Agenten in Raum und Zeit

Diskrete Simulation mit Multiagentensystemen und expliziter Raumrepräsentation

Dissertation zur Erlangung des Doktorgrades
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
der Universität Hamburg

vorgelegt von Ruth Meyer

Hamburg, 2014

Tag der Disputation: 26. Juni 2014

Gutachter:
Prof. Dr.-Ing. Bernd Page, Universität Hamburg
Prof. Dr.-Ing. Jochen Wittmann, HTW Berlin
Dr. Daniel Moldt, Universität Hamburg

And crawling on the planet's face
Some insects called the human race.
Lost in time, and lost in space . . .

The Rocky Horror Picture Show

Kurzfassung

Die letzten zehn Jahre haben eine Fülle von sogenannten agentenbasierten Modellen hervorgebracht. Diese Modelle zeichnen sich dadurch aus, dass Entitäten im beobachteten System und deren Interaktionen direkt auf Modellentitäten („Agenten“) und entsprechende Interaktionen zwischen Agenten abgebildet werden, die Struktur des modellierten Systems also erhalten bleibt, während sein Verhalten über die Zeit untersucht wird. Die Anwendungsgebiete reichen von Soziologie und Ökonomie über Biologie und Ökologie bis zu Archäologie und dem militärischen Bereich.

Da der agentenbasierte Modellierungsstil in diesen Gebieten oft der kontinuierlichen Simulation gegenübergestellt wird, in welcher ein System von Differentialgleichungen schrittweise gelöst wird, ist es nicht verwunderlich, dass die überwiegende Mehrzahl agentenbasierter Modelle mit Zeitsteuerung implementiert werden, d.h. die Simulationszeit wird in äquidistanten Schritten fortgeschrieben. Dies ist außerdem erheblich einfacher zu realisieren als die flexiblere und effizientere Ereignissteuerung, was besonders im Falle fehlender Simulationswerkzeug-Unterstützung eine Rolle spielt. Anwendungen und Werkzeuge werden meist von Domänen-Experten und nicht Simulationsexperten entwickelt.

Im Rahmen der vorliegenden Dissertation wird gezeigt, wie die agentenbasierte Simulation im Kontext der ereignisdiskreten Simulation als eine neue Weltsicht definiert werden kann. Diese in Anlehnung an die Bezeichnungen der klassischen Weltsichten als „agentenorientiert“ bezeichnete Weltsicht kombiniert die Metapher der Multiagentensysteme mit einem ereignisgesteuerten Zeitfortschritt. Verwendung der Ereignissteuerung setzt voraus, dass (a) die Dauer von Aktivitäten der Agenten und Umgebung vor ihrem Ende bestimmt, (b) jeder Agent sofort auf Veränderungen in der Umgebung reagieren und (c) die Aktualisierung des Umgebungszustands trotz asynchronem Agenten-Update effizient gehalten werden kann.

Darauf aufbauend wurde ein Konzept entwickelt für ein Simulationswerkzeug, das diese agentenorientierte Weltsicht umsetzt. Besonderer Schwerpunkt des Werkzeugs ist es, die explizite Repräsentation einer räumlichen Umgebung sowie die Bewegung von Agenten im Raum zu unterstützen. Dies sind Bereiche, in denen bestehende Werkzeuge für agentenbasierte Modellierung Schwächen aufweisen, obwohl ein Großteil der Multiagentenmodelle den räumlichen Aspekt der Umgebung explizit berücksichtigen und mobile Agenten vorsehen.

Das Konzept wurde prototypisch realisiert im Simulationsframework FAMOS als eine Erweiterung des bewährten diskreten Simulators DESMO-J. FAMOS konnte durch Re-Implementation zweier aus der Literatur bekannten Modelle (Schellings Segregationsmodell, Sugarscape) getestet und darüber hinaus in einer praxisrelevanten Anwendung zur Simulation von Stadtkurierdiensten ausführlich evaluiert werden.

Abstract

The last ten years have spawned a plethora of so-called agent-based models. This type of model is characterized by mapping entities and their interactions in the system under observation directly onto entities in the model ("agents") and interactions between agents, respectively. Thus the model preserves the structure of the modelled system while investigating its behaviour over time. The wide range of application domains includes sociology, economy, biology, ecology, archaeology and the military sector.

Since these domains often contrast the agent-based style of modelling with continuous simulation, in which a set of differential equations is solved numerically by stepwise integration, it is not surprising that the vast majority of agent-based models apply a time-driven approach, i.e. simulation time is advanced in equidistant steps. This time advance method is considerably easier to implement than the more flexible and efficient event-driven approach, which can make all the difference if a dedicated simulation toolkit is missing. Both applications and toolkits are usually developed by domain experts and not simulation experts.

The present dissertation thesis shows how agent-based simulation can be defined as a new world view within the context of discrete-event simulation. Named “agent-oriented” in accordance with the (German) terms for the classical world views, this world view combines the metaphor of multi-agent systems with an event-driven time advance. Applying the event-driven approach requires that (a) the durations for agent and environment actions are determined before they terminate so that the respective termination event can be scheduled in time, (b) each agent is able to instantly react to changes in its environment, and (c) the update of the state of the environment can be kept efficient despite updating agents asynchronously.

Based on these requirements the concept for a simulation toolkit was developed, which implements the agent-oriented world view. A major focus of this toolkit is to support an explicit representation of space within the environment and the movement of agents in that space. These are areas where existing toolkits for agent-based modelling show shortcomings, despite the fact that a majority of multi-agent models explicitly model space and allow for mobile agents.

A prototypical implementation of this concept is provided with the simulation framework FAMOS extending the tried and tested discrete-event simulator DESMO-J. FAMOS was first tested by re-implementing two well-known models (Schelling’s segregation model, Epstein and Axtell’s Sugarscape model) and then thoroughly evaluated in a practice-oriented simulation study of a city courier service.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

Whatever occurs, occurs in space and time.
– Michael Wegener

Die letzten zehn Jahre haben eine Fülle von sogenannten agentenbasierten Simulationsmodellen hervorgebracht. Diese Modelle zeichnen sich dadurch aus, dass Entitäten im beobachteten System direkt auf Modellentitäten („Agenten“) und deren Interaktionen auf entsprechende Interaktionen zwischen Agenten abgebildet werden. Die Struktur des modellierten Systems bleibt somit erhalten, während sein Verhalten über die Zeit untersucht wird. Das Spektrum der Anwendungsgebiete reicht von den Sozialwissenschaften über Biologie und Ökologie bis zu Archäologie und dem militärischen Bereich.

Der „Siegeszug“ der agentenbasierten Modellierung und Simulation (ABMS, auch als MABS für *multiagent-based simulation* abgekürzt) liegt in der breiten Anwendbarkeit der Agenten-Metapher begründet. In komplexen Systemen können die aktiven Entitäten in der Regel gewinnbringend als autonome Agenten betrachtet werden, welche einer Reihe von Verhaltensregeln folgen, um ihre Ziele zu erreichen, während sie miteinander und mit ihrer Umwelt interagieren.

Dieser Modellierungsstil wird häufig als neues Paradigma der ereignisdiskreten und der kontinuierlichen Simulation gegenübergestellt (vgl. z. B. Parunak et al. 1998; Dolk et al. 2001; Schieritz und Milling 2003; Chan et al. 2010; Siebers et al. 2010). Die Abgrenzung zur kontinuierlichen Simulation ist offensichtlich: Während die kontinuierliche Simulation eine makroskopische Sicht auf das zu modellierende System einnimmt und aggregierte Zustandsgrößen auf Basis eines durchschnittlichen Individuums betrachtet, handelt es sich bei der agentenbasierten Simulation um einen mikroskopischen Ansatz, der das Konzept der Multiagentensysteme zur Formulierung eines Simulationsmodells nutzt. Die Entitäten eines solchen Multiagentenmodells sind simulierte Agenten, welche in einer simulierten Umwelt miteinander (inter-)agieren (Klügl 2001, S. 68). Globale Systemeigenschaften ergeben sich aus dem Verhalten der individuellen Agenten.

Die Abgrenzung zur diskreten Simulation ist weniger eindeutig, da sowohl die prozessorientierte Weltsicht als auch die objektorientierte Simulation ebenfalls eine entitätenbasierte Perspektive einnehmen und das Verhalten einzelner Akteure beschreiben. Das Konzept eines Agenten ist jedoch allgemeiner und bietet eine höhere Abstraktionsebene als Prozesse oder Objekte, wie in Abschnitt 3.1.3 ausführlich dargelegt wird. Agentenbasierte Modelle können daher Systeme abbilden, deren Komplexität die Ausdruckskraft traditioneller Simulationsansätze übersteigt. Der nicht ganz ernst gemeinte Titel einer Podiumsdiskussion drückt dies wie folgt aus: „Discrete-event simulation is dead, long live agent-based simulation!“ (Siebers et al. 2010).

1.1. Zielsetzung

Die Agenten eines Multiagentenystems agieren in einer gemeinsamen Umwelt, welche eine von den Agenten unabhängige Dynamik aufweisen kann. Sowohl die Handlungen der Agenten als auch die Prozesse in der Umgebung finden konzeptuell nebenläufig statt. Der von bestehenden ABMS-Softwarewerkzeugen und einführenden Darstellungen (vgl. z. B. Klügl 2001; Gilbert und Troitzsch 2005; Macal und North 2008; North et al. 2013) propagierte Modellierungsstil sieht eine Diskretisierung der Simulationszeit in regelmäßige Intervalle vor. Um die Nebenläufigkeit von Aktionen abzubilden, werden in jedem Zeitschritt alle Agenten aktiviert, so dass sie ihre Handlung(en) durchführen und ihren Zustand aktualisieren können. Dies hat den Vorteil, dass sich ein Modellierer über die Dauer von Aktionen keine Gedanken machen muss.

Nachteilig ist dagegen, dass alle Ereignisse, welche in einen Zeitschritt fallen, behandelt werden, als ob sie zum Ende des Intervalls stattfinden. Die natürliche Reihenfolge von Ereignissen kann dabei verloren gehen; stattdessen wird die Verantwortung für eine korrekte Abfolge in die Hand des Modellierers (oder des verwendeten Simulationswerkzeugs) gelegt. Bei einer Verwendung der Ereignissteuerung ist dies nicht der Fall, da die Diskretisierung der Simulationszeit von den Abläufen im modellierten System selbst vorgegeben wird anstatt von einer künstlichen Taktrate. Systeme, welche inhärent von Ereignissen gesteuert werden, sind in verschiedenen Anwendungsbereichen zu finden; als Beispiele seien Dominanz und Konkurrenz in der Ökologie, wo die Vertreibung eines Individuums aus einem Habitat eine Reihe weiterer Dominanztests und Vertreibungen in benachbarten Habitaten nach sich ziehen kann (Grimm und Railsback 2005, S. 112), chemische Reaktionen (Barnes und Chu 2010, S. 26ff), Auftragseingänge an der Börse (Jacobs et al. 2004; Daniel 2006; Boer et al. 2007) oder die komplexen, voneinander abhängigen Prozesse der Auftragsabwicklung in dem in Kapitel 6 beschriebenen Kurierdienst genannt.

Ein Ziel dieser Arbeit ist es daher, die Agenten-Perspektive mit einem ereignisgesteuerten Zeitfortschritt zu verbinden und diese als agentenorientierte Simulation bezeichnete Weltsicht mit geeigneter Software zu unterstützen. Eine Ereignissteuerung ist nicht nur genauer, sondern auch effizienter als eine Zeitsteuerung, da jeweils nur die vom aktuellen Ereignis betroffenen Entitäten aktualisiert werden. Für die Verwendung in Multiagentenmodellen setzt dies voraus, dass folgende Probleme gelöst werden:

1. Da zeitverbrauchende Aktivitäten in der diskreten Simulation auf Ereignisse abgebildet werden, muss der Zeitpunkt für das Ende-Ereignis einer Aktivität spätestens zu ihrem Beginn bekannt sein. Eine Ausnahme bilden sogenannte Verzögerungen, deren Ende von bestimmten Bedingungen abhängen. Übertragen auf die agentenbasierte Simulation bedeutet dies, dass ein Modellierer für jede Aktion eines Agenten deren jeweilige Dauer festlegen muss. Ein Simulationswerkzeug sollte dafür entsprechende Unterstützung anbieten, indem z. B. stochastische Verteilungen zur Verfügung gestellt werden oder die Dauer modellunabhängiger Aktionen wie Wahrnehmung der Umgebung und Bewegung zu einer anderen Position automatisch berechnet wird.

2. Durch die Abbildung von Zeitverbrauch auf eine Folge von Ereignissen sind Agenten zwischen den Ereignissen programmtechnisch passiv. Trotzdem muss gewähr-

leistet sein, dass jeder Agent sofort auf Veränderungen in der Umgebung reagieren kann, da Agenten konzeptuell ständig die Umwelt beobachten. Die naheliegende Lösungsmöglichkeit, Agenten intern mit einer Zeitsteuerung auszustatten und sie so in regelmäßigen Abständen die Umwelt wahrnehmen zu lassen (vgl. z. B. Guessoum 2000), würde die Effizienz der Ereignissteuerung obsolet machen.

Ein weiteres Ziel ist eine flexible, anwendungsunabhängige Unterstützung der Raummodellierung. Natürliche oder sozio-technische Systeme, in denen die Lokalität des Agentenverhaltens die Systemdynamik beeinflusst und die Inhomogenitäten des Raums eine Rolle spielen (Klügl et al. 2004), benötigen eine explizite Repräsentation des Raums. Die in der agentenbasierten Simulation vorherrschenden zweidimensionalen regelmäßigen Gitter sind als alleiniges Raummodell unzureichend, da sie beispielsweise keine angemessene Modellierung von (Verkehrs-)Netzen zulassen. Ein flexibles Raummodell setzt Lösungen für die folgenden Aufgabenstellungen voraus:

1. Der Zugriff auf die räumliche Umgebung (Wahrnehmung, Manipulation von Objekten, Bewegung) ist idealerweise unabhängig vom verwendeten Raummodell. Auf diese Weise kann das Verhalten der Agenten ebenfalls unabhängig vom Raummodell spezifiziert werden, was einen einfachen Austausch bestimmter Raummodelle ermöglicht.

2. Bewegung im Raum ist in vielen Anwendungsbereichen ein zentrales Modellelement; in ökologischen Modellen stellt sie z. B. die elementare Methode dar, mit der Individuen auf Veränderungen in der Umgebung reagieren (Lamberson 2002). Ein Simulationswerkzeug sollte daher Bewegung im Raum auf relativ hoher Abstraktionsebene unterstützen, indem beispielsweise eine raummodell-unabhängige Methode *move* zur Verfügung gestellt wird, die einen Agenten an eine spezifizierte Position bewegt. Damit ist ein Modellierer nicht darauf angewiesen, die Bewegung aus primitiven Befehlen (Löschen von aktueller Position, Eintragen an neuer Position, Aktualisierung der im Agenten gespeicherten Koordinaten) selbst zusammenzusetzen, wie es in verfügbaren ABMS-Werkzeugen immer noch üblich ist (Railsback et al. 2006, S. 614).

3. Im Zusammenhang mit der Bewegung im Raum ist darüber hinaus zu gewährleisten, dass die Aktualisierung des Umgebungszustands – und damit auch die aktuellen Positionen der Agenten – trotz asynchroner Aktualisierung der Agenten effizient gehalten wird, d. h. nicht vor jedem Zugriff durch einen Agenten aktualisiert werden muss.

1.2. Überblick über die Arbeit

Die Arbeit gliedert sich wie folgt: Kapitel 2 stellt die methodischen Grundlagen für die Modellierung von Agenten in Raum und Zeit bereit. Neben einer Diskussion der Zeit- und Raumrepräsentation in der Simulation wird in den Bereich der Multiagentensysteme eingeführt. Um die agentenorientierte Simulation als Weltsicht der diskreten Simulation etablieren zu können, werden in Kapitel 3 zunächst die klassischen Weltsichten vorgestellt

und ihr Bezug zum Agentenkonzept untersucht. Daran anschließend werden detailliert verschiedene Aspekte der agentenbasierten Simulation betrachtet, bevor abschließend die agentenorientierte Weltsicht definiert wird.

Aufbauend auf dieser Definition wird in Kapitel 4 ein Konzept entwickelt für ein Simulationswerkzeug, das die agentenorientierte Weltsicht umsetzt. Dies bedeutet die Erstellung von Modellen zu unterstützen, deren Struktur ein Multiagentensystem bildet und deren dynamisches Verhalten ereignisgesteuert abläuft. Besonderer Schwerpunkt des Werkzeugs ist es, die explizite Repräsentation einer räumlichen Umgebung sowie die Bewegung von Agenten im Raum zu unterstützen.

Kapitel 5 beschreibt die Realisierung des Konzepts im Framework für agentenorientierte Modellierung und Simulation (FAMOS). Dieses baut auf dem am Fachbereich Informatik der Universität Hamburg entwickelten objektorientierten Framework für diskrete Simulation DESMO-J auf und erweitert es um Funktionalität zur Erstellung von Multiagentenmodellen wie leistungsfähige Konstrukte zur Modellierung des Raums sowie des Verhaltens von Agenten einschließlich der Bewegung im Raum. Die Nutzung des Frameworks und dessen flexible Zeit- und Raumrepräsentation werden anhand zweier klassischer agentenbasierter Modelle demonstriert: Schellings (1978) Segregationsmodell und Epstein und Axtells (1996) *Sugarscape*-Modell.

Über diese eher einfachen Modelle hinaus konnte FAMOS im Rahmen eines Forschungsprojekts in einer umfangreichen, praxisrelevanten Anwendung eingesetzt und evaluiert werden (siehe Kapitel 6). Gegenstand des Projekts war es, die betriebswirtschaftlichen, ökologischen und sozialen Auswirkungen möglicher neuer Organisationsformen für Stadtkurierdienste im voraus abzuschätzen. Hierzu bietet sich die Simulation auf Basis eines agentenorientierten Modells an, da die Arbeitsabläufe innerhalb eines Kurierdienstes durch geringe zentrale Kontrolle sowie hohe Autonomie und Eigenverantwortlichkeit der beteiligten Akteure geprägt sind. Zudem ist es notwendig, die Bewegung der Kuriere auf dem städtischen Verkehrsnetz explizit im Modell abzubilden, da diese das Verhalten der Kuriere entscheidend beeinflusst.

Kapitel 7 fasst abschließend die Ergebnisse der Dissertation zusammen und nennt Ansatzpunkte für Erweiterungen, die sich aus der vorliegenden Arbeit ergeben.

2. Grundlagen

In diesem Kapitel werden die zum Verständnis der Arbeit notwendigen Grundlagen gelegt. Vorausgesetzt wird eine Vertrautheit mit den elementaren Begriffen und Methoden der diskreten Simulation, wie sie in einschlägigen Lehrbüchern (z. B. Page 1991; Banks et al. 1999; Law und Kelton 2000; Page und Kreutzer 2005) erläutert sind. Da der Begriff „Simulation“ in zwei Bedeutungen verwendet wird, soll er dennoch hier kurz beleuchtet werden: Im *engeren* Sinne bedeutet Simulation die Durchführung von Experimenten an einem Modell, das an die Stelle des Originalsystems tritt. Während eines solchen Experiments wird das Verhalten des Modells erzeugt, indem schrittweise eine Folge von Zuständen berechnet wird. Der Umfang dieser Berechnung erfordert in der Regel den Einsatz von Computern, so dass das Modell als ausführbares Programm (Computermodell) vorliegen muss. Im *weiteren* Sinne umfasst der Begriff Simulation nicht nur die Nutzung, sondern auch die Erstellung des Modells; er wird also abkürzend für den gesamten Prozess der Modellbildung und Simulation verwendet.

In dieser Arbeit ist durchgängig der weitere Sinn von Simulation gemeint; gerade in zusammengesetzten Bezeichnungen wie „agentenbasierte Simulation“ ist ausdrücklich die Komponente der Modellbildung mit einbezogen. Wenn nicht anders angegeben, ist Simulation daher im Kontext dieser Arbeit als „Modellierung und Simulation dynamischer Systeme mit Hilfe von Computermodellen“ zu lesen.

2.1. Zeitrepräsentation in der Simulation

Time exists solely to prevent things from happening at once.
– unknown

Bei der Simulation dynamischer Systeme werden Systeme betrachtet, deren Zustand sich in Abhängigkeit von der Zeit ändert. Das Simulationsmodell als Repräsentation eines solchen Systems muss diese Zustandsänderungen in geeigneter Weise widerspiegeln; auch das Modell wird demnach seinen Zustand zeitabhängig ändern. Die Zeit ist somit ein Aspekt des Systems, der im Modell zu berücksichtigen ist.

2.1.1. Die verschiedenen Zeitbegriffe

So wie bei der Modellierung allgemein für die Systemstruktur, d. h. die relevanten Systemelemente und ihre Beziehungen untereinander, eine geeignete Abbildung zu finden ist, so gilt dies bei dynamischen Systemen auch für das Systemverhalten, d. h. die Änderung des Systemzustands in Abhängigkeit von der Zeit. Drei verschiedene Zeitbegriffe müssen unterschieden werden, die den drei wesentlichen Bestandteilen System, Modell und Experiment zugeordnet werden können:

- Die *wahre Zeit* (engl. *real time* oder *physical time*) ist die Zeit des zu simulierenden dynamischen Systems. Der in der Simulation betrachtete Zeitausschnitt kann in der Vergangenheit oder in der Zukunft liegen (wenn die Gegenwart als einzelner Zeitpunkt aufgefasst wird). Die wahre Zeit verläuft kontinuierlich.

- Die *Simulationszeit* (engl. *virtual time* oder *simulation time*) ist die modellierte wahre Zeit. Sie wird auch als Zeitbasis des Modells bezeichnet. Von der wahren Zeit wird eine Abbildung auf die Simulationszeit benötigt. Diese Abbildung kann – wie bei der Modellierung der Systemelemente – komplexitätsreduzierend sein und beispielsweise Zeit-Intervalle auf Zeitpunkte abbilden oder nur einzelne Zeitpunkte berücksichtigen. Die Simulationszeit ist daher nicht zwingend kontinuierlich, sondern kann auch diskret sein. Streng genommen bedingt die Verwendung von Digitalrechnern immer eine Diskretisierung kontinuierlicher Prozesse, so dass die Simulationszeit höchstens quasi-kontinuierlich verlaufen kann.

- Die *Ausführungszeit* (engl. *computation time* oder *wallclock time*), auch Rechenzeit genannt, ist die Zeit, die bei der eigentlichen Simulation, d. h. der Ausführung des Computermodells im Rahmen eines Experiments, benötigt wird. Diese Zeit ist wie die wahre Zeit kontinuierlich. Die Dauer eines Experiments muss aber keinesfalls mit der Länge des simulierten Zeitabschnitts identisch sein; noch muss die Ausführungszeit jeder einzelnen, simulierten Aktivität mit der tatsächlichen Aktivitätsdauer übereinstimmen.[1] Im Gegenteil, ein Vorteil der Simulation ist gerade die Möglichkeit, Systeme im Zeitraffer oder auch in Zeitlupe betrachten zu können.[2]

Abbildung 2.1 auf der nächsten Seite verdeutlicht den Zusammenhang zwischen den drei Zeitbegriffen. Dargestellt sind einzelne Zeitpunkte, an denen Änderungen im System eintreten (sog. Ereignisse). Bei der Abbildung der wahren Zeit auf die Simulationszeit müssen die Verhältnisse zwischen einzelnen Zeitpunkten gewahrt bleiben: Ist der Abstand zwischen den System-Zeitpunkten t_i und t_k doppelt so groß wie der zwischen t_i und t_j, so muss dies auch für die entsprechenden Modell-Zeitpunkte t'_i, t'_j und t'_k gelten. Anders ausgedrückt, besteht eine lineare Beziehung zwischen den Zeitdauern in wahrer Zeit und Simulationszeit:

$$\Delta T_w = a \cdot \Delta T_s \tag{2.1}$$

wobei ΔT_w ein Zeitintervall in wahrer Zeit, ΔT_s das entsprechende Intervall in Simulationszeit und a einen Skalierungsfaktor bezeichnen. Der Skalierungsfaktor bestimmt, wie eine Einheit Simulationszeit in physikalischen Zeit-Einheiten zu interpretieren ist.

Gleiches betrifft die Ordnung der Zeitpunkte: Aus $t_i < t_j$ im System folgt $t'_i < t'_j$ im Modell. Damit ist sichergestellt, dass Ursache-Wirkungsbeziehungen von Ereignissen erhalten bleiben.[3]

[1] Dieser Spezialfall wird als Echtzeit-Simulation bezeichnet (vgl. z. B. Fujimoto 1998; Crosbie 1982) und kommt u. a. in virtuellen Testumgebungen wie Flugsimulatoren zum Einsatz.

[2] In diesem Zusammenhang kann neben der Ausführungszeit noch die *Präsentationszeit* unterschieden werden, d. h. die Zeit, die für die Darstellung der Ergebnisse, beispielsweise in der Form simulationsbegleitender Animation, benötigt wird (Wittmann 2011).

[3] Zum Reihenfolgeproblem bei gleichzeitigen Ereignissen siehe Abschnitt 2.1.3.

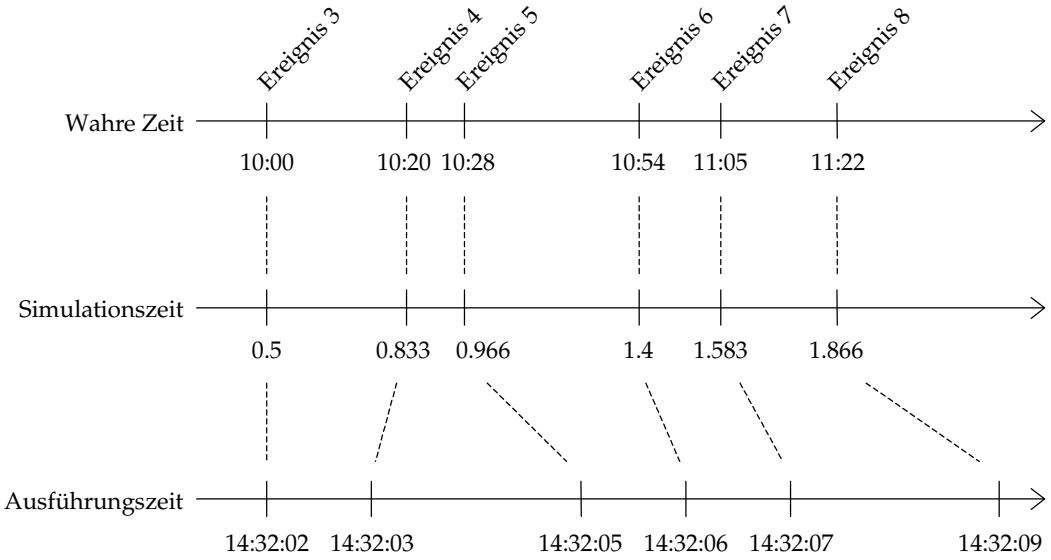

Abbildung 2.1.: Zusammenhang der Zeitbegriffe in der Simulation.

Die Abbildung der Simulationszeit auf die Ausführungszeit geschieht durch die Ausführung des Computermodells. Auch diese Abbildung muss der Kausalitätsregel genügen, d. h. in Simulationszeit aufeinanderfolgende Ereignisse oder Aktivitäten müssen auch nacheinander ausgeführt werden, so dass die simulierte Zukunft nicht die Vergangenheit beeinflussen kann. Im Gegensatz zur Abbildung von wahrer Zeit auf Simulationszeit wird die Abbildung von Simulations- auf Ausführungszeit in der Regel jedoch nicht linear sein. Die Dauer der Berechnung eines Ereignisses oder einer Aktivität ist abhängig sowohl von der Komplexität des Codes als auch der benutzten Hardware und der momentanen Rechnerauslastung. Selbst die exakte Wiederholung eines Experimentes auf demselben Rechner gewährleistet daher nicht, dass die Rechenzeiten identisch sind. Dies ist jedoch nur dann problematisch, wenn die Ausführungszeit direkt die Simulationszeit beeinflusst; so bei interaktiven Echtzeit-Simulationen, in denen bestimmte Antwortzeiten eingehalten werden müssen (Crosbie 1982), oder in der ereignis-diskreten Simulation, wenn die Rechenzeit verwendet wird, um die Dauer von Aktivitäten abzuschätzen (vgl. Abschnitt 2.1.3).

2.1.2. Zeitfortschritt und Aktualisierung in der diskreten Simulation

Die Art des zu modellierenden Systems beeinflusst oft, ob die Simulationszeit kontinuierlich oder diskret gewählt wird. Zwar sind nicht alle Systeme ausschließlich kontinuierlich oder ausschließlich diskret, doch können sie im Allgemeinen eindeutig einer der beiden Kategorien zugeordnet werden, da eine der beiden Arten von Zustandsänderung überwiegt. Es ist jedoch nicht zwingend erforderlich, ein diskretes System durch ein diskretes Modell und ein kontinuierliches System durch ein kontinuierliches Modell abzubilden.

Dies hängt von den Zielen und Anforderungen der durchgeführten Simulationsstudie ab. Bei der Modellierung des Verkehrsflusses auf einer Straße ist z. B. ein diskretes Modell zu wählen, wenn individuelle Fahrzeuge unterschieden werden müssen (mikroskopisches Modell). Zustandsänderungen wären hier beispielsweise das Hineinfahren oder Hinausfahren eines Fahrzeugs aus dem betrachteten Straßenabschnitt. Ein kontinuierliches Modell würde dagegen das Makro-Phänomen des Verkehrsflusses in Form von Differentialgleichungen beschreiben.

Im folgenden werden ausschließlich diskrete Modelle betrachtet. Grundannahmen sind hier, dass Zustandsänderungen nur zu bestimmten Zeitpunkten eintreten und dabei konzeptuell zeitverzugslos[4] ablaufen. Zwischen diesen sogenannten Ereignissen bleibt der Systemzustand konstant. Zeitverbrauchende Aktivitäten müssen daher in geeigneter Weise auf Folgen von Ereignissen abgebildet werden, z. B. auf ein Start- und ein Ende-Ereignis.

In der zeitdiskreten Simulation wird in der Regel eine Zeitbasis $T_s \subset \mathbb{R}_0^+$ verwendet, d. h. die Simulationszeitpunkte werden als reelle Zahlen repräsentiert und die Simulation beginnt mit dem Zeitpunkt $t_0 = 0.0$. Manchmal kommt auch eine Zeitbasis $T_s \subset \mathbb{N}$ zum Einsatz. Die Simulationsuhr, die den jeweils aktuellen Simulationszeitpunkt vorhält, wird von Ereigniszeitpunkt zu Ereigniszeitpunkt fortgeschaltet und zu jedem Ereigniszeitpunkt wird der Modellzustand aktualisiert. Dabei müssen Ereignisse bezüglich einer Entität in monoton steigender Folge ihres Auftretens in Simulationszeit behandelt werden, d. h. die Kausalitätsregel ist zu beachten:

$$e_1 \rightarrow e_2 \Rightarrow t_a(e_1) < t_a(e_2) \tag{2.2}$$

wobei e_i ein Ereignis, $\rightarrow$ mögliche kausale Beeinflussung und $t_a(e_i)$ die Ausführungszeit eines Ereignisses bezeichnen[5]. Dies stellt insbesondere für die verteilte Simulation ein Problem dar und hat zur Entwicklung verschiedener Synchronisationsmechanismen geführt, die sich in die Kategorien konservativ und optimistisch einteilen lassen (siehe z. B. Fujimoto 1990).

Zwei verschiedene Möglichkeiten, die Simulationszeit fortzuschreiben, und zwei verschiedene Möglichkeiten, die Zustandsgrößen zu aktualisieren, sind zu unterscheiden. Der Zeitfortschritt kann

1. in *regelmäßigen* Intervallen erfolgen, d. h. es existiert ein festes Δt, das jeweils zum aktuellen Zeitpunkt addiert wird, um den nächsten Zeitpunkt zu erhalten; hier stellt sich das Problem, ein geeignetes Δt zu finden;

2. in *unregelmäßigen* Intervallen erfolgen, d. h. Δt ist variabel. In diesem Fall muss die Frage gelöst werden, wie der nächste Ereigniszeitpunkt bestimmt wird.

