Man kann nun derartige Benutzertypen ausgehend von unterschiedlichen Anwendungsszenarien auch für SESAM identifizieren[1]. Dabei kann als Grundlage die in Abschnitt 6.2.1.1 vorgestellte Basisunterteilung in Domäne, Verhaltensmodell und Modellkonfigurierung für Multiagentenmodelle in SESAM dienen:

- Ohne Vorgaben, die über die generische Architektur und den vorgegebenen Primitiven hinausgehen, stellt SESAM eine Entwicklungsumgebung ohne konkrete Domänenbindung dar und kann als solche benutzt werden, um von Grund auf neue Multiagentenmodelle zu entwickeln.
- Verwendet man SESAM mit einer vorgegebenen Domäne, an die die allgemeine Entwicklungsumgebung angepasst wurde, kann ein Modellierer mit einem anwendungsspezifischen System arbeiten.

[1] Für jedes dieser Anwendungsszenarien kann aus SESAM ein neues, vollständiges und ausführbares Programm erzeugt werden.

- Ist in SESAM eine Domäne und ein Umweltmodell vorgegeben, kann man darin verschiedene Formen von Agentensystemen, z. B. mit unterschiedlichen Koordinationsformen oder individuellen Strategien untersuchen. Auf diese Weise kann ein in SESAM entwickeltes System als Testbed verwendet werden. Dabei kann man zwei Typen von Benutzern identifizieren: Testbed-Entwickler und Agentensystem-Designer.

- Eine auf SESAM basierende Experimentierumgebung ist dann vorhanden, wenn in einem Testbed Agentenverhalten modelliert wurde. Für verschiedene Einstellungen und Startsituationen lassen sich Simulationsexperimente durchführen. Hier kommt zu den obigen Rollen eines Testbed-Entwicklers und Agentensystem-Designers ein Experimentator.

- Wenn jedes Detail eines Multiagentenmodells und seiner "Einsatzumgebung" festgelegt ist, kann man einen Anwendungstyp finden: Über mehr oder weniger angepasste Eingriffmöglichkeiten des Benutzers, sei es bei der direkten Steuerung einzelner Agenten oder über das Anstoßen von Ereignissen, kann man ein so mit Vorgaben versehenes SESAM als ein Spiel identifizieren. Der Benutzer ist dann ein Experimentator mit beschränkter Macht, da er die Startkonfigurationen nicht in jedem Aspekt anpassen dürfen sollte.

An jede der so identifizierbaren Benutzertypen: Domänenentwickler, Modellierer, Experimentator oder Spieler, muß man unterschiedliche Anforderungen an die jeweils vorhandenen Modelliererfahrung, Kreativität oder "technisches" Geschick stellen.

Zur Erstellung einer Domäne muß man SESAM relativ gut kennen und möglicherweise auch auf der *CLOS*-Ebene programmieren, um gegebenenfalls mit neuen Primitiven die Grundfunktionalität zu erweitern. Um die Modellier- und Experimentierumgebung an die Domäne anzupassen, muß man die genaue Syntax und Semantik der Beschreibungen kennen und beachten.

Wurde diese Basis gelegt, kann ein Modellierer vollständig mittels graphischer Editoren vorgehen – unabhängig davon, ob er nur die Umwelt oder auch das Agentensystem konstruiert. Allerdings muß man zugeben, daß eine Kenntnis der grundlegenden Strukturen und Metaphern des Agentenverhaltens dringend notwendig ist, um das Verhalten geschickt zu formulieren – das System besitzt also nur beschränkte Fähigkeiten zur Selbsterklärung. Die Frage, in welcher Situation am besten eine Skelettplan-ähnliche Struktur oder doch eher ein Verhaltensautomat verwendet werden sollte, ist nicht immer sofort aus einem Konzeptmodell zu beantworten. Ebenso sollte der Modellierer auch die Primitive bis zu einem bestimmten Grad kennen[2], besonders wenn er auf die Laufzeit der Simulationsexperimente Rücksicht nehmen will. Bei kleineren Modellen ist dies unproblematisch, aber bei größeren Modellen kann es einen Unterschied machen, wieviele Objekte oder räumliche Einheiten abgetestet werden müssen, um einen Aspekt der Situation zu festzustellen. Allerdings ist das eher eine eher technische Entscheidung, konzeptionell ist eine geschickte Wahl der verwendeten Primitive weniger ausschlaggebend.

Dagegen muß ein Experimentator nur bestimmte Vorgehensweisen beachten, zum Beispiel im Bezug auf die Protokoll-Konfigurierung. Zunächst muß festgelegt werden, welche Daten aufgezeichnet werden sollen. Für die Auswertungen stehen dann genau diese Daten

[2] zumindest sollte der Modellierer, bevor er ein Primitiv erstmals verwendet, dessen Beschreibung im Handbuch gelesen haben.

zur Verfügung. Darüber hinaus ist die Bedienung einer an ein konkretes Modell angepassten Experimentierumgebung relativ intuitiv. Ein Benutzer, der nicht wie ein Experimentator für die technisch saubere Durchführung und Auswertung von systematischen Simulationsläufen zuständig ist, sondern mit einem vorgefertigten Modell "spielt", kann dies je nach vorgefertigter Umgebung ohne konkrete Kenntnisse über die Strukturen, die den jeweiligen Modellteilen zugrunde liegen, durchführen.

Um die praktische Verwendbarkeit eines Modellierungs- und Simulationswerkzeuges zu zeigen, muß dieses eingesetzt und gefordert werden. Dabei sollte nicht nur der Entwickler des Werkzeugs Beispielanwendungen implementieren, sondern auch Modellierer und andere Benutzergruppen ohne tiefgreifende Programmierkenntnisse sollten das System benutzen und zu ihren Erfahrungen mit dem System befragt werden.

Im folgenden soll nun anhand von ausgewählten Beispielanwendungen der bisherige Einsatz von SESAM dargestellt werden. Es konnte bisher keine systematische Evaluation der Werkzeuge durchgeführt werden. Dazu wäre eine weite Verbreitung eines wirklich stabilen Systems notwendig gewesen. SESAM befindet sich allerdings noch in einem Entwicklungsstadium, das durch persönlichen Kontakt zu Anwendern und deren direkte Befragung viele Verbesserungen erlaubt. Gerade bei der praktischen Verwendbarkeit der Oberfläche und bei der Modellierung und Experimentation, haben die Anmerkungen fachfremder Benutzer in vielerlei Hinsicht zu Erweiterungen beigetragen.

Im folgenden sollen zunächst einfache Reimplementationen bekannter Modelle, wie beispielsweise *Pengi* und anderer Simulationsmodelle mit expliziten Raumdarstellungen (Zelluläre Automaten) beschrieben werden. Danach sollen Anwendungen aus den beiden wichtigsten der bisherigen Gebiete vorgestellt werden: Simulation sozialer Insekten und die Anwendung als Entwicklungsumgebung für ein Testbed in der psychologischen Forschung. Dabei soll das Potential, das SESAM besitzt, deutlich werden. Der Abschnitt endet mit einer kurzen Darstellung des Einsatzes in der universitären Ausbildung und einer Zusammenfassung geeigneter Originalsysteme.

8.2 Einfache Reimplementationen

8.2.1 Pengi

Pengi [Agre & Chapman 1987], ein aus einem Videospiel entwickeltes Szenario, ist wohl eines der beliebtesten Beispielszenarien für agentenbasierte Systeme. Es gibt zwei Typen von Agenten mit unterschiedlichen Zielen: Pinguine wollen möglichst viele Diamanten aufsammeln. Die Aufgabe der Wespen ist es, Pinguine einzukreisen und sie auf diese Weise zu "töten". Darüber hinaus gibt es auf dem Spielfeld "Eisblöcke", die sich, wenn sie einmal von Wespen oder Pinguinen angeschubst wurden, solange gerade über das Spielfeld bewegen, bis sie auf ein Hindernis stoßen. Sowohl Pinguine als auch Wespen "sterben", wenn sie von einem Eisblock überrollt werden. In SESAM wurde eine vereinfachte Version dieses Spiels reimplementiert. Das Szenario wurde dahingehend vereinfacht, daß ein Pinguin nicht eingekreist, sondern mehrfach in einem Takt gestochen werden muß, um "getötet" zu werden. Das ist in so fern eine massive Vereinfachung des Szenarios, da die Wespen kein Einkreisungsverhalten koordinieren müssen. Daneben wurde eine höhere Zahl und Kommunikationsfähigkeit der Wespen durch eine größeren Wahrnehmungsradius und eine temporär höhere Geschwindigkeit der Pinguine ausgeglichen.

In einer diskreten Umwelt suchen die Pinguine nach Diamanten, indem sie zufällig über das Spielfeld laufen. Können sie einen Diamanten im Abstand von maximal zwei Feldern wahrnehmen, laufen sie darauf zu und sammeln ihn ein. Diese Grundtätigkeit wird in zwei Situationen kurz unterbrochen: Wenn sie eine Wespe hinter einem Eisblock wahrnehmen, stoßen sie diesen an. Andererseits "fliehen" sie wenige Takte mit höherer Geschwindigkeit, wenn sie mehrere Wespen gleichzeitig wahrnehmen. Wespen suchen auf der anderen Seite das Spielfeld zufällig nach Pinguinen ab. Wenn sie einen Pinguin in einem Nachbarfeld wahrnehmen, fokussieren sie diesen und schicken eine entsprechende Nachricht an alle anderen Wespen und folgen dem Pinguin solange, bis sie ihn nicht mehr wahrnehmen. Patrouliert eine Wespe und erhält eine "Pinguin-Alarm-Nachricht", folgt sie dieser. Durch diese einfachen Verhaltensmuster entstehen relativ ausgewogene Spiele.

Alle im letzten Kapitel verwendeten Screenshots zur Erklärung der Editoren und Mechanismen wurden mit diesem Modell und damit durchgeführten Experimenten aufgenommen. Deshalb werden an dieser Stelle keine weiteren Screenshots gezeigt.

Ein anderes klassisches Szenario ist das "Schafhund"-Modell, das P. Wavish [Wavish 1992] benutzte, um daran zu zeigen, wie nützliches Verhalten eines Gesamtsystem durch Emergenz erzeugt werden kann. Ein Schafhund läuft entlang einer Schafherde, die Schafe weichen dem Hund aus. Durch die Gleichzeitigkeit dieser beiden Verhaltensweisen entsteht ein "Zusammentreiben" der Herde. Auch dieses Szenario wurde in SESAM nachgebildet.

8.2.2 Zelluläre Automaten: LIFE und andere Modelle

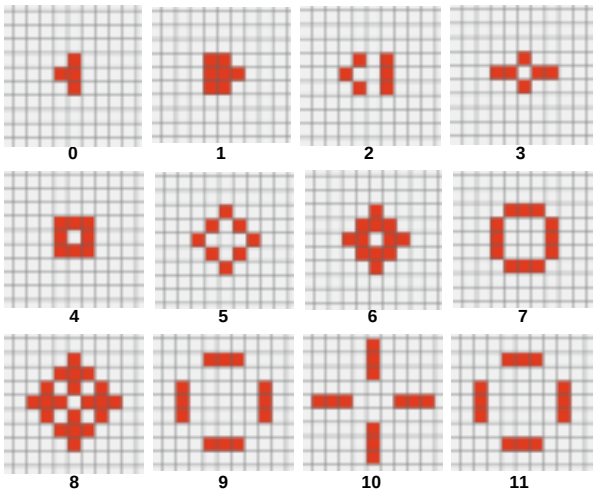
Ebenso unproblematisch war die Nachbildung zellulärer Automaten oder anderer Systeme, bei denen eine explizite Raumrepräsentation im Vordergrund steht. In SESAM konnten ohne größeren Aufwand verschiedene derartige Modelle reproduziert werden. *Life* als bekanntester Zellulärer Automat (siehe dazu Seite 48) bildete dabei den Ausgangspunkt. Dabei wird jede Position eines Bezugssystem durch einen Agenten besetzt, der den Zustand seiner unmittelbaren Nachbarn wahrnehmen kann. Jeder Agent ändert seinen eigenen Zustand abhängig von der wahrgenommenen Situation. Das funktioniert beim *Game of Life* nach folgender Formel. Dabei sind a_n die Zustände der Zellen der Moore-Nachbarschaft (siehe dazu Abschnitt 3.2.2.1):

$$a_{i,j}^{(t+1)} = \begin{cases} a_{i,j}^{(t)} & , \quad wenn \sum_n a_n = 2, \\ 1 & , \quad wenn \sum_n a_n = 3, \\ 0 & , \quad sonst \end{cases}$$

In Abbildung 8.1 wird eine Folge von Zustandskonfigurationen dargestellt, die nach dieser Formel erzeugt werden.

Das größte Problem, das im Zusammenhang mit diesen Zellulären Automaten gelöst werden mußte, lag in der fehlenden Gleichzeitigkeit der Zustandsänderung in SESAM. In SESAM wird jeder Agent nacheinander vollständig aktualisiert, bevor der nächste behandelt wird. Am Ende des Takts wird die Aktualisierungsreihenfolge zufällig geändert (siehe Algorithmus 6.3.1.2). Bei Zellulären Automaten wird zur Neuberechnung eines Zustands $a_{i,j}^{(t+1)}$ von allen Nachbarzellen zum Zeitpunkt t benötigt, der bei einer derartig unbestimmten Aktualisierungsreihenfolge nicht trivial zu verwalten ist. Eine einfache Lösung ist es, sich zum aktuellen Zustand zusätzlich den Zustand vor der letzten Aktualisierung mit dem entsprechenden Zeitstempel zu merken. Ein anderer, ähnlicher Zellulärer Automat, der nach den

Abbildung 8.1 Folge von Zellenkonfigurationen zur Entstehung eines "Blinkers" in der Reimplementierung von *LIFE* in SESAM



Erfahrungen mit *Life* relativ schnell[3] reimplementiert werden konnte, ist das in Abbildung 3.4 dargestellte Modell der Meinungspropagierung.

8.2.3 Das Nagel-Schreckenberg-Modell

Ein weiterer, interessanter Zellulärer Automat, der in SESAM reimplementiert wurde, ist das *Nagel-Schreckenberg-Modell* [Esser et al. 1998, Nagel & Rasmussen 1994] zur Verkehrssimulation. Dabei folgt in der ursprünglichen Version des Zellulären Automaten, wie in Abschnitt 3.2.2.3 bemerkt, die Beschreibung des Modells eindeutig einem agentenbasierten Paradigma, während die Darstellung als Zellulärer Automat etwas umständlich erscheint. Ein Fahrzeug i bestimmt nach folgenden Regeln seine Geschwindigkeit v_i und bewegt sich dementsprechend [Esser et al. 1998]:

1. Beschleunigung: $v_i \leftarrow min(v_i + 1, v_{max})$
2. Abbremsen zur Unfallvermeidung: $v_i \leftarrow min(v_i, g_i)$, mit g_i Abstand zum Vorderfahrzeug
3. Mit Trödelwahrscheinlichkeit p_{dec}: $v_i \leftarrow max(v_i - 1, 0)$

Diese Regeln können direkt in SESAM als Verhaltensbeschreibung von Fahrzeugagenten umgesetzt werden. Die Autorin benötigte zusammen mit J. Wahle vom Lehrstuhl für Transport und Verkehr der Universität Duisburg[4] wenige Stunden, um ein mit dem ursprünglichen Modell vergleichbares zu implementieren. Tabelle 8.1 zeigt, daß die Reimplementierung des Nagel-Schreckenberg-Modells in SESAM wirklich analoge Ergebnisse produziert. Für diese Messungen wurde ein System mit 1000 Zellen und periodischen Grenzen (siehe

[3] Der Aufwand lag für die Autorin im Bereich von etwa einer Stunde.

[4] An diesem Lehrstuhl wurde das Nagel-Schreckenberg-Modell entwickelt und wird unter anderen von J. Wahle in verschiedenen Szenarien eingesetzt.

Abschnitt 3.2.2.1) benutzt. 100 Agenten mit Maximalgeschwindigkeit $v_{max} = 6$ Zellen / Takt waren im System (entsprechend einer Dichte von $d = 0.1$ Agenten / Zelle). Die Werte sind gemittelt über 500 Messwerte nach einer Einschwingzeit von 500 Takten. Für $p_{dec} = 0.1$ und $p_{dec} = 0.25$ gilt die Bedingung $v = v_{max} - p_{dec}$, die andeutet, daß das System sich in einer Region des freien Flusses befindet. Bei $p_{dec} = 0.5$ ergibt sich eine Mischung aus Stau und freiem Fluß.

Tabelle 8.1 Vergleich zwischen Implementierung in SESAM und als Zellulärer Automat mit unterschiedlichen Trödelwahrscheinlichkeiten p_{dec}. Durchschnittsgeschwindigkeit $v_{average}$ und Abweichung in Zellen / Takt. Die Ergebnisse stimmen mit einem tolerierbaren Fehler überein.

	SESAM		**Zellulärer Automat**	
p_{dec}	$v_{average}$	Varianz	$v_{average}$	Varianz
0	6	0	6	0
0.1	5.8602	0.0338	5.8663	0.0354
0.25	5.7359	0.0480	5.7463	0.0437
0.5	3.1505	0.2090	3.1863	0.0356

Ein großer Vorteil der direkten Umsetzung als Multiagentenmodell ist dabei, daß die Umwelt, aber auch das Verhalten der Agenten ohne Probleme realistischer, bzw. reichhaltiger gestaltet werden kann. Der ursprüngliche Zelluläre Automat funktioniert sehr gut für Autobahnverkehr oder andere ununterbrochen lineare Strukturen. Einmündungen, Kreuzungen sind aufwendig, aber noch möglich.

Individuelle Routen-Entscheidungen der Agenten sind dagegen in einer Formulierung als Zellulärer Automaten kaum, bzw. sehr umständlich darstellbar. Gerade darin liegt aber die Stärke der Multiagentensimulation. A. Bazzan et al. [Bazzan et al. 1999] setzen genau an dieser Stelle an, indem sie zusätzlich zur taktischen Ebene der Nagel-Schreckenberg-Bewegung eine weitere Ebene für Entscheidungen zur Auswahl der Route abhängig von "beliefs" über angenommene Entscheidungen anderer Verkehrsteilnehmer und Inhalte von Verkehrsnachrichten einbauen. Daraus entwickelte sich eine interessante Kooperation mit J. Wahle und M. Schreckenberg vom Lehrstuhl für Physik von Transport und Verkehr der Universität Duisburg und A. Bazzan vom Lehrstuhl für Künstliche Intelligenz der Universität Porte Alegre in Brasilien.

Ein in SESAM implementiertes Modell erweitert das oben darstellte Nagel-Schreckenberg-Modell auf zwei Routen um sogenannte "floating cars", die am Ende ihrer Route ihre Fahrzeit als "Verkehrsnachricht" auf ein Blackboard schreiben. Ein bestimmter Anteil der Fahrzeugagenten entscheiden zu Beginn ihrer Fahrt abhängig von diesen Nachrichten, welche Route sie nehmen. Dieses Szenario mit dynamischen Verkehrsnachrichten und Fahrern, die auf möglicherweise nicht mehr aktuelles Zustandswissen reagieren, wird zur Zeit intensiv untersucht und bildet die Grundlage für diverse Experimente, die die Basis für vorausschauende Verkehrsnachrichten bilden (siehe dazu [Bazzan et al. 2000, Klügl et al. 2000]).

8.3 Verhalten sozialer Insekten

Im Gegensatz zu den oben beschriebenen, relativ einfachen Modellen, die von den SESAM-Entwicklern, bzw. Informatikstudenten entwickelt und implementiert wurden, wurden die folgenden Modelle zu sozialen Insekten hauptsächlich von Biologen erstellt, die damit eigene Fragestellungen untersuchen wollten. Keiner dieser Anwender war vor der Benutzung von SESAM sehr erfahren im Umgang mit Modellierungs- und Simulationssoftware, sondern ist eher in der empirischen Forschung beheimatet. Damit gehören diese Anwender zu der Benutzergruppe, für die SESAM entwickelt wurde. Jedes der im folgenden beschriebenen Modelle, das Ameisenmodell, in dem vor allem die Kombination mehrerer emergenter Phänomene formuliert werden sollte, die Modelle zur Arbeitsteilung bei Honigbienen und Interaktion zwischen Bienen und Milben auf Populationsebene, stellen jeweils durch ihre Charakteristika reizvolle Anforderungen an die Entwicklungsumgebung.

8.3.1 Emergente Phänomene bei Ameisen

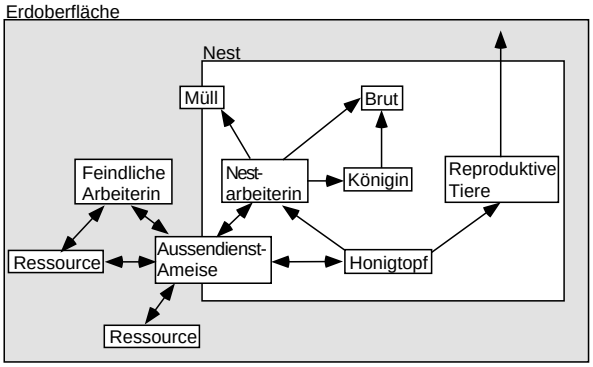
Im ersten, wirklich großen Modell, das mit SESAM entwickelt wurde, sollte untersucht werden, wie verschiedene, emergente Phänomene darstellt und kombiniert werden können. Dr. U. Raub entwickelte mit Unterstützung der Autorin zwischen 1996 und 1998 dieses Modell, das verschiedene Aspekte im Leben eines Ameisenstaates beleuchtet. SESAM wurde gleichzeitig mit diesem Modell entwickelt. Deshalb war die Autorin direkt als "Wisseningenieurin" an der Implementierung beteiligt. Einen Überblick über die Bestandteile des Modells findet man in [Klügl et al. 1996, Klügl & Raub 1997], die Ergebnisse der Kombination wurden in [Klügl et al. 1998] beschrieben.

Im einzelnen kann dieses nahezu vollständige Ameisenmodell wie folgt beschrieben werden. Es orientiert sich dabei am Verhalten der *myrmecocystes mimicus*, einer Ameisenart, die in der Wüste von Arizona beheimatet ist. Eine modellierte Kolonie besteht aus Ameisen, die zu verschiedenen Kasten, also Gruppen von Einheiten mit mehr oder weniger spezialisierten Aktivitäten, oder Entwicklungsstadien gehören. Jede dieser Kategorien wurde als einzelner Verhaltenstyp formuliert, der abhängig vom Alter und von der individuell wahrgenommenen Energiesituation gewechselt werden kann. Abbildung 8.2 zeigt diese verschiedenen Verhaltenstypen und skizziert Interaktionen zwischen einzelnen Tieren, aber auch zu passiven Objekten, wie Futter-Ressourcen und Müllobjekten. Feindliche Ameisen können dabei jeden Verhaltenstyp annehmen. Sie sind durch ihre Zugehörigkeit zu einem bestimmten Ameisenstaat charakterisiert.

Dabei wurden im einzelnen folgende Teilphänomene abgebildet, die im Leben einer Ameisenkolonie eine Rolle spielen:

- Produktion von neuen Ameisen, die sich über ein abstrahiertes Brutstadium zu Arbeiterinnen oder neuen Königinnen entwickeln. Während dieser Wachstumsphase werden sie nicht nur gefüttert und anderweitig gepflegt, sondern auch umgelagert.
- Erforschung neuer Terrains und Fouragierverhalten (d.h. Futtersuche) mit Spurablage, Spurfolgeverhalten und Futtereintrag in die Kolonie. Dabei entscheidet eine Aussendienst-Arbeiterin aufgrund der von ihr wahrgenommenen Energiesituation (in ihrem eigenen inneren Magen und dem sozialen Magen der Tiere, denen sie begegnet), ob sie in das Fouragierverhalten tritt (siehe Abbildung 8.3, Teil A).

Abbildung 8.2 Übersicht über Verhaltenstypen im Ameisenszenario und deren Interaktionen



- Das eingetragene Futter wird in "Honigtopf"-Ameisen gespeichert. Darüber hinaus wird jedesmal, wenn sich zwei Ameisenagenten, die zur selben Kolonie gehören, der Energiewert ihrer "sozialen Mägen" angeglichen ("Throphallaxis"), wodurch im Nest ein mehr oder weniger gleichmäßiger Energiezustand entsteht.

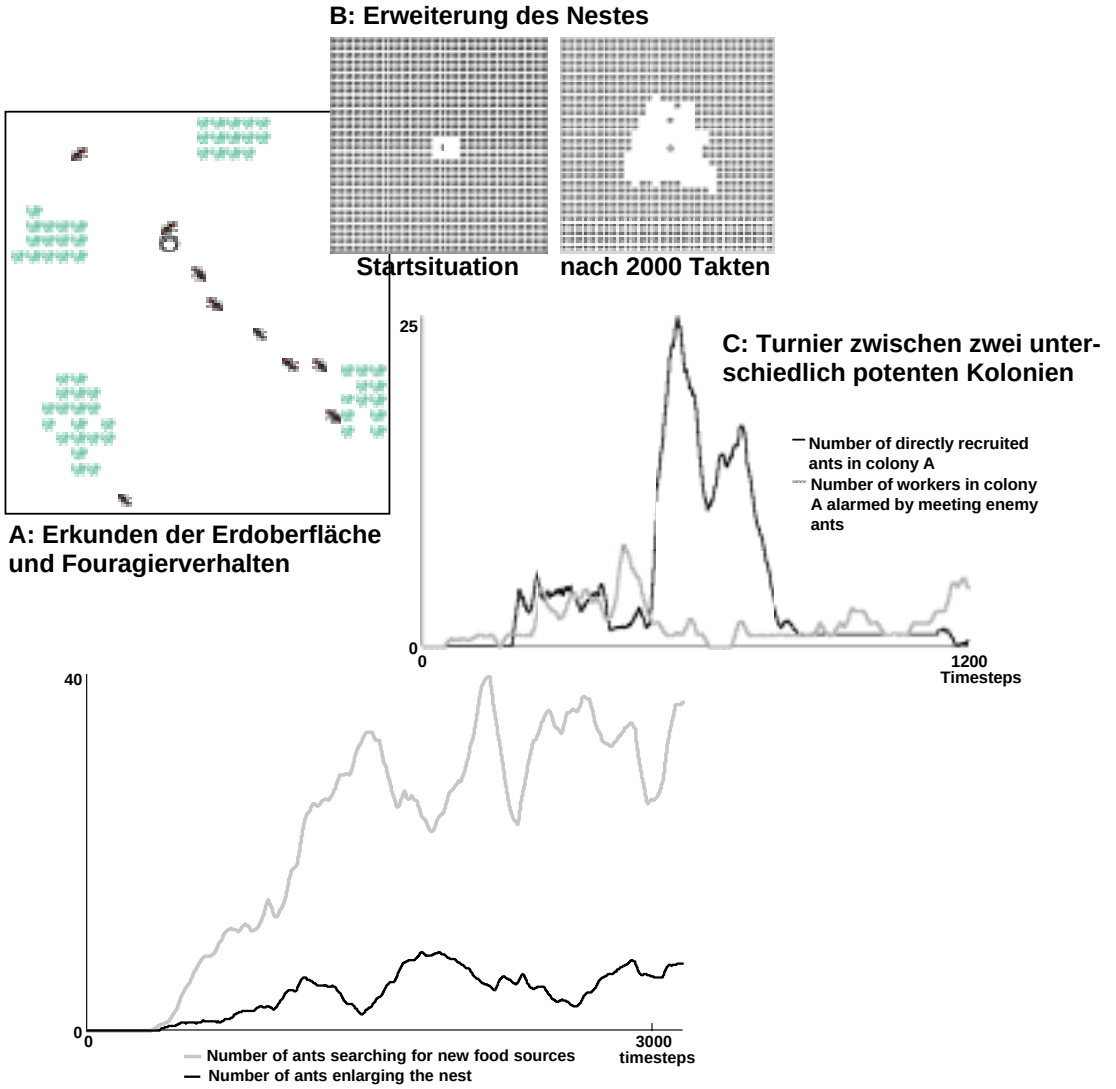
- Bei Wachstum der Kolonie entscheiden Arbeiterinnen aufgrund der von ihnen wahrgenommenen Ameisen-Dichte im Nest, ob sie Aktivitäten zur Nestvergrößerung ausführen. Dazu sucht das simulierte Tier nach anderen Arbeiterinnen, die ebenfalls graben und gräbt an der selben Stelle weiter, bzw. fängt an einer zufälligen Stelle an. Beim Austragen der Grababfälle werden wiederum Pheromone benutzt, um einen Müllberg zu finden (siehe Abbildung 8.3, Teil B).

- Auch interkoloniale Konflikte wurden in das Modell miteinbezogen, indem die Turnierkämpfe zwischen Tieren der verschiedenen Kolonien nachgebildet wurden. Trifft eine Aussendienst-Arbeiterin auf der Erdoberfläche auf eine Ameise aus einer anderen Kolonie, läuft sie "aufgeregt" herum. Zählt sie während dieser Aktivität mehr Ameisen aus anderen Kolonien als aus der eigenen, läuft sie zurück ins Nest und legt dabei eine Alarmierungspur, der andere Arbeiterinnen zum "Turnierplatz" folgen und dort dasselbe Verhalten beginnen (siehe Abbildung 8.3, Teil C).

- Zu bestimmten Zeiten, bzw. wenn ausreichend neue Königinnen entstanden sind, fliegen diese aus und gründen neue Kolonien und erschließen auf diese Weise neue Habitate. Männliche Tiere, die eigentlich an dieser Stelle ein Bestandteil des Modells sein müßten, wurden nicht miteinbezogen.

Durch die Kombination dieser Einzelverhaltensweisen in ein Gesamtmodell wurden auch Erkenntnisse über die ausgewogene Kalibrierung der natürlichen Verhaltensregulation gewonnen. Das Modell wurde iterativ erstellt. Jedes Mal, wenn ein neuer Bestandteil aufgenommen wurde, mußte das gesamte Modell neu balanciert werden, da die einzelnen Phänomene sehr stark von einander abhängen. So muß zum Beispiel der Energieverbrauch der Brut durch erweitertes Fouragieren ausgeglichen werden. Die Folge davon ist, daß die Zahl der Brutagenten in Relation zur Zahl der Aussendienst-Arbeiterinnen stehen muß. Dabei ergibt sich nicht nur ein massives Problem bei der Kalibrierung eines derartig umfangreichen

Abbildung 8.3 Zusammenstellung verschiedener Aspekte aus beispielhaften Simulationsläufen des Ameisenmodells



Modells, sondern auch bei der Validierung des Modells.

Diese müßte im Grunde auf drei Ebenen geschehen: Das Verhalten eines individuellen Agenten muß dem in der Realität ausreichend genau entsprechen, ein einzelnes Phänomen muß ebenso wie das Zusammenspiel aller Aspekte in der gesamten Kolonie dem natürlichen entsprechen. Die Validation geschah in diesem Fall durch Bewertung der Plausibilität der Parameterverläufe und der Ähnlichkeit zwischen dem animierten Simulationsexperimenten und dem natürlichen Verhalten. Der Nestbau wurde dabei nicht durch Vergleich der erzeugten mit natürlichen Strukturen validiert, sondern über plausible Beschäftigungsanteilen bei Arbeiterinnen. Wichtige Parameter sind dabei global messbar: So unterliegt die Größe einer Kolonie einem langfristigen Zyklus mit verschiedenen Stadien, der qualitativ reproduziert werden konnte. Auch die Wahrscheinlichkeit, mit der eine Kolonie, die jeweils

durch eine einzelne Königin gegründet wird, das Wachstumsstadium überlebt, ist plausibel.

Auf dieser Weise wurden allerdings mehr Erkenntnisse über den Modellierungsprozess und über die Grenzen eines kalibrierbaren und validierbaren Multiagentenmodells gewonnen, als inhaltliche Hypothesen zum Ameisenverhalten getestet. Die wichtigste Erfahrung war demnach, daß die Validierung bei großen Modellen trotz analoger Beobachtungsbasis – Agent als einzelne markierbare Ameise – schwierig und die Kalibrierung eines derartigen Modells aufwendig und diffizil ist. Beide Phasen des Entwicklungsprozessen müssen weiter und besser unterstützt werden.

Das Ziel, mit diesem Modell die Leistungsfähigkeit von SESAM zu testen, wurde dagegen erreicht. So besitzen zum Beispiel Aussendienst-Arbeiterinnen ein Verhaltensrepertoire von mehr als 60 Aktivitäten. Eine Übersicht über das Modell zu behalten, war nur durch eine stringente Strukturierung möglich. In Simulationsexperimenten konnten mehrere tausend Agenten gleichzeitig simuliert werden.

8.3.2 Arbeitsteilung bei Honigbienen

Bei einem Modell, das Dipl. Biologin A. Dornhaus zum Bienenverhalten entwickelte, war das Ziel herauszufinden, welchen Effekt individuelle Variationen bei der Reizreaktion auf den globalen Erfolg eines Bienenstocks besitzen (für weitere Ausführungen siehe [Dornhaus et al. 1998]). Das Modell beschreibt das Verhalten der individuellen Tiere im Innern eines Bienenstocks. Die Interaktion mit der Außenwelt wurde über Wahrscheinlichkeitsverteilungen beim Fouragiererfolg abstrakt dargestellt. Insgesamt ist das Modell strukturell dem oben beschriebenen Ameisenmodell ähnlich. Abbildung 8.4 liefert eine Übersicht über das gesamte Modell, in dem es die verschiedenen Verhaltenstypen und die möglichen Übergänge und Aktivitäten darstellt. Abbildung 8.5 ergänzt dieses Schema um einen Ausschnitt aus dem simulierten Bienenstock.

Für die Zielsetzung war es wichtig, ein umfangreicheres Modell zu konstruieren, bei dem viele unterschiedliche Aufgaben von den simulierten Bienen erfüllt werden müssen, da anzunehmen ist, daß sich andernfalls die Effekte der Schwellenvariabilität nicht signifikant zeigen würden [Dornhaus et al. 1998]. Alle relevanten Aktivitätsübergänge wurden dabei parametrisiert und konnten so für jeden Bienenagenten individuell gestaltet werden. Es wurden Simulationsläufe für Bienen mit gleichwertigen Schwellen mit Experimenten für zufällige Schwellwerten, die jeweils zwischen unterschiedlichen Intervallen gewählt wurden, verglichen. Dabei wurde festgestellt, daß die Gesellschaft der Spezialisten, also derjenigen, die über unterschiedliche Reizschwellen verfügten, eindeutig eine höhere Zahl an Verpuppungen erreichte.

Allerdings konnte dieses Ergebnis bei späteren Versuchen mit einem veränderten Modell nicht mehr reproduziert werden. Dabei wurden keine strukturellen Änderungen, sondern nur Parameteranpassungen vorgenommen. Auch aus den Erfahrungen mit diesem Modell kann man schließen, daß das Kalibrieren eines Multiagentenmodells diffizil ist und derartige Simulationen grundsätzlich sehr sensibel gegen Parameteränderungen sind. Diese Erkenntnis ist allerdings nicht überraschend, wenn man sich bewußt macht, daß ein Parameter im Verhaltensmodell, mit dem alle Agenten einer bestimmten Kategorie gesteuert werden, Auswirkungen für jeden dieser Agenten besitzt. So werden die Effekte eines Parameterwerts potenziert. Die Arbeiten an diesem Modell wurden aus Zeitgründen nicht weitergeführt. Frau Dornhaus bereitet zur Zeit ein Modell zu den Effekten unterschiedlicher Signalqualitäten bei der Kommunikation von Bienen im Vergleich zu Hummeln vor.