Die Aktualisierung des Modellzustands kann analog dazu

1. *synchron* erfolgen, d. h. alle Zustandsgrößen (Entitäten) werden zum selben Zeitpunkt aktualisiert; oder sie kann

[4] d. h. sie verbrauchen keine Simulationszeit, unabhängig davon, wieviel Ausführungszeit benötigt wird, um die Zustandsänderung zu berechnen.

[5] Diese Definition erfolgt in Anlehnung an die *Clock Condition* in (Lamport 1978, S. 560)

2. *asynchron* erfolgen, d. h. nur diejenige Zustandsgröße, die mit dem aktuellen Ereignis verknüpft ist, wird aktualisiert.

Die *zeitgesteuerte* Simulation verbindet regelmäßige Zeitfortschreibung mit synchroner Zustandsaktualisierung. Der nächste betrachtete Zeitpunkt ergibt sich direkt aus dem aktuellen Zeitpunkt und dem fixen Zeitinkrement: $t_{i+1} = t_i + \Delta t$. Nur zu diesen Zeitpunkten wird eine Änderung des Systemzustands in Betracht gezogen. Ereignisse, deren tatsächlicher Zeitpunkt innerhalb eines solchen Intervalls liegt, werden behandelt, als ob sie am Ende des Intervalls stattfänden. Wenn Δt zu groß gewählt ist, kann dies die Genauigkeit des Modells beeinträchtigen. Demgegenüber wird bei sehr kleinen Werten von Δt die Simulation ineffizient, da zu vielen Zeitpunkten keinerlei Änderung des Systems eintritt und diese Tatsache vergeblich überprüft werden muss.

Andererseits ist die Zeitsteuerung einfach zu realisieren und als Approximation bei der Simulation kontinuierlicher dynamischer Systeme verwendbar. Außerdem ist sie geeignet für alle Systeme, für die vernünftigerweise angenommen werden kann, dass Änderungen tatsächlich zu einem der Zeitpunkte $n \cdot \Delta t$ mit $n = 0, 1, 2...$ eintreten. Ein Beispiel sind Wirtschaftssysteme, für die Daten nur auf jährlicher Basis vorliegen; hier ist die Wahl von $\Delta t = 1$ Jahr natürlich (vgl. Law und Kelton 2000, S. 94).

Im Gegensatz dazu verbindet die *ereignisgesteuerte* Simulation unregelmäßigen Zeitfortschritt mit asynchroner Aktualisierung des Zustands. Da Zustandsänderungen nur zu Ereigniszeitpunkten auftreten und die Simulationsuhr von Ereigniszeitpunkt zu Ereigniszeitpunkt springt, werden inaktive Phasen des Systems auf effiziente Weise übergangen. Bei zeitgesteuerter Simulation ist dies nicht möglich, es sei denn, die Dauer inaktiver Phasen ist immer $\leq \Delta t$. Darüber hinaus können die Ereigniszeitpunkte – zumindest bei Verwendung einer reellwertigen Zeitbasis – beliebig gewählt werden, so dass die Ereignisse jeweils zum „richtigen“ Zeitpunkt stattfinden. Die Diskretisierung der Simulationszeit wird damit von den Abläufen im modellierten System selbst vorgegeben anstatt von einer künstlichen Taktrate. Variable Zeitinkremente sind insbesondere dann natürlicher, wenn die modellierten Entitäten auf unterschiedlichen Zeitskalen agieren.

Die ereignisgesteuerte Simulation ist somit flexibler, genauer und effizienter als die zeitgesteuerte Simulation. Durch Erzeugen eines speziellen „Aktualisierungsereignisses“ in Abständen von Δt Zeiteinheiten kann mit Hilfe der Ereignissteuerung eine Zeitsteuerung realisiert werden; die zeitgesteuerte Simulation kann daher als Spezialfall der ereignisgesteuerten Simulation aufgefasst werden.

2.1.3. Probleme der diskreten Simulationszeit

Die Diskretisierung der Simulationszeit ist zwar gerade in Form der Ereignissteuerung vorteilhaft, da u. a. überflüssige Zustandsberechnungen vermieden werden, doch birgt sie auch Probleme. Beim ereignisgesteuerten Ansatz wird der jeweils nächste Ereigniszeitpunkt benötigt; das Eintreten von Ereignissen muss demnach (rechtzeitig) im voraus bestimmt werden. Für zeitverbrauchende Aktivitäten muss nicht nur der Start-, sondern auch der Endzeitpunkt bekannt sein, d. h. die Dauer von Aktivitäten muss bestimmt werden. Beim zeitgesteuerten Ansatz sind dagegen die einzelnen Ereigniszeitpunkte durch die Taktrate vorgegeben; hier muss allerdings eine geeignete Taktrate gewählt werden.

Und in beiden Ansätzen ist auf das Problem gleichzeitig eintretender Ereignisse, die ggf. sogar in Abhängigkeit voneinander stehen, zu reagieren.

2.1.3.1. Wahl der Taktrate

Bei der zeitgesteuerten Simulation, zu der im Prinzip auch die quasi-kontinuierliche Simulation gehört, wird die Simulationsuhr in festen Schritten fortgeschrieben. Die Taktrate Δt ist die kleinste unterscheidbare Zeiteinheit, sie gibt die Auflösung der Simulationszeit vor. Aus Effizienzgründen ist ein möglichst großes Δt anzustreben, um unnötige Neuberechnungen des Zustands zu vermeiden. Wird Δt jedoch zu groß gewählt, kann es zu Verfälschungen führen: Änderungen, die eigentlich innerhalb eines Zeitintervalls eintreten, werden auf das Ende des Intervalls verschoben. Zusätzliche Probleme können dann eintreten, wenn bei dieser Verschiebung die Reihenfolge des Eintritts nicht beachtet wird (s. u.). Um solche Simulationsartefakte zu vermeiden, könnte auf Effizienz verzichtet und ein ausreichend kleines Δt gewählt werden. Ausreichend klein bedeutet, dass in jedem Intervall immer nur höchstens eine Zustandsänderung eintreten kann. Folglich muss der kleinste Abstand zwischen zwei Zustandsänderungen des simulierten Systems als Taktrate verwendet werden. Dieser ist jedoch oft vorher nicht bekannt. Es kann daher nötig sein, mehrere Simulationsläufe mit verschiedenen Werten für Δt durchzuführen, bis eine geeignete Taktrate gefunden ist.

Zu bedenken ist, dass die Taktrate nur ein technisches Hilfsmittel darstellt, um das dynamische Verhalten des simulierten Systems als zeitliche Abfolge von Modellzuständen zu berechnen. In vielen Modellen, insbesondere im Zusammenhang mit Zellularautomaten (siehe Abschnitt 2.2.3), wird sie jedoch implizit verwendet, um die Dauer von Aktivitäten zu bestimmen. Zwar ist dieses Vorgehen nicht *per se* inkorrekt, doch sollte es explizit gemacht und in die Modelldokumentation aufgenommen werden.

2.1.3.2. Gleichzeitigkeit

Um das dynamische Verhalten des modellierten Systems korrekt wiederzugeben, muss die Simulationssteuerung sicherstellen, dass die zeitliche Ordnung der Zustandsänderungen eingehalten wird. Dies ist solange unproblematisch, wie die einzelnen Ereigniszeitpunkte unterscheidbar sind. Gleichzeitig eintretende Ereignisse sind jedoch nicht ausgeschlossen und sollten von der Simulationssteuerung entsprechend behandelt werden. Da sie auf einem sequentiell arbeitenden Rechner nur nacheinander ausgeführt werden können, muss in diesem Fall eine geeignete Serialisierung gefunden werden. Hierfür ist in Simulationssoftware meist ein Prioritätsmechanismus vorgesehen (vgl. Page 1991, S. 28), der dem Modellentwickler die Möglichkeit bietet, die Serialisierung modellspezifisch zu beeinflussen. Eine echte parallele Ausführung gleichzeitiger Ereignisse ist nur bei einer Verteilung auf mehrere Prozessoren möglich. Diese setzt aber in der Regel eine entsprechende Implementation des Modells für Parallelrechner voraus (parallele Simulation, Fujimoto 1990) oder den Einsatz von (Simulations-)Software, welche die verteilte Ausführung von Modellkomponenten in einem Rechnernetz unterstützt (verteilte Simulation,

Bachmann 2003).[6] Im folgenden steht die sequentielle Simulation im Vordergrund.

Bei der Serialisierung ist darauf zu achten, dass durch die Reihenfolge, in der die parallelen Ereignisse abgearbeitet werden, keine Simulationsartefakte entstehen, indem beispielsweise die Kausalitätsregel verletzt oder eine Entität stets vor einer anderen Entität aktualisiert wird. Ein relativ einfaches Hilfsmittel, dies zu vermeiden oder zumindest die resultierenden Fehler zu minimieren, besteht darin, die parallelen Ereignisse immer in zufälliger Reihenfolge abzuarbeiten. Dieses Verfahren wird insbesondere im Zusammenhang mit synchroner Aktualisierung verwendet, da hier zu jedem Zeitpunkt meist viele Ereignisse vorliegen, die so auf einfache Weise serialisiert werden können. Allerdings ist diese Strategie nur sinnvoll anzuwenden auf zeitgleiche Ereignisse, die voneinander unabhängig sind.

Der bereits erwähnte Prioritätsmechanismus stellt eine andere Möglichkeit dar: hier zeichnet der Modellierer die verschiedenen Ereignistypen mit Prioritäten aus, die bei der Einsortierung der Ereignisse in die Ereignisliste zusätzlich zum Zeitpunkt berücksichtigt werden. Für den Fall, dass im Modell die Abhängigkeit von (nicht notwendigerweise zeitgleichen) Ereignissen bekannt ist, bieten manche Simulatoren außerdem an, ein Ereignis explizit vor oder hinter einem anderen Ereignis in die Ereignisliste einzusortieren. Beide Vorgehensweisen eignen sich jedoch in der Regel nur für Systeme mit einer überschaubaren Menge von Ereignistypen und relativ selten auftretenden parallelen Ereignissen.

In allen beschriebenen Fällen stellt sich darüber hinaus die Frage, ob die Zustandsänderung, die ein Ereignis auslöst, sofort wirksam wird oder erst, nachdem alle parallelen Ereignisse abgearbeitet sind. Ersteres bedeutet, dass der Modellzustand inkrementell aktualisiert wird und die einzelnen zeitgleich aktiven Entitäten auf leicht unterschiedlichen Zuständen operieren. Dies ist Standard in der ereignisgesteuerten Simulation. Letzteres realisiert ein zweiphasiges Schema: In einer ersten Phase der Aktualisierung wählen alle beteiligten Entitäten basierend auf dem (noch) aktuellen Modellzustand ihre nächste Handlung aus; in der zweiten Phase werden diese Handlungen dann tatsächlich durchgeführt und verändern den Zustand. Dieses Verfahren stellt sicher, dass alle simultanen, lokalen Zustandsänderungen auf demselben globalen Modellzustand operieren, und wird u. a. bei Zellularautomaten eingesetzt (siehe Abschnitt 2.2.3).

2.1.3.3. Dauer von Aktivitäten

In der ereignisgesteuerten Simulation wird die Simulationsuhr immer auf den nächsten bevorstehenden Ereigniszeitpunkt weitergeschaltet. Um ein Zurückspringen in Simulationszeit zu verhindern, müssen alle in der Zukunft liegenden Ereignisse rechtzeitig vorher bestimmt werden. Für zeitverbrauchende Aktivitäten muss daher spätestens zu ihrem Startzeitpunkt ihre jeweilige Dauer bekannt sein.[7] Aus der Dauer lässt sich dann der Zeit-

[6] Die Verteilung von Modellkomponenten auf verschiedene Rechner schließt prinzipiell nicht aus, dass innerhalb einer Komponente gleichzeitige Ereignisse auftreten. Das hängt von der Aufteilung des Modells in Komponenten und deren Interaktionen untereinander ab.

[7] Zu unterscheiden von Aktivitäten (engl. *activities*) sind sogenannte Verzögerungen (engl. *delays*, siehe Banks 1998, S. 8), bei denen die Dauer zu Beginn nicht bekannt ist, da sie nicht von der Handlung selbst abhängt, sondern von „äußeren“ Ereignissen. Ein solches Warten auf unbestimmte Zeit tritt in der Regel im Zusammenhang mit der Interaktion von Entitäten auf, beispielsweise bei der Konkurrenz um beschränkte Ressourcen.

punkt des Ende-Ereignisses berechnen, so dass das entsprechende Ende-Ereignis rechtzeitig in die Ereignisliste eingetragen werden kann. Für die Bestimmung der Aktivitätsdauer gibt es verschiedene Möglichkeiten (vgl. z. B. Banks 1998, S. 8), die je nach Modellkontext unterschiedlich geeignet sind:

konstant Einer Aktivität wird ein fester Wert als Dauer zugewiesen. Gilt dieser konstante Wert für alle Entitäten, so ist die Aktivitätsdauer nicht nur unabhängig von aktuellem Zustand und Startzeitpunkt, sondern auch von der ausführenden Entität. Wenn jeder im System möglichen Aktivität im Modell die gleiche Dauer zugewiesen wird, bedeutet das konzeptuell, dass die Dauer einer Handlung unabhängig von der Art der Handlung ist. Beginnen Aktivitäten nie zeitlich versetzt, erhält man technisch gesehen eine Zeitsteuerung mit der einen konstanten Handlungsdauer als Δt.

empirisch Liegen ausreichend detaillierte Daten über das System vor, können diese im Modell direkt verwendet werden, indem sie beispielsweise zu Beginn oder während der Simulation aus einer Datei eingelesen und der jeweiligen Aktivität zugeordnet werden.

stochastisch Weitaus häufiger ist der Fall, dass die Dauer einer Aktivität zufällig gewählt wird. Wenn Kenntnisse über durchschnittliche Dauer und Varianz der Werte vorhanden sind, wird die Auswahl eines Wertes einer geeigneten stochastischen Verteilung folgen. Auch detaillierte empirische Daten lassen sich durch entsprechende stochastische Verteilungen ersetzen, um Unabhängigkeit von konkreten Daten zu erlangen (vgl. Page 1991, S. 117).

zustands- oder zeitabhängig Hängt die Aktivitätsdauer im System vom aktuellen Zustand oder dem Zeitpunkt ihres Beginns ab, sollte dies auch im Modell umgesetzt werden. In der Regel wird die Dauer dann über eine mehr oder weniger komplexe Funktion berechnet. So ergibt sich z. B. die Zeitspanne, die für eine Bewegung benötigt wird, aus dem Quotienten der zurückgelegten Distanz und der verwendeten Geschwindigkeit.

aufwandsabhängig Liegen aus dem modellierten System keine Daten vor oder handelt es sich um eine Echtzeit-Simulation, kann die Dauer einer Aktivität abgeschätzt werden anhand des rechnerischen Aufwands, der zu ihrer Durchführung nötig ist. Die einfachste Möglichkeit besteht hier darin, die benötigte Rechenzeit zur Ausführung des entsprechenden Codefragments auf die Simulationszeit abzubilden. Nachteilig daran ist, dass die Simulationsergebnisse rechnerabhängig werden, da auf unterschiedlichen Plattformen oder gar nur bei unterschiedlicher Auslastung desselben Rechners andere Werte für die Rechenzeit bei derselben Aktivität zum selben Simulationszeitpunkt auftreten können. Um Rechnerunabhängigkeit zu gewährleisten, sollte der Aufwand anhand der Anzahl der auszuführenden Instruktionen berechnet werden (Anderson 1997).

2.2. Raumrepräsentation in der Simulation

Stranger. *Pooh! what do you know of Space? Define Space.*
I. *Space, my Lord, is height and breadth indefinitely prolonged.*
Stranger. *Exactly: you see you do not even know what Space is.*
– Edwin A. Abbott, Flatland (1884)

Raum und Zeit sind gleichwertige Eigenschaften von Ereignissen und Prozessen: Alles geschieht im raumzeitlichen Kontinuum. Obwohl daher Modelle, welche reale Prozesse abbilden, sowohl Zeit als auch Raum repräsentieren müssten, werden oft die räumliche oder die zeitliche Dimension ausgeblendet, um die Modelle einfacher und leichter verständlich zu machen. Viele Simulationsmodelle besitzen daher kein oder nur ein implizites Raummodell (vgl. Abschnitt 2.2.1). Für die explizite Repräsentation des Raums wird neben der vorherrschenden Verwendung einfacher Gitter in letzter Zeit verstärkt auf Geographische Informationssysteme zurückgegriffen (Abschnitt 2.2.2). Daneben existiert mit den in Abschnitt 2.2.3 beschriebenen Zellularautomaten eine Gruppe sowohl zeitlich als auch räumlich diskreter Modelle, welche im Zuge der Komplexitätsforschung erneutes Interesse gefunden haben und in individuen- bzw. agentenbasierten Modellen heute zur expliziten Repräsentation der räumlichen Umgebung eingesetzt werden.

2.2.1. Explizite versus implizite Raummodellierung

Modellierung ist prinzipiell mit Idealisierung und Abstraktion vom betrachteten realen System verbunden (Page 1991, S. 5). In Abhängigkeit von der zu untersuchenden Fragestellung entscheidet der Modellierer, welche Systemkomponenten und -eigenschaften als unwesentlich anzusehen sind und daher im Modell nicht berücksichtigt werden müssen. In vielen Fällen betrifft dies die räumliche Dimension des betrachteten Systems. Hierbei gibt es zwei Möglichkeiten:

1. Der Raum wird ganz vernachlässigt. Diese Vereinfachung ist die Regel, wenn aufgrund der gewählten Systemgrenzen oder Zeitskalen weder räumliche Strukturen noch die für Bewegung im Raum benötigte Zeit für das Modell relevant sind. So kann beispielsweise bei der Simulation von Computernetzwerken von den physikalischen Positionen der einzelnen Rechner problemlos abstrahiert werden.

2. Der Raum wird auf andere Variablen wie die Zeit oder Kosten abgebildet. Dies ist immer dann möglich, wenn nur die Überbrückung des Raums eine Rolle spielt, nicht jedoch dessen zugrundeliegende Struktur. Als Beispiel sei die Simulation von Betriebsabläufen durch Bedien-/Wartesysteme genannt: Der Transport eines Auftrags oder Werkstücks zwischen zwei Bedienstationen wird ausschließlich als Zeitintervall repräsentiert, welches für den Transport benötigt wird.

   Dies ist auch der Fall in baustein-orientierten graphischen Simulationsumgebungen wie z. B. ExtendSim (Krahl 2002, 2008) oder Arena (Kelton et al. 2010). Zwar können die aus einer Bibliothek ausgewählten Bausteine auf der graphischen Oberfläche dem abgebildeten System entsprechend räumlich angeordnet und verknüpft werden, doch wird Raumüberbrückung intern weiterhin auf Zeitverbrauch abgebildet.

> Die graphische Darstellung dient ausschließlich der Visualisierung und geht nicht als explizites Raummodell in das so spezifizierte Simulationsmodell ein.

Oft spielt jedoch auch die gewählte Simulationsmethode eine Rolle. In der kontinuierlichen Simulation ist ein Simulationsmodell als System von Differentialgleichungen formuliert, welche die Veränderung von Zustandsvariablen über die Zeit beschreiben. Obwohl oft für die Modellierung komplexer physikalischer oder biologischer Prozesse mit Raumbezug eingesetzt wie Klima- bzw. Wettervorhersage oder die Analyse von Ökosystemen, wird der Raum in der Regel nicht berücksichtigt. Stattdessen wird von der Annahme ausgegangen, dass alle Entitäten die gleiche Umgebung erleben und Interaktionen zwischen ihnen durch keinerlei räumliche Strukturen beschränkt sind, d. h. mit gleicher Wahrscheinlichkeit auftreten (die sogenannte *perfect mixing assumption*). Dies ermöglicht die Verwendung gewöhnlicher Differentialgleichungen (Durrett 1999). Interaktion wird einfach auf eine Rate abgebildet; in den bekannten Räuber-Beute-Modellen werden z. B. nur die für die Räuber-Population erfolgreichen Interaktionen mit der Beute-Population in deren Sterberate berücksichtigt.

Während solche abstrakten Modelle für viele Problemstellungen adäquat sind, haben sie sich beispielsweise in der Ökologie als unzureichend erwiesen. Wenn die räumliche Verteilung von Individuen Einfluss auf das Systemverhalten hat, weil Interaktionen nur zwischen benachbarten Individuen stattfinden und somit lokal beschränkt sind, liefern räumlich explizite Modelle andere Resultate als Modelle ohne Raummodell. Differentialgleichungssysteme können um ein kontinuierliches Raummodell erweitert werden, indem sie durch partielle Differentialgleichungen ersetzt werden (Gross 1996). Eine andere Möglichkeit besteht darin, den Raum in verschiedene Kompartimente zu diskretisieren, welche jeweils durch ein Differentialgleichungssystem beschrieben werden und per Ein- und Ausgabekanäle miteinander verbunden sind (vgl. z. B. Lorek 1998, S. 13f). Bei der Modellierung von Populationen führt dies zu sogenannten Metapopulationsmodellen (Hanski und Gilpin 1991), in denen die gesamte Population einer Spezies auf verschiedene Habitate aufgeteilt ist. Die einzelnen Habitate sind entweder voneinander isoliert oder ermöglichen den Austausch einer beschränkten Anzahl von Individuen (Migration).

Die Kritik an makroskopischen Ansätzen in der Ökologie beruhte nicht allein auf deren Verletzung des Lokalitätsprinzips. Die aggregierte Betrachtung einzelner Individuen als Gruppe geht von der Annahme aus, dass alle Individuen einer Population identisch sind oder zumindest, dass interindividuelle Unterschiede in Verhalten und Physiologie irrelevant sind (*homogeneity assumption*). Aus diesen Gründen hat sich heute nach mehr als einem Jahrzehnt der Diskussion ihrer Vor- und Nachteile (vgl. u. a. Huston et al. 1988; Hogeweg und Hesper 1990; DeAngelis und Gross 1992; Judson 1994; Uchmanski und Grimm 1996; Grimm 1999) die individuenbasierte Simulation als Methode in der theoretischen Ökologie etabliert (Grimm und Railsback 2005). Ein individuenbasiertes Modell betrachtet ein System auf der Basis einzelner Individuen, die sich in Eigenschaften und Verhalten unterscheiden können und sich in einem explizit modellierten Raum bewegen. Dieser Raum wird in der Regel diskretisiert und als zwei- oder dreidimensionales Gitter von Zellen repräsentiert.

Ein weiteres Beispiel *par excellence* für ein Anwendungsgebiet mit expliziter Raumrepräsentation ist die Verkehrsmodellierung. Im traditionellen Vier-Stufen-Algorithmus der

Verkehrsprognose werden, wie aus Abbildung 2.2 (Hilty et al. 1998, S. 67) zu entnehmen ist, zwei verschiedene Raummodelle verwendet: Zonen und Verkehrsnetze. Der Algorithmus umfasst eine Sequenz von vier Modellen:

1. Das Verkehrserzeugungsmodell berechnet die ungerichtete Verkehrsnachfrage, d. h. den von jeder Zone ausgehenden Verkehrsbedarf wie Fahrten zu Arbeit, Ausbildung, Einkauf, Erholung oder Güter-Transporte.

2. Das Verkehrsverteilungsmodell ermittelt daraus die gerichtete Verkehrsnachfrage, indem die Fahrten auf mögliche Ziele in den verschiedenen Zonen verteilt werden.

3. Das Verkehrsmittelwahlmodell bestimmt, welche Anteile der Fahrten auf welchen Verkehrsmodus (z. B. motorisierter Individualverkehr, öffentlicher Verkehr) entfallen.

4. Das Umlegungsmodell berechnet schließlich die Verteilung der Verkehrslast auf das entsprechende Wegenetz, d. h. die Fahrten werden auf einzelne Routen „umgelegt“. Hierbei wird angenommen, dass die Fahrer den sogenannten Verkehrswiderstand zu minimieren suchen, eine Größe, die häufig durch die Reisezeit approximiert wird, aber auch subjektive Faktoren wie die Bevorzugung bestimmter Straßentypen beinhalten kann (Ortúzar und Willumsen 1994, S. 294). Das Ergebnis der Verkehrsumlegung sind die Verkehrsstärken auf den einzelnen Verkehrswegen.

Für die Verkehrserzeugung wird das Untersuchungsgebiet in Zonen eingeteilt, um individuelle Haushalte und Betriebe zu aggregieren. Hierfür werden häufig Verwaltungsbezirke herangezogen, es können aber beispielsweise auch Regionen gleicher Reisezeit gewählt werden, so dass in Staugebieten eine feinere Auflösung erzielt wird. Die Zonen werden in der Regel durch einzelne Punkte, sogenannte Zentroide, repräsentiert, welche mit dem Verkehrsnetz über spezielle Konnektoren verbunden sind (Ortúzar und Willumsen 1994, S. 103f). Die Konnektoren bilden die durchschnittlichen Kosten (Zeit, Distanz) für Fahrten von und zu Zielen innerhalb der Zone ab.

Das Verkehrsnetz wird üblicherweise als gerichteter Graph modelliert, dessen Knoten Kreuzungen und dessen Kanten Straßenabschnitte darstellen mit Attributen wie Länge, Geschwindigkeit und Anzahl Spuren. Der Detaillierungsgrad des Netzes ist abhängig von der Datenlage und – damit zusammenhängend – von dem verwendeten Zonensystem: Ein detailliertes Verkehrsnetz, welches alle Straßentypen von Autobahnen bis zu Nebenstraßen berücksichtigt, ist sinnvollerweise nur mit einem ebenfalls feinräumig gegliederten Zonensystem zu kombinieren (Ortúzar und Willumsen 1994, S. 106f).

In ökonomischen und sozialwissenschaftlichen Modellen kann der physikalische Raum oft vom Interaktionsraum unterschieden werden (Parunak et al. 1998). Durch die moderne Kommunikationstechnologie sind Interaktionen zwischen Individuen oder Betrieben nicht mehr durch den physikalischen Raum festgelegt, weshalb dieser in Modellen vernachlässigt werden kann. Dagegen werden die Interaktionen durch explizit repräsentierte soziale Netze bestimmt: Akteure bilden die Knoten, welche über Kanten miteinander verknüpft sind, die soziale oder wirtschaftliche Beziehungen ausdrücken wie z. B. Freundschaft, Nachbarschaft oder Zulieferer/Kunde.

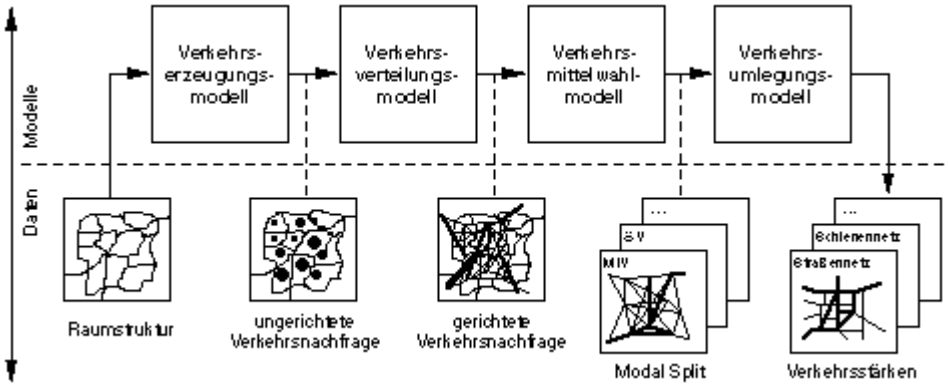

Abbildung 2.2.: Der Vier-Stufen-Algorithmus der Verkehrsprognose

Soziale Netzwerke zeichnen sich durch komplexe Topologien aus, deren wichtigste die sogenannten *Small-World*-Netze (Watts 1999) und skalenfreie Netze (Barabási und Bonabeau 2003) sind. Beide besitzen gegenüber zufälligen Netzen einen hohen Cluster-Koeffizienten, d. h. zwei Knoten, die jeweils über eine Kante mit einem dritten Knoten verbunden sind, sind mit hoher Wahrscheinlichkeit auch miteinander verbunden. In Small-World-Netzen können darüber hinaus die meisten Knoten über relativ kurze Pfade von einem beliebigen anderen Knoten aus erreicht werden. Dies beschreibt das sogenannte Small-World-Phänomen, dass beliebige, einander fremde Personen über einige wenige gemeinsame Bekannte verknüpft werden können. Skalenfreie Netze besitzen einige Knoten (*Hubs*), die über sehr viele Verbindungen zu anderen Knoten verfügen, während der durchschnittliche Knotengrad eher gering ist.

2.2.2. Ansätze aus der Geo-Informatik

Die Geo-Informatik beschäftigt sich mit dem Einsatz von Informatikmethoden zur Verarbeitung und Anwendung von Geodaten, d. h. Daten mit Raumbezug. Hierzu werden geographische Informationssysteme (GIS) verwendet, welche spezielle Funktionen für die Verwaltung, Manipulation und Analyse raumbezogener Daten bereitstellen [8]. Die Raummodellierung ist somit ein Schwerpunkt und es ist naheliegend, diese fachliche Kompetenz für die Simulation räumlich expliziter Prozesse zu nutzen.

2.2.2.1. Räumliche Datenmodelle in Geographischen Informationssystemen

Geodaten beschreiben Objekte der realen Welt durch sowohl räumliche als auch thematische Komponenten:

- Die *Geometrie* legt die absolute Lage auf der Erdoberfläche fest unter Verwendung eines räumlichen Bezugssystems. Dies können z. B. die sphärischen Koordinaten

[8] Für eine Einführung in GIS vgl. z. B. die Lehrbücher (Star und Estes 1990), (Bartelme 2005), (Bill 2010) und (Burrough et al. 2013) oder auch (Gerken und Meyer 1995) und (Worboys und Duckham 2004) für speziell auf Informatiker zugeschnittene Texte.

(Längen- und Breitengrade), eine bestimmte Projektion in ein planares Koordinatensystem oder – für kleine Flächen – ein lokal definiertes kartesischen Koordinatensystem sein.

- Die *Topologie* beschreibt Beziehungen zwischen räumlichen Objekten wie Nachbarschaft oder Enthaltensein in Flächen. Da nicht absolute Koordinaten, sondern ausschließlich die Lage der Objekte im Raum zueinander betrachtet wird, sind topologische Eigenschaften invariant bezüglich umkehrbar eindeutigen, metrischen Abbildungen wie Translation, Drehung oder Scherung. Das bedeutet, sie bleiben z. B. auch nach einer Überführung der Geometrie in eine andere Kartenprojektion erhalten.

- Die *Attribute* oder *Sachdaten* umfassen alle zusätzlichen Informationen, welche logisch den räumlichen Daten zugeordnet werden. Sie sind positionsunabhängig und können ein weites Spektrum abdecken, von Objekteigenschaften wie Eigentümer, Bodentyp, Landnutzung oder Höhe über Angaben zur graphischen Darstellung bis zu Metadaten, welche die Datenerhebung betreffen.

Das Datenmodell legt fest, wie diese Daten strukturiert sind, um sie in der Datenbank des GIS zu speichern. Dabei werden zwei grundlegend verschiedene Sichtweisen auf die abzubildenden realen Objekte eingenommen, welche in zwei unterschiedlichen Datenmodellen resultieren (Worboys und Duckham 2004, Kap. 4):

Bei der feld-basierten Sicht werden kontinuierliche Werteverteilungen geographischer Variablen betrachtet, z. B. Niederschlagsmenge oder Höhe über Normal-Null. Das *Raster-Datenmodell* liefert eine Approximation des räumlichen Kontinuums durch Diskretisierung des Raums in einzelne Elemente (Tesselation, Bill und Zehner 2001; auch Pflasterung oder Kachelung). Nach Form und Größe der Elemente unterscheidet man reguläre (regelmäßige) und irreguläre (unregelmäßige) Tesselationen; für die Ebene existieren beispielsweise genau drei verschiedene reguläre Tesselationen: aus gleichseitigen Drei-, Vier- bzw. Sechsecken. Solche regulären Tesselationen werden auch als Raster oder Gitter, ihre Elemente als Zellen bezeichnet. In GIS wird meist ein Raster aus quadratischen Zellen verwendet. Die Genauigkeit der Approximation hängt von der Größe der Rasterzellen ab.