Abbildung 8.4 Überblick über die Verhaltenstypen, deren Aktivitäten und Übergänge im Arbeitsteilungsmodell bei Honigbienen (aus [Dornhaus et al. 1998]).

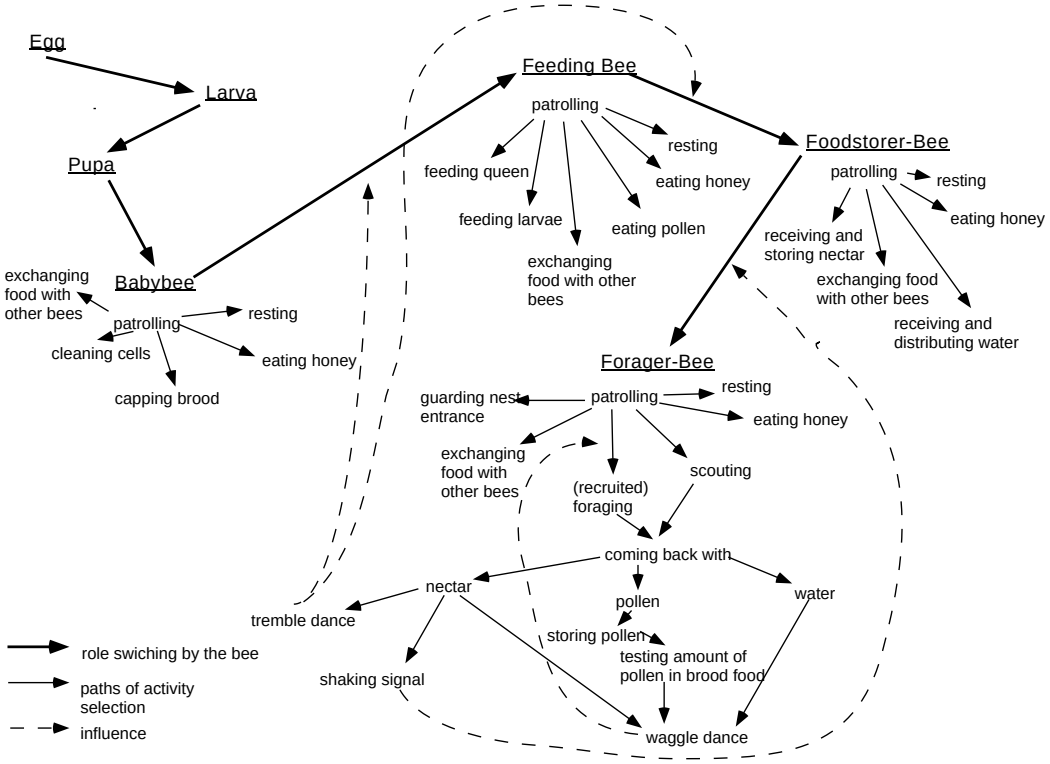
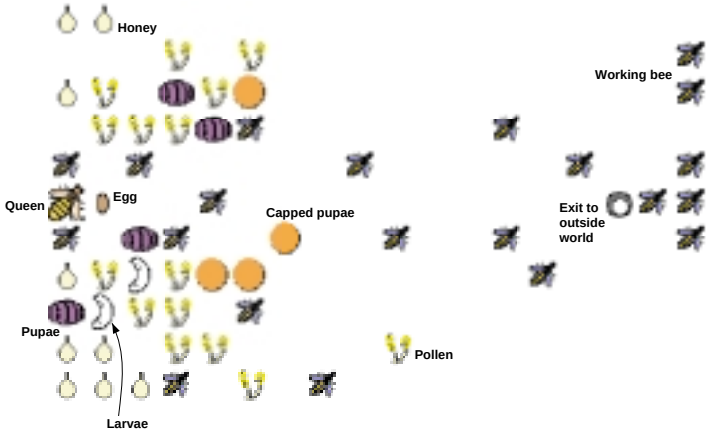


Abbildung 8.5 Bild (mit Beschreibung der Bestandteile) während der Simulation im Inneren eines Bienenstockes (aus [Dornhaus et al. 1998])



8.3.3 Interaktionen zwischen Bienenvölker und Varroa-Milben

Während bei den obigen Simulationen der Ausgangspunkt bei den Verhaltensweisen einzelner Tiere liegt, wird in einem Modell von Dr. S. Fuchs et al. [Fuchs et al. 1999], das von K.

Eggelbusch in ihrer Diplomarbeit überarbeitet und erweitert wurde [Eggelbusch 2000, Eggelbusch et al. 2000], Bienenvölker als Agenten betrachtet. Dabei spielen vielfältige Interaktionen auf der Ebene von Bienenvölkern zusätzlich zu klassischen Wachstumsformeln eine Rolle. Dabei soll die Frage nach einem möglichen Gleichgewicht zwischen Populationen von Honigbienen und Varroa-Milben beantwortet werden. Insbesondere sollen dabei verschiedene Übertragungswege und Virulenzen der Parasiten berücksichtigt werden.

Die Basis des Modells bildet ein Wachstumsmodell eines Bienenvolkes in Diffenzialgleichungsform, das über Parameter für die Individuenzahl bei einem Bienenvolk-Agenten realisiert wurde. Bienenvölker interagieren über "verfliegende Bienen", also einzelne Bienen – ebenfalls als Bienenvolk-Agenten mit geringeren Individuenzahlen realisiert –, die nach dem Fouragieren in einem fremden Stock landen. Eine weitere Möglichkeit der Interaktion sind "räubernde Bienen", die aus einem sterbenden Volk dessen Ressourcen entfernen. Über Bienenschwärme, also sich teilende, ausreichend große Völker, wurde ein natürlicher Ausbreitungsmechanismus der Bienen modelliert. In ein derartiges, stabiles Populationsmodell für interagierende Bienenvölker wurden einzelne simulierte Varroa-Milben eingeführt. Ein infiziertes Bienenvolk besitzt dabei eine flachere Wachstumskurve, bzw. stirbt, wenn der Milbenbefall zu hoch ist. Milben werden über die Interaktions- und Ausbreitungsformen mittransportiert und so in der Gesamtpopulation verbreitet.

K. Eggelbusch bezog sich bei der Wahl der Parameter und ihrer zeitlichen Relationen auf genaue Daten aus der Literatur. Das Modell besitzt durch die Differenzialgleichungen zum Wachstum der Bienenvölker und anderer exakt umgesetzter Wahrscheinlichkeitsverteilungen einen mathematischen Charakter und wurde durch Vergleich des Verlaufs der wichtigsten Kenngrößen, wie z.B. die Gesamtpopulation validiert. Dr. S. Fuchs vom Bienenforschungsinstitut Oberursel der Universität Frankfurt beurteilte als Experte der Impkereipraxis die Verlaufsdaten zusätzlich auf ihre Plausibilität.

Allerdings muß man zugeben, daß die Experimentier- und Auswertungsumgebung von SESAM zeitweise bei großen Karten mit Hunderten von Bienenvölker-Agenten an die Grenzen der Leistungsfähigkeit stieß. Die erhöhten Anforderungen an die Effizienz des Simulators und der Auswertungsroutinen veranlassten einige Verbesserungen auf technischer Ebene.

8.4 Strategien im Feuer-Szenario

In der Psychologie werden schon seit langem Simulationsspiele verwendet, um komplexes Problemlösen beim Menschen zu untersuchen. Versuchspersonen sollen dabei Teile eines Gesamtsystems steuern. Die dabei von ihnen verfolgten Strategien werden per Videogerät und anschließende Interviews aufgezeichnet [Dörner & Schaub 1992]. Ein bekanntes Beispiel ist das "Feuerszenario", bei dem Feuerlöscheinheiten, Raupen und Hubschrauber, mit unterschiedlichen Fähigkeiten, z.B. bei der Fahrgeschwindigkeit und Löschwasserkapazität, vom menschlichen Benutzer so gesteuert werden sollten, daß auftretende Waldbrände möglichst eingedämmt werden konnten.

In den Diplomarbeiten von E. Büttner [Büttner 1999] (Universität Bamberg unter Betreuung von H. Schaub) – weitergeführt von P. Huber [Huber 2000] sollte der Versuch unternommen werden, daß Versuchspersonen nicht mehr ausschließlich per Hand die Feuerlöscheinheiten "kommandieren", sondern einfache SESAM-Modelle für die Steuerung dieser Einheiten erstellen sollten. Das Ziel dabei war, daß die Versuchspersonen ihre Strategien explizit

beschreiben müssen, anstatt wie zuvor nur implizit anwenden brauchten. So stehen einem Versuchsleiter deutlich mehr und möglicherweise auch bessere Information als Grundlage für seine Interpretationen zur Verfügung

Für diese Experimente wurde das Feuerszenario, das von den Bamberger Psychologen benutzt wird, in SESAM nachgebildet.

In Abbildung 8.6 wird eine Situation während der Simulation dargestellt.

Abbildung 8.6 Bild aus dem Feuerszenario

Die Versuchspersonen mußten unter Anleitung von E. Büttner Verhaltensautomaten und Regeln für die Raupen und Hubschrauber formulieren. Während der Simulationsexperimente konnten zudem alle Benutzeraktionen zusätzlich zu anderen Simulationsdaten, wie Zahl der Feuer, u. a. aufgezeichnet werden. Der Einsatz von SESAM als Basis für derartige Modellierungsexperimente stellte hohe Anforderungen an das System. Die Entwicklungs- und Simulationsumgebung mußte dazu nicht nur völlig stabil laufen, da für die Versuche jeweils nur wenige Stunden zur Verfügung standen, sondern mußte weitestgehend intuitiv bedienbar sein. Durch die Experimente bzw. deren Vorbereitung kamen einige Fallstricke in der Modellierumgebung SESAM zum Vorschein und konnten deswegen behoben werden. Da nur eine der vier Versuchspersonen über Programmiererfahrungen verfügte bzw. die anderen “Neulinge” waren, kann man diese Versuche als echte Tests des “enduser programming” betrachten.

Bei diesem Einsatz von SESAM ist auch die Form der strukturellen und inhaltlichen Vorgaben interessant, da diese die Strategien, die die Versuchspersonen formulieren können, beeinflußt. Gerade für Untersuchungen dieser Fragestellung sind weitere Arbeiten geplant bzw. wurden bereits von P. Huber in seiner Diplomarbeit weitergeführt.

Während der Experimente von E. Büttner konnte auch festgestellt werden, daß an dieser Stelle ein weiteres Potential an Versuchen zur Delegation möglich ist. Durch die Kombination aus Aktivitätsbeschreibung und Eingriffmöglichkeiten während der Simulation, kann eine Versuchsperson selbst unterschiedliche Strategien verfolgen. Eine Einheit kann vollständig automatisiert werden. Andererseits können komplexe Aktionen beschrieben und

die Versuchsperson gibt während der Simulation den Rahmen für die Ausführung diese Aktionen vor. Die weitere Kooperation mit Prof. H. Schaub soll das Szenario ausbauen. Die dabei geplanten Fragestellungen reichen vom Effekt des Feedback der Agenten während der Ausführung der von den Versuchspersonen programmierten bzw. zugewiesenen Aktivitäten bis hin zu Aspekten des Blickwinkels der Versuchsperson. Bei letzterem ist die Umsetzung der globalen Betrachtungsweise von Außen auf die lokalen Sichtweisen der Feuerlöscheinheiten im System interessant.

8.5 Einsatz in der universitären Ausbildung

Seit der ersten Version des Simulators im Winter 1995 wurde SESAM in vier interdisziplinären – Informatik / Biologie – Praktika zur Modellierung des Verhaltens sozialer Insekten und in einem weiteren Praktikum von zwei Gruppen von Informatikstudenten zur Implementierung eines Modells der Computerwirtschaft und historischer Stadtentwicklung verwendet. Die dabei entstandenen Modelle sind sehr interessant, ich möchte sie aber aus Platzgründen hier nicht weiter besprechen.

Ein anderes Einsatzgebiet ergab sich aus der Verwendung von SESAM als Entwicklungsumgebung für Testbeds: Wegen der objekt-orientierten Implementierung von SESAM kann die Aktionsselektion eines Agenten als Methode, die wie bei "Standard" – Klassenbibliotheken für besondere Agentenklassen spezialisiert werden. Demzufolge können nicht nur über die vorgegebenen Architekturstrukturen Verhalten beschrieben, sondern auch vollständig andere Agentenformen eingebunden werden. Auf dieser Basis wurde SESAM für drei Informatik-Diplomarbeiten verwendet. Bei diesen wurden reaktive Planer in einem Einkaufszenario [Oechslein 1997], Reinforcement Lernen beim Fußballspielen [Hargesheimer 1998] und verschiedene Organisationsformen in einem Entwicklungsspiel [Schmidt 1998] untersucht.

8.6 Betrachtung zu passenden Anwendungsszenarien

Mit den oben skizzierten Anwendungen wurde gezeigt, daß SESAM einerseits in verschiedenen Szenarien zu Modellierung unterschiedlicher, auch komplexer Verhaltensmodelle geeignet ist und andererseits diese Modellierung auch von Fachexperten ohne Kenntnisse einer speziellen Programmiersprache bzw. ohne den Willen, auf dieser Ebene zu operieren, durchgeführt werden kann.

Bei der Betrachtung von Anwendungsmöglichkeiten und Potential stellt sich zunächst die Frage, welche Art von Multiagentenmodellen gut in dem vorgestellten Rahmen repräsentierbar sind bzw. für die Formulierung welcher Art von Systemen er dagegen weniger geeignet ist.

Besonders gut eignet sich SESAM mit seiner generischen Agentenarchitektur für die Beschreibung von Systemen mit folgenden Eigenschaften:

- Bei den Agenten sind Aktivitäten als Aktionszustände klar identifizierbar. Diese sind von eindeutigen Situationen abhängig.
- Es sind klare Stereotypen ausmachbar, also die Verhaltensbeschreibung läßt sich gut strukturieren.

- Die Agenten sind nicht zu einfach: Hunderttausende sehr einfacher Agenten können effektiver direkt in einer nicht-deklarativen Programmierumgebung gestaltet werden.
- Die Agenten sind nicht zu komplex: Bei sehr wenigen komplexen, mit intelligenten Planungs- oder Schedulingalgorithmen ausgestattete Agenten, könnte man die entsprechenden Reasoning-Mechanismen direkt verwenden und sie nicht über eine abstrakte Beschreibungssprache nachbilden.
- Die Beschreibung der Umwelt ist wichtig und nicht trivial.

Allerdings gilt auch an dieser Stelle: Der Modellierer entscheidet über den Abstraktionsgrad und die Strukturierung des Modells. Der Zweck seiner Untersuchung bestimmt den Detaillierungsgrad und somit auch, wie elegant sich das Agenten- und Umweltmodell in SESAM formulieren läßt.

9

Bewertung und Ausblick

Am Ende einer Arbeit stellt sich immer die Frage, wie das Geleistete zu beurteilen ist und wie man es weiterführen sollte. Doch zunächst wird ausgehend von den gestellten Zielen, ein Fazit gezogen.

9.1 Zusammenfassung

Der Beitrag dieser Arbeit lag darin, einen Rahmen für die Entwicklung von Multiagentensimulationen zu entwerfen: Dieser war in eine Modellier- und Simulationsumgebung zu integrieren, um so Domänenexperten ein Werkzeug an die Hand geben zu können, mit dem sie besser als bisher Modellbildung und Simulation für ihre Untersuchungen verwenden können. Es wurde versucht, dieses etwas vage Ziel durch folgende konkreten Schritte zu erreichen.

1. Bereitstellung eines deklarativen Rahmenwerks für die Darstellung von Strukturen in einem Multiagentenmodell und der Repräsentation ihres Verhaltens
2. Entwicklung eines effizienten Interpreters für eine derartige Beschreibung
3. Integration dieses Simulators in den größeren Rahmen einer Modellier- und Experimentierumgebung.

Die dabei zurückgelegten Schritte werden im Folgenden kurz zusammengefaßt:

9.1.1 Das Repräsentationsschema

Das Repräsentationsschema ermöglicht eine Verhaltensmodellierung für Agenten in einer Multiagentensimulation. Die Beschreibung eines Modells ist in verschiedenen Dimensionen strukturierbar, die jeweils Struktur und Verhalten der Agenten sowie des Gesamtsystems betreffen:

Der Ausgangspunkt für eine einheitliche Beschreibung des internen Zustand eines Bausteins in einem Multiagentenmodells ist eine Repräsentation in Form von Zustandsvariablenvektoren. Diese Darstellungsform kann aus KI-Sicht im regelbasierten Kontext etwas

ungewöhnlich gefunden werden, bildet aber in der Simulationsmethodik ein gängiges Konzept. Jede Zustandsvariable wird in dem Rahmen explizit beschrieben, bevor sie in einer Einheit verwendet werden kann. Die dabei mögliche Charakterisierung der Zustandsvariablen umfaßt Kategorien, die für eine strukturierte Darstellung des internen Zustands verwendet werden können. Zu domänenabhängigen Kategorien gehören dabei Informationen, die für eine Beschreibung der zu erwartenden Dynamik oder für eine Typisierung der Variablen hilfreich sind.

Auch für Strukturen auf der Ebene der Architektur des Gesamtsystems wurden domänenunabhängige Ansatzpunkte identifiziert und zugänglich gemacht. Wichtig ist dabei vor allem eine Unterscheidung zwischen Agenten als den aktiven Bestandteilen, Ressourcen als passive Einheiten und Bezugssystemen zur Umweltrepräsentation. Gerade letztere zeichnen sich dadurch aus, daß die Interaktionen zwischen den Agenten und einer Darstellung des Raumes einen anderen Charakter als zwischen den Agenten besitzen.