Die Geometrie wird üblicherweise in Form von Referenz-Koordinaten für mindestens eine Ecke des gesamten Rasters angegeben. Koordinaten einer Rasterzelle ergeben sich aus dann aus ihrer Position im Raster (Zeilen-/Spalten-Index) und der Größe der Zelle. Die Topologie ist implizit durch die Anordnung der Zellen gegeben; benachbarte Zellen sind einfach zu adressieren. Attribute werden als einzelne Werte den entsprechenden Zellen zugewiesen.

Bei der objekt-basierten Sicht werden diskrete Objekte wie z. B. Häuser, Straßen, Parzellen oder Landkreise betrachtet. Diese Objekte besitzen eine Position, eine gewisse räumliche Ausdehnung und zusätzliche Attribute und werden durch die geometrischen Basis-Objekte Punkt, Linie oder Fläche im *Vektor-Datenmodell* repräsentiert. Den Punkten werden Koordinaten zugewiesen und Linien und Flächen werden als Folge von Punkten, welche durch Geradenstücke verbunden sind, erzeugt. Attribute werden den jeweiligen Objekten zugeordnet. Die Topologie muss explizit gebildet werden, d. h. Adjazenz und Inzidenz von Punkten, Linien und Flächen wird berechnet und gespeichert.

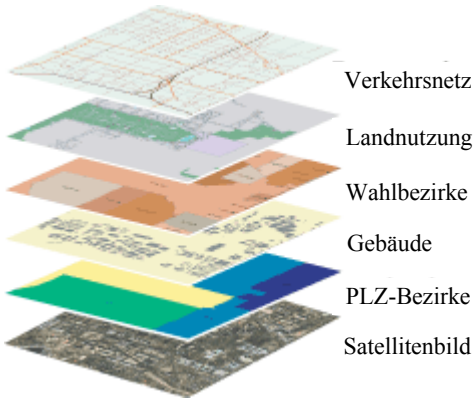

Abbildung 2.3.: Das Layer-Konzept in GIS erlaubt die Kombination verschiedener thematischer „Karten“. Satellitenbilder werden oft als Hintergrund verwendet.

Vektor-Daten können ebenfalls kontinuierlich verteilte Phänomene abbilden. Isolinien und Dreiecksvermaschung (engl. *Triangulated Irregular Network*, vgl. den folgenden Abschnitt) werden benutzt, um beispielsweise das Relief (Variation der Höhe) zu repräsentieren.

Unabhängig vom verwendeten Datenmodell benutzen GIS sogenannte *Layer*, um Daten für die Analyse logisch zu strukturieren. Jeder Layer enthält einen bestimmten Typ an Information wie Landnutzung, Flüsse, Bodenarten und kann als eine Karte aufgefasst werden (Abbildung 2.3). Durch ein gemeinsames Koordinatensystem können einzelne Layer miteinander verknüpft werden zu einer neuen Karte (Verschneidung, engl. *Overlay*) unter Anwendung verschiedener Operatoren. Bei Vektor-Daten kann dies z. B. die (rechnerisch aufwendige) Vereinigung oder Schnittbildung der geometrischen Objekte bedeuten, während im Raster-Datenmodell lediglich die Attribute der Zellen gleicher Position miteinander verglichen werden müssen. Diese Vorgehensweise ist abgeleitet von der Herstellung analoger Karten durch Übereinanderlegen themenspezifischer Folien in der traditionellen Kartographie (Bartelme 2005, S. 57).

Die Zeit spielt in den klassischen Datenmodellen keine Rolle und wird – wenn überhaupt – nur als Attribut behandelt. Seit Anfang der 1990er Jahre hat das Interesse an einer expliziten Behandlung der zeitlichen Dimension in GIS deutlich zugenommen. Für eine erfolgreiche Integration der Zeit müssen zwei wesentliche Probleme überwunden werden: Mangelnde Verfügbarkeit kontinuierlicher Daten über einen Zeitraum und fehlende Datenstrukturen für die Aufzeichnung, Speicherung und Visualisierung von Daten über ein Objekt in unterschiedlichen zeitlichen Zuständen (Heywood et al. 2011, S. 103). Verschiedene Erweiterungen der existierenden Datenmodelle wurden vorgeschlagen, um eine bessere Repräsentation der Zeit zu ermöglichen (für eine ausführlichere Übersicht vgl. z. B. Langran 1992, Ott und Swiaczny 2001 und Pelekis et al. 2004):

- Einführung von Zeitstempeln für ganze Layer (*Snapshot*-Modell, Armstrong 1988), einzelne Attribute bezogen auf Zellen eines Raster bzw. kleinste gemeinsame Polygone (*Space-Time Composites*, Langran und Chrisman 1988) oder räumliche Objek-

te (*Spatio-Temporal Objects*, Worboys 1994);

- Ereignis-orientierte Versionierung mittels sogenannter *Amendment*-Vektoren, welche die Änderung von Flächengrenzen nachführen (Langran 1992), oder Ereignisfolgen auf einem Raster (*Event-Oriented Spatio-Temporal Data Model* ESTDM, Peuquet und Duan 1995);
- Trennung der zeitlichen Komponente von sowohl räumlichen als auch thematischen Eigenschaften (*Three-Domain Model*, Yuan 1999), was im Gegensatz zu den vorher genannten Ansätzen die Abbildung von Bewegung im Raum ermöglicht (Pelekis et al. 2004).

Durch die explizite Repräsentation der Zeit erleichtern die erweiterten Datenmodelle die Visualisierung und Analyse dynamischer Attribute und geometrischer Eigenschaften. Für die Modellierung von Prozessen – Gegenstand der Simulation – reichen sie jedoch nicht aus, da sie vorwiegend darauf abzielen, die sich über die Zeit verändernde Struktur räumlicher Phänomene abzubilden, nicht jedoch die zugrundeliegende Abfolge von Aktionen (Claramunt et al. 1997; Brown et al. 2005). Dies zu erreichen ist einer der Gründe, welcher die Kopplung von GIS mit Simulationsmodellen motiviert, die im folgenden (ab Seite 21) näher diskutiert wird. Zuvor soll jedoch eine weitere relevante räumliche Datenstruktur näher betrachtet werden.

2.2.2.2. Voronoi-Diagramme

Ein Voronoi-Diagramm, im GIS-Zusammenhang auch Thiessen-Polygone oder Dirichlet-Tesselation genannt, ist eine geometrische Datenstruktur für die Repräsentation von räumlicher Nähe. Für eine gegebene Menge $S = \{s_1, s_2, ..., s_n\}$ von Stützpunkten, den sogenannten Generatoren, unterteilt das Voronoi-Diagramm die Ebene in n Bereiche (Voronoi-Zellen), einen für jeden Stützpunkt, so dass ein beliebiger Punkt p innerhalb desjenigen Bereiches liegt, dessen Stützpunkt s_i er am nächsten liegt. Die Bereiche bilden konvexe Polygone mit höchstens $n - 1$ Kanten und Eckpunkten. Jeder Punkt auf einer Kante ist von genau zwei Stützpunkten gleich weit entfernt, während Eckpunkte äquidistant zu mindestens drei Stützpunkten sind (Berg et al. 1997, Kap. 7, Okabe et al. 2000, Kap. 2).

Mit Hilfe des Voronoi-Diagramms lassen sich beispielsweise Abfragen nach nächsten Nachbarn beantworten: Welches ist die zu einem beliebigen Ort (Abfragepunkt) nächstgelegene Stätte von Interesse (Stützpunkt)? Diese als *Postamt-Problem* bekannte Aufgabe wird durch das Voronoi-Diagramm reduziert auf das Problem, diejenige Voronoi-Zelle zu bestimmen, welche den Abfragepunkt enthält. Eine weitere Anwendung aus dem GIS-Bereich ist der Einsatz als räumliches Interpolationsverfahren. Liegen nur Werte an bestimmten Messpunkten vor, kann durch Konstruktion eines Voronoi-Diagramms jedem beliebigen Punkt ein Wert zugewiesen werden. Diesem Verfahren liegt die – für geographische Eigenschaften oft begründete – Annahme zugrunde, dass nahe beieinander liegende Punkte mit größerer Wahrscheinlichkeit gleiche oder ähnliche Werte haben als weiter entfernte.

Eine dem Voronoi-Diagramm eng verwandte Struktur ist der sogenannte Delaunay-Graph, meist als Delaunay-Triangulation bezeichnet. Allgemein versteht man unter der

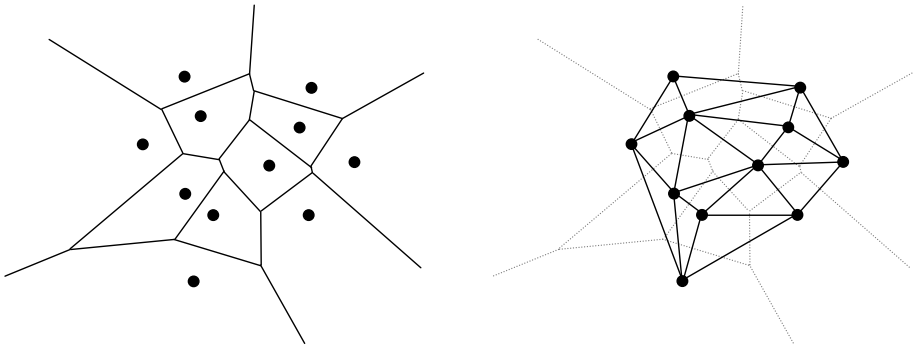

Abbildung 2.4.: Voronoi-Diagramm einer gegebenen Menge von Stützpunkten (links) und die zugehörige Delaunay-Triangulation (rechts)

Triangulation einer Menge von Stützpunkten eine Unterteilung der Ebene in Dreiecke, deren Eckpunkte die gegebenen Stützpunkte sind. Die Delaunay-Triangulation verbindet zwei Stützpunkte durch eine Kante genau dann, wenn deren Voronoi-Zellen in einer Kante aneinandergrenzen (Berg et al. 1997, Kap. 9). Damit bildet die Delaunay-Triangulation im graphentheoretischen Sinn den dualen Graphen zum Voronoi-Diagramm: Für jede Voronoi-Zelle besitzt dieser Graph einen Knoten, den vorgegebenen Stützpunkt, und für jede Kante zwischen benachbarten Zellen eine orthogonale Kante (vgl. Abbildung 2.4).

Delaunay-Triangulationen finden z. B. Einsatz in der digitalen Geländemodellierung mit GIS. Ausgehend von einer Menge unregelmäßig im abzubildenden Terrain verteilter Messpunkte wird zunächst eine Triangulation erstellt (*Triangulated Irregular Network*, TIN). Durch Projektion der Messpunkte auf die korrekte Höhe werden die einzelnen Dreiecke aus der Ebene in den dreidimensionalen Raum abgebildet und es entsteht eine kontinuierliche, stückweise lineare Oberfläche. Die Güte dieser Approximation des tatsächlichen Terrains hängt sowohl von der Auswahl der Messpunkte als auch von der gewählten Triangulation ab. Während signifikante Höhenunterschiede im Relief eine dichtere Verteilung von Messpunkten für eine realitätsnahe Abbildung benötigen, muss das Triangulationsverfahren Dreiecke mit möglichst großen Winkeln generieren. Dadurch wird erreicht, dass jeder beliebige Punkt so nahe wie möglich an einem Messpunkt liegt, was die Genauigkeit der Approximation verbessert. Die Delaunay-Triangulation besitzt genau diese Eigenschaft: Sie maximiert den minimalen Winkel über alle Dreiecke (Berg et al. 1997, S. 189) und wird daher in der Regel als Grundlage eines TIN gewählt.

Voronoi-Diagramme haben Anwendung in diversen anderen Disziplinen gefunden, von Untersuchungen zum Wachstum von Kristallen bis hin zur Routenplanung für Roboter (für eine Übersicht vgl. Aurenhammer 1991). In Kombination mit ihrem dualen Delaunay-Graph wurden sie sogar als grundlegende Datenstrukturen für GIS vorgeschlagen (Gold et al. 1997), da sie die Vorteile von Raster- und Vektordatenmodell verbinden und alle Basis-Operationen zur Manipulation und Abfrage von Daten wie Reklassifizierung, Pufferbildung und Verschneidung unterstützen. Sie müssen dafür nur um Liniensegmente als Generatoren erweitert werden.

Im Rahmen dieser Arbeit werden Voronoi-Diagramm und Delaunay-Triangulation ver-

wendet als eine Ausprägung des Raummodells des in Kapitel 4 entwickelten Frameworks für agentenbasierte Simulation. Ursprünglich für den zweidimensionalen Raum definiert, lassen sich Voronoi-Diagramm und Delaunay-Triangulation unter Erhalt ihrer Dualitätseigenschaft auf den n-dimensionalen Raum verallgemeinern (Berg et al. 1997, S. 159). Dies macht sie zu idealen Kandidaten für die Repräsentation unregelmäßiger Tesselationen in dem in Abschnitt 4.3.1 vorgestellten diskreten Raummodell.

2.2.2.3. Kopplung von GIS mit Simulationsmodellen

Mit ihren Kompetenzen in räumlicher bzw. zeitlicher Modellierung können sich GIS und Simulation gut ergänzen. Bestrebungen, die Raummodellierung von GIS mit der dynamischen Prozessmodellierung diskreter oder kontinuierlicher Simulation zu verbinden, kommen von zwei Seiten: Die Geowissenschaften versuchen, die Analysefähigkeiten von GIS durch Integration von Simulationsmodellen zu erweitern, so z. B. Klimamodelle für die Umweltplanung (Bernard und Streit 2000). Von Seiten der Simulation sind dagegen die Bereitstellung raumbezogener Daten sowie die speziellen räumlichen Analyse- und Präsentationsfunktionen eines GIS von Bedeutung (Meyer et al. 1995). Während sich einfache statische Modelle ohne Schwierigkeiten innerhalb eines GIS anwenden lassen, läuft die Anwendung komplexerer, dynamischer Modelle in der Regel immer noch separat ab.

Es bestehen verschiedene Möglichkeiten, Simulationsmodelle und GIS zu koppeln, wobei Terminologie und verwendete Kategorien der diskutierten Ansätze (Steyaert und Goodchild 1994; Deursen 1995; Fedra 1996; Sandhu und Treleaven 1996; Westervelt 2002) leicht voneinander abweichen. Das Spektrum der Ansätze reicht von loser bis enger Kopplung:

- Bei der *losen Kopplung* erfolgt die Kommunikation zwischen beiden Systemen über den Austausch von Dateien. Abhängig vom verwendeten Datenformat werden gegebenenfalls spezielle Datenkonverter benötigt. Dieses Problem wird durch das Aufkommen von GML entschärft, einem vom Open Geospatial Consortium (OGC)[9] festgelegten XML-Dialekt zur einheitlichen Repräsentation von Geo-Daten (Lake et al. 2004). Etliche GIS-Produkte unterstützen inzwischen GML, so dass die Abhängigkeit von proprietären Dateiformaten obsolet wird.

  Die lose Kopplung bedeutet, dass zwar Daten, nicht aber Funktionen ausgetauscht werden können. Das Simulationsmodell muss daher einen signifikanten Grad an traditioneller GIS-Funktionalität wie geeignete Datenstrukturen, raumbezogene Abfragen und ggf. Visualisierungen aufweisen. Auf der anderen Seite ist diese Form der Kopplung relativ einfach zu realisieren. Beide Systeme bleiben unabhängig und können asynchron ausgeführt werden. Lose Kopplung ist adäquat und effektiv, wenn es um den Input von Daten zu Beginn der Simulation und die Präsentation oder Analyse von Ergebnisdaten im Anschluss an einen Simulationslauf geht.

- Die *enge Kopplung* entspricht einer Einbettung eines Systems in das andere. Unabhängig davon, welche der beiden möglichen Kombinationen gewählt wird, ist dieser Ansatz mit erheblichem Programmieraufwand verbunden. Darüber hinaus kann

[9] http://www.opengeospatial.org/

das resultierende System möglicherweise die negativen Eigenschaften monolithischer Software-Systeme aufweisen (Schüle et al. 2004). Der wesentliche Vorteil der Einbettung mag diese Nachteile jedoch aufwiegen: Beide Systeme arbeiten nicht nur auf derselben internen Datenrepräsentation, die komplette GIS-Funktionalität steht darüber hinaus allen Modell-Komponenten zur Verfügung. Auf diese Weise lassen sich beispielsweise Agenten mit der Fähigkeit zum räumlichen Schließen realisieren: Die Agenten wenden GIS-Analysefunktionen wie die Berechnung von Distanzen, Hangneigung und Sichtbarkeit auf ihre lokale Umgebung an, um Entscheidungen über die Routenwahl zu treffen (Itami 2002).

- Weitere Ansätze zwischen diesen Extremen verwenden in der Regel eine Form von Client-Server-Architektur, wobei GIS und Simulationsmodell sowohl als Client als auch als Server fungieren können. Eine Variante ist die *verteilte Verarbeitung* (Westervelt 2002, S. 90f). Anstelle eines monolithischen Systems werden die einzelnen Komponenten separat ausgeführt und kommunizieren während des Simulationslaufs über geeignete Middleware. Voraussetzung ist, dass beide Systeme Client-Server-Konzepte unterstützen. Bernard et al. (2002) beschreiben ein Konzept für eine komponentenbasierte Integration von Simulationsmodellen in GIS auf der Grundlage existierender Standards wie HLA (Straßburger 2006) und OGC Web Services (Doyle und Reed 2001).

Ein weiterer Ansatz, GIS um Simulationsfunktionalität zu erweitern, ist die Integration von Zellularautomaten (vgl. Abschnitt 2.2.3). Raster-GIS und Zellularautomaten weisen viele Gemeinsamkeiten auf: Beide stützen sich auf dasselbe räumliche Datenmodell (Zellen in einem Gitter, welches die Nachbarschaftsrelation festlegt) und operieren lokal auf den einzelnen Zellen. Die GIS-Analysefunktionen können als spezielle Form der Übergangsfunktion eines Zellularautomaten interpretiert werden (Wagner 1997). Zellularautomaten ermöglichen darüber hinaus die Einbindung der zeitlichen Dimension. Takeyama und Couclelis (1997) definieren mit ihrer Geo-Algebra einen formalen mathematischen Unterbau für die Integration von GIS und Zellularautomaten, basierend auf dem Konzept der räumlichen Nachbarschaft.

Einen Schritt weiter gehen Benenson und Torrens (2003) mit dem Konzept der *Geographic Automata Systems* (GAS), welches GIS nicht nur mit Zellularautomaten, sondern auch mit Multi-Agenten-Simulation verbindet, um bewegliche Objekte abbilden zu können. Die hierfür von Grund auf neu implementierte Software OBEUS (Object-Based Environment for Urban Simulations) ist auf Anwendungen in der Stadtentwicklung beschränkt (Benenson und Torrens 2004).

2.2.3. Zellularautomaten

Zellularautomaten (engl. *cellular automata*, auch als zelluläre Automaten übersetzt) beschreiben dynamische Systeme, in welchen Raum, Zeit und Systemzustände diskret sind (Toffoli und Margolus 1987). Zustandsänderungen werden von einer lokal definierten Regel bestimmt, die parallel für alle Systemelemente ausgeführt wird. Zellularautomaten können somit als spezielle Ausprägung zeitdiskreter Simulationsmodelle aufgefasst werden. Sie finden Anwendung als Modelle in vielen Bereichen der Physik, Biologie, Ma-

thematik und Sozialwissenschaften. Ihre Stärke ist die Untersuchung von Makro-Phänomenen, die auf der Basis von sehr vielen Ereignissen auf der Mikro-Ebene auftreten (Gilbert und Troitzsch 2005, S. 131).

Besonders im Rahmen von biologischen und sozialwissenschaftlichen Untersuchungen werden Zellularautomaten zur Modellierung der dynamischen räumlichen Umgebung eingesetzt. Sie werden dann oft mit weiteren Modellkomponenten zu einem komplexen individuen- bzw. agentenbasierten Modell gekoppelt.

2.2.3.1. Definition

Zellularautomaten wurden Ender der 1940er Jahre von John von Neumann als formales Modell selbst-reproduzierender biologischer Systeme eingeführt (Sarkar 2000). Der ursprüngliche Automat bestand aus einem unendlichen zweidimensionalen Gitter gleichförmiger Zellen, welche jeweils mit ihren vier orthogonalen Nachbarzellen verbunden waren (heute als von-Neumann-Nachbarschaft bezeichnet).

Allgemein besteht ein zellulärer Automat aus einem regelmäßigen Gitter identischer Zellen, dem sogenannten Zellraum. Dieser Zellraum ist d-dimensional, wird jedoch in der Praxis meist auf ein oder zwei Dimensionen beschränkt. Jede der Zellen befindet sich in einem von k Zuständen aus einer endlichen Mengen von Zuständen. Zustandsübergänge treten synchron zu diskreten Zeitschritten auf und gehorchen einer deterministischen lokalen Funktion, welche für alle Zellen gleich ist. Diese Funktion bestimmt den Zustand einer Zelle zu einem Zeitpunkt t in Abhängigkeit von den Zuständen der Zellnachbarschaft zum Zeitpunkt $t - 1$.

Die Nachbarschaft einer Zelle besteht aus einer endlichen Menge von Zellen und kann prinzipiell beliebig definiert werden. Der Idee der lokalen Interaktion folgend, werden aber in der Regel die räumlich am nächsten liegenden Zellen als Nachbarzellen berücksichtigt. In eindimensionalen Zellularautomaten sind dies die links und rechts angrenzenden Zellen, in zweidimensionalen Automaten entweder die vier orthogonal angrenzenden Zellen (von-Neumann-Nachbarschaft) oder alle acht umgebenden Zellen (Moore-Nachbarschaft), siehe Abbildung 2.5. Die betrachtete Zelle selbst wird zwar im Allgemeinen nicht als Nachbar bezeichnet, jedoch in die Größe n der Zell-Nachbarschaft einbezogen.

Ist der Zellraum endlich, so treten an dessen Rändern besondere Bedingungen auf, da die hier liegenden Zellen weniger Nachbarn besitzen. Um das Problem zu umgehen, spezielle Zustandsübergangsfunktionen für die Randzellen zu definieren, wird meist angenommen, dass der Zellraum die Topologie eines (d-dimensionalen) Torus besitzt. Diese auch als *periodisch* bezeichnete Randbedingung simuliert einen unendlichen Zellraum, indem jeweils gegenüberliegende Ränder miteinander verbunden werden. Im eindimensionalen Fall ergibt sich ein Ring, im zweidimensionalen Fall ein Torus (vgl. Abbildung 2.6). Weitere Möglichkeiten sind, die Randzellen an der Gittergrenze zu spiegeln (symmetrische Randbedingung) oder die Randzellen als konstant anzunehmen (statische Randbedingung). Der Einfluss der gewählten Lösung auf das dynamische Verhalten des Zellularautomaten nimmt mit der Größe des Zellraums ab: Bei einem Gitter von $10x10$ Zellen sind 36% der Zellen Randzellen, während bei einem Gitter von $100x100$ Zellen die Randzellen nur noch ca. 4% ausmachen.

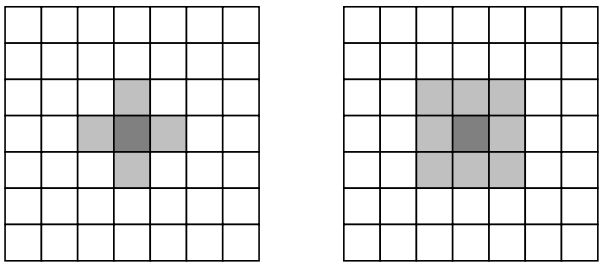

Abbildung 2.5.: Nachbarschaften in zweidimensionalen Zellularautomaten: Von-Neumann-Nachbarschaft (links) und Moore-Nachbarschaft (rechts). Die hellgrau getönten Zellen bilden die Nachbarzellen der dunkelgrau getönten Zelle in der Mitte.

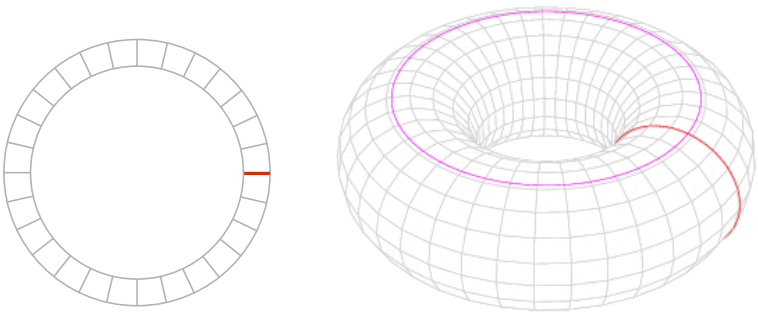

Abbildung 2.6.: Periodische Randbedingungen in Zellularautomaten: Werden die gegenüberliegenden Ränder miteinander verbunden, ergibt sich ein Ring für eindimensionale (links) und ein Torus für zweidimensionale Automaten.

Man kann einen zellulären Automaten als einen Verbund von endlichen Automaten auffassen (Smith 1976), d. h. jede Zelle ist ein endlicher Automat, definiert als Tupel

$$(Q, \Sigma, q_0, F, \delta) \tag{2.3}$$

mit

Q	endliche Menge der Zustände
$\Sigma = Q^n$	endliches Eingabealphabet (hier: die Zustände der Nachbarzellen)
q_0	Startzustand
$F \subset Q$	Menge der Endzustände (meist leer)
$\delta = (Q \times \Sigma \rightarrow Q)$	Zustandsübergangsfunktion

Aus theoretischer Sicht interessant ist, dass manche Zellularautomaten dennoch äquivalent zu einer universellen Turingmaschine sind (von Neumann 1966; Cook 2004).

2.2.3.2. Einfache Automaten

In sogenannten *elementaren Automaten* besteht die Zustandsmenge aus nur zwei Elementen, 0 und 1. Der Zellraum ist eindimensional mit periodischer Randbedingung und Zellen besitzen genau zwei Nachbarn (rechts und links). Betrachtet man eine Zelle mit ihren beiden Nachbarzellen, so können diese genau $2^3 = 8$ verschiedene Konfigurationen annehmen: $000, 001, 010, 100, 011, 110, 101, 111$. Die Zustandsübergangsfunktion kann als Tabelle angegeben werden, in der jeder möglichen Konfiguration der Zell-Nachbarschaft ein Folgezustand für die mittlere Zelle zugeordnet wird. In diesem Fall sind $2^8 = 256$ verschiedene Übergangsfunktionen möglich: Jede der acht Konfigurationen kann im nächsten Zeitschritt eine 0 oder 1 für die mittlere Zelle generieren.

Diese Automaten wurden umfassend wissenschaftlich untersucht (vgl. z. B. Langton 1990; Wolfram 1983, 1984b, a), zum einen, weil die Anzahl der verschiedenen Zustandsübergangsfunktionen noch relativ übersichtlich ist,[10] und zum anderen, weil ihre zeitliche Entwicklung gut zu erfassen ist in Raum-Zeit-Diagrammen (vgl. Abbildung 2.7). In einem solchen Diagramm werden die Zellen eines eindimensionalen Automaten (Raum) für jeden Zeitschritt untereinander aufgetragen, so dass sich die Zeit entlang der y-Achse entwickelt.

Basierend auf einer systematischen Analyse ihres Verhaltens, teilt Wolfram (1984b) Zellularautomaten in vier Kategorien ein:

1. Übergangsfunktionen, die einen stabilen gleichförmigen Zustand herleiten, unabhängig von der Startkonfiguration.

2. Übergangsfunktionen, die einfache Gebilde erzeugen, welche stabil oder periodisch sind. Die Position der Strukturen ist abhängig von der Startkonfiguration, lokale Änderungen in der Startkonfiguration sind lokal begrenzt, d. h. wirken sich nicht auf den gesamten Automaten aus.

[10] Allgemein ergeben sich bei k möglichen Zuständen und einer Nachbarschaft von n Zellen k^n verschiedene Konfigurationen sowie k^{k^n} verschiedene Zustandsübergangsfunktionen. Ein zweidimensionaler Automat mit von-Neumann-Nachbarschaft und zwei Zuständen besitzt damit bereits $2^{(2^5)} = 4.294.967.296$ mögliche Übergangsfunktionen.

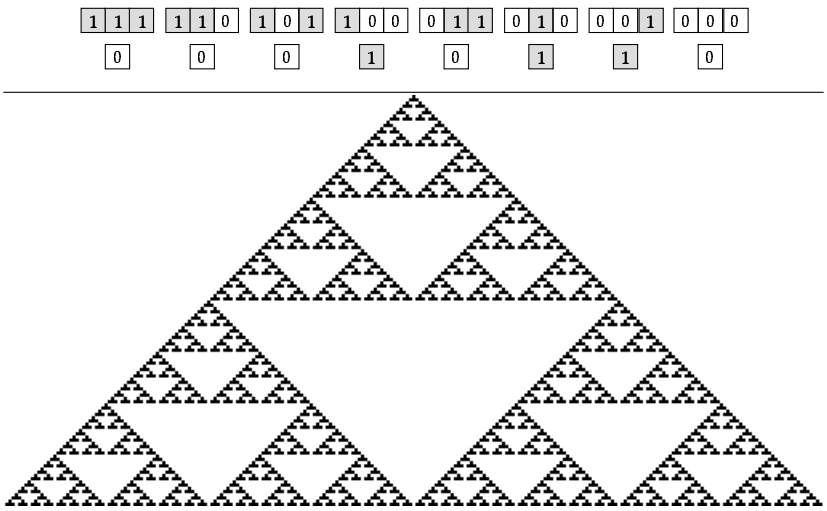

Abbildung 2.7.: Raum-Zeit-Diagramm eines eindimensionalen Zellularautomaten: Die zeitliche Entwicklung des Automaten mit der oben spezifizierten Zustandsübergangsfunktion erzeugt – ausgehend von einer einzelnen Zelle im Zustand 1 (schwarz) – ein fraktales Muster. Das Diagramm wurde mithilfe eines NetLogo-Beispielprogramms zu Zellularautomaten (Wilensky 1998) generiert.

3. Übergangsfunktionen, die chaotische Muster in Raum und Zeit erzeugen. Abhängig von der Startkonfiguration können die chaotischen Muster Fraktalen gleichen. Abbildung 2.7 zeigt als Beispiel eine Übergangsfunktion, welche von einer einzelnen Zelle im Zustand 1 aus eine Struktur generiert, die dem Sierpinski-Dreieck ähnelt.

4. Übergangsfunktionen, die komplexe räumliche und zeitliche Muster erzeugen. Informationsausbreitung, d. h. der Einfluss von Änderungen in der Startkonfiguration, ist nicht beschränkt.

Dieses Klassifikationsschema lässt sich auf mehr-dimensionale Automaten übertragen. Das bekannteste Beispiel für einen zweidimensionalen Zellularautomaten der Klasse 4 ist John Conways *Game of Life* (Gardner 1970, Berlekamp et al. 1982, Kap. 25). Dieser Automat besitzt zwei Zustände (0: tot, 1: lebendig), eine Moore-Nachbarschaft und eine Zustandsübergangsfunktion, welche nur die Anzahl der lebenden Zellen einer Nachbarschaft berücksichtigt, nicht deren exakte Positionen, so dass sie auf drei einfache Regeln reduziert werden kann:

1. Eine Zelle im Zustand 1 mit zwei oder drei lebenden Nachbarn bleibt im Zustand 1 (Überleben).

2. Eine Zelle im Zustand 0 mit genau drei lebenden Nachbarn geht in den Zustand 1 über (Geburt).

3. Eine Zelle im Zustand 1 geht in den Zustand 0 über, wenn sie weniger als zwei oder mehr als drei lebende Nachbarn besitzt (Tod aus Einsamkeit bzw. Überbevölkerung).

Conway wählte die Zustandsübergangsfunktion so aus, dass die Evolution einer Startkonfiguration (hier als Population bezeichnet) möglichst unvorhersehbar ist. Die große Popularität des *Game of Life* liegt darin begründet, dass es aus einfachen lokalen Regeln überraschende und komplexe Strukturen auf der Makroebene generieren kann. In diesem Sinn ist es ein Beispiel für Emergenz in komplexen Systemen (Christen und Franklin 2002).

2.2.3.3. Erweiterungen

Diverse Veränderungen und Erweiterungen der einfachen Zellularautomaten sind möglich und für verschiedene Anwendungsgebiete diskutiert worden (vgl. z. B. Gutowitz 1991). Eine gerade in biologischen und soziologischen Modellen häufig verwendete Erweiterung ist die Einführung stochastischer Zustandsübergänge anstelle rein deterministischer. Dies bedeutet, dass ein bestimmter Zustandsübergang nur mit einer gewissen Wahrscheinlichkeit stattfindet. Beispiele sind Modelle von Kontaktprozessen (Bagnoli 1998; Durrett 1999), wie die Ausbreitung von Epidemien, Waldbränden oder Meinungen. In allen diesen Fällen werden die Zellen des Automaten mit Individuen gleichgesetzt, die zwar ihren Zustand ändern, nicht aber ihre Position im (Zell-)Raum. Die stochastische Zustandsübergangsfunktion sorgt für Variabilität unter den Individuen und damit allgemein für eine bessere Übereinstimmung mit den modellierten natürlichen Systemen (Phipps 1992).

Letzteres ist ebenfalls ein Grund für die Wahl einer anderen als der vorherrschenden quadratischen Zellform. Im Rahmen der Definition eines Zellularautomaten sind für den zweidimensionalen Fall beispielsweise drei verschiedene regelmäßige Gitter möglich, aufgebaut aus gleichseitigen Drei-, Vier- oder Sechsecken. Da quadratische Gitter am einfachsten zu implementieren sind, werden sie für Computersimulationen meist vorgezogen. Andererseits weisen sie den Nachteil größerer Anisotropie auf, d. h. der Abstand zwischen benachbarten Zellen ist – abhängig von der gewählten Nachbarschaft – nicht immer gleich. So beträgt der Abstand für die vier diagonalen Zellen in der Moore-Nachbarschaft beispielsweise nicht die Kantenlänge x einer Quadratzelle, sondern $\sqrt{2}x$. Hexagonalgitter lassen daher simulierte Vorgänge oft natürlicher erscheinen (Weimar 1998) und haben sich für die diskrete Simulation von Flüssigkeiten und Gasen mit Zellularautomaten (sogenannte *Lattice Gas Automata*) als notwendig erwiesen (Frisch et al. 1986).

Durch Verzicht auf die Annahme der Regelmäßigkeit des Zellraums ergeben sich irreguläre Zellularautomaten, in denen sowohl die Form als auch die Nachbarschaftsrelation der einzelnen Zellen unterschiedlich ist. Auf diese Weise lassen sich räumlich heterogene Strukturen und Prozesse, wie sie in den Sozial- und Umweltwissenschaften häufig auftreten, adäquater modellieren. Im zweidimensionalen Raum können die Zellen als beliebige Polygone beschrieben werden, deren Nachbarschaftsrelation frei definiert werden kann (Sonnenschein und Vogel 2001). Eine Möglichkeit zur Umsetzung solcher Zellularautomaten ist die Verwendung von Voronoi-Diagrammen zur Partitionierung des Raums (Flache und Hegselmann 2001).

Weitere Generalisierungen der Grundidee eines Zellularautomaten betreffen den Zeitfortschritt und die Zustandsübergangsfunktion. Werden unterschiedliche Regeln für einzelne Zellen zugelassen, so erhält man inhomogene Zellularautomaten (Sipper 1994). Bei

Verzicht auf die geforderte Synchronität, d. h. die Aktualisierung aller Zellen zum selben Zeitpunkt, ergeben sich asynchrone Zellularautomaten. Diese sind insbesondere für Anwendungen in den Lebenswissenschaften interessant, da für die dort untersuchten Systeme die Annahme eines globalen Takts – wie ihn synchroner Zeitfortschritt voraussetzt – unrealistisch ist (Hogeweg 1988). Verschiedene Untersuchungen haben gezeigt, dass auch bei abstrakten Modellen wie z. B. dem *Game of Life* oder manchen elementaren Zellularautomaten das besondere Verhalten ein Artefakt der synchronen Aktualisierung ist (vgl. z. B. Bersini und Detours 1994; Cornforth et al. 2002).

2.2.3.4. Anwendungen in der diskreten Simulation

Zellularautomaten haben sich als wertvolles Hilfsmittel für die Untersuchung von selbstorganisierten Systemen erwiesen, in denen lokale Wechselwirkungen bedeutsam sind. Wenige, einfache Regeln und rein lokale Interaktionen können komplexe Strukturen auf der Makroebene generieren. Verschiedene Autoren haben sich daher für die Verwendung von Zellularautomaten-Modellen in ihren jeweiligen Fachgebieten eingesetzt, so u. a. Hogeweg (1988); Phipps (1992) für die Ökologie, Hegselmann (1996); Hegselmann und Flache (1998) für die Sozialwissenschaften und Batty und Xie (1994); White und Engelen (1997); White (1998) für die Geowissenschaften. In allen Fällen kommen erweiterte Zellularautomaten zum Einsatz wie im vorhergehenden Abschnitt beschrieben; in Abhängigkeit von der jeweiligen Anwendung werden z. B. stochastische Zustandsübergangsfunktionen, unregelmäßige Zellen, heterogene Nachbarschaften und asynchrone Aktualisierung verwendet (vgl. z. B. O'Sullivan und Torrens 2000; Wittmann 2000).

Allen Anwendungen gemeinsam ist, dass sie Zellen als räumliche Einheiten behandeln. Der Zellularautomat bildet somit eine explizite Repräsentation des Raums. Meist handelt es sich außerdem um individuen- bzw. agentenbasierte Modelle, d. h. Modelle, welche detailliert Verhalten und Interaktionen von einzelnen Akteuren abbilden – seien sie Menschen, Tiere, Pflanzen oder mehr oder weniger abstrakte Organisationen. Der wesentliche Unterschied liegt in der Bedeutung der Zellzustände und Zustandsübergangsfunktionen:

1. Zellzustände und Regeln beschreiben sowohl Raumeinheiten als auch auf ihnen positionierte Akteure (sofern diese verschieden von den Raumeinheiten sind). Akteure werden *implizit* repräsentiert über entsprechende Kodierungen der Zustände und Übergangsfunktionen. Auf diese Weise bleibt das gesamte Modell im Rahmen eines Zellularautomaten.

2. Zellzustände und Regeln beschreiben ausschließlich das Verhalten der Raumeinheiten. Akteure werden *explizit* repräsentiert als zusätzliche Modellkomponenten. Der Zellularautomat bildet dann ebenfalls eine Modellkomponente und dient beispielsweise als räumliche Umgebung in einem Multi-Agenten-Modell.

Beispiele für die erste Kategorie sind Populationsmodelle für Pflanzen, in denen der Zellzustand das Vorkommen einer Spezies, die Menge and Biomasse oder die Größe einer lokalen Subpopulation als Anteil der von ihr bedeckten Zellfläche abbildet (Grimm und Railsback 2005, S. 250f). Das bekannteste Beispiel aus der Verkehrssimulation ist das nach seinen Autoren benannte Nagel-Schreckenberg-Modell (Nagel und Schreckenberg 1992).

Es modelliert den Verkehrsfluss auf einer einspurigen Straße mittels eines eindimensionalen Zellularautomaten. Jede Zelle kann entweder leer sein oder genau ein Fahrzeug aufnehmen. Im letzteren Fall beschreibt der Zustand der Zelle die aktuelle Geschwindigkeit des Fahrzeug, repräsentiert als ganze Zahl zwischen 0 und der Maximalgeschwindigkeit v_{max}. Die Zustandsübergangsfunktion bestimmt die Geschwindigkeit zum nächsten Zeitpunkt und besteht aus drei Schritten: (1) Beschleunigung, falls die Maximalgeschwindigkeit noch nicht erreicht und der Abstand zum vorausfahrenden Fahrzeug groß genug ist, (2) Abbremsen, falls der Abstand zum nächsten Fahrzeug kleiner als die aktuelle Geschwindigkeit ist und (3) einer zusätzlichen Verringerung der Geschwindigkeit um 1 mit einer gewissen „Trödel-Wahrscheinlichkeit“ p. Anschließend werden die Fahrzeuge entsprechend ihrer Geschwindigkeit bewegt, d. h. die Zustände der jeweiligen Zellen aktualisiert.

Das Modell zeigt trotz seiner Einfachheit ein realistisches Verhalten und konnte das Phänomen des „Staus aus dem Nichts“ als Folge von Überreaktionen beim Bremsen erklären. Ausführung auf Parallelrechnern erlaubt es, mehrere Millionen Fahrzeuge gleichzeitig zu simulieren, wie die Weiterentwicklung zu TRANSIMS (Nagel et al. 2000) belegt. Das Modell wurde darüber hinaus als Grundlage für die Verkehrsprognose auf Nordrhein-Westfalens Autobahnen verwendet und entsprechend erweitert (Schreckenberg et al. 2003).

Neben diesen mehr oder wenigen „reinen“ Zellularautomaten wurden einige Versuche unternommen, die Ausdrucksfähigkeit von Zellularautomaten zu erhöhen, indem die Spezifikation von Zuständen und Zustandsübergangsfunktion mit komplexeren Modellierungsmethoden wie System-Dynamics-Modellen (Maxwell und Costanza 1997), Petrinetzen (Gronewold und Sonnenschein 1998) oder evolutionären Algorithmen (Hogeweg 2010) vorgenommen wird.

Der überwiegende Teil der Anwendungen fällt in die zweite Kategorie, d. h. ein Zellularautomat wird als diskretes Raummodell verwendet. Der große Vorteil dieses Ansatzes ist die Möglichkeit, den Raumeinheiten über die Zustandsübergangsfunktion ein eigenes Verhalten zuzuordnen. So kann beispielsweise in Ökosystemmodellen die Erneuerung von natürlichen Ressourcen (vgl. z. B. Clark und Rose 1997; Etienne et al. 2003; Nonaka und Holme 2007) oder die Ausbreitung von Pheromonen (u. a. Drogoul und Ferber 1994; Panait und Luke 2004; Bala et al. 2012) abgebildet werden.

Ersteres geschieht auch in dem als *Sugarscape* bekannt gewordenen Modell einer künstlichen Gesellschaft (Epstein und Axtell 1996). In diesem Modell wird eine Population von Agenten betrachtet, welche die nachwachsende Ressource Zucker zum Überleben benötigen. Der Zucker wird von ihrer Umgebung zur Verfügung gestellt, einem als Zellularautosmaten modellierten zweidimensionalen quadratischen Gitter. Jede Zelle besitzt eine bestimmte Zuckerkapazität (Zellzustand) und kann pro Zeitschritt den vorhandenen Zuckervorrat gemäß einer konstanten Regenerationsrate erneuern (Zustandsübergangsfunktion). Agenten können sich über das Gitter bewegen und die Ressourcen derjenigen Zelle verbrauchen, auf der sie sich gerade befinden. Für eine detailliertere Beschreibung des Modells sei auf Abschnitt 5.4 verwiesen.

2.3. Multiagentensysteme

Multiagentensysteme (MAS) sind ein relativ neues Teilgebiet der Informatik, dessen Ursprünge in der Künstlichen Intelligenz liegen. Inzwischen ist das dahinterstehende Konzept eines *Agenten* als einer unabhängig agierenden (computergestützten) Einheit von weiteren Bereichen aufgenommen worden und wird z. B. in der Softwaretechnik als neues Paradigma angesehen (Jennings 2000). Neben der Autonomie eines einzelnen Agenten wird in Multiagentensystemen besonders die Fähigkeit zur Interaktion mit anderen Agenten betont. Die Interaktion findet typischerweise in Form von Nachrichtenaustausch statt und ermöglicht den Agenten, mit anderen zu kooperieren oder zu verhandeln. Diese „sozialen" Aspekte von Multiagentensystemen werden nicht nur in verteilten, heterogenen Systemen wie dem Internet ausgenutzt, sondern bespielsweise auch bei der Untersuchung (künstlicher) sozialer Gesellschaften.

Zunächst sollen die Grundbegriffe Agent und Multiagentensystem diskutiert werden (Abschnitte 2.3.1 und 2.3.2). Abschnitt 2.3.3 gibt einen kurzen Abriss der bisherigen Entwicklung des Forschungsgebiets MAS, um später die agentenbasierte Simulation besser einordnen zu können. Eine Diskussion von Ansätzen zu Entwurf und Implementation von Agenten und Multiagentensystemen (Abschnitt 2.3.4) rundet die Einführung ab.

2.3.1. Begriffsbestimmung Agent

Don't trust anything that can think for itself if you can't see where it keeps its brain.
– J.K. Rowling

Das Wort Multiagentensystem impliziert, dass es sich um ein System aus einer Vielzahl von Agenten handelt. Daher ist es notwendig, zunächst den Begriff *Agent* zu definieren. Dies ist allerdings insofern schwierig, als trotz der zahlreichen Anwendungen und der noch umfangreicheren Literatur zu Agenten keine allgemein anerkannte Definition des Begriffs existiert. Statt dessen wird der Begriff mehr oder weniger kontrovers diskutiert, was zu einem großen Teil daran liegen mag, dass die Anwendungsgebiete so divers sind und daher sehr unterschiedliche Eigenschaften für Agenten als wesentlich angesehen werden (Wooldridge 2009, S. 21).

Ein Vorteil dieser Situation ist, dass man praktisch immer eine für seine Zwecke passende Definition finden wird. Nachteilig ist dagegen, dass jeder, der mit Agenten arbeitet, bereits seine „eigene" Definition im Hinterkopf hat, was zu Missverständnissen führen kann. Es ist daher unerlässlich, jedesmal die verwendete Bedeutung des Begriffs offenzulegen.

2.3.1.1. Definition als Zuschreibung

Es gibt zwei grundsätzlich unterschiedliche Herangehensweisen, den Begriff des Agenten zu fassen: Die eine besteht darin, einem Objekt das „Agentsein" zuzuschreiben (engl. *ascription*), die andere darin, die Eigenschaften eines Agenten zu beschreiben (engl. *description*) (vgl. Bradshaw 1997). Der erste Ansatz beruht darauf, bestimmte Dinge als Agent zu

betrachten – ob etwas ein Agent ist oder nicht, liegt somit im Auge des Betrachters. Die Definition von Russell und Norvig (2009, S. 35) macht dies deutlich:

> An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

Auch die Definition von Shoham (1993, S. 52) fällt in diese Kategorie:

> An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments.[...] In this view, therefore, agenthood is in the mind of the programmer: What makes any hardware or software component an agent is precisely the fact that one has chosen to analyze and control it in these mental terms.

Zwar kann prinzipiell alles als Agent aufgefasst werden, fraglich bleibt jedoch, ob es auch von Vorteil ist. Shoham (1993, S. 53) illustriert diesen Punkt mit dem Beispiel eines Lichtschalters, der als (sehr kooperativer) Agent betrachtet werden kann: Er lässt immer genau dann Strom fließen, wenn er glaubt, dass wir Strom haben möchten. Durch Ein- bzw. Ausschalten teilen wir dem Lichtschalter unsere Wünsche mit. Shoham argumentiert, dass diese Sichtweise zwar schlüssig ist, uns aber keine neuen Erkenntnisse über Lichtschalter bringt, da wir auf einer einfacheren, mechanischen Ebene bereits ausreichend das Verhalten von Lichtschaltern beschreiben können. Es sollte daher zwischen Zulässigkeit und Zweckmäßigkeit bei der Zuschreibung von Agenten-Eigenschaften – in diesem Fall gleichgesetzt mit mentalen Eigenschaften – differenziert werden.

Zweckmäßig erscheint es insbesondere bei komplexen Systemen, über deren Funktionsweise keine ausreichenden Kenntnisse bestehen. Es ist daher nicht verwunderlich, dass die Forschung zu komplexen Systemen wesentlich zur Entstehung der agentenbasierten Simulation beigetragen hat (siehe Abschnitt 3.2).

2.3.1.2. Definition als Beschreibung

Beim zweiten Ansatz besteht eine Definition des Begriffs *Agent* darin, die charakteristischen Eigenschaften aufzuzählen, die einen Agenten ausmachen. Als Folge der eingangs erwähnten kontroversen Diskussion existiert inzwischen eine Vielzahl mehr oder weniger unterschiedlicher Definitionen, welche die Bandbreite bestehender Ansätze und Anwendungen widerspiegeln. Manche Definitionen beschränken sich dabei auf eine spezielle Ausprägung des allgemeinen Agentenbegriffs, was in der Regel durch ein entsprechendes Adjektiv („intelligent“, „mobil“, „rational“) kenntlich gemacht wird. In dieser Arbeit wird ein sehr allgemeiner Agentenbegriff zugrundegelegt; deshalb sollen im folgenden exemplarisch einige in Frage kommende Definitionen betrachtet werden.

In ihrer ausführlichen Übersicht über die Agentenlandschaft von 1995, die große Anerkennung gefunden hat, unterscheiden Wooldridge und Jennings explizit zwischen dem weiteren und dem engeren Agentenbegriff (*weak notion* vs. *stronger notion of agency*, Wooldridge und Jennings 1995): Agenten im weiteren Sinne sind Computersysteme mit den Eigenschaften Autonomie, Reaktivität, Pro-Aktivität und der Fähigkeit zu kommunizieren (was als *social ability* bezeichnet wird). Agenten im engeren Sinne – wie sie insbesondere innerhalb der Künstlichen Intelligenz vertreten werden – sind darüber hinaus mit

„menschlichen“ Attributen versehen wie Wissen, Überzeugungen, Wünschen und Zielen. In späteren Publikationen differenziert Wooldridge den weiteren Agentenbegriff in allgemeine Agenten und intelligente Agenten[11]:

> An *agent* is a computer system that is *situated* in some environment, and that is capable of *autonomous action* in this environment in order to meet its design objectives. (Wooldridge 2002, S. 15)

> [A]n intelligent agent is one that is capable of *flexible* autonomous action in order to meet its design objectives, where flexibility means three things: *reactivity:* intelligent agents are able to perceive their environment, and respond in a timely fashion to changes [...]; *pro-activeness:* [...] exhibit goal-directed behaviour by *taking the initiative* [...]; *social ability:* [...] are capable of interacting with other agents [...] (Wooldridge 1999, S. 32)

Franklin und Graesser (1996) diskutieren zunächst eine Reihe von bestehenden Definitionen, ehe sie darauf aufbauend ihre eigene Definition bringen. Dabei erheben sie ausdrücklich den Anspruch, eine möglichst umfassende Definition des Begriffs zu geben, die nichtsdestotrotz den Kern des „Agentseins“ fasst:

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

Ihre Definition baut u. a. auf dieser Definition von Pattie Maes (1995, S. 108) auf:

> Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.

Jacques Ferber definiert in seiner Einführung in Multiagentensysteme, die eines der ersten Lehrbücher zu diesem Thema darstellt[12], den Agentenbegriff wie folgt:

> An agent is a physical or virtual entity (a) which is capable of acting in an environment, (b) which can communicate directly with other agents, (c) which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimise), (d) which possesses resources of its own, (e) which is capable of perceiving its environment (but to a limited extent), (f) which has only a partial representation of this environment (and perhaps none at all), (g) which possesses skills and can offer services, (h) which may be able to reproduce itself, (i) whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representations and the communications it receives. (Ferber 1999, S. 9)

[11] Der Wortlaut dieser Definitionen wurde in den neueren Auflagen leicht verändert, vgl. (Wooldridge 2009, S. 21) bzw. (Wooldridge 2013, S. 8). Für die hier verwendete direkte Gegenüberstellung erschienen die Originalversionen prägnanter.

[12] Das französische Original erschien 1995. Bücher mit einem vergleichbaren Anspruch sind (Weiss 2013) und (Wooldridge 2009).

Ferbers Definition ist die umfangreichste, da er neben den notwendigen und hinreichenden Eigenschaften, die das Wesen eines Agenten ausmachen, auch optionale Kriterien aufführt. Weitere Unterschiede betreffen die Art des betrachteten Systems, Einschränkungen bezüglich der Umgebung und die zeitliche Existenz der Agenten. Konsens besteht dagegen bei allen aufgeführten Definitionen bezüglich der Situiertheit eines Agenten in einer Umgebung, der Fähigkeit zu (autonomer) Handlung und der Ausrichtung dieser Handlung an internen Zielvorgaben. Diese zentralen Eigenschaften sollen im folgenden näher betrachtet werden.

Situiertheit bedeutet, ein Agent ist in eine bestimmte Umgebung eingebettet. Er nimmt diese Umgebung wahr über Sensoren und wirkt auf die Umgebung ein über Effektoren. Auf diese Weise ist es ihm möglich, die Umgebung zu verändern und auf die von ihm oder anderen Einflüssen verursachten Änderungen zu reagieren. Kybernetisch kann ein Agent als ein mit der Umwelt rückgekoppeltes System betrachtet werden. Sein Wahrnehmungs- wie sein Einflussbereich sind in der Regel lokal beschränkt. Dabei ist es nicht zwingend erforderlich, dass beide identisch sind, wie in Abbildung 2.8 der Einfachheit halber dargestellt. Von der Beschaffenheit der Sensoren und Effektoren wird zunächst abstrahiert; gefordert ist nur, dass sie dem Agenten Informationen über den Zustand der Umgebung liefern bzw. aktive Einflussnahme auf diesen Zustand ermöglichen. Daraus folgt, dass sie auf den Typ der Umgebung (siehe unten) abgestimmt sein müssen. Die Situiertheit grenzt

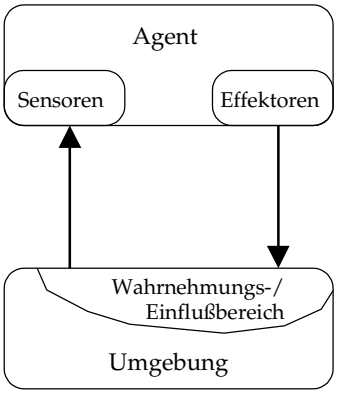

Abbildung 2.8.: Schematische Darstellung eines Agenten in seiner Umgebung. Angedeutet ist die lokale Beschränkung des Agenten beim Zugriff auf die Umwelt, indem Wahrnehmungs- und Einflussbereich abgegrenzt sind.

Agenten gegen andere KI-Techniken wie z. B. Expertensysteme ab. Diese sind inhärent körperlos, d. h. sie interagieren nicht direkt mit einer Umwelt: Weder nehmen sie Informationen über Sensoren auf, noch wirken sie auf eine Umgebung ein. Statt dessen fungiert in der Regel ein Benutzer als Schnittstelle, der Informationen aus der bzw. über die Umwelt liefert und Aussagen des Expertensystems ggf. in Handlungen umsetzt (Wooldridge 2009, S. 30; Jennings et al. 1998, S. 8).

Zwar besteht Konsens darüber, dass die Fähigkeit zu autonomem Handeln eine zentrale Eigenschaft von Agenten ausmacht, doch herrscht Uneinigkeit, was genau unter *Au-*

tonomie zu verstehen ist. Wie schon beim Agentenbegriff selbst gibt es beim Konzept der Autonomie eine engere und eine weitere Auffassung. Autonomie im weiteren Sinne besagt, dass Agenten die Kontrolle über ihr Verhalten besitzen, d. h. sie können eigenständig handeln, ohne auf den Eingriff menschlicher Benutzer oder anderer Systeme angewiesen zu sein (Jennings et al. 1998, S. 8; Wooldridge 2013, S. 5). Autonomie im engeren Sinne fordert, dass das Verhalten zumindest teilweise von eigenen Erfahrungen bestimmt wird (Russell und Norvig 2009, S. 40), der Agent demnach sein Verhaltensrepertoire durch Lernen anpassen kann. Diese strengere Form von Autonomie findet sich in der Regel im Kontext intelligenter Agenten. Huhns und Singh (1998, S. 3) diskutieren den Autonomiebegriff zusätzlich aus der Sicht von Multiagentensystemen und unterscheiden nach abnehmenden Freiheitsgraden zwischen absoluter, sozialer, Schnittstellen- und Ausführungsautonomie.

Strenggenommen beschränkt sich die Handlungsautonomie des Agenten auf die Fähigkeit, eigenständig zu entscheiden, welche der ihm zur Verfügung stehenden Aktionen in der aktuellen Situation auszuführen ist. Selbst die Eigenständigkeit der Entscheidung kann in Frage gestellt werden, da bei nicht-adaptiven Agenten auch die Entscheidungskriterien vom Entwickler vorgegeben sind. Dagegen lässt sich argumentieren, dass dies das Wesen eines Agenten nicht berührt, da auch menschliche Agenten, die – im umgangssprachlichen Sinne des Begriffs – im Auftrag eines anderen selbstständig handeln, nur über die konkrete Ausführung ihres Auftrags entscheiden, während der Auftrag selbst und eventuelle Hilfsmittel vom Auftraggeber vorgegeben sind (Burkhard 1998).

Ein Agent besitzt nicht nur Fähigkeiten, sondern auch *Zielvorgaben*. Diese bestimmen, wann er welche Fähigkeiten einsetzt. Die Abbildung der Wahrnehmungen auf die möglichen Handlungen wird so von den Zielen des Agenten beeinflusst. In der Regel wird angenommen, dass der Agent versucht, in jeder Situation die bestmögliche Handlung auszuführen (rationaler Agent, Russell und Norvig 2009, S. 38). Dabei ist es durchaus möglich, dass Zielkonflikte auftreten oder dass Ziele sich verändern. Technisch gesehen müssen die Ziele nicht explizit repräsentiert werden wie in BDI-Architekturen, sondern können beispielsweise auch in Verhaltensregeln kodiert sein wie bei der *Subsumption*-Architektur (siehe Abschnitt 2.3.4). Zielgerichtetes Verhalten wird oft mit Pro-Aktivität (*pro-activeness*) gleichgesetzt (z. B. Wooldridge 2009, S. 27), der Fähigkeit eines Agenten, selbst aktiv zu werden, statt nur auf die Umgebung zu reagieren.

Wooldridge gibt zu bedenken, dass Zielorientierung *per se* keine agentenspezifische Eigenschaft ist. Fasst man die Vorbedingungen einer Funktion oder Prozedur als Annahmen auf und die Nachbedingungen als Ziele, so handelt bereits ein einfaches funktionales System, dass auf die Eingabe x die Ausgabe $f(x)$ produziert, zielgerichtet. Allerdings stellt sich dieses einfache Modell sofort als unzureichend heraus in dynamischen Umgebungen, in denen die beiden impliziten Voraussetzungen nicht gegeben sind: Dass sich die Umgebung während der Ausführung der Funktion nicht ändert und dass das Ziel, das mit der Ausführung erreicht werden soll, gültig bleibt. In solchen Umgebungen macht also eine ausgewogene Balance zwischen Zielorientierung und Reaktivität einen (erfolgreichen) Agenten aus (Wooldridge 2009, S. 27f).

Der in dieser Arbeit zugrundegelegte Agentenbegriff umfasst Software-Komponenten, die in eine Umgebung eingebettet sind und in dieser Umgebung autonom und zielgerichtet handeln. Damit stütze ich mich auf die oben angeführte Definition von Woold-

ridge (2002, S. 15). Da im Kontext von Multiagentensystemen die Interaktion zwischen Agenten eine wesentliche Rolle spielt, soll die Fähigkeit zur Kommunikation zusätzlich als wesentliche Eigenschaft eines Agenten betrachtet werden. Ob diese Kommunikation jedoch direkt durch Austausch von Nachrichten zwischen Agenten oder indirekt durch Veränderung des Umgebungszustands stattfindet, sei dahingestellt.

2.3.1.3. Eigenschaften der Umgebung

Der Typ der Umgebung hat Einfluss auf den Entwurf von Agenten. So müssen zumindest Sensoren und Effektoren auf die Umgebung abgestimmt sein, sollen sie den Agenten in die Lage versetzen, im Hinblick auf die Zielvorgaben „richtig" zu handeln. Verschiedene Typen von Umgebungen stellen Agenten-Designer dabei vor unterschiedlich schwere Aufgaben.

Die folgende Zusammenstellung der Umgebungseigenschaften stützt sich auf die Klassifikationen von Huhns und Singh (1998, Kapitel 1) und Russell und Norvig (2009, S. 43ff)). Es werden zu den deutschen Bezeichnungen die entsprechenden englischen Ausdrücke genannt, wobei jeweils zuerst die von Russel und Norvig verwendete Bezeichnung und dann die von Huhns und Singh gewählte aufgeführt sind.

Zugänglich. Eine zugängliche (engl. *fully observable/knowable*) Umgebung erlaubt dem Agenten Zugriff auf ihren vollständigen Zustand. In der Regel ist der Wahrnehmungsbereich des Agenten jedoch auf einen Ausschnitt der Umgebung beschränkt; eine solche Umgebung wird als unzugänglich bezeichnet. Ist der Agent bei der Auswahl der nächsten Aktion auf mehr als die lokal verfügbare Information angewiesen, so muss er in unzugänglichen Umgebungen in der Lage sein, Informationen über die Umgebung zu speichern. Auch dies garantiert jedoch in dynamischen Umgebungen nicht, dass der Agent den aktuellen Zustand der Umgebung immer hinreichend genau bestimmen kann, um die jeweils optimale Aktion auszuwählen.

Deterministisch. Die Umwelt ist deterministisch (*deterministic/predictable*), wenn der Folgezustand vollständig durch den aktuellen Umgebungszustand und die aktuelle Handlung des Agenten bestimmt ist. In nicht-deterministischen Umgebungen kann dagegen dieselbe Aktion, zu unterschiedlichen Zeitpunkten, aber unter sonst praktisch identischen Umständen ausgeführt, ganz verschiedene Ergebnisse haben. Die Auswirkung einer Handlung ist daher nicht eindeutig vorhersagbar. Insbesondere ist es möglich, dass eine Handlung fehlschlägt, indem sie nicht den gewünschten Effekt produziert (Wooldridge 2009, S. 22). Russel und Norvig weisen darauf hin, dass diese Kategorie aus Sicht des Agenten interpretiert werden sollte, da ausreichend komplexe Umgebungen zwar deterministisch sein können, aufgrund ihrer Unzugänglichkeit dem Agenten jedoch als nicht-deterministisch erscheinen.

Episodisch. Eine episodische (*episodic/historical*) Umgebung ist im Prinzip gedächtnislos: Zukünftige Zustände hängen nicht vom aktuellen Zustand und der Aktion des Agenten ab. Im Gegensatz dazu muss ein Agent in einer nicht-episodischen Umgebung vorausplanen, da jede seiner Handlungen nicht nur den aktuellen, sondern einen beliebigen, zukünftigen Umgebungszustand beeinflussen kann. Russell und

Norvig (2009, S. 44) führen Arbeit an einem Fließband als Beispiel einer episodischen Umgebung an. Jede Episode beginnt mit der Ankunft eines neuen Werkstücks, das auf Defekte zu untersuchen ist, und endet mit der vom Agenten getroffenen Entscheidung. Welche Entscheidungen über die vorher eingetroffenen Werkstücke getroffen worden sind, hat keinerlei Einfluss auf die Entscheidung, wie das aktuelle Stück zu behandeln ist, noch hat die aktuelle Entscheidung Einfluss auf die nächste Episode.

Dynamisch. Wenn sich der Zustand der Umgebung zwischen Wahrnehmung und Handlungsauswahl durch den Agenten ändern kann, so handelt es sich um eine dynamische (*dynamic/real time*) Umgebung. Das bedeutet zum einen, dass neben dem Agenten noch andere aktive Prozesse auf die Umgebung einwirken. Zum anderen muss sich der Agent (bzw. sein Entwickler) bei der Abbildung der Sensor-Informationen auf die auszuwählende Handlung mit dem Verstreichen von Zeit auseinandersetzen. Eine Möglichkeit besteht darin, während der Aktionsselektion weiterhin die Umgebung zu beobachten, um Änderungen sofort in den Entscheidungsprozess einbeziehen zu können; wobei hier die Gefahr droht, dass der Agent über reine Reaktion auf Umgebungsveränderungen nicht hinaus kommt.[13] Eine andere, wohl eher theoretische Möglichkeit ist es, den Entscheidungsprozess des Agenten so zu beschleunigen, dass die Dynamik der Umgebung keine Rolle mehr spielt. In einer statischen Umgebung treten solche Echtzeit-Anforderungen nicht auf, da sich hier der Zustand nur durch Aktionen des Agenten ändert.