Der Schwerpunkt des Repräsentationsschemas liegt allerdings in einem deklarativen Rahmen für die Darstellung des Agentenverhaltens. Ausgehend von einer regelbasierten Beschreibung des Eingabe-Ausgabe-Verhaltens einer aktiven Einheit werden verschiedene Möglichkeiten geschaffen, damit dieses so explizit und strukturiert dargestellt werden kann, daß ein Modellierer nicht nur bei der Formulierung des Modells, sondern auch bei der Analyse der Beschreibung unterstützt wird: Zu einer Unterteilung der verhaltensbestimmenden Regelmenge in Teilmengen, mit denen typisches Verhalten entsprechend einer Rolle charakterisiert werden kann, kommt die Einführung des Konzepts einer Aktivität. Damit wird ein Behälter für zusammengehörende Aktionen bezeichnet, die solange immer wieder ausgeführt werden, bis in einer Situation eine andere Aktivität ausgewählt werden muß. Aktivitäten kann man auf verschiedenen Abstraktionsebenen anlegen und über Regeln auswählen. Eine im weiterem Sinne ebenfalls regelbasierte Vorverarbeitung und Aktivitätsverfeinerung strukturieren das Verhaltensmodell eines Agenten weiter und erlauben eine einfache Beschreibung der zentralen Aktivitätsauswahl. Besonders wertvoll sind die sich aus dieser Strukturierung ergebenden Metaphern, wie “Aktivitätsautomat” oder “Skelettplan”, denn sie bieten einem Modellierer eine auch konzeptionelle Hilfe bei der Umsetzung seines Modells.

Für eine Verwendbarkeit dieses Repräsentationsschemas ist jedoch auch die Durchführbarkeit von Simulationsexperimenten entscheidend. Nur wenn eine hinreichend große Zahl an Einheiten mit entsprechendem Verhaltensrepertoire dargestellt werden kann, ist eine Modellier- und Experimentierumgebung für Fachexperten interessant. Durch Regelindizierung auf der Basis der Aktivitäten und Ausnutzung von Information über die wirklich aktiven Bestandteile (Agenten, dynamische Zustandsvariablen) war es möglich, einen effizienten Interpretationsmechanismus für diese deklarativ repräsentierte Beschreibung eines Agentensystems und seiner Umwelt zu realisieren. Für die Dauer eines Simulationsexperiments ist so vor allem entscheidend, ob der Modellierer diese Möglichkeiten zur Kategorisierung bzw. Strukturierung von Einheiten, Zustandsvariablen und Verhaltensbeschreibungen nutzt. Ein weiterer limitierender Faktor ist die Wahrnehmungsfähigkeit der Agenten. Auch hier entscheidet vor allem der Modellierer, z.B. wie groß er den Wahrnehmungsradius der Agenten wählt, usw. Auf der Basis des vorgegebenen Repräsentationsschema war es zudem möglich, weitere automatische Verbesserungen, wie beispielsweise ein Caching der einzelnen erfüllten Bedingungen, zu realisieren. Auf diese Weise wurden Experimente mit Modellen in relevanten Größenordungen möglich und die wichtigste Voraussetzung für den Einsatz des Repräsentationsschemas erfüllt.

9.1.2 Die Entwicklungs- und Experimentierumgebung

Ein deklaratives Schema für die Verhaltensbeschreibung in einem Multiagentenmodell wird besonders wertvoll, wenn es in den Kontext einer Entwicklungs- und Experimentierumgebung gestellt wird. Zum eigentlichen Simulator kommen dabei noch weitere Werkzeuge hinzu, die ein Modellieren und Experimentieren unterstützen können:

Eine graphische Modellierumgebung setzt auf dem expliziten Rahmen für die Verhaltensbeschreibung auf und ermöglicht durch die strukturierte Visualisierung der Beschreibung eine einfache Umsetzung und Analyse des Modells durch den Modellierer. Durch eine Erweiterung des Simulators zu einer Experimentierumgebung mit Möglichkeiten zur Instrumentierung des Modells kann ein Modellierer die von ihm ausgewählten systematischen Experimente durchführen und Auswerten.

Die Evaluation der Entwicklungs- und Experimentierumgebung erfolgte bisher in zahlreichen Anwendungen mit unterschiedlicher Komplexität. Insbesondere konnten dabei Fachexperten ohne Informatikausbildung (Biologen, Psychologen) Modelle in relevanten Grössenordnungen entwickeln und mit diesen Experimente durchgeführen.

Die Anwendungen der Entwicklungsumgebung mit den höchsten Anforderungen an die Modellieroberfläche waren die Diplomarbeiten von Emanuel Büttner [Büttner 1999] bzw. Patrick Huber [Huber 2000] (siehe auch Abschnitt 8.4). Dabei ging es unter anderem um die Bewertung der graphischen Modellier- und Experimentierumgebung – angepaßt an ein konkretes Szenario – in Hinblick auf die Frage, ob und wieweit damit Versuchspersonen innerhalb weniger Stunden und ohne Vorkenntnisse Agenten programmieren können. Eine besondere Schwierigkeit bei den Experimenten bestand darin, daß der Versuchsleiter selbst – also E. Büttner oder P. Huber – den Umgang mit SESAM erst lernen und so Erfahrung bei der Umsetzung beliebiger Strategien aufbauen mußten. Beide beurteilten den Einsatz von SESAM bei derart anspruchsvollen Experimenten als positiv.

Während frühe Anwender – ca. in den Jahren 1995-1998 – noch Unterstützung bei der Benutzung von SESAM benötigten, konnten durch kontinuierliche Verbesserungen spätere Anwender fast völlig ohne weitere Hilfe durch die Autorin selbst ihre Modelle umsetzen[1] Insgesamt kann man also aufgrund der durchgeführten Anwendungen zurecht behaupten, daß mit SESAM eine Entwicklungsumgebung für Multiagentensimulationen zur Verfügung steht, die für Fachexperten interessante Perspektiven bietet.

9.1.3 Bewertung im Kontext anderer Arbeiten

Der Vergleich mit anderen Rahmenwerken, Systemen oder Werkzeugen ist nicht einfach, da die Einordnung in Kategorien – die ja die Voraussetzung für einen Vergleich darstellt – nur schwer möglich ist. Es gibt viele Systeme, die für eine Umsetzung und Experimentierung mit einem Multiagentenmodell verwendbar wären, bzw. dafür entwickelt wurden (siehe dazu auch Abschnitt 4.4.2). Allerdings decken viele der dort umrissenen Systeme nur Teile dessen ab, was ein Modellierer mit einem Modell machen möchte; es ist also schwer, eine Agentenarchitektur wie *Soar* mit einer Simulationsumgebung wie *Swarm* zu vergleichen, obwohl beide häufig für die Entwicklung von Multiagentensimulationen verwendet werden. SESAM wurde dazu entwickelt, mehrere Phasen des Modellierprozesses – Umsetzung,

[1] Dies gilt in besonderer Weise, nachdem A. Hörnlein das Tutorial mit dem Aufbau eines Beispielmodells schrieb [Hörnlein 1999].

Validation und Auswertung des Modells – zu unterstützen und bietet dazu das gesamte "Programm" von einer Agentenarchitektur bis zur Animation und Auswertungsumgebung.

Deshalb soll an dieser Stelle ein Vergleich mit Werkzeugen und Rahmenwerken angestellt werden, die als besonders repräsentativ für ihre Kategorie gelten können oder besondere Ähnlichkeit zu dem hier vorgestellten Entwicklungssystem besitzen: Die auf der übernächsten Seite folgende Tabelle zeigt eine Zusammenfassung des direkten Vergleichs verschiedener Systeme mit SESAM. Die dabei verwendeten Kriterien benötigen noch eine kleine Erläuterung, insbesondere im Hinblick auf die Entwicklungs- und Experimentierwerkzeuge, die in dieser Tabelle nicht behandelt werden.

- Der **Hintergrund** des Simulationssystems liefert noch kein direktes Kriterium, das für den Vergleich relevant ist, allerdings trägt es viel zum Verständnis der Charakteristika des Systems bei.
- Eine **charakteristische Anwendung** zeigt, daß und für welche Art von Modellen das System bereits angewendet wurde. Bei *PECS-SIMPLEX II* wurde von der Anforderung bereits durchgeführter Simulationen zurückgetreten, da die bisher implementierten Beispiele (Räuber-Beute-Systeme u.ä.) das Potential der Agentenarchitektur nicht wiederspiegeln; stattdessen wurde die Zielanwendung aufgelistet.
- Die Repräsentationsform für das Verhalten der Agenten bzw. die Teilsysteme ist ein zentraler Punkt für SESAM, bei dem davon ausgegangen wird, daß ein gut strukturiertes Beschreibungsschema die passende Ausgangsbasis für eine effizient verwendbare Modellier- und Experimentierumgebung bietet. Im einzelnen kann man folgende Merkmale für den näheren Vergleich identifizieren.
  - Das **grundlegende Schema** erläutert die Form der Basisrepräsentation.
  - **Abstraktionsgrad der Beschreibung** zeigt die Ebene, auf welcher der Modellierer das Verhalten der Agenten beschreiben muß,
  - und **Abstraktionsgrad der Schnittstellen** bestimmt die Ebene der Beschreibung der Interaktionen zwischen den Agenten bzw. anderen Systemteilen.
  - **Deklarativität und Explizitheit** deuten auf das Ausmaß der Trennung zwischen Repräsentation und ihrer Simulation. Ist zur Durchführung eines Simulationsexperimentes ein Interpreter des Verhaltensmodells notwendig oder wird das Modell in einer direkt ausführbaren Form repräsentiert?
  - **Strukturierung** des Agentenmodells ist eines der Hauptziele der vorliegenden Arbeit. Welches Formen und welches Ausmaß an Strukturierung findet man in anderen Systemen? Dabei wird eine Unterteilung zwischen Strukturierung des Agentensystems und einer Struktur bei der Darstellung des Verhaltens der Bausteine gemacht.

  Einen Vergleich der hier verwendeten Agentenarchitektur mit anderen, insbesondere *PRS* oder *RAP*, findet man im Abschnitt 5.3.5.2 bei der abschließenden Darstellung des Repräsentationsschemas. Dadurch, daß die *Soar*-Architektur auch bei der Simulation von Multiagentenmodellen eine wichtige Rolle spielt, wird sie in der folgenden Tabelle behandelt, obwohl sie eigentlich der Kategorie einer Agentenarchitektur zugeordnet werden müsste.

- Die Integration eines Simulators in eine Modellierumgebung ist entscheidend für die Benutzbarkeit durch einen Fachexperten. Eine derartige **Entwicklungsumgebung** sollte folgende Werkzeuge bereitstellen, um einen Modellierer bei allen Aspekten seiner Arbeit mit dem Modell zu unterstützen. Genau an dieser Stelle weisen viele Systeme Defizite auf, die bei der Repräsentationsform gute Unterstützung leisten.

  - **Graphische Werkzeuge zur Formulierung des Verhaltens** der Modellbestandteile – d.h. existieren visuelle Mittel, um Regeln, Funktionen, also die Dynamik zu beschreiben.
  - **Graphische Werkzeuge zur Konfigurierung** des Modells sind dafür vorgesehen, beispielsweise Agenten auf einem Bezugssystem zu verteilen und eine Startkonfiguration des Modells zu manipulieren.
  - Werkzeuge zur **Animation** (Informationen dazu waren aus den vorhandenen Materialien zu SDML in [SDML 2000] nicht ersehbar).
  - Werkzeuge zur Instrumentierung des Modells, zum Datensammeln und zur **Auswertung** der Daten – bzw. zumindest zum Export der Daten.

Die Tabelle, die anhand dieser Kriterien verschiedene Systeme mit SESAM vergleicht, ist auf der nächsten Seite zu finden und wird im Anschluß näher erläutert.

Tabelle 9.1 Vergleich der wichtigsten Systeme zur Modellierung und Simulation von Multiagentenmodellen mit SESAM

Kriterium	*Swarm*	*PECS-Simplex II*	*SDML*	*Soar*	*AgentSheets*	SESAM
Hintergrund	Komplexe Systeme	Simulationstechnik	Theoretische Sozialwissenschaft	KI	Visuelles Programm.	KI
Aktuelle Referenz	*www.swarm.org*	[Urban 2000]	[SDML 2000]	[Soar 2000]	*www.agentsheets.com*	(diese Arbeit)
Charakteristische Anwendung	Emergente Phänomene	Fabrik mit Menschen	Gesellschaften kogn. Agenten	Modell für einzelne kogn. Agenten	Interaktive Spiele mit einfachem Agentenverhalten	Sim. komplexer Agenten in der Biologie
Repräsentation						
Grundlegendes Schema	Objektive C Klassenbibliothek	Simplex II Simulationssprache	Regeln mit "Datenbank"	Regeln mit Zustand, Operator Kontext-Stack	Graphische Java-Regeln	Regeln mit Aktivitäten und ZVariablen
Abstraktionsgrad der Beschreibung	Programmiersprache Skript-Erweiterung	Programmiersprache Skript-Erweiterung	rel. niedrig *Soar* wurde reimplementiert	hoch	visuelle Programmiersprache	hoch
Abstraktionsgrad der Schnittstellen	(Update-) Methoden	KQML-ähnliche Nachrichten	Faktenaustausch in Datenbanken	Tcl/Tk	Regelaktionen	Regelaktionen, einfache symb. Nachrichten
Deklarativität und Explizitheit	–	–	vollständig explizit	deklarative Architektur	–	deklarative Architektur
Strukturierung Agentensystem	Hierarchische "Swarms"	PECS-Architektur Umwelt, Hierarch. Modellieren	Hierarchische Regelmengen	–	Verfeinerung des diskreten Raums	Agenten, Ressourcen, Bezugssysteme
Strukturierung Verhalten	–	–	–	Problemräume	–	Aktivitäten, usw.
Entwicklungs-umgebung						
Graphische Verhaltens-modellierung	–	–	Rudimentär	Prototyp aus Klapphierarchien und Texteditoren	Vollständig graphisch	Vollständig graphisch
Graphische Konfigurierung	–	Modellkonfiguration Simplex II	Rudimentär		Vollständig graphisch	graphisch nach Beschreibung
Animation?	ja	ja	??		ja	ja
Auswertung?	Stärke des Systems	im System und Export	??		–	im System und Export

Die in den Spalten dieser Tabelle aufgeführten Systeme sind folgende, von denen jedes bei einem oder mehreren Kriterien direkt mit SESAM verglichen werden muß:

- *Swarm* kann in der wissenschaftlichen Gemeinschaft – gerade in biologischen Anwendungen – als das wohl am weitesten verbreitete System zur Programmierung von Multiagentensimulationen betrachtet werden. Es wurde ursprünglich am Santa Fe Institute entwickelt, mittlerweile wird seine Anwendung und neue Versionen von einer Firma vorangetrieben. Obwohl *Swarm* über keinerlei deklarative Verhaltensrepräsentation verfügt und die Strukturierung des Modells nur über hierarchisch zusammengesetzte "Swarm"-Objekte geschieht – welche die Agenten der Multiagentensimulation darstellen –, muß das Simulationssystem in dieser Aufstellung auftauchen. Die Stärke von *Swarm* liegt in seinen Möglichkeiten zur Instrumentierung des Modells. Datensammeln, Protokollfunktionen, Animationen sind für das wissenschaftliche Arbeiten mit einem Modell unerläßlich und genau diese Möglichkeiten machen *Swarm* für Fachexperten mit Programmierkenntnissen sehr attraktiv. Dennoch ist für das Gros der potentiellen Anwender eine Umsetzung von Multiagentensimulationen in Objective C nicht praktikabel. An dieser Stelle ist SESAM klar im Vorteil, sowohl durch das höhere Abstraktionsniveau der deklarativen Repräsentation, als auch durch die darauf aufbauende graphische Modellieroberfläche.

- *PECS-Simplex II* baut auf der Standard-Simulationssprache *Simplex II* auf, die an der Universität Passau entwickelt wurde. Wie viele derartige Systeme stellt dieses Werkzeuge zur Modellverwaltung, Animation, Datenerhebung und -auswertung zur Verfügung. Ein graphisches Editieren der Modellkonfiguration ist möglich. Im Gegensatz zu *Swarm* wird bei der Strukturierung des Agentensystems vor allem die Ebene der Agenten selbst betrachtet. Für diese wird eine spezielle Struktur zur Anordnung der internen Zustandsvariablen vorgeschlagen, die sich an psychologischen Modellen orientiert. Auch SESAM nutzt eine Kategorisierung des Beschreibung der Bausteine des internen Zustands, zwar auf einem rudimentäreren Niveau, das allerdings völlig domänenunabhängig ist. Zudem verfügt *PECS-Simplex II* über abstraktere Vorgaben bei den verschickbaren Nachrichten. Allerdings muß das Verhalten der Agenten und die Dynamik der Zustandsvariablen textuell unter Benutzung einer Programmiersprache – bzw. einer Skriptsprache für eine etwas abstraktere Beschreibung der Aktionen eines Agenten – realisiert werden. Durch seine regel-basierte Ausgangsbasis mit den beschriebenen Strukturierungmöglichkeiten kann SESAM die Darstellung von Verhalten besser unterstützen. Dies gilt insbesondere wegen der deklarativen Repräsentation und den graphischen Spezifikationswerkzeugen.

- *SDML* wurde am Center for Policy Modelling der Universität Manchester entwickelt und hat mehr den Charakter eines Multiagenten-Produktionssystem bei dem Agenten hierarchisch zusammengefaßt werden können, als den einer höheren Simulationsumgebung. *SDML* ist vollständig deklarativ und explizit, d.h. die Bedeutung eines jeden Symbols wird im System definiert – es gibt keine Atome, deren Semantik auf einer anderen programmiersprachlichen Ebene definiert ist – wie dies z.B. die Primitive in SESAM sind. Ebenso wie in dieser regel-basierten Programmiersprache *Soar* implementiert werden konnte, dürfte es auch möglich sein, die Schemata zur Verhaltensrepräsentation von SESAM und ihre Interpretation in *SDML* zu realisieren.

- *Soar* bietet "nur" eine Agentenarchitektur. Diese kann über Schnittstellen mit ebenfalls durch *Soar* gesteuerte Agenten und ihre Umwelt kommunizieren. Die Strukturierung der Verhaltensbeschreibung geschieht über Problemräume, die nächste Aktion wird auf der Basis von Ziel-Repräsentation in Form von Zuständen rückwärtsverkettet hergeleitet, wenn keine direkt verwendbare Information vorhanden ist, bzw. vorwärtsverkettet verknüft, wenn dies möglich ist. Die *Soar*-Architektur besitzt ein weites Potential zur Repräsentation von Agentenverhalten und ist deutlich flexibler als die Agentenarchitektur von SESAM. *Soar* bietet eine einheitlichen Repräsentationsform für unterschiedliche Aspekte und Phasen des Reasoning eines Agenten. Allerdings kann eine Beschreibung von Verhalten mit SESAM für Domänenexperten ohne Informatikausbildung einfacher sein, da eine klare Trennung in Vorverarbeitung, Auswahl und Ausführung weniger Abstraktion erfordert. Eine graphische Modellieroberfläche *"VisualSoar"* existiert als ein Prototyp.

- *AgentSheets* ist insgesamt wohl das Entwicklungssystem, das SESAM am ähnlichsten ist. Es bietet eine durchgängig visuelle Programmiersprache für das Verhalten der Agenten. Dabei spielen besondere Primitive zur Interaktion (graphische Effekte, usw.) mit dem Benutzer eine wichtige Rolle. Wegen der ursprünglichen Ausrichtung auf die Konstruktion von intelligenten graphischen Benutzeroberflächen durch die Anwender selbst – die erst später auf interaktive Spiele erweitert wurde – besitzt *AgentSheets* einige Merkmale, die es zwar für die Implementierung von interaktiven Simulationsspielen gut geeignet macht, die sich allerdings beim wissenschaftlichen Arbeiten mit Multiagentenmodellen als nachteilig erweisen.

  – Das Verhalten der Agenten wird durch eine unstrukturierte Regelmenge formuliert. Diese ist zwar visuell behandelbar, aber durch ihre Unstrukturiertheit ist ein komplexeres Verhaltensmodell für einen Agenten nicht praktikabel.
  – Das Modell der Agenten ist sehr "raum-orientiert". Eine Strukturierung des Agentensystems geschieht immer über eine räumliche Zusammenfassung in einem "Sheet", also einem Blatt mit diskreten Felden.
  – Durch den Fokus auf interaktive Simulationsspiele wurde die Bereitstellung von Werkzeugen zur Auswertung von Experimenten vernachläßigt. Gerade beim Arbeiten mit komplexeren Multiagentensimulationen ist aber das Sammeln und Abstrahieren von Experimentdaten ein wichtiger Bestandteil der Arbeiten mit dem Modell.

  AgentSheets lebt von seiner visuellen Programmiersprache zur Konfiguration des Agentensystems und Beschreibung des Agentenverhaltens. Es ist keinerlei textuelle Anpassung notwendig – wie dies bei SESAM durch die Beschreibungsobjekte für die Primitive und domänenspezifischen Klassen der Fall ist.

9.2 Fazit

Am Ende dieser Zusammenfassung muß man also das Fazit ziehen, daß SESAM auf der Basis seines deklarativen und strukturierten Schemas einen wirklichen Fortschritt für die Modellbildung und Simulation darstellt. Bisher war kein Simulationssystem für Multiagentenmodelle verfügbar, das einen Modellierer auf so umfassende Weise unterstützt. Für die

technischen Probleme, auf die ein Benutzer bei der Realisierung, Validierung und Auswertung seiner Multiagentensimulation stoßen kann, wird jeweils eine Lösung bereitgestellt. In Anwendungen aus verschiedenen Domänen wurde gezeigt, daß die Modellier- und Experimentierumgebung von Fachexperten wirklich selbständig benutzt werden kann. Die dabei möglichen Modelle und Experimente gehen deutlich über kleine “Spielzeug”-Szenarien hinaus, sie liefern für Fachexperten relevante Ergebnisse.

Also hat sich das Konzept, auf einer regelbasierten Verhaltensdarstellung eine mehrdimensionale Strukturierung aufzusetzen, um ein Verhaltensmodell explizit zu repräsentieren, als tragfähig erwiesen. Diese deklarative Form der Modellrepräsentation für eine Multiagentensimulation bildet eine passende Basis für graphische Modellier- und Experimentierwerkzeuge. Als nächster Schritt für die Verbreitung von Modellbildung und Simulation als Werkzeug für Fachexperten wäre eine forcierte Anwendung von SESAM in weiteren Szenarien, für erweiterte und andere Fragestellungen, in Forschungsprojekten und in der Lehre nötig.

Die technischen Aspekte der Unterstützung von Fachexperten bei Modellbildung und Simulation bekommt man demnach gut in den Griff. Allerdings mußten wir die Erfahrung machen, daß mit der Erstellung einer Multiagentensimulation doch Probleme verbunden sind, die durch die Bereitstellung einer graphischen Modellierumgebung, sowie Werkzeugen zur Beschreibung, Beobachtung und Auswertung von Experimenten nicht beseitigt werden können. Diese Probleme, gerade bei der Modellkalibrierung, sind eher methodischer Natur, es gibt aber vielversprechende Ansätze, deren Lösung zu unterstützen.

9.3 Ausblick: Unterstützung der Kalibrierung und Optimierung

Wie in Abschnitt 4.2.3 behandelt, zeigen Erfahrungen mit der Multiagentensimulation, daß diese Modelle sehr sensibel gegenüber Parameteränderungen auf der Agentenebene reagieren. Das bedeutet, wenn man das Verhalten einer Kategorie von Individuen ändert, besitzt dies Effekte an vielen Stellen. Aber gerade die Vielzahl der Agenten, die dieses modifizierte Verhalten ausführen oder nur damit interagieren, beeinflußt das Gesamtsystem oft überraschend stark. Das macht die Kalibrierung eines Modells, oder die Optimierung einer Simulation diffizil. Gerade im Hinblick auf die eher aufwendigen Simulationsexperimente ist eine Unterstützung unerläßlich.

Man kann diesen Prozess der Kalibrierung auf zwei verschiedene Arten unterstützen:

1. Verwendung von Wissen über Zusammenhänge im Modell, um Fehlläufe schneller zu erkennen oder Hinweise auf Parameteränderungen zu gewinnen.

2. Automatische Parameteroptimierung durch lernende Agenten

Diese Ansätze sollen zum Abschluß dieser Arbeit näher erläutert werden:

9.3.1 Verwendung zusätzlicher Formen von Wissen

Häufig besitzt ein Fachexperte Wissen über das zu modellierende System, das nicht direkt in der Verhaltensbeschreibung verwendet wird: Das können Invarianten sein, die Zusammenhänge im Verhaltensmodell beschreiben – wie z.B. der Wert einer Zustandsvariable muß immer kleiner als ein anderer sein, oder ein bestimmter Wert verändert sich zeitversetzt zu einem zweiten. Dieses Wissen verwendet der Modellierer mehr oder weniger explizit, wenn

er das Modell validiert, also dessen Nähe zum Originalsystem beurteilt. Stellt man dem Modellierer Möglichkeiten zur Formalisierung dieses Wissens zur Verfügung, kann er diese Invarianten, Constraints, usw. dazu benutzen, die Beobachtung von Simulationsläufen und gegebenenfalls deren Abbruch zu automatisieren. Dieses Wissen kann man auch dazu benutzen, die Identifikation kritischer Parameter zu unterstützen. In bestimmten Anwendungen ist vollstellbar, daß dieses Wissen sogar ausreichen kann, um den Wertebereich dieser Parameter einzuschränken oder bei bestimmten Beobachtungen eine Modifikation automatisch vornehmen zu können.

Interessant ist für letzteres explizites Meta-Wissen über das Modell. Experten für Organisationsdesign oder -management verfügen über Kenntnisse darüber, welche lokale Änderung in einem "natürlichen" Multiagentensystem zu welchen Effekten auf der Ebene des Gesamtsystems führen. In den Wirtschaftswissenschaften wird derartiges Wissen im Studium teilweise sogar gelehrt. Kann man solches Wissen über ein Multiagentensystem und somit über ein dieses abbildende Multiagentenmodell explizit formalisieren, kann der Prozess der Kalibrierung eines Modells gemäß dem gekannten Gesamtverhalten durch Werkzeuge unterstützt werden, die damit verbundene Probleme auflösen.

9.3.2 Automatische Parameterkalibrierung

Wie oben angedeutet, kann man die Kalibrierung eines Multiagentenmodells auch als Optimierungsproblem betrachten – man minimiert die Distanz zwischen dem Verhalten des Originalsystems und dem des Modells. Kann das Zielverhalten explizit gemacht werden, kann man ein automatisches Optimierungsverfahren auf das Verhaltensmodell anwenden. Bei diesem Ansatz zeigt sich wieder, daß eine deklarative Repräsentation besonders hilfreich sein kann, da so die Parameter des Verhaltensmodells explizit dargestellt werden. Allerdings wird man die Menge der zu betrachtenden Parameter einschränken müssen.

Diese Optimierung kann durch ein externes Modul gesteuert werden, das einen modifizierten Parametersatz an den Simulator weitergibt, der durch ein Simulationsexperiment eine "Ansteuerung" des Modells bewertet. Das ist ein übliches Vorgehen in kombinierten simulationsbasierten Optimierungssystemen, wie z.B. in [Schneider & Schwarz 2000]. Allerdings ist eine derartige Vorgehensweise wegen des hohen Aufwands für ein Simulationsexperiment nicht immer tragbar. Anstatt nun oben beschriebenes Wissen zu benutzen, um Simulationsläufe zu verkürzen oder Heuristiken zur Parameteranpassung zu formulieren, kann man Agenten ihr Verhaltensmodell selbständig anpassen, also lernen lassen. In biologischen Anwendungsszenarien ist ein derartiges Vorgehen besonders reizvoll. Der Modellierer kann bestimmte Umweltcharakteristika vorgegeben und das Agentensystem adaptiert sein Verhaltensmodell beispielsweise mittels Evolution. Dadurch, daß das Verhaltensmodell deklarativ repräsentiert wird, sind die dazu notwendigen Ansatzpunkte relativ klar identifizierbar. Durch Evolutions- und Selektionsexperimente lassen sich nicht nur Verhaltensmodelle von Agenten an eine vorgegebene Umwelt zwecks Kalibrierung behandeln, sondern auch Hypothesen zur Entstehung von Verhaltensweisen oder Organisationsformen können auf eine innovative Art und Weise untersucht werden. Erste Ergebnisse zu diesen Fragestellungen findet man in [Oechslein et al. 1999, Hörnlein et al. 2000].

Dieser kurze Ausblick zeigt nicht nur, daß für eine umfassende Unterstützung der Entwicklung, Validation und Auswertung von Multiagentenmodellen die Ausführung weiterer Ansätze notwendig ist. Gerade die deklarative, strukturierte Modellbeschreibung bietet da-

zu eine wichtige Voraussetzung, um Verfahren zur Parameteradaptierung für die Kalibrierung und Optimierung von Agentenmodellen verwenden zu können. Auf diese Weise sind innovative Weiterentwicklungen möglich, die nicht nur das Skalieren der Multiagentensimulationen durch unterstützende Werkzeuge verbessern, sondern auch neue Experimentiermöglichkeiten bieten und so die Methodik der Multiagentensimulation für Fachexperten noch attraktiver machen.

Literaturverzeichnis

[AAII 1996] AAII: *dMARS Technical Overview*. http://www.aaii.oz.au/proj/dMARS-prod-brief.html, 1996.

[Agha & Jamali 1999] G. A. Agha and N. Jamali: *Concurrent Programming for DAI*. In G. Weiss (ed.): *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, chapter 12, pages 505–537. MIT Press, 1999.

[Agre 1989] P. E. Agre: *The Dynamic Structure of Every Day Life*. PhD thesis, MIT Artificial Intelligence Laboratory, 1989.

[Agre 1997] P. E. Agre: *Computation and Human Experience*. Cambridge University Press, Cambridge, 1997.

[Agre & Chapman 1987] P. E. Agre and D. Chapman: *Pengi: An Implementation of a Theory of Activity*. In *Proceedings of the AAAI-87*, pages 268–272, 1987.

[Anderson 1993] J. R. Anderson: *Rules of the Mind*. Lawrence Erlbaum Publishers, Hillsdale, 1993.

[Anderson & Evans 1995] J. Anderson and M. Evans: *A Generic Simulation System for Intelligent Agent Designs*. Applied Artificial Intelligence, 9:535–560, 1995.

[Atkin et al. 1998] M. S. Atkin, D. L. Westbrook, P. R. Cohen, and G. D. Jorstad: *AFS and HAC: Domain-General Agent Simulation and Control*. In *Proc. of the AAAI-98 Workshop on Software Tools for Agent Development*. http://www.cs.bham.uk/bst/aaai-98/, 1998.

[Axelrod 1995] R. Axelrod: *A model of the emergence of new political actors*. In N. Gilbert and R. Conte (eds.): *Artificial Societies - the computer simulation of social life*, pages 19–39. UCL Press, 1995.

[Axelrod 1997a] R. Axelrod: *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princton University Press, Princton, 1997.

[Axelrod 1997b] R. Axelrod: *The Dissemination of Culture: A Model with Local Convergence and Global Polarization*. Journal of Conflict Resolution, 41:203–226, 1997.

[Axtell et al. 1996] R. Axtell, R. Axelrod, J. M. Epstein, and M. D. Cohen: *Aligning Simulation Models: A Case Study and Results*. Computational and Mathematical Organization Theory, 1:123–141, 1996.

[Bagnoli 1998] F. Bagnoli: *Cellular Automata*. Dynamical Modelling in Biotechnologies, 1998.

[Balci 1994] O. Balci: *Validation, Verification and Testing Techniques Throughout the Life Cycle of a Simulation Study*. Annals of the Operations Research Society, 53:121–173, 1994.

[Balci 1997] O. Balci: *Principles of Simulation Model Validation, Verification and Testing*. Transactions of the Society for Computer Simulation, 14:3–12, 1997.

[Balmann 1997] A. Balmann: *Farm Based Modelling of Regional Structural Change: A Cellular Automata Approach*. European Review of Agricultural Economics, 24:85–108, 1997.

[Bamberger 1999] S. Bamberger: *Verteiltes Problemlösen mit wissensbasierten Diagnosesystemen*, Band 203 der Reihe *Diski*. infix, 1999.

[Bamberger et al. 1997] S. Bamberger, U. Gappa, F. Klügl und F. Puppe: *Komplexitätsreduktion durch grafische Wissensabstraktion*. In: *XPS'97 - Proc. der 4. Deutschen Tagung Expertensysteme*, Seiten 61–78. infix, 1997.

[Barbuceanu & Fox 1997] M. Barbuceanu and M. Fox: *The Design of a Coordination Language for Multi-Agent-Systems*. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings (eds.): *Intelligent Agents III (= Proceedings of ATAL'96)*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 341–355. Springer, 1997.

[Barr & Feigenbaum 1989] A. Barr and E. A. Feigenbaum: *The Handbook of Artificial Intelligence - Volume I*. Addison Wesley, 2. edition, 1989.

[Bates et al. 1994] J. Bates, A. B. Loyall, and W. S. Reilly: *An Architecture for Action, Emotion and Social Behavior*. In C. Chastelfranchi and E. Werner (eds.): *Artificial Social Systems - 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'92*, pages 55–68. Springer, July 1994.

[Baumgarten 1990] B. Baumgarten: *Petri-Netze - Grundlagen und Anwendungen*. BI-Wissenschaftsverlag, Mannheim, 1990.

[Bazzan et al. 1999] A. Bazzan, J. Wahle, and F. Klügl: *Agents in Traffic Modelling -from Reactive to Social Behaviour*. In W. Burgard, T. Christaller, and A. Cremers (eds.): *KI-99: Advances in Artificial Intelligence (LNAI 1701)*, pages 303–307. Springer, 1999.

[Bazzan et al. 2000] A. Bazzan, J. Wahle, F. Klügl, and M. Schreckenberg: *Anticipatory Traffic Forecast Using Multi-Agent Techniques*. In D. Helbing, H. Hermann, M. Schreckenberg, and D. Wolf (eds.): *Traffic and Granular Flow'99*. Springer, 2000.

[Beer 1995] R. D. Beer: *A dynamical systems perspective on agent-environment interaction*. Artificial Intelligence, 72:173–215, 1995.

[Belli & Hain 1998] V. Belli und R. Hain: *Verteilte Simulation mit SeSAm - Praktikumsbericht, Wintersemester 1997-1998*. unpublished, 1998.

[Bernardinello & Cindio 1992] L. Bernardinello and F. D. Cindio: *A Survey of Basic Net Models and Modular Net Classes*. In G. Rozenberg (ed.): *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 304–351, Springer, Berlin, 1992.

[Blumberg 1994] B. Blumberg: *Action Selection in Hamsterdam: Lessons from Ethology*. In J.-A. Meyer and S. Wilson (eds.): *From Animals to Animats 3: 3d Conference on the Simulation of Adaptive Behavior*, pages 108–117. MIT Press, 1994.

[Bond & Gasser 1988a] A. Bond and L. Gasser: *An Analysis of Problems and Research in Distributed Artificial Intelligence*. In A. Bond and L. Gasser (eds.): *Readings in Distributed Artificial Intelligence*, chapter 1, pages 3–35. Morgen Kaufmann, 1988.