Diskret. Der Wertebereich der in der Umgebung möglichen Wahrnehmungen und Aktionen kann diskret oder kontinuierlich sein. In einer diskreten (*discrete/–*) Umgebung steht nur eine endliche Menge von klar abgrenzbaren Wahrnehmungen und Aktionen zur Verfügung. Für eine solche Umgebung kann daher garantiert werden, dass sie eine endliche Menge diskreter Zustände besitzt (Wooldridge 2009, S. 25). Diese Tatsache vereinfacht den Agenten-Entwurf zumindest theoretisch, ließe sich doch im Prinzip für jeden Zustand die optimale Aktion in Form einer Tabelle vorhalten.

Kontrollierbar. Die Umwelt ist in gewissem Grade kontrollierbar (*–/controllable*) durch den Agenten, wenn der Agent sie (teilweise) verändern kann. Die Unterscheidung zwischen Kontrollierbarkeit und Zugänglichkeit macht deutlich, dass Einfluss- und Wahrnehmungsbereich eines Agenten nicht übereinstimmen müssen.

Teleologisch. Eine Umgebung kann als teleologisch (*multiagent/teleological*) bezeichnet werden, wenn Teile von ihr zweckgerichtet handeln, d. h. andere Agenten vorhanden sind. Diese Eigenschaft ist bei Multiagentensystemen immer erfüllt.

Die komplexeste Klasse von Umgebungen wird im Allgemeinen durch die Kombination der Eigenschaften unzugänglich, nicht-deterministisch, dynamisch und kontinuierlich gebildet (Wooldridge 2009, S. 25). Dabei wird die Komplexität immer aus der Sicht des Agenten bzw. seines Konstrukteurs beurteilt: Die komplexeste Umgebungsklasse stellt

[13] Die erfolgreiche Integration von zielgerichtetem und reaktivem Verhalten ist ein Schlüsselproblem beim Entwurf von Agenten (Wooldridge 2013, S. 10).

die höchsten Anforderungen an die Flexibilität des Agentenverhaltens. Darüber hinaus wird die Umgebung in der Regel als gegeben betrachtet, was in der agentenbasierten Simulation nicht zutrifft. Hier ist die Klassifikation der Umgebungseigenschaften also auch für den Entwurf der Umgebung relevant und sollte von Werkzeugen entsprechend berücksichtigt werden (siehe Abschnitt 4.1).

2.3.2. Begriffsbestimmung Multiagentensystem

Ein intuitives Verständnis von Multiagentensystem (MAS) lässt sich direkt aus der Bezeichnung selbst sowie der Definition von Agent ableiten: Es handelt sich um ein System, das aus mehreren Agenten in einer gemeinsamen Umgebung gebildet wird. Jeder einzelne Agent interagiert über seine Sensoren und Effektoren mit dieser Umgebung, wobei sich sein lokaler Wahrnehmungsbereich mit denen anderer Agenten überschneiden kann. Da die Art der Agenten nicht näher spezifiziert ist, müssen sie sich weder in ihrer Struktur noch in ihren Zielen ähneln; sie können heterogen sein.

2.3.2.1. Eigenschaften von Multiagentensystemen

Multiagentensysteme zeichnen sich durch folgende charakteristische Eigenschaften aus (Jennings et al. 1998, S. 17):

- Jeder Agent besitzt nur eine beschränkte Sicht auf das Gesamtsystem. Sein Wissen über den globalen Zustand ist daher immer unvollständig.
- Das System wird nicht von einer zentralen Instanz gesteuert; dies widerspräche auch der Autonomie von Agenten. Dass sich alle anderen Agenten freiwillig einem Agenten unterordnen, der ihre Handlungen steuert, ist zwar denkbar, stellt aber nur einen Spezialfall dar. In der agentenbasierten Simulation, in der jedes Modell ein MAS bildet (vgl. Kapitel 3.2), könnte allerdings die modellglobale Simulationsuhr als eine Form von globaler Steuerung interpretiert werden (Klügl 2001, S. 17).
- Nicht nur die Steuerung, sondern auch die Datenhaltung ist dezentral. Dies folgt direkt aus dem Agentenkonzept: Jeder Agent kapselt seinen Zustand und damit sein Wissen über die Umwelt und die anderen Agenten. Durch die Beschränkung seiner Wahrnehmung ist er außerdem auf die lokal verfügbaren Daten angewiesen, um sein Wissen zu aktualisieren. Hierbei ist es möglich, dass Inkonsistenzen auftreten – sowohl in Bezug auf die Daten eines Agenten als auch bezüglich der Daten verschiedener Agenten.
- Berechnungen geschehen asynchron. Auch diese Eigenschaft folgt direkt aus dem Agentenkonzept: Jeder Agent handelt autonom und entscheidet in Reaktion auf Veränderungen der Umwelt und seine internen Zielvorgaben, wann er welche Aktion ausführen möchte. Die einzelnen Agenten müssen sich weder untereinander noch zwangsläufig mit der Umgebung in ihren Aktionen synchronisieren.[14]

[14] Auch in Situationen, die ein gewisses Maß an Synchronisation voraussetzen, wie z. B. die Kooperation mehrerer Agenten, können die Berechnungen, die jeder einzelne Agent durchführt, durchaus asynchron

Damit sind MAS offene Systeme im Sinne von Hewitt (1986, S. 272f), der neben den oben genannten Eigenschaften noch die Nebenläufigkeit der Systemkomponenten betont. Diese ist für Multiagentensysteme ebenfalls erfüllt und kann sogar als definierendes Kriterium gelten: Agenten, die gleichzeitig in einer gemeinsamen Umgebung existieren, werden aufgrund ihrer Autonomie immer nebenläufig sein, d. h. sie können unabhängig voneinander handeln[15]. Ob und wie sie diese Unabhängigkeit zugunsten gemeinsamer Ziele aufgeben, ist eine andere Frage. Darüber hinaus kann auch die Umgebung selbst nebenläufig zu den Agenten sein, wenn in ihr zusätzlich zu den Agenten noch weitere dynamische Prozesse agieren.

Die Offenheit des Systems bezieht sich bei Hewitt auf den Zufluss von Information über Kommunikation mit außerhalb liegenden Quellen. Systemtheoretisch gesehen unterhalten offene Systeme mindestens eine Interaktionsbeziehung zu einem umgebenden System, d. h. mindestens ein Element des offenen Systems wird in seinem Zustand von außen kausal beeinflusst (es existiert ein Systemeingang) oder beeinflusst selbst den Zustand eines außerhalb liegenden Elements (es existiert ein Systemausgang) (Page 1991, S. 2f). Unter diesem Gesichtspunkt ließe sich die Offenheit auch auf die Agenten selbst beziehen: Interpretiert man die Menge der vorhandenen Agenten als (Teil-)Zustand der Umgebung innerhalb des MAS, so verändert sich dieser, wenn ein Agent hinzukommt oder ein Agent das System verlässt. In diesem Fall wird das gesamte MAS, also Agenten einschließlich Umgebung, als offenes System betrachtet. Die Systemumgebung ist dann nicht identisch mit der MAS-Umgebung, die ein Element des Systems bildet.

Diese Sichtweise nehmen offenbar auch Huhns und Stephens ein, wenn sie folgende charakteristische Eigenschaften einer MAS-Umgebung identifizieren (1999, S. 81f), die zu den im vorigen Abschnitt eingeführten Umgebungseigenschaften hinzukommen:

1. Die Umgebung stellt eine Infrastruktur zur Verfügung, die Kommunikations- und Interaktionsprotokolle bestimmt. Während Kommunikationsprotokolle den Austausch einzelner Nachrichten zwischen Agenten regeln, spezifizieren Interaktionsprotokolle den Austausch von Nachrichtenfolgen.

2. Die Umgebung ist typischerweise offen und hat keinen zentralen Designer. Ich interpretiere *offen* hier in dem Sinne, dass Agenten das System betreten und wieder verlassen können.

3. Die Umgebung enthält Agenten, die autonom und verteilt sind und die egoistisch oder kooperativ sein können. Damit gehen Huhns und Stephens implizit davon aus, dass ein MAS ein verteiltes System darstellt. Auf konzeptueller Ebene kann die Verteilung der Agenten als Nebenläufigkeit interpretiert werden; schließlich ist Verteilung eine Möglichkeit, Nebenläufigkeit zu realisieren.

Die Fähigkeit zu kommunizieren, d. h. Nachrichten zu senden und zu empfangen, wird für Agenten eines Multiagentensystems in der Regel vorausgesetzt (Huhns und Stephens

erfolgen. Zudem ist die Synchronisation nur zeitweise nötig und betrifft nur die an der Aufgabe beteiligten Agenten.

[15] Diese Bedeutung von Nebenläufigkeit ist von der Definition nebenläufiger Transitionen in (Jessen und Valk 1987, S. 27 und S. 47) abgeleitet.

1999, S. 83). Manchmal können Sensoren und Effektoren sogar auf diese Kommunikationsfähigkeit beschränkt sein. Jacques Ferber bezeichnet den Spezialfall eines Multiagentensystems, das ausschließlich aus solchen Agenten besteht, als *rein kommunizierendes* MAS (Ferber 1999, S. 11). Er differenziert zwischen kommunizierenden und situierten MAS: In kommunizierenden MAS findet die Interaktion zwischen den Agenten direkt durch Nachrichtenaustausch statt, in *rein situierten* MAS dagegen nur indirekt über die Ausbreitung von Signalen in der Umgebung. Situiert ist ein Multiagentensystem genau dann, wenn die Umgebung eine räumliche Ausprägung besitzt und die Agenten in ihr positioniert sind. Damit schließen sich kommunizierend und rein situiert gegenseitig aus; ebenso situiert und rein kommunizierend, da bei letzterem die räumliche Ausdehnung der Umgebung nicht betrachtet wird.

In Ferbers Definition von MAS ist eine räumliche Umgebung dagegen der Normalfall (Ferber 1999, S. 11). Zusätzlich zu den Agenten befinden sich in der Umgebung noch andere Objekte, die von den Agenten beliebig manipuliert werden können. Beziehungen verknüpfen Objekte und Agenten untereinander und miteinander. Dies können die bereits erwähnten Kommunikationsbeziehungen zwischen Agenten sein, ebenso aber z. B. Abhängigkeitsbeziehungen (Sichman et al. 1994; Sichman und Conte 2002) oder durch einen organisatorischen oder sozialen Kontext vorgegebene Beziehungen (Horling und Lesser 2005). Da solche Beziehungen das Verhalten der Agenten beeinflussen, sollten sie explizit repräsentiert werden (Jennings 2000, S. 281).

2.3.2.2. Beziehungen in Multiagentensystemen

Abbildung 2.9 (nach Jennings 2000, S. 281) zeigt die *typische Struktur* eines Multiagentensystems: In einer gemeinsamen, ggf. räumlichen Umgebung befindet sich eine Menge von Agenten, deren Wahrnehmungs- und Einflussbereiche sich überschneiden können; zwischen den Agenten bestehen Interaktionsbeziehungen, die in der Regel den Austausch von getypten Nachrichten bedeuten; zusätzliche Organisationsbeziehungen zwischen den Agenten ordnen diese beispielsweise bestimmten Gruppen zu oder weisen ihnen bestimmte Rollen zu.

Bei der Interaktion zwischen Agenten können zwei Ebenen unterschieden werden, die aufeinander aufbauen: Auf der unteren Ebene werden einzelne Nachrichten ausgetauscht (Kommunikation), auf der höheren Ebene strukturierte Folgen von Nachrichten (Konversation) (Huhns und Stephens 1999, S. 96). Die technischen Voraussetzungen für *Kommunikation* werden von der Umgebung des Multiagentensystems bereitgestellt; sie sind nicht agentenspezifisch und werden in der Regel als vorhanden vorausgesetzt (Klügl 2001, S. 92). Im Hinblick auf die Anzahl der Empfänger einer Nachricht werden wie bei der Kommunikation nebenläufiger Prozesse *Punkt-zu-Punkt-Kommunikation* (ein Sender, ein Empfänger) und *Gruppenkommunikation* (ein Sender, mehrere Empfänger) unterschieden, wobei letztere danach differenziert werden kann, ob alle Teilnehmer der Gruppe (*broadcast*) adressiert werden oder nur eine Teilmenge (*multicast*) (Burkhart 2006, S. 637).

Damit die Agenten in MAS die ausgetauschten Nachrichten auch verstehen können, ist eine gemeinsame Sprache nötig, die Aufbau, Typ und Bedeutung von Nachrichten eindeutig festlegt. Gerade in offenen Systemen mit heterogenen Agenten kommt dieser Sprache eine Schlüsselrolle zu; es ist daher wichtig, dass sie möglichst standardisiert ist. Die

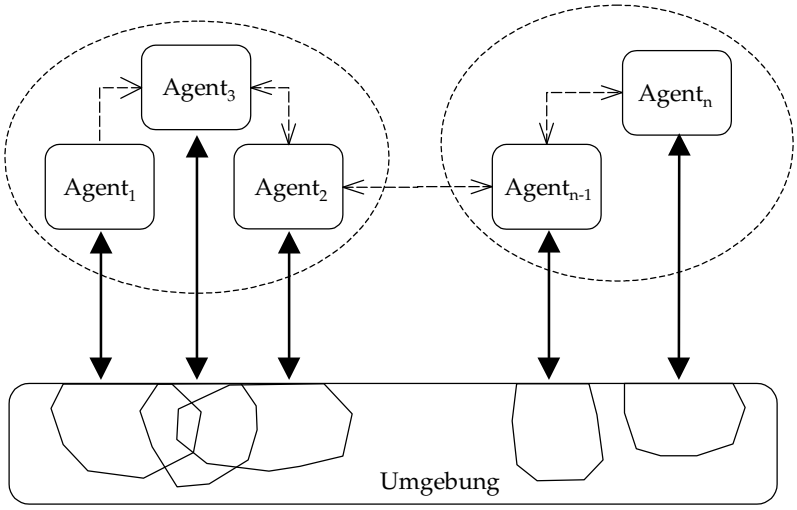

Abbildung 2.9.: Schematische Darstellung eines Multiagentensystems. Die durchgezogenen Pfeile entsprechen wie in Abbildung 2.8 der Interaktion zwischen Agent und Umgebung, die gestrichelten Pfeile repräsentieren direkte Interaktionen zwischen Agenten. Zusätzlich können organisatorische Beziehungen zwischen Agenten bestehen (gestrichelte Ellipsen).

bekanntesten Agenten-Kommunikationssprachen (*agent communication languages*, ACL), die dies anstreben, sind KQML und FIPA ACL (Labrou et al. 1999). Beide orientieren die möglichen Nachrichten-Typen an der Theorie der Sprechakte (*speech acts*, begründet von Austin 1962 und Searle 1969), die sprachliche Äußerungen als Handlungen auffasst, welche u. a. dazu dienen, beim Empfänger eine Wirkung hervorzurufen. Beispiele für Sprechakte (und Nachrichten-Typen) sind Zusicherung (*assert*), Frage (*query*) und Anweisung (*command*).

Für die Koordination von Agenten ist es in der Regel nötig, eine Reihe von Nachrichten in einer bestimmten Abfolge auszutauschen. Um solche *Konversationen* zu modellieren, werden Interaktionsprotokolle verwendet, die eine gültige Reihenfolge definieren. Ein Beispiel für ein Interaktionsprotokoll ist das weit verbreitete Kontraktnetz-Protokoll (Smith 1980) für die Verteilung von Aufgaben auf Agenten; es wird im folgenden Abschnitt 2.3.3 näher erläutert. Verfolgen die beteiligten Agenten ein gemeinsames Ziel, so verhalten sie sich kooperativ; Ziel des Interaktionsprotokolls ist dann die Aufstellung eines kohärenten Plans für gemeinsame Aktion, ohne die Autonomie der Agenten zu verletzen. Sind die Agenten dagegen egoistisch oder verfolgen konfligierende Ziele, wird das Interaktionsprotokoll eher dem Aushandeln eines Kompromisses gleichen, bei dem alle beteiligten Agenten ihren eigenen Nutzen soweit wie möglich maximieren können (Huhns und Stephens 1999, S. 96). Zur Spezifikation solcher Protokolle werden häufig Endliche Automaten oder Petrinetze eingesetzt (Ferber 1999, S. 326).

Die Beziehungen zwischen den Agenten können vorgegeben und damit statisch sein; im Allgemeinen werden sie sich jedoch im Laufe der Zeit verändern: Neue Beziehungen

kommen hinzu und bestehende entwickeln sich weiter oder verschwinden ganz. Dies gilt um so mehr, wenn MAS als offene Systeme aufgefasst werden, die Agenten beliebig betreten und wieder verlassen können. Für die Beschreibung des Zusammenspiels der Agenten innerhalb eines Multiagentensystems werden häufig soziale Begriffe herangezogen und das MAS nach dem Vorbild der menschlichen Gesellschaft als (künstliche) Gesellschaft von Agenten betrachtet, um bessere Koordinationsmechanismen für Agenten zu finden (z. B. Bond und Gasser 1988b; Weiss 1999). An der Schnittstelle zwischen Soziologie und KI hat sich seit Ende der 1990er Jahre die Sozionik als interdisziplinäres Forschungsfeld entwickelt. Sie befasst sich mit der Übertragung soziologischer Erkenntnisse auf MAS und der Erforschung sogenannter „hybrider Gesellschaften“, in denen menschliche Benutzer mit künstlichen Agenten interagieren (Müller et al. 1998; Malsch und Schulz-Schaeffer 2007).

Für den Kontext dieser Arbeit ist festzuhalten, dass MAS offene Systeme darstellen, in denen zwischen den Systemelementen – den Agenten und der Umgebung – vielfältige Beziehungen bestehen können. Diese Beziehungen lassen sich einteilen in Interaktionen zwischen Agent und Umgebung, Interaktionen zwischen Agenten (Kommunikation) und organisatorisch bedingte Beziehungen. Die Kommunikation zwischen Agenten kann dabei direkt durch Austausch von beliebig komplex strukturierten Nachrichten erfolgen oder indirekt über Veränderung der Umgebung. Diese wird in MAS, welche agentenbasierte Modelle realisieren, immer dann eine explizite Raumrepräsentation benötigen, wenn die räumliche Verteilung der als Agenten modellierten Akteure Einfluss auf die Systemdynamik hat.

2.3.3. Entwicklungsgeschichte

Multiagentensysteme sind heute ein interdisziplinäres Forschungsgebiet, das sich nicht nur auf die Informatik, insbesondere deren Teilgebiete Künstliche Intelligenz, Verteilte Systeme und Softwaretechnik, erstreckt, sondern auch auf Disziplinen wie Soziologie, Wirtschaftswissenschaften und Ökologie (Niazi und Hussain 2011). Entstanden ist das Forschungsgebiet zu Beginn der achtziger Jahre im Rahmen der Verteilten Künstlichen Intelligenz (VKI), welche sich mit Informationsverarbeitung in verteilten intelligenten Systemen beschäftigt (Weiss 1998). Der Schwerpunkt der Arbeiten lag zunächst auf dem sogenannten verteilten Problemlösen (*distributed problem solving*), d. h. der Frage, wie das Lösen einer speziellen – nicht notwendigerweise ebenfalls verteilten – Aufgabe aufgeteilt werden kann auf verschiedene, miteinander kooperierende Module (Bond und Gasser 1988a, S. 3). Die beteiligten Problemlöse-Module (Agenten) verfügen nur über begrenzte Fähigkeiten und begrenztes Wissen und müssen sich daher koordinieren, um das gemeinsame Ziel zu erreichen.

Das bekannteste und am weitesten verbreitete Koordinationsverfahren ist das Kontraktnetz-Protokoll (*Contract Net Protocol*, Smith 1980; Davis und Smith 1983), dem die Idee des Aushandelns von Verträgen zugrunde liegt. Es liefert eine Lösung für das sogenannte Verbindungsproblem: Wie kann ein Agent (Manager) für eine bestimmte Aufgabe einen anderen Agenten (Vertragsnehmer) finden, der am besten geeignet ist, diese Aufgabe zu bearbeiten? Das Protokoll sieht folgende Schritte vor:

1. Der Manager sendet die Aufgabenbeschreibung an alle oder eine ausgewählte Teilmenge der anderen Agenten. Die Beschreibung enthält u. a. eine Aufzählung der benötigten Fähigkeiten und einen Zeitpunkt, bis zu dem Rückmeldungen entgegengenommen werden.

2. Die potentiellen Vertragsnehmer evaluieren die Aufgabenbeschreibung und senden, falls sie in Frage kommen und interessiert sind, ein Gebot an den Manager.

3. Der Manager bewertet die erhaltenen Gebote und erteilt demjenigen Agenten, der das am besten bewertete Gebot abgegeben hat, den Zuschlag. Damit ist der Vertrag zustandegekommen. Jede weitere Kommunikation bezüglich der Aufgabe, wie z. B. das Übermitteln von Resultaten, findet jetzt nur noch zwischen den beiden beteiligten Agenten statt. Falls kein Agent innerhalb des festgelegten Zeitraums ein Gebot abgibt, kann der Manager das Protokoll erneut beginnen.

Der große Vorteil des Kontraktnetz-Protokolls ist seine Flexibilität: Es ist unabhängig von der Anzahl der beteiligten Agenten und erlaubt eine hierarchische Aufgaben-Zuordnung. Da die Rollen Manager und Vertragsnehmer nicht *a priori* festgelegt sind, kann beispielsweise ein Agent, der eine Aufgabe übernommen hat, diese wiederum in Teilaufgaben zerlegen und für diese Teilaufgaben als Manager auftreten. Andererseits gewährleistet das Verfahren keine optimale Zuordnung von Aufgaben zu Agenten, da der von seinen Fähigkeiten her am besten geeignete Agent zum Zeitpunkt der Aufgabenausschreibung ggf. aufgrund von Arbeitsüberlastung kein Gebot abgibt.

Von dem beschriebenen generellen Ablaufschema existieren verschiedene Varianten und Erweiterungen, welche die Performanz verbessern oder es an anwendungsspezifische Anforderungen anpassen. Auch in dem in Kapitel 6 beschriebenen Modell eines Stadtkurierdienstes wird eine Variante des Kontraktnetz-Protokolls eingesetzt, um das Auftragsvermittlungsverfahren abzubilden.

Das globale gemeinsame Ziel unterscheidet verteilte Problemlöser von Multiagentensystemen im engeren Sinne, in denen die Agenten individuelle, ggf. konträre Ziele verfolgen können (Bond und Gasser 1988a, S. 3). Da sich die Aktionen der Agenten in der gemeinsamen Umgebung gegenseitig beeinflussen, ist Koordination auch hier unerlässlich; zumal wenn nur begrenzte Ressourcen zur Verfügung stehen, um die die Agenten konkurrieren. In diesem Fall besteht die Koordination darin, Konflikte mittels Verhandlungen zu lösen. Dabei können sich die Agenten unterschiedlicher Strategien bedienen, um das von ihnen angestrebte Ergebnis zu erhalten.

Die Komplexität der Verhandlungsprotokolle ist abhängig von Art und Anzahl der strittigen Punkte, der Anzahl der beteiligten Agenten und der Art und Weise ihrer Interaktion. Drei Kategorien zunehmender Komplexität bezüglich der Teilnehmer lassen sich unterscheiden: Ein Agent verhandelt mit genau einem anderen Agenten ($1 : 1$), ein Agent verhandelt mit mehreren anderen Agenten gleichzeitig ($1 : n$), und mehrere Agenten verhandeln mit mehreren Agenten gleichzeitig ($m : n$). Durch das zunehmende Interesse an elektronischem Handel (*e-commerce*) ist die Entwicklung automatisierter Verhandlungen zwischen mehreren Teilnehmern, beispielsweise basierend auf dem Kontraktnetz-Protokoll oder auf Auktionsmechanismen, ein aktives Forschungsgebiet (Jennings et al. 2001; Ito et al. 2010).

Eine häufig gemachte Annahme ist, dass sich Agenten rational verhalten, d. h. immer diejenige Aktion wählen, die in der aktuellen Situation hilft, den eigenen Nutzen zu maximieren. Ein Agent wird somit als *homo oeconomicus* betrachtet. Wenn für jede Situation die Ergebnisse möglicher Aktionen bekannt sind, kann mit Hilfe von spieltheoretischen Modellen die jeweils rationale Handlung bestimmt werden. Die Spieltheorie, deren Gegenstand die Untersuchung interaktiver Strategien von Entitäten mit widerstreitenden Interessen ist (Holler und Illing 2009), wird auch im Rahmen von Multiagentensystemen eingesetzt, um die Interaktionen zwischen Agenten zu analysieren. Dies ist naheliegend, können doch die Begegnungen zweier Agenten als Spiel aufgefasst werden, in dem die Agenten strategisch handeln müssen, um das Ergebnis zu ihren Gunsten zu beeinflussen (Wooldridge 2009, S. 151). Eines der klassischen abstrakten Zwei-Personen-Spiele der Spieltheorie ist das sogenannte Gefangenendilemma (*Prisoner's Dilemma*), in dem zwei Gefangene sich unabhängig voneinander entscheiden müssen, ob sie eine gemeinsam verübte Tat gestehen oder nicht. Die Spielsituation ist so konstruiert, dass einseitiges Gestehen den größten Gewinn verspricht, beide Gefangene jedoch bei gegenseitiger Kooperation (beide schweigen) besser abschneiden, als wenn beide gestehen. Wird dieses Spiel mehrfach mit den gleichen Partnern wiederholt, so dass die Entscheidungen in der Vergangenheit das aktuelle Verhalten beeinflussen können, eignet sich das Gefangenendilemma beispielsweise, um die Evolution von Strategien zu untersuchen (Axelrod 1997, Kap. 1).

In der bisher beschriebenen ersten Forschungsrichtung, die auch als erste Generation von Agenten und MAS bezeichnet wird (Nwana und Ndumu 1997), stehen Makro-Aspekte im Vordergrund, d. h. Aspekte auf der Ebene des Systems, der Gesellschaft der Agenten. Forschungsarbeiten konzentrieren sich auf Mechanismen zur Koordination von Agenten, die entweder einander wohlgesonnen sind (*benevolent agents*, Rosenschein und Genesereth 1985) und sich daher kooperativ verhalten, oder eigennützig (*self-interested agents*) und daher auftretende Konflikte lösen müssen. Die Kommunikation zwischen Agenten findet in der Regel auf einem hohen Abstraktionsniveau statt und die Agenten selbst sind eher deliberativ, d. h. sind anhand des Symbolverarbeitungs-Paradigmas der traditionellen KI konstruiert (vgl. die Übersicht über Agenten-Architekturen in Abschnitt 2.3.4).

In der zweiten Forschungsrichtung bzw. zweiten Generation, die ungefähr 1990 begann, ist dies nicht mehr der Fall. Hier wird von der Hypothese ausgegangen, dass die einzelnen Agenten nicht notwendigerweise intelligent sein müssen, damit das Gesamtsystem intelligentes Verhalten zeigt (Ferber 1999, S. 16). Indem sie auf Ereignisse in der Umgebung reagieren, können Agenten auch ohne eine symbolische Repräsentation der Umwelt entscheiden, welche Aktion der aktuellen Situation angemessen ist. Kommunikation zwischen solchen reaktiven Agenten findet in der Regel auf einem niedrigeren Abstraktionsniveau statt, ggf. sogar nur indirekt durch Veränderung der Umgebung. Koordination ist dennoch möglich, wie das – besonders in dem als *Artificial Life* (ALife) bezeichneten Teilgebiet (Langton 1988; Bedau 2003) beliebte – Vorbild sozialer Insekten zeigt, die in komplexen Strukturen (Staaten) organisiert sind und z. B. bei der Futtersuche kooperieren.[16] Im Gegensatz zu der explizit vereinbarten Koordination zwischen intelli-

[16] Ameisen sind laut Mitchell Resnick (2000, S. 59) das inoffizielle Maskottchen der ALife-Gemeinde.

genten Agenten entsteht in reaktiven Systemen die Koordination aus den Interaktionen der Agenten mit ihrer Umgebung – sie ist eine *emergente* Eigenschaft des Systems.

Den wohl größten Einfluss auf die Ausbildung dieser Forschungsrichtung hatte Rodney Brooks mit seiner Kritik am symbolischen Ansatz der KI (Brooks 1991a, b), die aus seinen Arbeiten in der Robotik resultierte. Er betonte die Situiertheit eines Agenten in seiner Umgebung: Intelligentes Verhalten entsteht aus der Interaktion des Agenten mit dieser Umgebung und durch die Interaktion verschiedener einfacherer Verhaltensbausteine. Die von ihm vorgeschlagene Subsumption-Architektur (siehe Abschnitt 2.3.4) vereinfachte die Konstruktion robuster autonomer Roboter.

Die Themen Situiertheit und Emergenz stehen im Vordergrund dieses neuen Ansatzes, der als situiert, verhaltensorientiert oder reaktiv bezeichnet wird (Wooldridge 2009, S. 85). Emergenz bezieht sich hier sowohl auf das Verhalten eines einzelnen Agenten, das sich aus dem Zusammenspiel mehrerer einfacher Verhaltensmodule ergibt, als auch auf das Verhalten des Multiagentensystems als Ganzes, das sich aus der Interaktion der einzelnen Agenten miteinander bzw. mit ihrer Umwelt ergibt. Letzteres wird auch in der agentenbasierten Simulation vertreten (vgl. Kapitel 3.2). Verbunden mit diesem Ansatz ist eine Konzentration auf die Mikro-Aspekte eines Multiagentensystems, z. B. die Konstruktion des Verhaltens der einzelnen Agenten.

2.3.4. Ansätze zur Realisierung

Die bisherige Diskussion von Agenten und Multiagentensystemen hat im Wesentlichen auf der konzeptuellen Ebene stattgefunden. Um das Agentenkonzept nutzen zu können, müssen solche Systeme realisiert werden. Hierfür wurden verschiedene Ansätze entwickelt, die sich entweder mit dem Aufbau einzelner Agenten befassen (sogenannte Agenten-Architekturen) oder mit der Konstruktion des gesamten MAS.

2.3.4.1. Entwurf und Implementation von Agenten

Vorausgesetzt, dass Sensoren und Effektoren gegeben sind, besteht die zentrale Aufgabe bei der Konstruktion eines Agenten in der Realisierung seines Verhaltens: Wie wählt der Agent die zur aktuellen Situation passende Aktion aus? Und wie wird dabei eine Balance zwischen reaktivem und zielgerichtetem Verhalten erreicht? Das Problem liegt darin, dass die Entscheidung über die jeweilige Aktion in der Regel nicht zum Zeitpunkt der Programmierung, sondern zum Zeitpunkt der Ausführung des Agenten getroffen wird. Die Eigenschaften der Umgebung beeinflussen dabei die Komplexität des Entscheidungsprozesses (siehe Abschnitt 2.3.1).