[Bond & Gasser 1988b] A. Bond and L. Gasser (eds.): *Readings in Distributed Artificial Intelligence*. Morgen Kaufmann, 1988.

[Bossel 1994] H. Bossel: *Modellbildung und Simulation. Konzepte, Verfahren und Modelle zum Verhalten dynamischer Systeme*. Vieweg Verlag, Braunschweig, 2. Auflage, 1994.

[Bouron et al. 1991] T. Bouron, J. Ferber, and F. Samuel: *MAGES: A Multi-Agent Testbed for Heterogenous Agents*. In Y. Demazeau and J.-P. Mueller (eds.): *Decentralized A.I. 2*, pages 195–214. North Holland, 1991.

[Bradshaw et al. 1997] J. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley: *KAoS: Toward an industrial strength open agent architecture*. In J. Bradshaw (ed.): *Software Agents*, chapter 17, pages 371–418. MIT Press, 1997.

[Braitenberg 1984] V. Braitenberg: *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambrigde, 1984.

[Brazier et al. 1995] F. Brazier, B. Dunin-Keplicz, N. Jennings, and J. Treur: *Formal Specification of Multi-Agent Systems: A Real World Case*. In V. Lesser (ed.): *Proceedings of the ICMAS-95 (= First International Conference on Multi-Agent Systems)*, pages 25–32. AAAI Press, 1995.

[Brooks 1986] R. A. Brooks: *A Robust Layered Control System For a Mobile Robot*. IEEE Journal of Robotics and Automation, 2:14–23, 1986.

[Brooks 1991a] R. A. Brooks: *Intelligence without Reason*. In *Proceedings of 12th Int. Joint Conf. on Artificial Intelligence, Sydney, Australia, August 1991*, pages 565–595, 1991.

[Brooks 1991b] R. A. Brooks: *Intelligence without Representation*. Artificial Intelligence, 47:139–159, 1991.

[Bruderer & Maiers 1997] E. Bruderer and M. Maiers: *From the Margin to the Mainstream: An Agenda for Computer Simulation in the Social Sciences*. In R. Conte, R. Hegselmann, and P. Terna (eds.): *Simulating Social Phenomena*, volume 456 of *Lecture Notes in Economics and Mathematical Systems*, pages 89–95, Springer, Berlin, 1997.

[Buerckert & Müller 1991] H.-J. Buerckert and J. Müller: *RATMAN: Rational Agents Testbed for Multi Agent Networks*. In Y. Demazeau and J.-P. Mueller (eds.): *Decentralized A.I. 2*, pages 217–230. North Holland, 1991.

[Burmeister 1996] B. Burmeister: *Models and Methodology for Agent-Oriented Analysis and Design*. In K. Fischer (ed.): *Proceedings of the KI-96 Workshop Agentenorientertes Programmieren und Verteilte Systeme*. DFKI-Dokument D-96-06, 1996.

[Burnett et al. 1995] M. M. Burnett, M. J. Baker, C. Bohus, P. Carlson, S. Yang, and P. van Zee: *Scaling Up Visual Languages*. IEEE Computer, 28(3):45–54, 1995.

[Büttner 1999] E. Büttner: *Kann man Strategien programmieren?*. Diplomarbeit, Institut für Psychologie, Universität Bamberg, Dez. 1999.

[Cariani 1991] P. Cariani: *Emergence and Artificial Life*. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*, volume 10 of *SFI Studies in the Sciences of Complexity*, pages 775–796. Addison Wesley, 1991.

[Carley & Newell 1994] K. Carley and A. Newell: *The Nature of the Social Agent*. Journal of Mathematical Sociology, 19:221–262, 1994.

[Castelfranchi 1998] C. Castelfranchi: *Modelling Social Action for AI Agents*. Artificial Intelligence, 103:157–182, 1998.

[Castelfranchi & Conte 1996] C. Castelfranchi and R. Conte: *Distributed Artificial Intelligence and Social Science: Critical Issues*. In G. M. P. O'Hare and N. R. Jennings (eds.): *Foundations of Distributed Artificial Intelligence*, chapter 20, pages 527–542. John Wiley and Sons, 1996.

[Casti 1997] J. L. Casti: *Reality Rules: I - Picturing the World in Mathematics – The Fundamentals*. Wiley Interscience, New York, 1997.

[CEU 2000] CEU: *MAML Homepage*. http://www.syslab.ceu.hu/maml/maml.html, 2000.

[Chauhan 1997] C. Chauhan: *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*. PhD thesis, University of Cincinnati, 1997.

[Coderre 1988] B. Coderre: *Modelling Behavior in Petworld*. In C. Langton (ed.): *Artificial Life*, SFI Studies in Complexity, pages 407–420. Addison Wesley Publishing Company, 1988.

[Cohen et al. 1989] P. R. Cohen, M. L. Greenberg, D. Hart, and A. Howe: *TRIAL BY FIRE. Understanding the Design Requirements for Agents in a Complex Environment*. AI Magazine, 10(3):32–48, 1989.

[Cohen & Levesque 1995] P. R. Cohen and H. J. Levesque: *Communicative Actions for Artificial Agents*. In V. Lesser (ed.): *Proceedings of the first International Conference on Multi-Agent Systems*, pages 65–72, MIT Press, Cambrigde, 1995.

[Collins & Jefferson 1991a] R. J. Collins and D. R. Jefferson: *AntFarm: Towards Simulated Evolution*. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*, pages 579–601. Addison-Wesley, 1991.

[Collins & Jefferson 1991b] R. J. Collins and D. R. Jefferson: *Representations for Artificial Organisms*. In J.-A. Meyer and S. W. Wilson (eds.): *From Animals to Animats: 1st Conference on the Simulation of Adative Behaviour*, pages 382–390. MIT Press, 1991.

[Cost et al. 1999] R. S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng, and I. Soboroff: *Agent Development with Jackal*. In *Proceedings of the 3rd Internation Conference on Autonomous Agents,1999*. ACM Press, 1999.

[Cypher 1998] A. Cypher: *Cocoa Homepage*. http://www.dnai.com/cypher/Cocoa/, 1998.

[Darley 1994] V. Darley: *Emergent Phenomena and Complexity*. In R. A. Brooks and P. Maes (eds.): *Artificial Life IV*, pages 411–416. MIT-Press, 1994.

[Dautenhahn 1995] K. Dautenhahn: *Biologically inspired robotic experiments on interaction and dynamic agent-environment couplings*. In: K. Dautenhahn und others (Hrsg.): *Proc. des 1. Workshops on Artificial Life (= GMD Studien Nr. 271)*, Seiten 195–200. GMD, 1995.

[Dautenhahn 1997] K. Dautenhahn: *Soziale Intelligenz für Roboter*. In H.-M. Gross (ed.): *Proceedings Workshop SOAVE'97, Selbstorganization von Adaptivem Verhalten, Ilmenau, 23.-24. September 1997 (= Fortschrittberichte VDI, Reihe 8, Nr. 663)*, pages 14–24, 1997.

[DeAngelis & Gross 1992] D. L. DeAngelis and L. J. Gross: *Individual Based Models and Approaches in Ecology*. Chapman and Hall, 1992.

[DECAF 2000] DECAF: *DECAF Agent Framework*. http://www.eecis.udel.edu/decaf/, 2000.

[Decker 1996] K. Decker: *Distributed Artificial Intelligence Testbeds*. In G. M. P. O'Hare and N. R. Jennings (eds.): *Foundations of Distributed Artificial Intelligence*, chapter 3, pages 119–138. John Wiley and Sons, 1996.

[Deneubourg et al. 1991] J.-L. Deneubourg, G. Theraulaz, and R. Becker: *Swarm-Made Architectures*. In P. Bourgine and F. Varela (eds.): *Towards a Practice of Autonomous Systems, Proceedings of the first European Conference on Artificial Life*, pages 123–133, MIT Press, Cambridge, 1991.

[DFKI 2000] DFKI: *Social Interaction Framework SIF*. http://www.dfki.de/sif/, 2000.

[Dignum & Linder 1997] F. Dignum and B. van Linder: *Modelling Social Agents: Communication as Action*. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings (eds.): *Intelligent Agents III (= Proceedings of ATAL'96)*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 205–218. Springer, 1997.

[Doran 1996] J. Doran: *Simulating Societies using Distributed Artificial Intelligence*. In K. G. Troitzsch, U. Mueller, G. N. Gilbert, and J. Doran (eds.): *Social Science Microsimulation*, chapter 17, pages 381–393. Springer, 1996.

[Doran & Palmer 1995] J. Doran and M. Palmer: *The EOS-Project: Integrating Two Models of Palaeolithic Social Change*. In N. Gilbert and R. Conte (eds.): *Artificial Societies - the computer simulation of social life*, pages 103–125. UCL Press, 1995.

[Doran et al. 1994] J. Doran, M. Palmer, N. Gilbert, and P. Mellars: *The EOS-Project: Modelling Upper Palaeolithic social change*. In J. Doran and N. Gilbert (eds.): *Simulating Societies: the Computer Simulation of Social Phenomenon*, pages 195–222. UCL Press, 1994.

[Dörner 1999] D. Dörner: *Bauplan für eine Seele*. Rowolt, 1999.

[Dörner & Schaub 1992] D. Dörner und H. Schaub: *Spiel und Wirklichkeit: über die Verwendung und den Nutzen computersimulierter Planspiele*. Kölner Zeitschrift für Wirtschaft und Pädagogik, 12:55–78, 1992.

[Dornhaus et al. 1998] A. Dornhaus, F. Klügl, F. Puppe, and J. Tautz: *Task Selection in Honey Bees - Experiments Using Multi-Agent Simulation*. In C. Wilke, S. Altmeyer, and T. Martinetz (eds.): *Proc. of the Third German Workshop on Artificial Life, GWAL-98*, pages 171–184. Verlag Harri Deutsch AG, 1998.

[Drogoul & Ferber 1994] A. Drogoul and J. Ferber: *Multi-Agent Simulation as a Tool for Modelling Societies: Application to Social Differentiation in Ant Colonies.*. In C. Chastelfranchi and E. Werner (eds.): *Artificial Social Systems - 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'92*, pages 3–23. Springer, July 1994.

[Drogoul et al. 1991] A. Drogoul, J. Ferber, B. Corbara, and D. Fresneau: *A Behavioural Simulation Model for the Study of Emergent Social Structures*. In P. Bourgine and F. Varela (eds.): *Towards a Practice of Autonomous Systems, Proceedings of the first European Conference on Artificial Life*, pages 161–170, MIT Press, Cambridge, 1991.

[Dryer 1999] D. C. Dryer: *Getting Personal with Computers: How to Design Personalities for Agents*. Applied Artificial Intelligence, 13(3):273–295, 1999.

[Dunin-Keplicz & Treur 1995] B. Dunin-Keplicz and J. Treur: *Compositional Formal Specification of Multi-Agent Systems*. In M. Wooldridge and N. R. Jennings (eds.): *Intelligent Agents: Proceedings of the ATAL'94*, volume 890 of *LNAI*, pages 102–117. Springer, 1995.

[Durfee & Montgomery 1989] E. H. Durfee and T. A. Montgomery: *MICE: A Flexible Testbed for Intelligent Coordination Experiments*. In *Proc. of the 9th Int. Workshop on Distributed Artificial Intelligence, Rosario, WA*, 1989.

[Durfee & Rosenschein 1994] E. H. Durfee and J. S. Rosenschein: *Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples*. In *Proc. of the 13th Int. Workshop on Distributed Artificial Intelligence*, pages 94–104, 1994.

[ECOSIM 1999] ECOSIM: *ECOTOOLS: ECOSIM*. http://www.office.uni-oldenburg.de/projekte/ecotools/, 1999.

[Eggelbusch 2000] K. Eggelbusch: *Experimentelle Beiträge und Modellbildung zum Ausbreitungsverhalten des Honigbienenschädlings Varroa jacobsoni*. Diplomarbeit, Institut für Zoologie, Universiät Mainz, April 2000.

[Eggelbusch et al. 2000] K. Eggelbusch, S. Fuchs und J. Tautz: *Agentenbasierte Simulation zum Ausbreitungsverhalten des Honigbienenschädlings Varroa jacobsoni Oudemans*. In: *Proc. des Workshop "Multiagentensysteme und Individuenbasierte Simulation", März 2000, Würzburg*, Seiten 73–82. Institut für Informatik, Bericht Nr. 253, 2000.

[Epstein & Axtrell 1996] J. M. Epstein and R. Axtrell: *Growing Artificial Societies. Social Science from the Bottom Up*. MIT Press, Cambridge, MA, 1996.

[Ermentrout & Edelstein-Keshet 1993] G. B. Ermentrout and L. Edelstein-Keshet: *Cellular Automata Approaches to Biological Modeling*. Journal of Theoretical Biology, 160:97–133, 1993.

[Esser et al. 1998] J. Esser, L. Neubert, J. Wahle, and M. Schreckenberg: *Traffic Simulation using Cellular Automata*. In H. K. Büning (ed.): *Proc. of the Workshop Simulation and Knowledge-based Systems 1998*, 1998.

[Fenton & Beck 1989] J. Fenton and K. Beck: *Playground: An Objekt Oriented Simulation System with Agent Rules for Children of All Ages*. In *Proc. of the OOPSLA'89*, 1989.

[Ferber 1999] J. Ferber: *Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence*. Addison Wesley, 1999.

[Ferber & Jacopin 1991] J. Ferber and E. Jacopin: *The Framework of ECO-Problem Solving*. In Y. Demazeau and J.-P. Mueller (eds.): *Decentralized A.I. 2*, pages 181–193. North Holland, 1991.

[Ferguson 1995] I. A. Ferguson: *Integrated Control and Coordinated Behavior: a Case for Agent Models*. In M. J. Wooldridge and N. R. Jennings (eds.): *Intelligent Agents. ECAI-94 Workshop on Agent Theories, Architectures and Languages ATAL-94*, volume 890 of *LNAI*, pages 203–218. Springer, August 1995. Amsterdam.

[Finin et al. 1997] T. Finin, Y. Labrou, and J. Mayfield: *KQML as an Agent Communication Language*. In J. Bradshaw (ed.): *Software Agents*, chapter 14, pages 291–316. MIT Press, 1997.

[Firby 1989] J. Firby: *Adaptive Execution in Complex, Dynamic Domains*. PhD thesis, Yale, 1989.

[Fischer 1993] K. Fischer: *The Rule-Based Multi-Agent System MAGSY*. In *Proc. of the CKBS'92 DAKE-Center - Keele University*, 1993.

[Fischer et al. 1995] K. Fischer, J. Mueller, M. Pischel, and D. Schier: *A Model for Cooperative Transportation Scheduling*. In V. Lesser (ed.): *Proceedings of the ICMAS-95 (= First International Conference on Multi-Agent Systems)*, pages 109–116. AAAI Press, 1995.

[Fisher 1995] M. Fisher: *Representing and Executing Agent-Based Systems*. In M. Wooldridge and N. R. Jennings (eds.): *Intelligent Agents: Proceedings of the ATAL'94*, volume 890 of *LNAI*, pages 307–323. Springer, 1995.

[Fisher 1997a] M. Fisher: *If Z Is the Answer, What Could the Question Possibly Be?*. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings (eds.): *Intelligent Agents III (= Proceedings of ATAL'96)*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 65–66. Springer, 1997.

[Fisher 1997b] M. Fisher: *Implementing BDI-like Systems by Direct Execution*. In *Proc. of IJCAI-97, Nagoya, Japan, August 1997*, pages 316–321, 1997.

[Fisher & Wooldridge 1995] M. Fisher and M. Wooldridge: *A logical approach to simulating societies*. In N. Gilbert and R. Conte (eds.): *Artificial Societies - the computer simulation of social life*, chapter 14, pages 268–283. UCL-Press, 1995.

[Fisher & Wooldridge 1997] M. Fisher and M. Wooldridge: *On the Formal Specification and Verification of Multi-Agent Systems*. International Journal of Cooperative Information Systems, 6(1):37–65, 1997.

[Fishwick 1990] P. A. Fishwick: *Methods for Qualitative Modelling in Simulation*. In P. A. Fishwick and R. B. Modjeski (eds.): *Knowledge-based Simulation: Methodology and Application*, volume 4 of *Advances in Simulation*, chapter 3, pages 36–52. Springer, 1990.

[Fishwick 1995] P. A. Fishwick: *Simulation Model Design and Analysis, Building Digital Worlds*. Prentice Hall, 1995.

[Fishwick & Zeigler 1992] P. A. Fishwick and B. P. Zeigler: *A Multimodel Methodology for Qualitative Model Engineering*. ACM Transactions on Modeling and Computer Simulation, 2:52–81, 1992.

[Fowler & Scott 1998] M. Fowler und K. Scott: *UML konzentriert - Die neue Standard-Objektmodellierungssprache anwenden*. Addison Wesley, 1998. Deutsche Übersetzung.

[Franklin & Graesser 1997] S. Franklin and A. Graesser: *It is an Agent, or just a Program? A Taxonomy for Autonomous Agents*. In N. Jennings and M. Wooldridge (eds.): *Intelligent Agents*, volume III. Springer, 1997.

[Fuchs et al. 1999] S. Fuchs, A. Dornhaus, R. Hager, F. Klügl, F. Puppe, and J. Tautz: *A Parasite – Host Simulation Perspective on Virulence in Varroa jacobsoni*. In M. P. Schwarz and K. Hogendoorn (eds.): *Social Insects at the turn of the Millennium, 13th Congress of the International Union for the Study of Social Insects IUSSI, Adelaide, Australia*, 1999.

[Fulbright & Stephens 1994] R. Fulbright and L. M. Stephens: *Classification of Multiagent Systems*. Technical report, University of South Carolina, Center for Information Technology, 1994.

[Funk et al. 1998] P. Funk, C. Gerber, J. Lind, and M. Schillo: *SIF: an Agent-Based Simulation Toolbox using the EMS Pradigm*. In *Proceedings of the 1998 EUROSIM Conference*, 1998.

[Gappa 1996] U. Gappa: *Generierbare Wissensakquisitionswerkzeuge*, Band 100 der Reihe *Diski*. infix, 1996.

[Gasser et al. 1988] L. Gasser, C. Braganza, and N. Herman: *Implementing distributed AI Systems using MACE*. In A. Bond and L. Gasser (eds.): *Readings in Distributed Artificial Intelligence*, chapter 6.3, pages 445–450. Morgen Kaufmann, 1988.

[Gavrila & Treur 1994] I. Gavrila and J. Treur: *A Formal Model for the Dynamics of Compositional Reasoning Systems*. In A. Cohn (ed.): *Proc. of ECAI-94*, pages 307–311. John Wiley, 1994.

[Genesereth 1995] M. Genesereth: *Knowledge Interchange Format Specification*. http://logic.stanford.edu/kif/specification.html, 1995.

[Genesereth & Ketchpel 1994] M. R. Genesereth and S. P. Ketchpel: *Software Agents*. Communication of the ACM, 37(7):48–53, 1994.

[Genesereth & Nilson 1987] M. L. Genesereth and N. J. Nilson: *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[Georgeff 1998] M. P. Georgeff: *Reasoning About Plans and Actions*. In H. E. Strobe and AAAI (eds.): *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artifical Intelligence*, chapter 5, pages 173–196. Morgan Kaufmann, 1998.

[Georgeff & Ingrand 1989] M. P. Georgeff and F. F. Ingrand: *Decision Making in an Embedded Reasoning System*. In *Proc. of the IJCAI'89*, pages 972–978, 1989.

[Gilbert 1995] N. Gilbert: *Emergence in social simulation*. In N. Gilbert and R. Conte (eds.): *Artificial Societies - the computer simulation of social life*, chapter 8, pages 144–156. UCL-Press, 1995.

[Gilbert & Troitzsch 1999] N. Gilbert and K. G. Troitzsch: *Simulation for the social scientist*. Open University Press, in press 1999. http://www.uni-koblenz.de/kgt/SimForSocSci.html.

[Goldberg 1989] D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, MA, 1989.

[Goodwin 1994] R. Goodwin: *Formalizing Properties of Agents*. Journal of Logic and Computation, 5(6), 1994.

[Gordon et al. 1992] D. M. Gordon, B. C. Goodwin, and L. E. H. Trainor: *A Parallel Distributed Model of the Behaviour of Ant Colonies*. Journal of theoretical Biology, 156:293–307, 1992.

[Graham 1996] P. Graham: *ANSI Common Lisp*. Prentice Hall, London, 1996.

[Grand et al. 1997] S. Grand, D. Cliff, and N. Malhotra: *Creatures: Artificial Life Autonomous Software Agents for Home Entertainment*. In *Proc. of the first Conference on Autonomous Agents (Agents'97)*. ACM, 1997.

[Gulyas & Kozsik 1999] L. Gulyas and T. Kozsik: *Model Design Interface – A CASE Tool for the Multi-Agent Modeling Language*. In *Int. Conference of PHD Students, Univ. of Miskolc, Ungarn, 1999*, 1999.

[Haddadi & Sundermeyer 1996] A. Haddadi and K. Sundermeyer: *Belief-Desire-Intention Agent Architectures*. In G. M. P. O'Hare and N. R. Jennings (eds.): *Foundations of Distributed Artificial Intelligence*, chapter 5, pages 169–186. John Wiley and Sons, 1996.

[Hanks et al. 1993] S. Hanks, M. E. Pollack, and P. E. Cohen: *Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures*. AI Magazine, 14(4):17–42, 1993.

[Hargesheimer 1998] H. Hargesheimer: *Adaptive Agenten in Konkurrenz zu Nicht-Adaptiven Agenten am Beispiel eines simulierten Fussballspieles*. Diplomarbeit, Institut für Informatik, Universiät Würzburg, Jan. 1998.

[Hayes-Roth & Doyle 1998] B. Hayes-Roth and P. Doyle: *Animate Characters*. Autonomous Agents and Multi-Agent Systems, 1:195–230, 1998.

[Hegselmann & Flache 1998] R. Hegselmann and A. Flache: *Understanding Complex Social Dynamics: A Plea for Cellular Automata Based Modelling*. Journal of Artificial Societies and Social Simulation, 1(3), 1998.

[Heisenberg 1994] M. Heisenberg: *Voluntariness (Willkürfähigkeit) and the General Organization of Behavior*. In R. J. Greenspan and C. P. Kyriacon (eds.): *Flexibility and Constraint in Behavioral Systems*, pages 147–156. John Wiley, 1994.

[Hogeweg & Hesper 1985] P. Hogeweg and B. Hesper: *Socioinformatic Processes: MIRROR Modelling Methodolgy*. Journal of Theoretical Biology, 113:311–330, 1985.

[Holland 1998] J. H. Holland: *Emergence*. Helix Books / Addison Wesley, Reading MA, 1998.

[Holvoet 1995] T. Holvoet: *Agents and Petri Nets*. Petri Net Newsletter, 49:3–8, 1995.

[Horn & Reinke 1999] E. Horn und T. Reinke: *Musterarchitekturen und Entwicklungsmethoden für Multiagentensysteme in betriebswirtschaftlichen Anwendungen*. In: S. Kirn und M. Petsch (Hrsg.): *Proc. of Workshop Intelligente Softwareagenten und betriebswirtschaftliche Anwendungsszenarien*, Band 14 der Reihe *Reports*, Seiten 23–34. TU Ilmenau, Juli 1999. Ilmenau.

[Hörnlein 1999] A. Hörnlein: *Tutorial: Modellbildung mit SeSAm*. unveröffentlicht, 1999.

[Hörnlein et al. 2000] A. Hörnlein, C. Oechslein und F. Puppe: *Optimierung in simulierten biologischen Multiagentensystemen mit Hilfe evolutionärer Verfahren*. In: *Proc. des Workshop "Multiagentensysteme und Individuenbasierte Simulation", März 2000, Würzburg*, Seiten 57–66. Institut für Informatik, Bericht Nr. 253, 2000.

[Hraber et al. 1997] P. T. Hraber, T. Jones, and S. Forrest: *The Ecology of Echo*. Artificial Life, 3:165–190, 1997.

[Huber 2000] P. Huber: *Formalisierung von Strategien beim komplexen Problemlösen unter Zuhilfenahme einer Multiagentensimulation*. Diplomarbeit, Institut für Psychologie, Universiät Würzburg, März 2000.

[Huhns & Singh 1997] M. N. Huhns and M. P. Singh: *Agents and Multiagent Systems: Themes, Approaches, and Challenges*. In M. N. Huhns and M. P. Singh (eds.): *Readings in Agents*, chapter 1, pages 1–24. Morgan Kaufmann, 1997.

[Huhns & Singh 1999] M. N. Huhns and M. Singh: *Multi-Agent Treatment of Agenthood*. Applied Artificial Intelligence, 13:3–10, 1999.

[Iglesias et al. 1999] C. Iglesias, M. Garijo, and J. C. Gonzales: *A Survey of Agent-Oriented Methodologies*. In J. P. Müller, M. Singh, and A. S. Rao (eds.): *Intelligent Agents V: Proceedings of the ATAL'98*, volume 1555 of *LNAI*. Springer, 1999.

[Iglesias et al. 1998] C. Iglesias, M. Garijo, J. C. Gonzales, and J. R.Velasco: *Analysis and Design of Multiagent Systems Using MAS-CommonKADS*. In M. Singh, A. Rao, and M. Wooldridge (eds.): *Intelligent Agents IV (= Proceedings of ATAL'97)*, volume 1365 of *Lecture Notes in Artificial Intelligence*, pages 313–327. Springer, 1998.

[Iglesias et al. 1996] C. Iglesias, J. C. Gonzales, and J. R. Velasco: *MIX: A General Purpose Multiagent Architecture*. In M. J. Wooldridge, J. P. Müller, and M. Tambe (eds.): *Intelligent Agents II (= Proceedings of ATAL'95)*, volume 1037 of *Lecture Notes in Artificial Intelligence*, pages 251–266. Springer, 1996.

[Ingrand et al. 1992] F. F. Ingrand, M. P. Georgeff, and A. S. Rao: *An Architecture for Real-Time Reasoning and System Control*. IEEE Expert, 7(6):34–44, 1992.

[Iwasaki 1989] Y. Iwasaki: *Qualitative Physics*. In A. Barr, , P. R. Cohen, and E. A. Feigenbaum (eds.): *The Handbook of Artificial Intelligence - Volume IV*, chapter XXI, pages 323–414. Addison Wesley, 2. edition, 1989.

[Jennings 1994] N. R. Jennings: *The ARCHON-System and its Applications*. In S. M. Deen (ed.): *Proc. of the CKBS'94 (International Working Conference on Cooperating Knowledge Based Systems*, pages 13–29. Springer, 1994. (invited paper).

[Jennings 1996] N. R. Jennings: *Coordination Techniques for Distributed Artificial Intelligence*. In G. M. P. O'Hare and N. R. Jennings (eds.): *Foundations of Distributed Artificial Intelligence*, chapter 6, pages 187–210. John Wiley and Sons, 1996.

[Jennings 2000] N. R. Jennings: *On agent-based software engineering*. Artificial Intelligence, 117:277–296, 2000.

[Jennings et al. 1998] N. R. Jennings, K. Sycara, and M. Wooldridge: *A Roadmap of Agent Research and Development*. Autonomous Agents and Multi-Agent Systems, 1(1):7–38, 1998.

[Jennings & Wooldridge 1998] N. R. Jennings and M. Wooldridge: *Applications of Intelligent Agents*. In N. R. Jennings and M. Wooldridge (eds.): *Agent Technology Foundations, Applications and Markets*, chapter 1, pages 1–27. Springer, 1998.

[Judson 1994] O. Judson: *The rise of the individual-based model in ecology*. Trends Ecol. Evol., 9(1):9–14, 1994.

[Kaelbling 1988] L. P. Kaelbling: *Goals as Parallel Program Specifications*. In *Proc. of the 8th National Conference on Artificial Intelligence AAAI-88*, pages 60–65, 1988.

[Kaelbling & Rosenschein 1991] L. P. Kaelbling and S. J. Rosenschein: *Action and Planning in Embedded Agents*. In P. Maes (ed.): *Designing Autonomous Agents*, pages 35–48. The MIT Press, Cambridge, 1991.

[Kinny & Georgeff 1997] D. Kinny and M. Georgeff: *Modelling and Design of Multi-Agent Systems*. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings (eds.): *Intelligent Agents III (= Proceedings of ATAL'96)*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 1–20. Springer, 1997.

[Kinny et al. 1996] D. Kinny, M. Georgeff, and A. Rao: *A Methodology and Modelling Technique for Systems of BDI Agents*. In W. V. de Velde and J. Perram (eds.): *Agents breaking away - Proc. of the MAAMAW'96*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 56–71. Springer, 1996.

[Kiss 1992] G. Kiss: *Variable Coupling of Agents to their environment: Combining Situated and Symbolic Automata*. In E. Werner and Y. Demazeau (eds.): *Decentralized A.I. 3 - 3rd European Workshop on Modellings Autonomous Agents in a Multi-Agent World, MAAMAW'91*, pages 231–248. North Holland, 1992.

[Klügl 2000] F. Klügl: *Visuelles Programmieren als Basis einer Modellierumgebung: Eigenschaften, Konzepte, Anforderungen*. Technischer Bericht, Institut für Informatik, Universität Würzburg, Technischer Bericht Nr. 260, 2000.

[Klügl & Oechslein 1999] F. Klügl und C. Oechslein: *SeSAm - Handbuch und Dokumentation*. Lehrstuhl für Künstliche Intelligenz und Angewandte, Universität Würzburg, 1999.

[Klügl et al. 1996] F. Klügl, F. Puppe, U. Raub und J. Tautz: *Ein Simulationssystem zur Darstellung emergenten Verhaltens*. In: ASIM (Hrsg.): *Proc. der Fachtagung Simulation und Animation 1996, Magdeburg, März 1996*, 1996. Mitteilungen aus den Arbeitskreisen Nr. 54.

[Klügl et al. 1998] F. Klügl, F. Puppe, U. Raub, and J. Tautz: *Simulating Multiple Emergent Behaviors - exemplified in an Ant Colony*. In C. Adami, R. Belew, H. Kitano, and C. Taylor (eds.): *Artificial Life VI*, pages 408–412. MIT Press, 1998.

[Klügl & Raub 1997] F. Klügl and U. Raub: *Reproducing the behavior of a complete ant colony*. In *Proc. of the Second German Workshop on Artificial Life, GWAL-97*, 1997.

[Klügl et al. 2000] F. Klügl, J. Wahle, A. Bazzan, and M. Schreckenberg: *Towards Anticipatory Traffic Forecast - Modelling of Route Choice Behavior*. In *Proc. of the Agents 2000 Workshop "Agents in Traffic Modelling", Barcelona, June 2000*, 2000.

[Kraetzschmar & Reinema 1993] G. Kraetzschmar und R. Reinema: *VKI Tools und Experimentierumgebungen*. In: H. J. Müller (Hrsg.): *Verteilte Künstliche Intelligenz - Methoden und Anwendungen*, Kapitel 8, Seiten 222–266. BI Wissenschaftsverlag, 1993.

[Kuipers 1986] B. Kuipers: *Qualitative Simulation*. Artificial Intelligence, 29:289–339, 1986.

[Kuljis 1994] J. Kuljis: *User Interfaces and Discrete Event Simulation Models*. Simulation Practice and Theory, 1:207–221, 1994.

[Laird et al. 1987] J. E. Laird, A. Newell, and P. S. Rosenbloom: *SOAR: An Architecture for General Intelligence*. Artificial Intelligence, 33:1–64, 1987.

[Laird et al. 1991] J. E. Laird, E. S. Yager, M. Hucka, and C. M. Tuck: *Robo-Soar: An Integration of External Interaction, Planning and Learning Using Soar*. Robotics and Autonomous Systems, pages 113–130, 1991.

[Langton 1991] C. Langton: *Life at the Edge of Chaos*. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*, pages 41–91. Addison-Wesley, 1991.

[Levitt et al. 1994] R. E. Levitt, G. P. Cohen, J. C. Kunz, C. I. Nass, T. Christiansen, and Y. Jin: *The Virtual Design Team: Simulating How Organization Structure and Information Processing Tools Affect Team Performance*. In K. Carley and M. J. Prietula (eds.): *Computational Organization Theory*, chapter 1, pages 1–18. Lawrence Erlbaum Publishers, 1994.

[Luck et al. 1997] M. Luck, N. Griffiths, and M. d'Inverno: *From Agent Theory to Agent Construction*. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings (eds.): *Intelligent Agents III (= Proceedings of ATAL'96)*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 49–63. Springer, 1997.

[Luger & Stubblefield 1993] G. L. Luger and W. A. Stubblefield: *Artificial Intelligence: Structures and Strategies for complex problem solving*. Benjamin / Cummings Publishing Company, 2 edition, 1993.

[MadKit 2000] MadKit: *The MadKit Project (O. Gutknecht and J. Ferber)*. http: / / www.madkit.org, 2000.

[Maes 1991] P. Maes: *A Bottom-Up Mechanism for Behavior Selection in an Artificial Creature*. In J. A. Meyer and S. W. Wilson (eds.): *From Animals to Animats. Proceedings of the first International Conference on the Simulation of Adaptive Behavior*, pages 238–246, MIT Press, Cambridge, 1991.

[Maes 1994] P. Maes: *Modeling Adaptive Autonomous Agents*. Artificial Life, 1:135–162, 1994.

[Maes 1995] P. Maes: *Artificial Life Meets Entertainment: Lifelike Autonomous Agents*. Communication of the ACM, 38(11):108–114, November 1995.

[Marcenac & Giroux 1998] P. Marcenac and S. Giroux: *GEAMAS: A Generic Architecture for Agent-Oriented Simulaion of Complex Processes*. Applied Intelligence, 8:247–267, 1998.

[Maxwell & Costanza 1995] T. Maxwell and R. Costanza: *Distributed Modular Spatial Ecosystem Modeling*. International Journal of Computer Simulation, 5:247–262, 1995.

[Mentges 1999] E. Mentges: *Concepts for an agent-based framework for interdisciplinary social science simulation*. Journal of Artificial Societies and Social Simulation, 2(2), 1999.

[Millonas 1994] M. Millonas: *Swarms, Phase Transitions, and Collective Intelligence*. In C. Langton (ed.): *Artificial Life III*, pages 417–445. Addison-Wesley, 1994.

[Minar et al. 1996] N. Minar, R. Burkhart, C. Langton, and M. Askenazi: *The SWARM Simulation System: A Toolkit for Building Multi-Agent Simulation*. http://www.santafe.edu/projects/swarm/, 1996.

[Möhring 1994] M. Möhring: *Grundlagen der Prozessmodellierung.* Fachberichte Informatik 14, Universität Koblenz, 1994.

[Möhring 1996] M. Möhring: *Social Science Multilevel Simulation with MIMOSE*. In K. G. Troitzsch, U. Mueller, G. N. Gilbert, and J. E. Doran (eds.): *Social Science Microsimulation*, chapter 6, pages 123–137. Springer, 1996.

[Monsef 1997] Y. Monsef: *Modelling and Simulation of Complex Systems - Concepts, Methods and Tools.* Frontiers in Simulation. Society for Computer Simulation, 1997.

[Moss et al. 1997] S. Moss, H. Gaylard, S. Wallis, and B. Edmonds: *SDML: A Multi-Agent Language for Organizational Modelling.* CPM-Report 16, Centre for Policy Modelling, 1997.