Das hierfür zuständige Agenten-Programm ist im Prinzip eine Software-Architektur für ein in eine Umgebung eingebettetes Entscheidungssystem (Wooldridge 2013, S. 6). Wie bei anderen Softwaresystemen auch legt die Architektur fest, aus welchen Komponenten das System – in diesem Fall der Agent – aufgebaut ist und wie deren Schnittstellen, Interaktionen und Abhängigkeiten untereinander beschaffen sind Ähnlich wie bei der Definition des Agentenbegriffs gibt es auch bei Agenten-Architekturen eine große Anzahl verschiedener Ansätze, die sich in ihrer Komplexität und ihrer Eignung für bestimmte Typen von

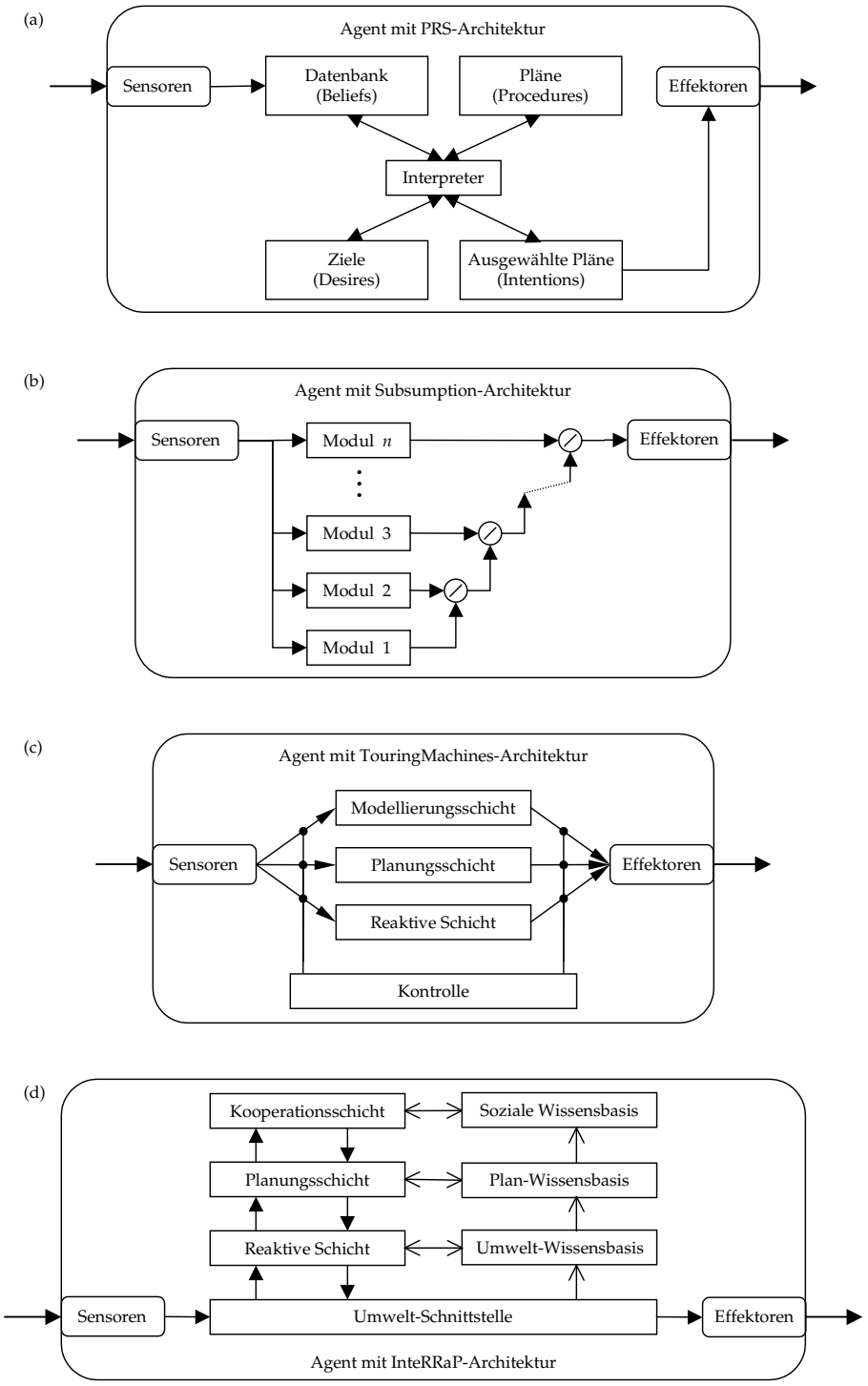

Abbildung 2.10.: Beispiele für verschiedene Agenten-Architekturen: deliberativ (a), reaktiv (b) und hybrid mit horizontaler (c) bzw. vertikaler Schichtung (d).

Umgebungen unterscheiden. Sie lassen sich nach dem Grad der verwendeten „internen Intelligenz“ klassifizieren:[17]

- *Deliberative* Architekturen zeichnen sich durch eine explizite, symbolische Repräsentation der Umgebung aus, dem internen Weltmodell. Dieses Weltmodell wird in Verbindung mit den ebenfalls explizit repräsentierten Zielen verwendet, um Aktionsfolgen zu planen. Durch schrittweise Abarbeitung solcher Pläne versucht der Agent, seine Zielvorgaben zu erreichen. Deliberative Architekturen lassen sich der klassischen, wissensbasierten Ausrichtung der KI zuordnen: Das Weltmodell bildet die Wissenbasis, aus der z. B. mittels logischem Schließen die geeigneten Aktionen abgeleitet werden.

  Anhand der verwendeten Wissensstrukturen und Inferenzmechanismen kann diese Kategorie weiter unterteilt werden in rein logik-basierte Architekturen, die Agenten im Prinzip als Theorem-Beweiser auffassen (Wooldridge 2009, S. 50f), und kognitive Architekturen, die auf kognitionswissenschaftlichen oder psychologischen Modellen beruhen (Klügl 2001, S. 27). Letztere umfassen im Wesentlichen einen als *BDI* bekannten Ansatz, der die interne Wissensrepräsentation in Form von mentalen Kategorien (*beliefs, desires, intentions*) beschreibt und einen Steuerungsmechanismus definiert, mit dessen Hilfe der Agent entscheidet, was zu tun ist (Auswahl der Intentionen) und wie dies zu erreichen ist (Folge von Aktionen). Abbildung 2.10 (a) auf Seite 45 zeigt als Beispiel für eine deliberative Architektur das *Procedural Reasoning System* (PRS; Georgeff und Lansky (1987); Georgeff und Ingrand (1989); Ingrand et al. (1992)), die bekannteste BDI-Architektur.

- *Reaktive* Architekturen verwenden kein internes Weltmodell und nur einfache symbolische Repräsentationen der Perzepte. Ihr Grundprinzip ist die Aktionsauswahl in Reaktion auf die Umgebung[18]; dies erfolgt meist durch Regeln der Form $Situation \rightarrow Aktion$. Diese Architekturen lassen sich der neueren, verhaltensorientierten Ausrichtung der KI zuordnen und werden daher auch als *situiert* oder *verhaltensorientiert* bezeichnet, je nachdem, ob die Situiertheit des Agenten oder der Aufbau seines Verhaltens aus einzelnen Verhaltensbausteinen betont wird (Wooldridge 2009, S. 85). Da kein Wissen über die Welt aufgebaut und verwendet wird, können solche Architekturen in Abgrenzung zu den deliberativen Architekturen auch als *subkognitiv* bezeichnet werden (Klügl 2001, S. 24).

  Die bekannteste reaktive Architektur ist die von Rodney Brooks für mobile Roboter vorgeschlagene *Subsumption*-Architektur (Brooks 1986). Sie besteht aus einer Menge einfacher Verhaltensmodule, die jeweils genau eine Aufgabe erfüllen. Diese Module arbeiten parallel, d. h. jedes Modul erhält die aktuellen Sensordaten und wählt ggf. eine passende Aktion aus. Die Entscheidung über die tatsächlich auszuführende Aktion wird über die Subsumptionshierarchie getroffen, in der die Module

[17] Für den Anwendungsbereich der agentenbasierten Simulation, der in dieser Arbeit im Vordergrund steht, erscheint diese Klassifikation ausreichend; sie orientiert sich an dem von Klügl (2001, S. 21ff) vorgestellten Schema. Andere, stärker von der KI-Perspektive beeinflusste Klassifikationen geben z. B. Wooldridge (2009) und Russell und Norvig (2009, S. 47ff).

[18] Proaktives Verhalten ist gleichwohl möglich, insofern ist die Bezeichnung „reaktiv“ etwas irreführend.

nach absteigender Priorität in Schichten angeordnet sind (siehe Abbildung 2.10 (b)). Schichten höherer Priorität blockieren die Ausführung der von Schichten niedrigerer Priorität gewählten Aktionen, so dass immer die jeweils höchstpriorisierte Aktion ausgeführt wird. Durch die Anordnung überlebenswichtiger Funktionen wie Hindernis-Vermeidung und Energie-Versorgung in den unteren Schichten mit hoher Priorität lässt sich ein Agent mit robustem, flexiblem Verhalten konstruieren, der nur dann, wenn die Umgebung es zulässt, sich mit den niedriger priorisierten, aber komplexeren Aufgaben beschäftigt.

- Ganz ohne inneren Zustand und in der Regel auch ohne Symbolmanipulation kommen *tropistische* Agenten aus (Genesereth und Nilsson 1987); die zugehörigen Architekturen können daher auch als *subsymbolisch* bezeichnet werden (Klügl 2001, S. 22). Typischerweise findet die Aktionsauswahl als relativ direkter Reflex auf die aktuellen Sensorwerte statt. Als klassisches Beispiel für eine sehr einfache subsymbolische Architektur dieser Art können die Vehikel von Valentino Braitenberg (1986) gelten, die in Form eines Gedankenexperiments demonstrieren sollen, dass komplexes Verhalten auf simple interne Strukturen zurückgeführt werden kann. Die Vehikel stellen kleine situierte Roboter dar, die sich in Reaktion auf Reizquellen in der Umgebung bewegen. Der Steuerungsmechanismus wird von einem einfachen neuronalen Netz mit fester Kantenbewertung gebildet, so dass das mögliche Verhalten eines Vehikels vordefiniert ist. Durch die Verwendung mehrschichtiger neuronaler Netze, deren Kantenbewertungen oder ggf. Verknüpfungsstrukturen z. B. mittels genetischer Algorithmen veränderbar sind, lässt sich auch adaptives Verhalten für tropistische Agenten erzielen (Ferber 1999, S. 138f).

  Da ihnen interne Zustände und somit ein interner Wissensspeicher fehlen, sind tropistische Agenten besonders stark von ihrer Umgebung abhängig: Jede Entscheidung wird ausschließlich aufgrund des lokal wahrnehmbaren Umgebungszustands getroffen. Die Handlungen eines solchen Agenten können daher als situiert bezeichnet werden (Ferber 1999, S. 193). Durch Nutzung der Umgebung als „externen“ Speicher lässt sich die Beschränkung des fehlenden internen Speichers abmildern.

- *Hybride* Architekturen, zum Teil auch als Schichtenmodelle (*layered architectures*) bezeichnet, kombinieren Komponenten verschiedener Architektur-Kategorien mit dem Ziel, die Schwächen der einzelnen Ansätze zu eliminieren. In der Regel werden mindestens eine reaktive und eine deliberative Komponente miteinander verbunden, um einen Agenten mit sowohl reaktivem als auch proaktivem Verhalten auszustatten. Nach der Art des Informations- und Kontrollflusses lassen sich horizontal und vertikal geschichtete Architekturen unterscheiden (Wooldridge 2009, S. 93).

  Bei horizontaler Schichtung sind alle Module direkt mit den Sensoren und den Effektoren verbunden, so dass sie alle parallel mit denselben Eingangsdaten arbeiten. Um konsistentes Verhalten zu garantieren, wird üblicherweise eine zentrale Vermittlungsinstanz benötigt, die entscheidet, welche Komponente das aktuelle Agentenverhalten bestimmt. Die *TouringMachines*-Architektur von Ferguson (1992) ist ein typische Beispiel einer horizontal geschichteten hybriden Architektur (siehe Abbildung 2.10 (c)).

Bei vertikaler Schichtung wird kein zentraler Vermittler benötigt, da nur jeweils eine Komponente mit Sensoren bzw. Effektoren verbunden ist. Informationen werden von einer Komponente zur nächsten weitergereicht und die Entscheidung über die auszuführende Aktion in der mit den Effektoren verbundenen Komponente getroffen. Beschränkt sich die Schnittstelle zu Sensoren und Effektoren auf genau eine Komponente, so ergibt sich ein doppelter Durchlauf durch alle Komponenten: zunächst ein aufsteigender Informationsfluss, gefolgt von einem absteigenden Kontrollfluss (Wooldridge 2009, S. 94). Als Beispiel einer solchen Architektur sei InteRRaP (Integration of Reactive Behavior and Rational Planning) genannt (Müller 1996), dargestellt in Abbildung 2.10 (d).

Welche Programmiersprache für die Implementation gewählt wird, ist – wie bei anderen Softwaresystemen auch – prinzipiell unabhängig von der jeweiligen Architektur. So wurde das Procedural Reasoning System ursprünglich in Lisp realisiert und mit PRS-CL[19] existiert ein in Common Lisp geschriebener Nachfolger. Doch gibt es auch Versionen in C (Ingrand et al. 1996), C++ (Lee et al. 1994; d'Inverno et al. 1997) und Java (Huber 1999). Java hat sich als Programmiersprache für Agentensysteme durchgesetzt. Stand der Technik heute sind in der Regel spezielle Frameworks oder Sprach-Erweiterungen von Java, welche die Implementation von Agenten erleichtern (vgl. Mangina 2002; Mascardi et al. 2005; Bordini et al. 2005; Bordini und Dix 2013 für detaillierte Übersichten über derzeit verfügbare Programmierumgebungen).

2.3.4.2. Entwurf und Implementation von MAS

Wie in Abschnitt 2.3.2 eingeführt, besteht ein Multiagentensystem aus einer Menge von Agenten, die in einer gemeinsamen Umgebung agieren. Zwischen den Agenten bestehen Beziehungen, die sich einteilen lassen in Interaktionsbeziehungen zur Kommunikation und organisatorisch bedingte Beziehungen. Um ein MAS zu realisieren, sind daher neben den Agenten auch deren Kommunikations- und Organisationsstrukturen sowie die Umgebung zu entwerfen und zu implementieren.

Für diese softwaretechnisch komplexe Aufgabe werden passende Entwurfsmethoden benötigt, die Entwicklern gestatten, ihre Anwendungen als MAS zu spezifizieren und strukturieren, sowie entsprechende Software-Toolkits, welche die Implementation des so entworfenen MAS unterstützen (Nwana und Ndumu 1999). In den Anfängen der Agenten-Technologie fehlten diese technischen Voraussetzungen, so dass existierende Anwendungen *ad hoc* entworfen wurden; entweder durch Anpassung einer vorhandenen Methode oder indem auf eine systematische Methode ganz verzichtet und das System auf Basis von Intuition und Erfahrung entwickelt wurde. Darüber hinaus ging ein signifikanter Anteil des Aufwands in die Realisierung der benötigten grundlegenden Infrastruktur, bevor mit der Realisierung von Agenten und deren Interaktionen begonnen werden konnte (Jennings et al. 1998, S. 30f).

In den letzten Jahren hat sich diese Situation deutlich verbessert. Blieben in der Anfangsphase designierte Methoden für den Entwurf agentenbasierter Systeme in der Regel

[19] `http://www.ai.sri.com/~prs/` [16.1.2014]

auf das akademische Umfeld beschränkt (Giorgini und Henderson-Sellers 2005), beginnen sie heute industrielle Reife zu zeigen (Winikoff und Padgham 2013). Um das Rad nicht neu erfinden zu müssen, sind die meisten MAS-Entwurfsmethoden von existierenden Methoden und Techniken abgeleitet, entweder von objekt-orientierten oder von wissensbasierten Methoden (Iglesias et al. 1999). Zu letzteren zählt MAS-CommonKADS (Iglesias et al. 1998), eine Erweiterung der etablierten Knowledge-Engineering-Methode CommonKADS. Kernstück beider Entwurfsmethoden ist die Spezifikation des zu entwickelnden Systems mittels einer Reihe von einzelnen Modellen für die wesentlichen Systemmerkmale. In MAS-CommonKADS sind dies das Agentenmodell und das Wissensmodell für die Modellierung der Agenten und ihrer Fähigkeiten, das Kommunikations- und Koordinationsmodell für die Beschreibung der Interaktionen sowie das Organisationsmodell und Aufgabenmodell für die Beschreibung des organisatorischen Rahmens und der dort anfallenden Aufgaben, welche die Agenten zu lösen haben. Die Modelle der Analysephase münden schließlich im Designmodell, das die Systemarchitektur, Agentenplattform und Software-Module beschreibt.

Wesentlich zahlreicher sind MAS-Entwurfsmethoden, welche von objektorientierten Methoden inspiriert wurden. In diese Kategorie fallen u. a. Gaia (Wooldridge et al. 2000; Zambonelli et al. 2003) und deren Erweiterung ROADMAP (Juan et al. 2002), Tropos (Bresciani et al. 2004), MaSe (Wood und DeLoach 2001; DeLoach 2004) und dessen Nachfolger O-MaSe (DeLoach und Garcia-Ojeda 2010) sowie AML/ADEM (Cervenka und Trencansky 2007) als allgemeine und ADELFE (Bernon et al., 2005), MESSAGE (Caire et al. 2001), Prometheus (Padgham und Winikoff 2002, 2004) und PASSI (Cossentino und Potts 2002) als Methoden mit einem speziellen Anwendungsbereich. Diese Ansätze unterscheiden sich darin, inwieweit sie den Software-Entwicklungsprozess unterstützen. Einige Methoden decken alle Phasen von der Anforderungsanalyse bis zur Inbetriebnahme ab, während andere nur ein Vorgehensmodell für Analyse und Design definieren. Allen gemeinsam ist, dass sie von intelligenten Agenten mit explizit repräsentierten mentalen Konzepten wie Wissen, Überzeugungen und Zielen ausgehen, welche über Mittel zum Erreichen jener Ziele wie Pläne oder Intentionen verfügen (Luck et al. 2004, Kap. 4).

Viele Methoden setzen zur Beschreibung des Multiagentensystems die Metapher der (menschlichen) Organisation als sozialer Struktur ein, in welcher Agenten eine oder mehrere verschiedene Rollen annehmen. Die Betonung von Rollen kann sogar als ein wesentliches Unterscheidungsmerkmal zu objektorientierten Entwurfsmethoden angesehen werden (Giorgini und Henderson-Sellers 2005, S. 10). Als Modellierungssprache wird überwiegend UML bzw. Derivate wie Agent UML (auch als AUML bezeichnet, Bauer et al. 2001; Bauer und Odell 2005) eingesetzt. Für eine vergleichende Übersicht über die momentan verfügbaren MAS-Entwurfsmethoden sei auf die aktuellen Referenzwerke zur agenten-orientierten Softwaretechnik (Bergenti et al. 2004; Luck et al. 2004; Henderson-Sellers und Giorgini 2005; Weiss und Jakob 2005; Winikoff und Padgham 2013) verwiesen.

Praktisch allen MAS-Entwurfsmethoden gemeinsam ist, dass ihr Schwerpunkt auf dem Entwurf von Agenten und deren Interaktionen und Organisationsstrukturen liegt. Die Umgebung wird dagegen auf die Kommunikationsinfrastruktur reduziert bzw. ganz vernachlässigt (Weyns et al. 2004). Dies ist insofern verständlich, als die meisten Anwendungen rein kommunizierende MAS darstellen (vgl. die Definition in Abschnitt 2.3.2). Weitere Aspekte der Umgebung zu berücksichtigen hat erst in den letzten Jahren Aufmerksamkeit

in der Forschungsgemeinde gefunden, wie z. B. die *E4MAS*-Workshops[20] belegen.

Parallel zu den Entwurfsmethoden sind diverse Toolkits für die Implementation und Ausführung von Multiagentensystemen entwickelt worden. Sie bieten Unterstützung für die Realisierung der Agenten – mit unterschiedlichen Freiheitsgraden bezüglich der Auswahl der Agentenarchitektur – und stellen die Grundlagen für Kommunikations- und Organisationsstrukturen zur Verfügung, so dass MAS-Entwickler sich auf die anwendungsspezifischen Probleme konzentrieren können (Luck et al. 2004, Kap. 3). Für eine aktuelle Übersicht bestehender Werkzeuge sei auf (Pokahr und Braubach 2009) verwiesen.

Auch wenn diese technischen Fortschritte die Aufgabe, ein funktionierendes MAS zu erstellen, deutlich erleichtert haben, bleibt deren Verifikation und Validierung, d. h. die Sicherstellung, dass das System korrekt arbeitet, eine Herausforderung an die Entwickler (vgl. z. B. Dix und Fisher 2013). Dies gilt ebenso für MAS, welche Multiagentenmodelle in der agentenbasierten Simulation darstellen (Denz 2013).

[20] Die Abkürzung *E4MAS* steht für *Environments for Multiagent Systems* und bezeichnet eine Workshop-Reihe im Rahmen der internationalen AAMAS-Konferenzen. Der erste Workshop fand 2004 statt.

3. Agentenbasierte Simulation als neuer Ansatz der diskreten Simulation

The purpose of modelling is insight not numbers.
– R. W. Hamming

Die agentenbasierte Simulation zeichnet sich aus durch die Verwendung des Agentenkonzepts als Abstraktionsmittel. Dabei werden die aktiven Entitäten eines Systems durch Agenten modelliert, d. h. ihr autonomes Verhalten und ihre Interaktionen mit anderen Agenten bzw. der Umgebung stehen im Vordergrund. Dies erlaubt eine adäquate Modellierung komplexer Systeme, die durch Interaktion autonomer Akteure geprägt sind. Um zu begründen, dass die agentenbasierte Simulation ein neuer Ansatz in der zeitdiskreten Simulation ist, muss sie mit den bisherigen Ansätzen verglichen werden. Insbesondere ist die These zu prüfen, dass das von diesen Ansätzen gebotene Abstraktionsniveau unzureichend ist.

3.1. Bisherige Ansätze und ihr Bezug zur agentenbasierten Simulation

Innerhalb der diskreten Simulation haben sich verschiedene Modellierungsstile entwickelt, die – zurückgehend auf Lackner (1962) – auch als Weltsichten (*world views*) bezeichnet werden. Je nach Art des zu modellierenden Systems und der zu untersuchenden Fragestellung erscheint der eine oder andere Modellierungsstil natürlicher. Allen gemeinsam ist, dass zwischen Struktur und Verhalten eines Systems unterschieden wird: Die meist statische Struktur ergibt sich aus den einzelnen Systemelementen und deren Beziehungen untereinander, das dynamische Verhalten aus den Handlungen der aktiven Systemelemente. Diese sind allein für Zustandsänderungen verantwortlich, während die passiven Elemente in der Regel als – ggf. kritische – Ressourcen fungieren (vgl. z. B. Zeigler 1976, S. 146 und Hill 1996, S. 4).

Für eine Abgrenzung der agentenbasierten Simulation als eigenständige Weltsicht seien die existierenden Weltsichten im folgenden kurz dargestellt, bevor ihr Verhältnis zur agentenbasierten Sicht diskutiert wird.

3.1.1. Die vier klassischen Weltsichten der diskreten Simulation

Vier klassische Weltsichten können unterschieden werden: die ereignisorientierte, die aktivitätsorientierte, die prozessorientierte und die transaktionsorientierte Sicht. Jede Weltsicht wurde durch paradigmatische Simulationssprachen geprägt. Die Verwendung einer solchen Simulationssprache legt damit eine bestimmte Sichtweise der Realsysteme und einen Modellierungsstil fest (Page 1991, S. 28).

Zentrale Begriffe für alle vier Weltsichten sind Ereignis, Aktivität und Prozess. Hier haben sich die folgenden, von Nance (1981) erstmals vereinheitlichten Definitionen durch-

gesetzt:

- Ein *Ereignis* (*event*) ist die Zustandsänderung einer Entität zu einem bestimmten Zeitpunkt. Die Änderung des Zustands geschieht somit konzeptuell zeitverzugslos. Zwei Arten von Ereignissen können unterschieden werden: Ereignisse, deren Eintreten ausschließlich von der Zeit abhängen, heißen unbedingt (*determined*, lt. (Banks et al. 1999, S. 62) auch *primary*); Ereignisse, deren Eintreten dagegen von bestimmten Systemzuständen abhängen, werden als bedingt (*contingent* bzw. *secondary*) bezeichnet.

- Eine *Aktivität* (*activity*) beschreibt eine Menge von Operationen, die zu einer Zustandsänderung führen. Jede Aktivität eines Modellobjekts ist begrenzt durch zwei Ereignisse, die aufeinanderfolgende Zustandsänderungen dieses Objekts markieren[1]. Anders ausgedrückt, entspricht eine Aktivität dem Zustand einer Entität (Modellobjekt) während eines Zeitintervalls.

- Ein *Prozess* (*process*) fasst direkt aufeinanderfolgende Aktivitäten einer Entität zusammen. Er beschreibt damit eine Folge von Zuständen einer Entität über eine Zeitspanne, d. h. über eine Folge zusammenhängender Zeitintervalle.

Die einzelnen Weltsichten unterscheiden sich v. a. darin, welchen der drei Begriffe sie als strukturierendes Element bei der Modellierung der Systemdynamik verwenden. Dieses charakteristische Merkmal findet sich daher auch in der Bezeichnung der jeweiligen Weltsicht wieder; wobei sich im Deutschen die Zusammensetzungen mit „-orientiert“ durchgesetzt haben, während sie im englischen Sprachgebrauch durch die von Fishman (1973) geprägten Bezeichnungen ersetzt wurden[2].

3.1.1.1. Ereignisorientierte Sicht

In der ereignisorientierten Weltsicht wird ein System modelliert, indem die charakteristischen Ereignisse identifiziert werden. Für jedes Ereignis wird in Form von Ereignisroutinen beschrieben, welche Zustandsänderungen zum Zeitpunkt des Ereignisses stattfinden. Da sich ein Ereignis nach der obigen Definition jeweils nur auf eine Entität bezieht, gleichzeitige Zustandsänderungen mehrerer Entitäten aber nicht ausgeschlossen sind, können mehrere Ereignisse zum selben Zeitpunkt auftreten. Die Gleichzeitigkeit von Ereignissen wird konzeptuell vermieden, wenn alle Zustandsänderungen, die zum selben Zeitpunkt stattfinden, gemeinsam als ein Ereignis betrachtet werden (Nance 1981); in diesem Fall bleibt es in der Verantwortung des Modellierers, die (richtige) Reihenfolge festzulegen. Diese Vorgehensweise wird auch gewählt, wenn zeitgleiche Ereignisse zwar zugelassen sind, es aber dem Anwender überlassen bleibt, geeignete Prioritäten zu vergeben, um eine Sequentialisierung zu erreichen (vgl. Abschnitt 2.1.3).

[1] Banks, Carson und Nelson (1999, S. 61f) differenzieren weiter in eigentliche Aktivitäten, die durch unbedingte Ereignisse beendet, und Verzögerungen (*delays*), die durch bedingte Ereignisse beendet werden, so dass ihre Dauer zum Startzeitpunkt noch nicht feststeht.

[2] Die ursprünglichen Bezeichnungen *event-oriented*, *activity-oriented* und *process-oriented* führt Nance (1981) zurück auf Lackner (1964) und Kiviat (1967; 1969).

Während der Simulation eines ereignisorientierten Modells werden bereits bekannte unbedingte Ereignisse auf einer Ereignisliste eingetragen, in der die zukünftigen Ereignisse zeitlich aufsteigend sortiert sind. Die Simulationszeit schreitet von Ereigniszeitpunkt zu Ereigniszeitpunkt voran – der ereignisorientierte Ansatz ist also inhärent ereignisgesteuert. Da nicht alle Ereigniszeitpunkte bereits zu Beginn der Simulation feststehen, müssen in den Ereignisroutinen neben den entsprechenden Zustandsänderungen auch neue Ereignisse erzeugt und auf die Ereignisliste gesetzt werden. Dieser Tatsache wird mit der englischen Bezeichnung *event scheduling* Rechnung getragen. Eine weitere charakteristische Eigenschaft des ereignisorientierten Ansatzes folgt aus der Definition des Ereignis-Begriffs: Die Zustandsänderungen geschehen konzeptuell zeitverzugslos, so dass bei der Ausführung einer Ereignisroutine keine Simulationszeit vergeht.

Vorteile dieser Weltsicht bestehen darin, auch in einer allgemeinen Programmiersprache einfach zu implementieren (Kreutzer 1986, S. 57) und – bei Wahl entsprechender Datenstrukturen für die Ereignisliste – effizient in der Ausführung zu sein. Nachteilig ist dagegen, dass die Logik des Modells auf die einzelnen Ereignisroutinen verteilt ist, was insbesondere bei komplexen Modellen schnell unübersichtlich wird. Dies ist darin begründet, dass zeitliche Verzögerungen über das Vormerken neuer Ereignisse abgebildet werden müssen; wobei bedingte Ereignisse nicht vorgemerkt werden können, so dass ihr Eintreten durch ggf. umfangreiche Tests in anderen Ereignisroutinen überprüft werden muss.

Die Nachteile lassen sich abmildern, indem eine eher objektorientierte Sichtweise eingenommen wird, d. h. die Entitäten die auf sie bezogenen Ereignisroutinen zusammenfassen und je nach Ereignistyp zu der entsprechenden Routine verzweigen (vgl. die Darstellung in Spaniol und Hoff 1995 und auch Zeigler et al. 2000, S. 159).

3.1.1.2. Aktivitätsorientierte Sicht

In der aktivitätsorientierten Weltsicht (engl. *activity scanning*) konzentriert sich der Modellierer auf die Aktivitäten in einem System und die Bedingungen, die den Beginn einer Aktivität erlauben. Die Simulation eines aktivitätsorientierten Modells geschieht zeitgesteuert: Bei jedem Zeitschritt werden die Bedingungen aller Aktivitäten überprüft und diejenigen Aktivitäten gestartet, deren ggf. komplexe Bedingungen erfüllt sind. Vergleichen lässt sich dieser Ansatz mit regelbasierter Programmierung, bei der ebenfalls mit dem Eintreten einer spezifizierten Bedingung die zugehörige Regel „feuert“ (Banks 1998, S. 9). Da hier der Zeitbezug fehlt, ist es nicht verwunderlich, dass er in der „reinen“ Aktivitätsorientierung künstlich durch eine Taktrate hergestellt werden muss.

Die deklarative Modellbeschreibung ist ein Vorteil der aktivitätsorientierten Weltsicht. Das einfache Konzept führt zu Modellen, die leicht verständlich und gut zu warten sind (Banks et al. 1999, S. 68). Demgegenüber steht die ineffiziente Ausführung, die sich durch die benötigte Zeitsteuerung und das ständige Überprüfen aller Bedingungen ergibt.

Die Weiterentwicklung zum sogenannten Drei-Phasen-Ansatz (*three-phase approach*) benutzt daher Konzepte aus der ereignisorientierten Sicht, um einen ereignisgesteuerten Zeitfortschritt zu ermöglichen und so unnötiges Testen von Bedingungen zu vermeiden (Liebl 1995, S. 105). Dies verbessert die Ausführungsgeschwindigkeit, macht den Modellierungsstil jedoch konzeptuell schwieriger, da praktisch zwei Weltsichten miteinander

vermischt werden.

Im Drei-Phasen-Ansatz werden Ereignisse als Aktivitäten mit Zeitdauer 0 aufgefasst und alle Aktivitäten in zwei Kategorien eingeteilt: *B (bound to occur)* für Aktivitäten, deren Eintrittszeitpunkt vorherbestimmt werden kann und die deshalb auf einer Ereignisliste eingetragen werden können, und *C (conditional)* für Aktivitäten, die vom Eintreten bestimmter Bedingungen abhängen (Banks et al. 1999, S. 68). Die Simulation läuft dann wiederholt in drei Phasen ab:

1. Die Simulationsuhr wird auf den nächsten Ereigniszeitpunkt vorgestellt und alle zeitgleichen Aktivitäten aus der Ereignisliste entfernt.
2. Diese B-Aktivitäten werden ausgeführt, was zu Zustandsänderungen führt und z. B. Ressourcen freigibt.
3. Die Bedingungen der C-Aktivitäten werden überprüft und alle Aktivitäten gestartet, deren Bedingungen wahr geworden sind. Dies generiert ggf. neue B-Aktivitäten und kann ebenfalls zu Zustandsänderungen führen. Die Überprüfung wird solange wiederholt, bis sichergestellt ist, dass keine weiteren C-Aktivitäten beginnen können.

3.1.1.3. Prozessorientierte Sicht

Die prozessorientierte Weltsicht wird in der Literatur zum Teil dargestellt als eine Verbindung von Ereignis- und Aktivitätsorientierung (vgl. z. B. Zeigler et al. 2000, S. 161; Zeigler 1984, S. 118; Fishman 1973, S. 40). Dabei unterscheidet sich dieser Ansatz wesentlich von den beiden vorigen, indem hier statt der globalen Sicht auf die im zu modellierenden System eintretenden Ereignisse bzw. ablaufenden Aktivitäten die Perspektive der einzelnen Entitäten eingenommen wird. Ein Modellierer identifiziert die Systemobjekte und beschreibt deren Lebenszyklus, d. h. die zeitliche Abfolge der sie betreffenden Ereignisse und Aktivitäten. Wartezeiten werden explizit mitmodelliert, so dass bei der Simulation eines prozessorientierten Modells während der Ausführung eines Prozesses gewöhnlich mehrmals Simulationszeit vergeht.

Ein prozessorientiertes Modell besteht demnach aus ebenso vielen parallelen und ggf. interagierenden Prozessen wie gleichzeitig handelnde Entitäten im abgebildeten System existieren; die englische Bezeichnung *process interaction approach* macht dies deutlich. Die quasi-parallele Ausführung solcher Prozesse auf einem sequentiell arbeitenden Rechner erfordert, dass Prozesse wechselseitig die Kontrolle abgeben und ihr Zustand für die Reaktivierung gespeichert wird. Sobald ein Prozess in eine simulationszeit-verbrauchende Phase (Aktivität) eintritt, – sei sie konzeptuell aktiv oder passiv, – wird er programmtechnisch inaktiv. Die programmtechnisch aktiven Phasen entsprechen dagegen Ereignissen und laufen simulationszeit-verzugslos ab.

In den programmtechnisch inaktiven Phasen kann ein Prozess zwei verschiedene Zustände einnehmen: unterbrochen oder passiv. Diese Zustände entsprechen den beiden Arten möglicher Aktivitäten: unbedingtes und bedingtes Warten. Bei unterbrochenen Prozessen ist die Dauer der Unterbrechung bekannt (unbedingtes Warten), so dass der Prozess zur Reaktivierung vorgemerkt werden kann. Bei passiven Prozessen hängt der Zeit-

punkt der Reaktivierung jedoch von einem bestimmten Systemzustand ab (bedingtes Warten), dessen Eintreten nicht im voraus berechnet werden kann.

Simulationssoftware, welche die prozessorientierte Weltsicht unterstützt, muss Sprachkonstrukte für beide Formen von Aktivitäten anbieten und die benötigte Koroutinen- oder Thread-Steuerung übernehmen. Für die Realisierung des bedingten Wartens existieren zwei unterschiedliche Ansätze, die als bedingte bzw. imperative Ablaufsteuerung (vgl. Kreutzer 1986, S. 42) bezeichnet werden. Bei ersterer spezifiziert ein Prozess in einem Konstrukt wie `wait until(<condition>)` selbst die Bedingungen für das Ende des Wartens. Das Überprüfen der Bedingungen wird von der Simulationsinfrastruktur übernommen, die den Prozess zum richtigen Zeitpunkt wieder reaktivieren muss. Dieser Ansatz stammt aus der aktivitätsorientierten Weltsicht, die ebenfalls das Testen von Bedingungen voraussetzt; ein Prozess, der ein `wait until()` ausführt, lässt sich mit einer C-Aktivität vergleichen.

Die imperative Ablaufsteuerung legt dagegen das Überprüfen der Bedingung einschließlich Reaktivierung des wartenden Prozesses in die Verantwortung des Anwenders. Hier passiviert sich ein Prozess über ein Konstrukt wie `passivate()` zwar auch selbst, seine Reaktivierung muss jedoch explizit von einem anderen Prozess übernommen werden. Dieser Ansatz entspricht somit eher der ereignisorientierten Weltsicht, in der ebenfalls das Eintreten bedingter Ereignisse explizit in Ereignisroutinen zu codieren ist.

Beide Arten der Ablaufsteuerung haben ihre Vorteile: Die bedingte Form ist einfacher zu benutzen und führt zu in sich geschlosseneren Prozessen, während die imperative Form deutlich effizienter in der Ausführung ist und darüber hinaus die Sequentialisierung der Prozesse transparent macht, da der Anwender sie selbst bestimmt (Nygaard und Dahl 1978, S. 253)[3]. Mit der imperativen Ablaufsteuerung reicht es aus, die Prozesse analog zu Ereignissen in der ereignisorientierten Weltsicht in aufsteigender Reihenfolge ihrer Reaktivierungszeitpunkte in einer internen Ereignisliste zu führen; unterbrochene Prozesse tragen sich selbst auf dieser Liste ein, während passive Prozesse zu gegebener Zeit von anderen Prozessen vorgemerkt werden. Die Zeit schreitet dann von Reaktivierungszeitpunkt zu Reaktivierungszeitpunkt voran. Die bedingte Ablaufsteuerung benötigt dagegen noch eine zusätzliche Liste für passive Prozesse, deren Bedingungen zu jedem Reaktivierungszeitpunkt überprüft werden müssen (Liebl 1995, S. 109f).

Der wesentliche Vorteil der prozessorientierten Weltsicht besteht darin, dass Systemelemente als Einheit aus Attributen und Verhalten beschrieben und implementiert werden. Damit bewahren sowohl konzeptuelles als auch Computer-Modell die Struktur des abgebildeten Systems, was den hohen intuitiven Reiz dieses Modellierungsstils erklärt (vgl. Banks 1998, S. 9; Zeigler et al. 2000, S. 161). Der Nachteil der aufwendigeren Simulationsinfrastruktur spielt bei Wahl entsprechender Software für den Anwender dagegen nur eine untergeordnete Rolle, indem er ggf. zu Lasten der Ausführungseffizienz geht. Da jedes prozessorientierte Modell in ein äquivalentes ereignisorientiertes Modell abgebildet werden kann, ließe sich dieser Nachteil sogar eliminieren z. B. durch automatische Modell-Transformation mittels eines speziellen Compilers (Huang und Iyer 1998).

[3] In der Simulationssprache Simula, die den prozessorientierten Modellierungsstil wesentlich geprägt hat, wurde die imperative Ablaufsteuerung gewählt.

3.1.1.4. Transaktionsorientierte Sicht

In der transaktionsorientierten Weltsicht wird ein System modelliert als eine Menge von dynamischen Entitäten, den sogenannten Transaktionen, die ein Netz von statischen Entitäten (Bedienstationen unterschiedlichster Art) durchlaufen und dabei um die Benutzung knapper Ressourcen konkurrieren (Schriber und Brunner 1998, S. 766). Die englische Bezeichnung *transaction flow* unterstreicht die grundlegende Idee, dass Objekte durch ein System „fließen“. Zurückführen lässt sich dieser Ansatz auf eine aus der Systemanalyse bekannte Blockdiagrammtechnik, mit der das Verhalten von Systemen beschrieben wird (Page 1991, S. 32). Blöcke stellen permanente Systemelemente dar mit fest vorgegebenen Funktionen, Transaktionen sind temporäre Systemelemente. Auf ihrem Weg durch die Blöcke können die Attribute der Transaktionen verändert werden.

Die transaktionsorientierte Weltsicht wird zum Teil nicht als eigenständiger Ansatz aufgefasst, sondern explizit als Spezialfall der prozessorientierten Sicht dargestellt (z. B. Zeigler et al. 2000, S. 161) oder implizit der prozessorientierten Sicht untergeordnet (u. a. Kreutzer 1986, S. 58f; Banks et al. 1999, S. 67). Andererseits lässt sich argumentieren, dass die grundlegenden Unterschiede zwischen einer prozessorientierten Sprache wie Simula und der den transaktionsorientierten Ansatz prägenden Simulationssprache GPSS eine Trennung beider Weltsichten rechtfertigen (Page 1991, S. 29). Im Unterschied zur prozessorientierten Weltsicht werden hier die dynamischen, temporären Modellkomponenten als aktiv, die statischen, permanenten Modellkomponenten dagegen als passiv betrachtet. Damit ist die transaktionsorientierte Sicht material-orientiert, während die prozessorientierte Sicht als maschinen-orientiert bezeichnet werden kann (vgl. Page 1991, S. 28).

Unabhängig davon, ob man dem transaktionsorientierten Ansatz Eigenständigkeit zubilligt oder nicht, kann auch dieser Modellierungsstil auf den grundlegenden ereignisorientierten Ansatz abgebildet werden. Die Ereignisse sind in diesem Fall das Aufeinandertreffen von Transaktionen und Bedienstationen.

3.1.2. Objektorientierte Simulation

Objektorientierte Simulation (OOS) ist ein eher unscharfer Begriff, der in leicht unterschiedlichen Konnotationen verwendet wird. Unter dem Aspekt der Softwareentwicklung bedeutet objektorientierte Simulation die Anwendung objektorientierter Techniken während der Phase der Modellbildung, vom Entwurf bis zur Implementation eines Modells (Roberts und Dessouky 1998, S. 360). In diesem Sinne kehrt Objektorientierung quasi in ihr ursprüngliches Anwendungsgebiet zurück, lassen sich doch wesentliche Konzepte wie Klassenbildung und Vererbung auf die Mitte der sechziger Jahre entwickelte Simulationssprache Simula (Dahl und Nygaard 1966; Nygaard und Dahl 1978) zurückführen.

Andererseits kann objektorientierte Simulation auch als besondere Weltsicht aufgefasst werden, in der Objekte die strukturierenden Elemente darstellen, analog zu den Ereignissen, Aktivitäten bzw. Prozessen der klassischen Weltsichten. Dies beinhaltet, dass das Verhalten der Objekte in Bezug zur Simulationszeit gesetzt wird.

Selbst die objektorientierte Realisierung von Simulationswerkzeugen kann unter dem Begriff subsumiert werden (vgl. Uhrmacher 1997); dies ist jedoch für die Abgrenzung zur agentenbasierten Simulation nicht relevant und soll daher im folgenden nicht näher

betrachtet werden.

3.1.2.1. Objektorientierung als Entwurfs- und Implementationsgrundlage

Seit Anfang der 1990er Jahre hat sich Objektorientierung auch im Bereich der Simulation als vorherrschende Softwareentwicklungmethode durchgesetzt. Dies umfasst sowohl Analyse und Design, d. h. die Entwicklung des konzeptuellen Modells, als auch die Implementation, d. h. die Umsetzung des konzeptuellen Modells in ein ausführbares Computermodell. Objektorientierte Simulation lässt sich demnach mit objektorientierter Modellbildung gleichsetzen; die reine Implementation eines Modells in einer objektorientierten Programmiersprache sollte dagegen nicht als objektorientierte Simulation bezeichnet werden (Roberts und Dessouky 1998, S. 360).

Die Verwendung objektorientierter Entwurfsmethoden und -werkzeuge ermöglicht ein einheitliches, durchgängiges Konzept für Analyse, Entwurf und Programmierung eines Simulationsmodells (Page et al. 2000, S. 20), wodurch die semantische Lücke zwischen Modellierung und Implementation verkleinert wird. Da in der Simulation mehr oder weniger komplexe Systeme aus interagierenden Entitäten betrachtet werden, erlaubt der objektorientierte Ansatz eine „natürliche“ Modellierung, indem reale Entitäten auf simulierte Objekte abgebildet werden. Diese konzeptuelle Nähe zum Problembereich ist einer der Vorzüge, die sich aus dem Einsatz objektorientierter Methoden in der Modellbildung ergeben (Page 1991, S. 231). Auch weitere Vorteile der objektorientierten Softwareentwicklung wie Wiederverwendbarkeit, leichte Erweiterbarkeit und Modularität lassen sich direkt auf den Anwendungsbereich Simulation übertragen.

Durch den Einsatz von objektorientierter Analyse und Entwurf hat auch die graphische Modellierungssprache UML für die Simulation zunehmend an Bedeutung gewonnen (Knaak und Page 2006, S. 33). Dies betrifft insbesondere Klassendiagramme für die Identifikation der statischen Struktur (Objekttypen und ihre Beziehungen), Aktivitäts- oder Zustandsdiagramme für die Modellierung des Objektverhaltens und Sequenz- oder Zeitverlaufsdiagramme, um die Interaktionen mehrerer Objekte zu verdeutlichen (Knaak und Meyer 2005). Zwar liegt der Schwerpunkt bei den Diagrammen zur Beschreibung des dynamischen Verhaltens eher auf einer Definition der Reihenfolge, in der Aktionen oder Ereignisse eintreten, doch ermöglicht UML seit Einführung der wesentlich überarbeiteten Version 2.0 auch die explizite Behandlung der Zeit (OMG 2005). Zeitintervalle, Dauern und relative bzw. absolute Zeitpunkte können definiert und Ereignisse so in Bezug zur Simulationszeit gesetzt werden. Die Dauer einer Aktivität lässt sich beispielsweise modellieren, indem das Schalten der entsprechenden Transition eines Zustandsautomaten an ein Zeitintervall geknüpft wird.

Objektorientierte Simulation im Sinne von objektorientierter Modellierung ist weder auf eine der klassischen Weltsichten beschränkt noch muss sie zwingend als eigene Weltsicht aufgefasst werden:

> Instead of thinking of the object orientation as a separate world view or simulation orientation, it is more useful to think of it as a different way to define entities—the way they act, interact, and communicate with each other—and to organize the types of entities in the model and the data or attributes which each entity possesses. (Bischak und Roberts 1991, S. 195)

Zwar besitzt die Objektorientierung aufgrund ihrer historischen Wurzeln in der Sprache Simula eine besondere Affinität zur prozessorientierten Weltsicht, doch lassen sich die ereignisorientierte oder aktivitätsorientierte Weltsicht ebenfalls objektorientiert umsetzen (vgl. z. B. Pidd 1995; Tyszer 1999).

3.1.2.2. Objektorientierung als Weltsicht

Auch wenn objektorientierte Simulation in der Regel – wie oben diskutiert – als objektorientierte Modellierung der zu untersuchenden Systeme aufgefasst wird, ist es dennoch möglich, sie als spezielle Weltsicht zu interpretieren. Allgemein wird in der Objektorientierung die Sicht eingenommen, dass ein System sich aus interagierenden Objekten zusammensetzt. Dieser Ansatz zur Modellierung von Systemen deckt sich mit dem der diskreten Simulation, was sich aus der Entstehungsgeschichte der Objektorientierung erklären lässt. Der fundamentale Unterschied liegt in der expliziten Zeitrepräsentation:

> Explicitly defining the time-dependent behaviors is the major difference between OOS and other OO systems (Feng und Feng 1996, S. 312).

Um als Weltsicht der diskreten Simulation zu gelten, muss die objektorientierte Simulation somit klar spezifizieren, wie die Beziehung zwischen Modellzustand und Simulationszeit herzustellen ist.

Eine potentielle Herangehensweise besteht darin, die objektorientierte Sicht auf eine der klassischen Weltsichten abzubilden, also quasi die objektorientierte Variante einer bestehenden Weltsicht zu schaffen. Für die von einer bestimmten Weltsicht unabhängige statische Struktur eines Modells, d. h. die Entitäten mit ihren Attributen und ihren Beziehungen untereinander, ist diese Abbildung trivial: Ein Objekt entspricht genau einer Entität. Das dynamische Verhalten des Modells, d. h. die Interaktionen der Entitäten (Objekte) über die simulierte Zeit und die dadurch hervorgerufenen Zustandsänderungen, ist unter Berücksichtigung objektorientierter Prinzipien auf eines der zeitbehafteten Elemente Ereignis, Aktivität oder Prozess zurückzuführen.

Da Objekte Zustand (Attribute) und Verhalten (Methoden) kapseln, bedeutet dies beispielsweise für den ereignisorientierten Ansatz, dass jedes Objekt die auf es bezogenen Ereignisse selbst verarbeitet, anstatt von einer entsprechenden Ereignisroutine bearbeitet zu werden. Spaniol und Hoff beschreiben einen solchen Ansatz und definieren eine Entität als „ein Objekt, welches in der Lage ist, sich (aktiv) in der Simulationszeit fortzubewegen“ (1995, S. 9). Jedes Objekt verfügt über eine spezielle Methode `handleEvent()`, die bei Eintreten eines Ereignisses aufgerufen wird und je nach Ereignistyp die entsprechenden Zustandsänderungen durchführt. Zeitverbrauch ist weiterhin über das Vormerken von Ereignissen zu modellieren, alle Methoden arbeiten prinzipiell zeitverzugslos. Die darunterliegende Simulationsinfrastruktur (Zeitführungsroutine, Ereignisliste) kann unverändert beibehalten werden.

Bei Verwendung der prozessorientierten Weltsicht ist die Abbildung noch einfacher: Ein Objekt entspricht exakt einem Prozess; beide kapseln Zustand und Verhalten. Für die technische Umsetzung ergeben sich dieselben Anforderungen wie bei der reinen Prozessorientierung, d. h. ein Objekt benötigt die Möglichkeit, die Ausführung einer Methode zu unterbrechen und nach (un)bestimmter Zeit an der Unterbrechungsstelle wieder

aufzunehmen. Dies ist von einer Verwendung der Objektorientierung als Entwurfs- und Implementierungsgrundlage des prozessorientierten Ansatzes nicht mehr zu unterscheiden, weshalb die Objektorientierung generell als Variante der prozessorientierten Weltsicht klassifiziert werden kann (vgl. Kreutzer 1986, S. 63).

Heute verfügbare kommerzielle Simulationssoftware verbindet eine objektorientierte Sicht oft mit einer Kombination von Ereignis- und Prozessorientierung, d. h. erlaubt einem Anwender, das Verhalten eines Modellobjekts über die Definition modellspezifischer Ereignisse oder Prozesse zu steuern (Pegden 2010, S. 214).

Im Gegensatz dazu entwickeln Feng und Feng (1996) eine eigenständige objektorientierte Weltsicht, aufbauend auf dem DEVS-Formalismus zur Spezifikation von ereignisdiskreten Systemen (Zeigler 1984, S. 62ff). Bausteine sind Objekte, die Zustände, Verhalten und spezielle Ports für die Kommunikation mit anderen Objekten besitzen. Das Verhalten wird in Form von Zustandsautomaten beschrieben, deren Transitionen von Ereignissen ausgelöst werden. Jede Transition spezifiziert ggf. zusätzliche Bedingungen und die auszuführenden Aktionen wie Zustandsänderungen, Vormerken von Ereignissen oder das Senden von Ereignissen an andere Objekte. Ein Objekt ändert folglich seinen internen Zustand ausschließlich in Reaktion auf interne (selbst generierte) oder externe Ereignisse (Interaktionen mit anderen Objekten). Die Simulation eines solchen objektorientierten Modells verläuft ereignisgesteuert.

Zusammenfassend lässt sich feststellen, dass – im Unterschied zu den klassischen Weltsichten – mit der objektorientierten Simulation kein einheitliches Verfahren für die Modellierung des zeitbehafteten Verhaltens etabliert ist. Dies mag darin begründet sein, dass im konventionellen objektorientierten Paradigma expliziter Zeitbezug keine Rolle spielt und darüber hinaus Objekte im Wesentlichen passiv sind, d. h. nur auf Methodenaufrufe reagieren. Für die Beschreibung der Beziehung zwischen Objekten und Ereignissen als den zentralen zeitbehafteten Elementen der ereignisdiskreten Simulation bleibt einem Modellierer somit ein gewisser Spielraum.

3.1.3. Bezug zum Agentenkonzept

Ein Vergleich der von den vorgestellten Weltsichten propagierten Konzepte mit dem Agentenkonzept ermöglicht es, Gemeinsamkeiten und Unterschiede herauszuarbeiten. Die auf diese Weise zu überprüfende These lautet, dass mit dem Konzept des Agenten eine höhere Abstraktionsebene erreicht wird, die von keinem bestehenden Modellierungsstil geboten wird. Damit ist jedoch nicht ausgeschlossen, dass sich die agentenorientierte Weltsicht als Erweiterung einer anderen Weltsicht definieren lässt.

3.1.3.1. Entitäten versus Agenten

Allen klassischen Weltsichten gemeinsam ist die Modellierung der statischen Struktur eines Systems basierend auf Entitäten, die sich durch zustandsbeschreibende Attribute auszeichnen. Für die Modellierung des Systemverhaltens in der Zeit verwenden sie dagegen unterschiedliche Ansätze. Die Unterschiede lassen sich sehr gut mit Hilfe des Lokalitätsprinzips der Softwaretechnik ausdrücken (Overstreet 1987, S. 583):

- Die ereignisorientierte Sicht betont die Lokalität der *Zeit*: Jede Ereignisroutine eines Modells beschreibt alle Operationen, die zu einem gemeinsamen Zeitpunkt auftreten.
- Die aktivitätsorientierte Sicht betont die Lokalität des *Zustands*: Jede Aktivitätsroutine eines Modells beschreibt alle Operationen, die auszuführen sind, sobald das System in einen bestimmten Zustand übergeht.
- Die prozessorientierte Sicht (auch in ihrer transaktionsorientierten Ausprägung) betont die Lokalität des *Objekts*: Jede Prozessroutine eines Modells beschreibt die vollständige Folge von Operationen, die von einem bestimmten Modellobjekt (Entität) durchgeführt werden.

Sowohl der ereignisorientierte als auch der aktivitätsorientierte Ansatz nehmen bei der Verhaltensbeschreibung eine makroskopische Sicht auf das System ein. Entitäten besitzen nur Attribute, ihr Verhalten ist dagegen über systemglobale Ereignisroutinen bzw. Aktivitätsroutinen verteilt, wobei eine Routine mehrere Entitäten betreffen kann. Interaktion zwischen Entitäten muss über gegenseitiges Vormerken von Ereignissen bzw. entsprechend komplexe Bedingungen abgebildet werden. In beiden Weltsichten sind die Entitäten somit passiv, während die in Ereignis- bzw. Aktivitätsroutinen festgelegten Operationen von außen ihren Zustand verändern. Dieses Konzept ist mit autonom handelnden Agenten nicht vereinbar; beide Weltsichten bieten ein zu niedriges Abstraktionsniveau, um die Modellierung von Agenten angemessen zu unterstützen. Auch der Drei-Phasen-Ansatz der Aktivitätsorientierung stellt konzeptuell keine Verbesserung dar, ermöglicht er doch nur eine effizientere Simulationssteuerung.

In der prozessorientierten Weltsicht sind die als Prozesse modellierten Entitäten dagegen aktiv. Ihr Verhalten wird aus lokaler Perspektive beschrieben; ein Prozess kapselt somit Zustand (Attribute) und Verhalten.[4] Gleiches gilt für die Transaktionen des transaktionsorientierten Ansatzes. Allerdings ist dieser konzeptuell am stärksten dem typischen Anwendungsgebiet der Bedien-/Wartesysteme verhaftet und daher für die Abbildung von Agenten wenig geeignet: Die Auffassung von durch ein System von Bedienstationen fließenden Transaktionen hat mit dem Konzept von autonomen Agenten, die als gleichberechtigte Partner miteinander kommunizieren, nichts gemeinsam.

Einzig die aktiven Entitäten (Prozesse) der prozessorientierten Sicht weisen eine konzeptuelle Nähe zu Agenten auf. Sie verbinden Zustands- und Verhaltensbeschreibung zu einer Einheit und sind – da sie potentiell gleichzeitig handelnde Systemelemente repräsentieren – inhärent nebenläufig. Dennoch ist das von ihnen gebotene Abstraktionsniveau unzureichend und zwar aus folgenden Gründen:

1. Der Lebenszyklus eines Simulationsprozesses bildet per Definition (siehe S. 52) einen linearen Ablauf: Die einzelnen Aktivitäten werden sequentiell ausgeführt, parallele

[4] Cota und Sargent argumentieren, dass Kapselung in den klassischen Weltsichten auch im prozessorientierten Ansatz nicht wirklich unterstützt wird. So ist beispielsweise Verdrängung bei der Bedienung von Aufträgen (Prozessen) in der Regel dadurch modelliert, dass der höher priorisierte Prozess den Reaktivierungszeitpunkt des in Bearbeitung befindlichen niedriger priorisierten Prozesses verschiebt (1992, S. 119f). Fasst man den Reaktivierungszeitpunkt als Attribut des Prozesses auf, wird somit direkt dessen Zustand manipuliert.

Aktivitäten innerhalb eines Prozesses sind nicht vorgesehen. Bei Agenten ist dagegen die Zahl nebenläufiger Verhaltensweisen prinzipiell nicht beschränkt und z. T. sogar explizit erforderlich. So wird ein Agent in dynamischen Umgebungen in der Regel eine permanente Beobachtung der Umwelt parallel zur jeweils aktuellen Aktivität durchführen, um ggf. sofort auf Umweltereignisse reagieren zu können. Eine Reaktion auf asynchron zum Prozesslauf auftretende Ereignisse ließe sich zwar über die Unterbrechung eines Prozesses von außen nachbilden, die von einigen Simulatoren wie z. B. DESMO-J unterstützt wird, doch entspricht dieses Konstrukt eher einer Ausnahmebehandlung und ist daher konzeptuell unpassend. Auch der Ansatz, das Verhalten eines Agenten durch mehrere Prozesse zu modellieren, löst das Problem nicht, da die einzelnen Verhaltensprozesse nicht voneinander unabhängig sind, sondern sich gegenseitig beeinflussen und somit weiterhin synchronisiert werden müssen.

2. Interaktion zwischen Prozessen findet auf der Ebene von Prozesssynchronisation statt, indem ein Prozess auf Reaktivierung durch einen anderen Prozess wartet. Dies umfasst sowohl direkte Synchronisation bei gemeinsamen Aktivitäten als auch indirekte Synchronisation bei Zugriff auf gemeinsam genutzte Ressourcen. Die Kommunikation zwischen Prozessen ist dabei auf die Übermittlung von Ereignissen bzw. Interrupt-Codes beschränkt. Für die Kommunikation zwischen Agenten gelten derartige Einschränkungen nicht; in der Regel bedeutet Kommunikation in Multiagentensystemen den Austausch getypter Nachrichten, oft anhand komplexer Interaktionsprotokolle (Konversationen, vgl. Abschnitt 2.3.2.2).

3. In Prozessen wird bereits während des Entwurfs festgelegt, an welchen Stellen im Lebenszyklus welche Form von Synchronisation zu verwenden ist. Damit bleibt einem Prozess keinerlei Entscheidungsfreiheit. Dies widerspricht dem Autonomie-Konzept bei Agenten; sie benötigen die Möglichkeit, eine Entscheidung über die zu verwendenden Synchronisationsmechanismen in Abhängigkeit von ihrer aktuellen Situation zu treffen, d. h. erst zur Laufzeit (Wooldridge 2009, S. 13).

Das Konzept des Simulationsprozesses unterstützt die Modellierung von Systemelementen demnach auf einer niedrigeren Ebene als das Agentenkonzept. Da die prozessorientierte Weltsicht wie alle klassischen Weltsichten im Hinblick auf eine allgemeinere Klasse von ereignisdiskreten Systemen entwickelt wurde als es die aus der (verteilten) KI stammenden Multiagentensysteme sind, ist dies nicht weiter verwunderlich. Für die Simulation von Maschinen einer Fertigungsanlage oder Prozessoren eines Rechnernetzes bieten Prozesse durchaus den richtigen Abstraktionsgrad, während das Konzept autonomer, miteinander kommunizierender Akteure übertrieben ist.

Ein Prozess lässt sich jedoch als einfacher proaktiver Agent mit eingeschränkten Kommunikationsmöglichkeiten auffassen. Mit ihrer prinzipiellen Nebenläufigkeit erfüllen Prozesse eine Minimalanforderung, um Agenten Autonomie zu sichern (Travers 1996, S. 80).

3.1.3.2. Objekte versus Agenten

Bedingt durch die Entstehungsgeschichte der Objektorientierung weisen Objekt und Simulationsprozess gewisse Gemeinsamkeiten auf: Ein Objekt entspricht insofern einem

Prozess, als es ebenfalls Zustand (Attribute) und Verhalten (Methoden) zu einer Einheit verbindet. Anders als bei Simulationsprozessen ist bei Objekten das Verhalten jedoch weder mit einem expliziten Zeitbezug verbunden noch von einer festgelegten Folge von Aktivitäten geprägt. Darüber hinaus erfolgt die Kommunikation zwischen Objekten ausschließlich durch das Senden von Nachrichten, was einem Aufruf von Methoden gleichkommt. Dies setzt voraus, dass zumindest ein Teil der Methoden für den Zugriff von außen zur Verfügung gestellt wird. Ein Objekt besitzt somit die Kontrolle über seinen Zustand, nicht jedoch über sein Verhalten. Die Entscheidung darüber, ob eine öffentlich zugängliche Methode ausgeführt wird, wird vom Aufrufer der Methode getroffen, nicht vom ausführenden Objekt.

Im Gegensatz dazu ist mit dem Konzept des Agenten Kontrolle über Zustand und Verhalten verbunden. Andere Agenten können zwar die Ausführung einer bestimmten Aktion anfordern, der Agent entscheidet jedoch selbst, ob er diese Aktion tatsächlich ausführt. Die Entscheidung darüber wird in der Regel nicht nur von seinem aktuellen Zustand, sondern auch von seinen aktuell angestrebten Zielen beeinflusst. Dieser höhere Grad an Autonomie wird gut in folgendem Spruch zusammengefasst: *„Objects do it for free, agents do it because they want to“* (Wooldridge 2009, S. 29).

In Abbildung 3.1 ist der Aspekt der Verhaltenskontrolle dargestellt als zusätzliche Schicht, die die Fähigkeiten eines Agenten vor dem Zugriff von außen schützt und entsprechende Anfragen im Hinblick auf die Ziele eines Agenten filtert (Ferber 1999, S. 58). Auch die Kommunikation zwischen Agenten erfolgt auf höherer Ebene als zwischen Objekten, da die Nachrichten nicht wie bei Objekten auf (konzeptuell nicht abweisbare) Methodenaufrufe beschränkt sind.

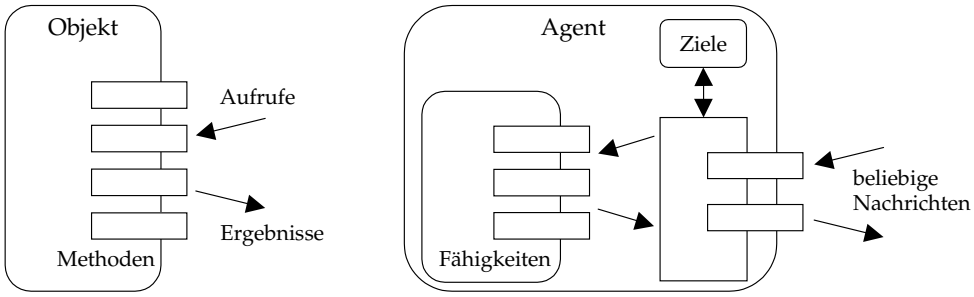

Abbildung 3.1.: Die grundsätzlichen Unterschiede zwischen den Konzepten Objekt und Agent (nach Ferber 1999, S. 58).

Nebenläufigkeit ist ein weiteres Unterscheidungsmerkmal zwischen Agenten und Objekten und steht in direktem Zusammenhang mit dem Autonomiekonzept. Da Objekte passiv sind in dem Sinne, dass sie ausschließlich auf Methodenaufrufe reagieren und nur dann ggf. ihren Zustand ändern, ist in einem System aus interagierenden Objekten (objektorientiertes Programm) konventionell genau ein Objekt zu jedem Zeitpunkt aktiv. Agenten werden dagegen als aktiv aufgefasst, können selbstständig Zustandsänderungen initiieren und sind nicht auf Einflüsse von außen angewiesen. In einem Multiagentensy-

stem agieren die Agenten daher nebenläufig zueinander und können gleichzeitig aktiv sein.

Auf konzeptueller Ebene haben Objekte und Agenten daher wenig gemein. Ein Objekt kann allenfalls als ein degenerierter Agent (Ferber 1999, S. 58) aufgefasst werden, dessen Kommunikationsfähigkeit auf den Aufruf von Methoden und dessen Ziele auf korrekte Methodenausführung beschränkt sind. Davon abgesehen stellen Objekte durchaus eine gute und häufig verwendete Grundlage zur Implementation von Agenten und Multiagentensystemen dar, vergleichbar dem Einsatz objektorientierter Techniken in der Simulation.

Mit der Integration von Nebenläufigkeit in das objektorientierte Paradigma ergibt sich ein fließender Übergang von Objekten zu Agenten: Das Konzept des aktiven Objekts erweitert ein passives Objekt um eigenständige „Aktivität“[5], meist realisiert als eigener *Thread* (Guessoum und Briot 1999, S. 68). Ein aktives Objekt besitzt folglich einen höheren Grad an Autonomie und Verhaltenskontrolle als ein passives Objekt, da es unabhängig vom Erhalt einer Nachricht Aktionen veranlassen kann (Ellis und Gibbs 1989, S. 562). Es existieren unterschiedliche Interpretationen des allgemeinen Konzepts des aktiven Objekts; eine verbreitete Variante basiert auf dem von Hewitt (1977) als Modell für nebenläufige Berechnung eingeführten *Actor*-Konzept. *Actors* kommunizieren miteinander über asynchrone Nachrichten. Auf den Erhalt einer Nachricht kann ein *Actor* reagieren mit dem Senden weiterer Nachrichten, Erzeugen neuer *Actors* (z. B. für die Bearbeitung der Nachricht) oder der Auswahl eines neuen Verhaltens für die Reaktion auf die nächste Nachricht (Zustandsänderung). Das Grundprinzip der Entkoppelung von Aufruf und Ausführung einer Methode, das sich im asynchronen Nachrichtenaustausch zwischen Sender und Empfänger ausdrückt, ist als Entwurfsmuster (Lavender und Schmidt 1996) ins Repertoire der objektorientierten Softwareentwicklung übernommen worden.