[Mössinger et al. 1995] P. Mössinger, D. Polani, R. Spalt, and T. Uthmann: *XRaptor - A Synthetic Multi-Agent Environment for Evaluation of Adaptive Control Mechanisms.* In Breitenecker and Husinsky (eds.): *Proceedings of the 1995 EUROSIM Conference, TU Wien*, 1995.

[Moulin & Chaib-Draa 1996] B. Moulin and B. Chaib-Draa: *An Overview of Distributed Artificial Intelligence.* In G. O'Hare and N. Jennings (eds.): *Foundations of Distributed Artificial Intelligence*, chapter 1, pages 3–56. John Wiley and Sons, 1996.

[Mulder et al. 1998] M. Mulder, J. Treur, and M. Fisher: *Agent Modelling in METATEM and DESIRE.* In M. Singh, A. Rao, and M. Wooldridge (eds.): *Intelligent Agents IV (= Proceedings of ATAL'97)*, volume 1365 of *Lecture Notes in Artificial Intelligence*, pages 193–207. Springer, 1998.

[Müller 1993] H. J. Müller (Hrsg.): *Verteilte Künstliche Intelligenz - Methoden und Anwendungen*. BI Wissenschaftsverlag, 1993.

[Müller 1996] J. P. Müller (ed.): *The Design of Intelligent Agents - A Layered Approach*, volume 1177 of *LNAI*. Springer, 1996.

[Müller 1999] J. P. Müller: *Architectures and applications of intelligent agents: A survey*. Knowledge Engineering Review, 13(4):353–380, 1999.

[Müller 1999] C. Müller: *Modellkalibrierung mittels Parameteroptimierung.* Diplomarbeit, Institut für Informatik, Universiät Würzburg, Mai 1999.

[Nagel & Rasmussen 1994] K. Nagel and S. Rasmussen: *Traffic at the Edge of Chaos.* In R. A. Brooks and P. Maes (eds.): *Artificial Life IV*, pages 222–235. MIT-Press, 1994.

[Newell 1982] A. Newell: *The Knowledge Level*. Artificial Intelligence, 18:87–127, 1982.

[Newell & Simon 1976] A. Newell and H. A. Simon: *Computer Science as Empirical Inquiry: Symbols and Search.* Communications of the ACM, 19:113–126, 1976.

[Nolfi & Parisi 1995] S. Nolfi and D. Parisi: *Evolving Artificial Neural Networks that Develop in Time*. In F. Moran, A. Moreno, J. J. Merelo, and P. Chacon (eds.): *Advances in Artificial Life (ECAL-95)*, volume 929 of *Lecture Notes in Artificial Intelligence*, pages 353–367. Springer, 1995.

[Nwana et al. 1999] H. Nwana, D. Ndumu, L. Lee, and J. Collis: *ZEUS: A Toolkit For Building Distributed Multi-Agent Systems.* Applied Artificial Intelligence, 13:129–185, 1999.

[Oechslein 1997] C. Oechslein: *Emergente versus explizite Strategien in simulierten Multi-Agenten Systemen*. Diplomarbeit, Institut für Informatik, Universiät Würzburg, Dez. 1997.

[Oechslein et al. 1999] C. Oechslein, F. Klügl und F. Puppe: *Kalibrierung von Multiagentenmodellen*. In: *Proc. of the Simulation und Künstliche Intelligenz Workshop: KI-Methoden in der Simulationsbasierten Optimierung, 12./13. April 1999, Chemnitz*. Informatik-Berichte der Universität Chemnitz, 1999.

[O'Hare & Jennings 1996] G. M. P. O'Hare and N. R. Jennings (eds.): *Foundations of Distributed Artificial Intelligence.* John Wiley and Sons, 1996.

[Parunak 1996] H. V. D. Parunak: *Applications of Distributed Artificial Intelligence in Industry*. In G. M. P. O'Hare and N. R. Jennings (eds.): *Foundations of Distributed Artificial Intelligence*, chapter 4, pages 139–164. John Wiley and Sons, 1996.

[Poggi 1995] A. Poggi: *DAISY: an Objekt-Oriented System for Distributed Artificial Intelligence.* In M. J. Wooldridge and N. R. Jennings (eds.): *Intelligent Agents. ECAI-94 Workshop on Agent Theories, Architectures and Languages ATAL-94*, volume 890 of *LNAI*, pages 341–354. Springer, August 1995.

[Pollack & Ringuette 1990] M. E. Pollack and M. Ringuette: *Introducing the Tileworld: Experimentally Evaluating Agent Architectures*. In *Proceedings of the Ninth National Conference on Artificial Intelligence, Boston, 1990*, pages 183–189, 1990.

[Prietula & Carley 1994] M. Prietula and K. Carley: *Computational organization theory: Autonomous agents and emergent behavior*. Journal of Organizational Computing, 41(1):41–83, 1994.

[Prietula et al. 1998] M. J. Prietula, K. M. Carley, and L. Gasser (eds.): *Simulating Organizations - Computational Models of Institutions and Groups*. AAAI Press / MIT Press, Cambridge, 1998.

[Puppe 1990] F. Puppe: *Problemlösungsmethoden in Expertensystemen*. Springer, 1990.

[Puppe 1991] F. Puppe: *Einführung in Expertensysteme*. Springer, 2 Auflage, 1991.

[Puppe et al. 1996] F. Puppe, U. Gappa, K. Poeck und S. Bamberger: *Wissensbasierte Diagnose- und Informationssysteme*. Springer, 1996.

[Rao & Georgeff 1991] A. S. Rao and M. P. Georgeff: *Modeling rational agents within a BDI-Architecture*. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, 1991.

[Rao & Georgeff 1995] A. S. Rao and M. P. Georgeff: *BDI Agents: From Theory to Practice*. In V. Lesser (ed.): *Proceedings of the first International Conference on Multi-Agent Systems*, pages 312–319. MIT Press, 1995.

[RePast 2000] RePast: *RePast: Recursive Porous Agent Simulation Toolkit*. http://repast.sourceforge.net/, 2000.

[Repenning 2000] A. Repenning: *AgentSheets: an Interactive Simulation Environment with End-User Programmable Agents*. In *Proceedings of INTERACTION 2000, Tokyo, Japan*, 2000.

[Repenning & Summner 1994] A. Repenning and T. Summner: *Programming as Problem Solving: A Participatory Theater Approach*. In *Proceedings of the Workshop on Advanced Visual Interfaces, 1994, Bari, Italy*, pages 182–191, 1994.

[Resnick 1991] M. Resnick: *Animal Simulation with StarLogo: Massive Parallelism for the Masses*. In J. A. Meyer and S. W. Wilson (eds.): *From Animals to Animats. Proceedings of the first International Conference on the Simulation of Adaptive Behavior*, pages 534–539, MIT Press, Cambridge, 1991.

[Rich & Knight 1991] E. Rich and K. Knight: *Artificial Intelligence*. McGraw Hill Publisher, 2 edition, 1991.

[Richardson 1991] G. P. Richardson: *System Dynamics: Simulation for Policy Analysis from a Feedback Perspective*. In P. A. Fishwick and P. Luker (eds.): *Qualitative Simulation and Analysis*, volume 5 of *Advances in Simulation*, pages 145–169. Springer, 1991.

[Richardson & Pugh 1981] G. Richardson and A. L. Pugh: *Introduction to System Dynamics Modeling with DYNAMO*. MIT Press, Cambridge, 1981.

[Richmond et al. 1987] B. Richmond and others: *STELLA for Buisness*. High Performance Systems, Hanover N.H., 1987.

[RoboCup 2000] RoboCup: *RoboCup Initiative*. http://www.robocup.org/, 2000.

[Rosenbloom et al. 1991] P. S. Rosenbloom, J. E. Laird, A. Newell, and R. McCarl: *A preliminary analysis of the Soar architecture as a basis for general intelligence*. Artificial Intelligence, 47:289–325, 1991.

[Rumelhart 1989] D. E. Rumelhart: *The Architecture of Mind: A Connectionist Approach*. In M. I. Posner (ed.): *Foundations of Cognitive Science*. MIT Press, Cambridge, 1989. reprinted in J. Haugeland (ed.) Mind Design II, MIT Press, 2. Auflage, 1997.

[Russell & Norvig 1995] S. Russell and P. Norvig: *Artificial Intelligence - A Modern Approach*. Prentice Hall, 1995.

[Schmidt 1998] N. Schmidt: *Implementierung und Evaluation verschiedener Organisationsformen in Multiagentensystemen am Beispiel einer Multi-Agenten Version des Spiels Civilization*. Diplomarbeit, Institut für Informatik, Universiät Würzburg, Juli 1998.

[Schneider 1991] H. J. Schneider: *Deklarative Programierung*. In: *Lexikon der Informatik und Datenverarbeitung*, Seite 624. Oldenburg Verlag, 3 Auflage, 1991.

[Schneider & Schwarz 2000] A. Schneider und P. Schwarz: *Agentenbasierte, simulationsgestützte Optimierung im Internet*. In: *Proc. des Workshop "Multiagentensysteme und Individuenbasierte Simulation", März 2000, Würzburg*, Seite 23f. Institut für Informatik, Bericht Nr. 253, 2000.

[SDML 2000] SDML: *SDML: a Strictly Declarative Modelling Language*. http://www.cpm.mmu.ac.uk/sdml/, 2000.

[Shoham 1993] Y. Shoham: *Agent-oriented programming*. Artificial Intelligence, 60:51–92, 1993.

[Shoham 1999] Y. Shoham: *What we talk about, when we talk about software agents*. IEEE Intelligent Systems, 14(2):28–32, 1999. Expert Opinion.

[Shoham & Tennenholtz 1995] Y. Shoham and M. Tennenholtz: *On social laws for artificial agent societies: off-line design*. Artificial Intelligence, 73:231–252, 1995.

[Sloman & Logan 1999] A. Sloman and B. Logan: *Building Cognitively Rich Agents Using the Sim_Agent Toolkit*. Communication of the ACM, 43(2):71–77, March 1999.

[Smith 1980] R. G. Smith: *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*. IEEE Transactions on Computers, 12:1104–1113, 1980.

[Smith et al. 1994] D. C. Smith, A. Cypher, and J. Spohrer: *KIDSIM: Programming Agents Without a Programming Language*. Communication of the ACM, 37(7):55–67, July 1994.

[Soar 2000] Soar: *Soar Homepage*. http://www.bigfoot.eecs.umich.edu/soar/, 2000.

[Sommerville 1996] I. Sommerville: *Software Engineering*. Addison Wesley, 5. edition, 1996.

[Spriet & Vansteenkiste 1982] J. A. Spriet and G. C. Vansteenkiste: *Computer-aided modelling and simulation*. Academic Press, London, 1982.

[Steele 1990] G. Steele: *COMMON LISP*. Digital Press, 2.0 edition, 1990.

[Steels 1994] L. Steels: *The Artificial Life Roots of Artificial Intelligence*. Artificial Life, 1, 1994.

[Strippgen 1995] S. Strippgen: *INSIGHT: Ein virtuelles Labor für das Studium von behavior-basierten autonomen Agenten*. In: K. D. et al. (Hrsg.): *Proceedings of the first German Workshop on Artificial Life GWAL-96*, Band 271 der Reihe *GMD-Studien*, Seiten 107–112, 1995.

[Sundermeyer 1993] K. Sundermeyer: *Modellierung von Agentensystemen*. In: H. J. Müller (Hrsg.): *Verteilte Künstliche Intelligenz - Methoden und Anwendungen*, Kapitel 2, Seiten 22–44. BI Wissenschaftsverlag, 1993.

[Sutton & Barto 1998] R. S. Sutton and A. G. Barto: *Reinforcement Learning: An Introduction*. A Bradford Book, MIT Press, Cambridge, 1998.

[Tambe 1997] M. Tambe: *Agent architectures for flexible, practical teamwork*. In *Proc. of the National Conference on Artificial Intelligence (AAAI-97)*, August 1997.

[Tambe et al. 1995] M. Tambe, W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb: *Intelligent Agents for Interactive Simulation Environments*. AI Magazine, 16:15–39, 1995.

[Terzopoulos et al. 1994] D. Terzopoulos, X. Tu, and R. Grzeszczuk: *Artificial Fishes: Automonomous Locomotion, Perception, Behavior and Learning in a Simulated Physical World*. Artificial Life, 1:327–351, 1994.

[Thomas 1995] S. R. Thomas: *The PLACA Agent Programming Language*. In M. J. Wooldridge and N. R. Jennings (eds.): *Intelligent Agents. ECAI-94 Workshop on Agent Theories, Architectures and Languages ATAL-94*, volume 890 of *LNAI*, pages 355–370. Springer, August 1995. Amsterdam.

[Toffoli & Margolos 1987] T. Toffoli and N. Margolos: *Cellular Automata Machines: A new environment for modeling*. MIT Press, Cambridge, MA, 1987.

[Travers 1988] M. D. Travers: *Animal Construction Kits*. In C. Langton (ed.): *Artificial Life*, SFI Studies in Complexity, pages 421–442. Addison Wesley Publishing Company, 1988.

[Travers 1996] M. D. Travers: *Programming with Agents: New metaphors of thinking about computation*. PhD thesis, Program in Media Arts and Sciences, School of Architecture and Planning, MIT, http://mt.www.media.mit.edu/people/mt/diss/index.html, May 1996.

[Troitzsch 1990] K. G. Troitzsch: *Modellbildung und Simulation in den Sozialwissenschaften*. Westdeutscher Verlag, Opladen, 1990.

[Troitzsch 1996] K. G. Troitzsch: *Multilevel Simulation*. In K. G. Troitzsch, U. Mueller, G. N. Gilbert, and J. E. Doran (eds.): *Social Science Microsimulation*, chapter 5, pages 107–122. Springer, 1996.

[Troitzsch 1997] K. G. Troitzsch: *Social Science Simulation - Origin, Prospects, Purposes*. In R. Conte, R. Hegselmann, and P. Terna (eds.): *Simulating Social Phenomena*, volume 456 of *Lecture Notes in Economic and Mathematical Systems*, pages 41–54. Springer, 1997.

[Uhrmacher 1996] A. M. Uhrmacher: *Object-Oriented and Agent-Oriented Simulation: Implications for Social Science Application*. In K. G. Troitzsch, U. Mueller, G. N. Gilbert, and J. E. Doran (eds.): *Social Science Microsimulation*, chapter 20, pages 432–447. Springer, 1996.

[Uhrmacher & Schattenberg 1998] A. M. Uhrmacher and B. Schattenberg: *Agents in Discrete Event Simulation*. In *Proc. of the ESS'98, October 26-28, Nottingham*, pages 129–136. SCS Publications, 1998.

[UMBC 2000] UMBC: *KQML Software Homepage*. http://www.csee.umbc.edu/kqml/software/, 2000.

[Urban 1997] C. Urban: *Agentenbasierte Simulation eines Lotka-Volterra-Systems mit Hilfe von Simplex II*. In: *7. Treffen des Arbeitskreises Werkzeuge für Modellbildung und Simulation in Umweltanwendungen. Wissenschaftliche Berichte des Forschungszentrums Karlsruhe*, 1997.

[Urban 2000] C. Urban: *PECS: A Reference Model for the Simulation of Multi-Agent Systems*. In R. Suleiman, K. G. Troitzsch, and G. N. Gilbert (eds.): *Tools and Techniques for Social Science Simulation*. Physica-Verlag, Heidelberg, 2000.

[Vetterle 1986] H. Vetterle: *Konstruktion und Simulation mikroanalytischer Modelle*, Band 1 der Reihe *BASYS*. Maro-Verlag, 1986.

[Wagner 1997] G. Wagner: *Practical Theory and Theory-Based Practice*. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings (eds.): *Intelligent Agents III (= Proceedings of ATAL'96)*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 67–70. Springer, 1997.

[Wavish 1992] P. Wavish: *Exploiting Emergent Behavior in Multi-Agent Systems*. In E. Werner and Y. Demazeau (eds.): *Decentralized A.I. 3 - 3rd European Workshop on Modellings Autonomous Agents in a Multi-Agent World, MAAMAW'91*. North Holland, 1992.

[Wavish & Graham 1995] P. Wavish and M. Graham: *Roles, Skills and Behaviour: a Situated Action Approach to Organising Systems of Interacting Agents*. In M. J. Wooldridge and N. R. Jennings (eds.): *Intelligent Agents. ECAI-94 Workshop on Agent Theories, Architectures and Languages ATAL-94*, volume 890 of *LNAI*, pages 371–385. Springer, August 1995. Amsterdam.

[Weiss 1999] G. Weiss (ed.): *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[Wilson 1998] W. G. Wilson: *Resolving Discrepancies between Deterministic Population Models and Individual-Based Simulations*. The American Naturalist, 151:116–134, 1998.

[Wolfram 1986] S. Wolfram (ed.): *Theory and Application of Cellular Automata*, volume 1 of *Advanced Series on Complex Systems*. World Scientific, 1986.

[Wooldridge 1997] M. Wooldridge: *Agent-Based Software Engineering*. IEE Proc. on Software Engineering, 144(1):26–37, 1997. Erweiterte Fassung vom September 97: http://www.elec.qmw.ac.uk/dai/people/mikew/pubs/iee-se.ps.gz.

[Wooldridge & Jennings 1995] M. Wooldridge and N. Jennings: *Intelligent agents: theory and practice*. The Knowledge Engineering Review, 10(2):115–152, 1995.

[Wooldridge et al. 1999] M. Wooldridge, N. R. Jennings, and D. Kinny: *A Methodology for Agent-Oriented Analysis and Design*. In *Proceedings of the 3rd Internation Conference on Autonomous Agents,1999*. ACM Press, 1999.

[Zeigler 1976] B. P. Zeigler: *Theory of Modelling and Simulation*. John Wiley and Sons, New York, 1976.

[Zeigler 1984] B. P. Zeigler: *Multi-facetted Modelling and Discrete Event Simulation*. Academic Press, New York, 1984.