Ein anderer Einfluss auf das Konzept des aktiven Objekts stammt aus der diskreten Simulation. Wie bei der Entwicklung der Objektorientierung im Allgemeinen ist hier insbesondere die Sprache Simula zu nennen; deren Prozess-Konzept ist im Grunde identisch mit der Auffassung eines aktiven Objekts als eines Elementes mit eigener Ablaufsteuerung (*thread of control*). Der Lebenszyklus eines Simulationsprozesses, d. h. die Abfolge der Handlungen eines modellierten Systemelements, stellt die oben genannte „Aktivität“ des aktiven Objekts dar. Einzig der explizite Bezug zur simulierten Zeit und die Simulationsprozessen inhärente Beschränkung auf einen sequentiellen Ablauf fehlen im allgemeinen Konzept des aktiven Objekts.

Abweichend vom traditionellen objektorientierten Paradigma werden in der objektorientierten Simulation auf konzeptueller Ebene sowohl passive als auch aktive Objekte betrachtet, um die in der Modellierung ereignisdiskreter Systeme übliche Unterscheidung in passive und aktive Systemelemente nachzuvollziehen. Aktive Systemelemente – und damit die sie repräsentierenden aktiven Modellobjekte ebenfalls – besitzen eine gewisse Autonomie, indem sie selbstständig Zustandsänderungen initiieren können, während die passiven Elemente bzw. Objekte auf Einwirkung von außen angewiesen sind, um ihren Zustand zu ändern. Allgemein beruht die autonome Handlungsfähigkeit aktiver Objekte

[5] Hier im Sinne von Handlungsfähigkeit, nicht zu verwechseln mit dem Begriff der Aktivität in der diskreten Simulation als eine von zwei Ereignissen begrenzte Handlung (siehe die Definition auf Seite 52).

in der objektorientierten Simulation darauf, auf sich selbst bezogene zukünftige Ereignisse vorzumerken, statt ausschließlich auf von anderen erzeugte Ereignisse zu reagieren. Die Umsetzung im Einzelnen hängt ab von dem oder den Verfahren, die von der entsprechenden Weltsicht bzw. dem verwendeten Simulationswerkzeug unterstützt werden.

Solche aktiven Objekte stehen dem Agentenkonzept deutlich näher als die passiven Objekte des traditionellen objektorientierten Ansatzes und bieten damit zumindest eine gute Grundlage für Entwurf und Implementierung von Agenten. Die objektorientierte Variante der ereignisorientierten Weltsicht (vgl. Spaniol und Hoff 1995) ist hier die technisch einfachste Lösung: Jedes aktive Objekt (Entität) kapselt praktisch die auf es bezogenen Teile der Ereignisroutinen, um so Ereignisse selbstständig zu verarbeiten und die zugehörigen Zustandstransformationen durchzuführen. Ein einfacher reaktiver Agent ließe sich auf diese Weise durchaus modellieren. Im Vergleich dazu erlauben die in der objektorientierten Weltsicht von Feng und Feng (1996) eingesetzten Zustandsautomaten eine Verhaltensbeschreibung auf höherer Ebene. Selbstverständlich müssten die aktiven Objekte jeweils noch um charakteristische Agentenfähigkeiten wie Kommunikation oder Situiertheit erweitert werden. Damit erscheint die objektorientierte Weltsicht (in ihren verschiedenen Ausprägungen) als beste Ausgangsbasis für die Entwicklung einer agentenorientierten Weltsicht.

3.2. Agentenbasierte Simulation

In der agentenbasierten Modellierung und Simulation (im folgenden auch als ABMS abgekürzt) werden Systeme autonomer, interagierender Agenten betrachtet. In verschiedenen Disziplinen ist sie inzwischen als neues Forschungsparadigma etabliert; so z. B. in den Sozialwissenschaften (Epstein 2007), den Wirtschaftswissenschaften (Tesfatsion 2002), der Geographie (Heppenstall et al. 2012) und der Ökologie (Grimm und Railsback 2005), wo sie aus historischen Gründen als individuenbasierte Simulation bezeichnet wird. Disziplin-übergreifend haben sich agentenbasierte Modelle für die Simulation komplexer Systeme durchgesetzt (Edmonds und Meyer 2013). Einer der wesentlichen Gründe für diese Expansion ist die zunehmende Verfügbarkeit von Software-Werkzeugen, welche die agentenbasierte Modellierung und Simulation gut genug unterstützen, um sie attraktiv für Anwendungsexperten in vielen Fachgebieten zu machen (Samuelson und Macal 2006).

Im folgenden wird die Entstehungsgeschichte der agentenbasierten Simulation dargestellt (Abschnitt 3.2.1), bevor die Konzepte auf Basis der von den verbreitetsten Simulationswerkzeugen zur Verfügung gestellten Modellierungskonstrukte erläutert werden (Abschnitt 3.2.2). Vor- und Nachteile der Zeit- und Ereignissteuerung für agentenbasierte Modelle (Abschnitt 3.2.3) sowie Ansätze zur formalen Definition (Abschnitt 3.2.4) werden diskutiert. Darauf aufbauend wird in Abschnitt 3.2.5 die agentenorientierte Weltsicht eingeführt, welche das agentenbasierte Modellierungsparadigma in Bezug zum ereignisgesteuerten Zeitfortschritt setzt.

3.2.1. Entstehungsgeschichte

Die Geschichte der agentenbasierten Modellierung und Simulation ist nicht gut dokumentiert. Dies liegt höchstwahrscheinlich darin begründet, dass eine allgemein anerkannte Definition des Begriffs „Agent" fehlt und daher die Ansichten, wo die Anfänge der ABMS zu suchen sind, innerhalb der Forschungsgemeinde divergieren (Siebers und Aickelin 2008). Verschiedene Disziplinen haben zur Entwicklung der agentenbasierten Simulation beigetragen. Die grundlegende Methode, zahlreiche interagierende Agenten in einem Computermodell zu repräsentieren, lässt sich zurückführen auf von Neumanns Arbeiten zu selbst-reproduzierenden Systemen Ende der 1940er Jahre, welche in der Definition von Zellularautomaten resultierte (siehe Abschnitt 2.2.3). John Conway's *Game of Life* ist eines der bekanntesten Beispiele eines zweidimensionalen Zellularautomaten, in dem sehr einfache Regeln auf der Ebene der einzelnen Zellen zu interessanten und überraschenden Strukturen auf der Makroebene führen können. In diesem Sinn ist es ein Beispiel für Emergenz in komplexen Systemen (Christen und Franklin 2002).

Die Forschung zu komplexen Systemen hat die Entwicklung der agentenbasierten Simulation wesentlich beeinflusst. Das Gebiet der komplexen adaptiven Systeme (*complex adaptive systems, CAS*) entstand aus der Untersuchung von Selbstorganisation, Anpassungsvermögen und Emergenz in biologischen Systemen. CAS sind Systeme, die aus vielen Komponenten (oft Agenten genannt) bestehen, welche interagieren und ihr Verhalten in Reaktion auf ihre Umwelt verändern können (Holland 2006). John Holland (1995) identifizierte Eigenschaften und Mechanismen, welche allen CAS gemeinsam sind, wie z. B. Nichtlinearität, Kommunikation zwischen Agenten, hierarchischer Aufbau (Komponenten können wiederum aus Komponenten bestehen) sowie Heterogenität von Agenten. Um solche Systeme besser analysieren zu können, wurde am Santa Fe Institut das Simulationswerkzeug *Swarm* für agentenbasierte Modelle entwickelt (Minar et al. 1996), das erste seiner Art. Es hat neben seinem ursprünglichen Verwendungszweck, die Synthese von biologisch-inspirierten CAS in dem als Künstliches Leben (*Artificial Life*, Langton 2000) bezeichneten Forschungsgebiet zu unterstützen, die Erstellung agentenbasierter Modelle in diversen Anwendungsgebieten inspiriert (Samuelson und Macal 2006). Gemeinsam ist diesen Modellen, dass sie als Forschungsmittel zur Untersuchung komplexer, nichtlinearer Systeme dienen; es lässt sich daher argumentieren, dass agentenbasierte Modelle das Resultat der menschlichen Unfähigkeit, nichtlineare Systeme zu verstehen, sind (Heath und Hill 2010).

Parallel dazu wurden in der Ökologie individuenbasierte Modelle eingeführt, um die Defizite der in diesem Bereich vorherrschenden kontinuierlichen Simulation zu überwinden. Ein individuenbasiertes Modell betrachtet ein System auf der Basis einzelner Individuen, die sich in Eigenschaften und Verhalten unterscheiden können und sich in einem explizit modellierten Raum bewegen. Solche Modelle erlauben daher Heterogenität in der Population von Individuen sowie lokale Beschränktheit der Interaktionen zwischen diesen Individuen zu berücksichtigen (vgl. Abschnitt 2.2.1). Die Idee, dass (einfache) Regeln auf der Mikroebene das Verhalten des Systems auf der Makroebene bestimmen und mikroskopische Ansätze zur Modellierung daher besser geeignet sind als der makroskopische Ansatz der traditionellen kontinuierlichen Simulation, liegt bereits den Zellularautomaten zugrunde. Diese kommen in vielen individuenbasierten Modellen zum Einsatz.

Thomas Schelling wird die Entwicklung des ersten sozialwissenschaftlichen agentenbasierten Modells zugeschrieben (Macal und North 2006). In diesem als Segregations- oder einfach Schelling-Modell bekannt gewordenen Modell repräsentieren die Agenten individuelle Personen, während die Interaktionen zwischen Agenten einen sozial relevanten Prozess abbilden: die Entstehung ethnisch getrennter Wohnviertel trotz relativ hoher Toleranz der einzelnen Individuen (Schelling 1978, S. 147–155). Generell betrachtet zeigt das Schellingmodell die Emergenz von Phänomenen auf der Makroebene (in diesem Fall: Segregation), welche unerwartet sind, wenn man nur die individuellen Entscheidungen und Motivationen der Akteure auf der Mikroebene betrachtet (Umzug in eine andere Gegend nur bei Überschreiten einer Toleranzgrenze von z. B. 65% Nachbarn der „falschen“ Art). Auf dieses Modell wird in Abschnitt 5.3 näher eingegangen.

Ein weiterer bedeutender Einfluss auf die agentenbasierte Simulation kam von der Künstlichen Intelligenz. Die verteilte KI hat den Begriff der Multiagentensysteme geprägt, d. h. offene Systeme, in denen mehrere Agenten in einer gemeinsamen Umwelt interagieren (siehe Abschnitt 2.3). Während die Forschung zu MAS sich zunächst auf Mechanismen zur Koordination intelligenter Agenten konzentrierte, setzte sich – wie in Abschnitt 2.3.3 dargelegt – Anfang der 1990er Jahre die Hypothese durch, dass die einzelnen Agenten nicht notwendigerweise intelligent sein müssen, damit das Gesamtsystem intelligentes Verhalten zeigt. Dieser Forschungsansatz betonte die Situiertheit der Agenten in ihrer Umgebung und die Emergenz intelligenten Verhaltens aus der Interaktion der Agenten mit dieser Umgebung.

Simulation wurde verwendet, um den Entwurf von Multiagentensystemen zu unterstützen. Sogenannte Testumgebungen (*testbeds*) stellen eine virtuelle Umgebung für die Agenten eines MAS zur Verfügung, in welchen anhand bestimmter Szenarien beispielsweise das Verhältnis von Entscheidungsqualität zu Zeit- und Ressourcenanforderung einer Agentenarchitektur getestet werden kann (Uhrmacher 2000). Die ersten Testumgebungen wurden für spezifische Anwendungsdomänen oder Agentenarchitekturen entwickelt; so z. B. DVMT (*Distributed Vehicle Monitoring Testbed*, Lesser und Corkill 1983), um die Koordination verteilter Problemlösungskomponenten (Agenten) anhand der kooperativen Überwachung von sich in einer zweidimensionalen Welt bewegenden Fahrzeugen zu untersuchen, und *Phoenix* (Cohen et al. 1989; Hanks et al. 1993), um adaptive Planer für die Bekämpfung von Waldbränden in einer realistischen dynamischen Umwelt zu testen. Spätere Testumgebungen wie MACE (*Multi-Agent Computing Environment*, Gasser et al. 1987) wurden explizit mit dem Ziel entworfen, domänen- und architekturunabhängig zu sein und somit die Entwicklung diverser Arten von MAS mit unterschiedlichem Detaillierungsgrad der Agenten zu ermöglichen. MACE kann als Vorfahre vieler der heute verfügbaren MAS-Toolkits angesehen werden (Michel et al. 2009).

Der Übergang von MAS-Toolkits, d. h. Softwarewerkzeugen zur Entwicklung von Multiagentensystemen, und Simulationswerkzeugen für die Entwicklung agentenbasierter Modelle ist fließend. Dies ist nicht weiter verwunderlich, kann doch ein agentenbasiertes Modell als Multiagentensystem aufgefasst werden. Klügl (2001) macht in ihrer Klassifikation von MAS anhand der Umgebung die Nähe von Agenten in Testumgebungen zu Agenten in einem agentenbasierten Modell deutlich (siehe Abbildung 3.2): Bei beiden handelt es sich um eine simulierte Umwelt, mit dem einzigen Unterschied, dass diese bei agentenbasierten Modellen als Teil des Modells mit entworfen werden muss.

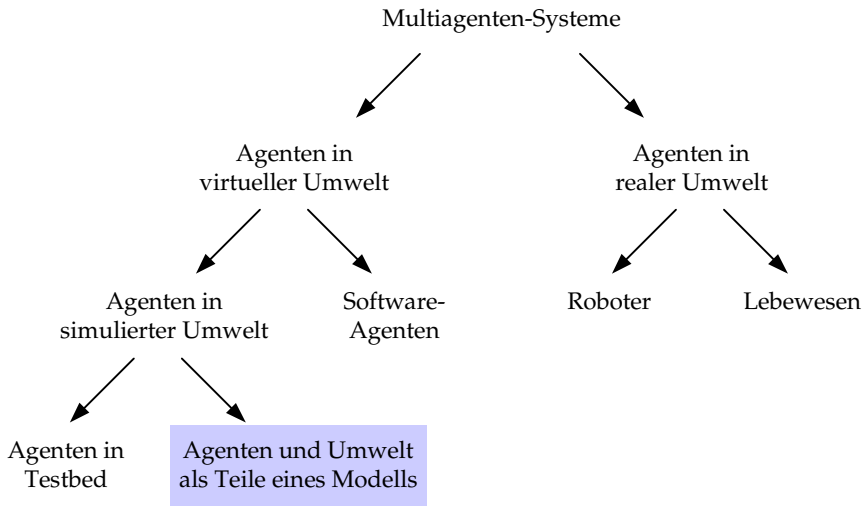

Abbildung 3.2.: Einordnung der agentenbasierten Simulation (grau hinterlegt) in Multiagentensysteme (nach Klügl 2001, S. 71)

In heutigen agentenbasierten Modellen werden die historischen Einflüsse aus der Komplexitätsforschung und der KI miteinander verbunden, indem die Konzepte der Multiagentensysteme auf die Modellierung und Simulation komplexer Systeme angewandt werden.

3.2.2. De-facto-Weltsicht der bestehenden Simulationswerkzeuge

In der agentenbasierten Simulation wird das Konzept der Multiagentensysteme zur Formulierung eines mikroskopischen Simulationsmodells genutzt. Die Entitäten eines Multiagentenmodells sind simulierte Agenten, welche in einer simulierten Umwelt miteinander agieren (Klügl 2001, S. 68). Zwar existiert, wie in Abschnitt 2.3.1 diskutiert, keine allgemein akzeptierte Definition des Begriffs „Agent“, doch stimmen die verschiedenen Auffassungen in der Regel in mehr Punkten überein als sie voneinander abweichen. Ein Vorteil dieser Situation ist, dass das Agentenkonzept vielseitig eingesetzt werden kann, von der Modellierung einzelner Zellen bis zu Personen oder Organisationen, so dass das Spektrum agentenbasierter Modelle von der Ausbreitung von Krebszellen im menschlichen Körper (vgl. z. B. Macklin und Edgerton 2010) über das Schwarmverhalten von Vögeln (als bekanntestes Beispiel sei Reynolds 1987 zitiert) bis zur Modellierung von (realen oder künstlichen) Gesellschaften (Kohler und Gumerman 2000; Epstein und Axtell 1996) reicht.

Allen Anwendungen gemeinsam ist, dass sie eine bestimmte Weltsicht einnehmen: Die aktiven Entitäten des zu modellierenden Systems werden als autonome Agenten betrachtet, welche einer Reihe von Verhaltensregeln folgen, um ihre Ziele zu erreichen, während sie miteinander und mit ihrer Umwelt interagieren. Um ein agentenbasiertes Simulati-

onsmodell zu erstellen, muss ein Modellierer sowohl die Struktur des Modells als auch dessen dynamisches Verhalten über die Zeit beschreiben. Aus der Definition eines Multiagentensystems als einer Menge von autonomen Agenten in einer gemeinsamen Umgebung (siehe Abschnitt 2.3.2) lassen sich die Komponenten eines agentenbasierten Modells ableiten (vgl. z. B. Klügl 2001; Macal und North 2010b):

1. Eine Menge von *Agenten*, die sich in ihren Attributen und ihrem Verhalten unterscheiden können.

2. *Beziehungen* zwischen diesen Agenten, welche die Interaktionen der Agenten beeinflussen; diese können neben Kommunikationsbeziehungen auch organisatorische oder soziale Beziehungen umfassen.

3. Eine *Umgebung*, mit der die Agenten interagieren. Diese kann eine räumliche Ausprägung besitzen und über eine eigene Dynamik verfügen, so dass Zustandsänderungen möglich sind, die nicht auf Aktionen von Agenten zurückzuführen sind.

Simulationswerkzeuge für agentenbasierte Modelle bieten Konstrukte für jede dieser Komponenten an. Während Swarm (Minar et al. 1996) als erstes allgemeines Werkzeug die Anfänge der agentenbasierten Simulation beherrschte, wird es heute von NetLogo (Wilensky 1999) und Repast (North et al. 2006) abgelöst, wie eine neuere Analyse von Publikationen zu agentenbasierten Modellen zeigt (LePage et al. 2012). Im Unterschied zu Swarm, Repast und vergleichbaren Simulationsframeworks wie z. B. Mason (Luke et al. 2005) und Ascape (Parker 2001) handelt es sich bei NetLogo um eine integrierte Entwicklungsumgebung, welche die rasche Erstellung von Modellen erleichtert. Die neueste Version von Repast, Repast Simphony (North et al. 2013), fällt ebenfalls in diese Kategorie, da es Repast um Konzepte von NetLogo erweitert und Eclipse als Entwicklungsumgebung verwendet. Die Simulationswerkzeuge unterscheiden sich leicht in Funktionsumfang und der verwendeten Terminologie für die zur Verfügung gestellten Konstrukte (Railsback et al. 2006); so bietet beispielsweise nur NetLogo eine Agenten-Klasse mit vordefinierten Attributen und Standardfunktionalität für Bewegung im Raum.

Für die Beschreibung des dynamischen Verhaltens muss festgelegt werden, zu welchen Simulationszeitpunkten und unter welchen Bedingungen die Agenten (und ggf. die Umwelt) ihre Aktionen ausführen. In Abhängigkeit von der gewählten Zeitführung existieren verschiedene Möglichkeiten, Aktionen mit der Simulationszeit in Beziehung zu setzen. Am weitesten verbreitet ist Zeitsteuerung (Michel et al. 2009, S. 24), d. h. Zeitfortschreibung in regelmäßigen Abständen mit synchroner Aktualisierung aller Entitäten (vgl. Abschnitt 2.1.2). In zeitgesteuerten Modellen wird angenommen, dass ein Agent pro Zeitschritt eine Folge von konzeptuell gleichzeitigen Aktionen ausführt (Grimm und Railsback 2005, S. 109). Diese werden in jedem Zeitschritt wiederholt, bis der Agent ggf. aufgrund bestimmter Bedingungen zu einer anderen Aktionsfolge wechselt (Klügl 2001, S. 122).

Um Simulationsartefakte zu vermeiden, wird die Aktualisierung der Agenten in der Regel in zufälliger Reihenfolge vorgenommen. Während manche Simulationsumgebungen, wie z. B. NetLogo, dies transparent intern umsetzen, muss der Benutzer in anderen die zufällige Abarbeitung explizit festlegen; Funktionalität dafür wird jedoch meist von den

Umgebungen zur Verfügung gestellt. So erläutert beispielsweise ein Repast-Tutorial, wie für jeden Zeitschritt die Liste der Agenten zuerst gemischt, bevor für jeden Agenten dessen Aktualisierungsmethode aufgerufen wird (Murphy 2003, Step 21). Eine Variante der zufälligen Abarbeitung bietet das Toolkit LEE (Latent Energy Environment), welches für die Untersuchung evolutionärer komplexer Systeme mit neuronalen Netzen zugeschnitten ist: In jedem Zeitschritt wird ein Agent nur mit einer bestimmten Wahrscheinlichkeit ausgeführt (Theodoropoulos et al. 2009, S. 80).

In vielen agentenbasierten Modellen wird der Bezug zur Simulationszeit nicht explizit hergestellt. Statt Ereignisse mit bestimmten Zeitpunkten zu verknüpfen, wird das Konzept eines Ereignisses als Zustandsänderung zu einem Zeitpunkt verallgemeinert zu der Ausführung des Verhaltens eines Agenten, einschließlich Interaktion mit anderen Agenten und der Umgebung. Dies mag mit einem Zeitpunkt in der realen (modellierten) Zeit korrelieren, ein Zeitbezug ist jedoch nicht nötig, da eine geordnete Folge von Ereignissen ausreicht, um das Verhalten des Modells korrekt zu beschreiben (Macal 2013).

Zwar bieten einige Simulationswerkzeuge wie z. B. Repast und Mason die Möglichkeit an, zukünftige Ereignisse dynamisch, d. h. während der Simulation, auf der Ereignisliste einzutragen, doch fokussieren zugehörige Beispielmodelle und Einführungen darauf, wie zeitgesteuerte Modelle mit dem jeweiligen Werkzeug erstellt werden können (vgl. z. B. Minar et al. 1996, S. 3; Luke 2013, S. 16f; North et al. 2013, S. 18ff). Eine Ausnahme stellt das von Swarm übernommene *Mousetrap*-Modell dar, das eine Kettenreaktion mit Tischtennisbällen und Mausefallen ereignisgesteuert abbildet. Railsback und Grimm (2011, S. 189f) demonstrieren, wie dieses Modell sogar in NetLogo unter Umgehung der internen Zeitsteuerung implementiert werden kann. Dennoch sind Multiagenten-Modelle mit expliziter Ereignissteuerung rar. Ein Beispiel ist das in (Troitzsch 2004) beschriebene Modell, welches die Entwicklung von Zweisprachigkeit in einer kleinen Population von Individuen untersucht. Die demographischen Aspekte des Modells (Geburt, Tod, Partnerwahl und Nachwuchs) sind in ereignisorientierter Form realisiert. Ein weiteres Beispiel liefern Lawson und Park (2000) mit der ereignisgesteuerten Re-Implementation des Sugarscape-Modells (Epstein und Axtell 1996).

Geht man von der naheliegenden Annahme aus, dass die agentenbasierte Simulation ebenso von den verfügbaren Modellierungswerkzeugen geprägt wird, wie die klassischen Weltsichten der diskreten Simulation von paradigmatischen Programmiersprachen geprägt wurden, ergibt sich die folgende, De-facto-Weltsicht:

- Das zu modellierende System wird als Multiagentensystem aufgefasst, d. h. die aktiven Systemelemente werden als Agenten modelliert, welche in einer gemeinsamen Umgebung interagieren. Wie bei der traditionellen diskreten Simulation wird zwischen Struktur und Verhalten des Systems unterschieden: Die Struktur ergibt sich aus den einzelnen Systemelementen und deren Beziehungen untereinander, das dynamische Verhalten aus den Handlungen der aktiven Systemelemente.

- Ein agentenbasiertes Modell bildet daher ebenfalls ein Multiagentensystem, bestehend aus den Agenten, deren Beziehungen und der Umgebung. Die Umgebung kann eine räumliche Ausprägung besitzen, über eine eigene Dynamik verfügen und zusätzliche (passive) Objekte wie z. B. Ressourcen enthalten.

- Agenten und ggf. die Umgebung handeln konzeptuell nebenläufig. Dies lässt sich abbilden, indem die möglichen Aktionen identifiziert und die Reihenfolge ihrer wiederholten Ausführung festgelegt wird (Grimm und Railsback 2005, S. 111). Dieser Modellierungsstil wird von NetLogo und Swarm propagiert. Eine andere Möglichkeit ist es, alle Agenten einmal pro Zeitschritt zu aktivieren, so dass sie ihr Verhalten ausführen und ihren Zustand aktualisieren können (Chan et al. 2010, S. 138). Dies ist der Standard in Mason und vielen Repast-Modellen. In beiden Fällen ist die Simulation zeitgesteuert, d. h. die Simulationssteuerung erfolgt in regelmäßigen Zeitschritten mit synchroner Aktualisierung der Agenten.

3.2.3. Zeitsteuerung vs. Ereignissteuerung

Die Dominanz der Zeitsteuerung in der agentenbasierten Simulation hat verschiedene Gründe. Die Entstehungsgeschichte der ABMS veranschaulicht, dass der agentenbasierte Ansatz zur Modellierung und Simulation – und insbesondere die Softwarewerkzeuge für dessen Umsetzung – nicht aus der traditionellen zeitdiskreten Simulation entstanden ist (Macal und North 2010a). Während die beteiligten Disziplinen keinen Hintergrund in diskreter Simulation hatten oder andere Ansätze wie kontinuierliche Simulation verwendeten, hat die Forschungsgemeinde der diskreten Simulation den entstehenden ABMS-Ansatz zunächst weitgehend ignoriert, wie beispielsweise die Tagungsbände der Winter Simulation Conference (INFORMS Simulation Society 2013), der bedeutendsten Konferenz für diskrete Simulation, zeigen. Anwendungen und Werkzeuge für agentenbasierte Simulation wurden daher meist von Domänen-Experten entwickelt.

Eine Zeitsteuerung ist leichter umzusetzen, da die Ablaufsteuerung bei gleichmäßigem Zeitfortschritt und synchroner Aktualisierung aller Entitäten auf die Verwaltung einer Ereignisliste verzichten kann (vgl. Abschnitt 2.1.2). Bei Werkzeugen und Modellen, welche die Simulationsaspekte selbst implementieren müssen, wird daher in der Regel die einfachere Zeitsteuerung umgesetzt. Ein weiterer Einfluss mag aus der kontinuierlichen Simulation kommen, in welcher ein System von Differentialgleichungen schrittweise gelöst wird.

Zum anderen vereinfacht eine Zeitsteuerung die Modellierung des Agentenverhaltens, da Aktionen durch die Taktrate implizit eine Dauer erhalten. Der Modellierer muss sich keine Gedanken darüber machen, welche Zeitdauer bestimmte Handlungen haben und bei mehreren zeitgleichen Handlungen nur deren Reihenfolge festlegen. Dies kann als Vorteil gegenüber der Ereignissteuerung gelten, in der jeder zeitverbrauchenden Aktion eine bestimmte Dauer zugeordnet werden muss. Andererseits lässt sich mit gleicher Berechtigung argumentieren, dass die Ereignissteuerung ein Modell transparenter macht, da sie den Modellierer zwingt, Aktionsdauern explizit zu machen. Für Systeme, in denen die Eintrittszeitpunkte von Ereignissen bedeutsam sind, wie beispielsweise Auftragseingänge an der Börse (vgl. z. B. Jacobs et al. 2004; Daniel 2006; Boer et al. 2007) oder in dem in Kapitel 6 beschriebenen Kurierdienst, ist Ereignissteuerung mit der daraus resultierenden asynchronen Aktualisierung von Agenten nötig, um eine korrekte Abbildung des Realsystems zu erreichen.

Im Zusammenhang mit der Aktualisierung von Agenten wurden in der diskreten Simulation bekannte Probleme (siehe Abschnitt 2.1.3) von Anwendern der agentenbasier-

ten Simulation erneut entdeckt. So zeigen beispielsweise Huberman und Glance (1993), welche Auswirkungen der Wechsel von synchroner zu asynchroner Aktualisierung auf die Resultate eines Modells haben kann: Während sich bei synchroner Aktualisierung aller auf einem zweidimensionalen Gitter angeordneten Agenten Bereiche mit Kooperation im Gefangenendilemma ergeben, welche interessante räumliche Muster auf der Makro-Ebene bilden, entwickeln sich die Agenten bei asynchroner Aktualisierung einheitlich zu Verrätern. Page (1997) erweitert diese Analyse um positions- und nutzen-basierte Prioritätsmechanismen, angewandt auf das *Game of Life* und ein einfaches Meinungsbildungsmodell, in dem ein Agent jeweils die Meinung der Mehrheit seiner Nachbarn annimmt. Axtell (2001) betont, dass synchrone Aktualisierung auf die zufällige Abarbeitung der Agentenliste angewiesen ist, um Simulationsartefakte zu vermeiden. Er untersucht, welcher Aufwand benötigt wird, d.h. wie viele Agenten ihre Positionen in der Liste tauschen müssen, so dass im nächsten Zeitschritt die meisten Agenten mindestens einen oder genau zwei neue Nachbarn haben.

Im Unterschied zu den ABMS-Werkzeugen, welche eine Zeitsteuerung ausschließlich oder zumindest bevorzugt anbieten, basierten die Testumgebungen für (Multi-)Agentensysteme in der Regel auf einer Ereignissteuerung (Uhrmacher 2000, S. 21ff). Die Zeit, welche ein Agent zum „Nachdenken“ benötigt, wird dabei abgebildet auf die Rechenzeit, wodurch die Performanz eines Agenten von Experiment zu Experiment variieren kann, auf eine rechenzeit-unabhängige Konstante oder auf die Anzahl der auszuführenden Instruktionen bzw. die Größe der verwendeten Wissensbasis.

Gegen die Verwendung von Ereignissteuerung in der agentenbasierten Simulation kann der Einwand erhoben werden, die benötigte Ablaufsteuerung widerspreche dem grundlegenden Prinzip, dass Multiagentensysteme keiner zentralen Kontrollinstanz unterliegen (Davidsson 2001, S. 100). Dies gilt jedoch auch für zeitgesteuerte Simulation, die eine globale Uhr für den gemeinsamen Zeittakt verwendet. Fasst man die Simulationsinfrastruktur nur als technisches Hilfsmittel zur Ausführung der nebenläufig agierende Agenten und Umweltprozesse auf, stellen Multiagentenmodelle weiterhin eine Form von Multiagentensystemen dar; auf konzeptueller Ebene ist die Autonomie der Agenten sichergestellt.

Andere Autoren argumentieren dagegen, dass gerade ein gemeinsamer Zeittakt und synchrone Aktualisierung aller Agenten dem Autonomie-Konzept widersprechen: „Forcing all agents of the MAS to act in lock step does not fit with autonomy of agents“ (Weyns und Holvoet 2003, S. 177). Ereignissteuerung ist angemessener, da sie den Agenten nicht *a priori* eine Synchronisation aufzwingt (Baveco und Lingeman 1992, S. 269). Darüber hinaus sind soziale Prozesse – Hauptanwendungsgebiet agentenbasierter Modelle – nur selten synchron, weshalb es ebenso selten adäquat ist, sie durch synchrone Aktualisierung aller Agenten zu modellieren (Axtell 2001, S. 41).

Ein inhärentes Problem der Zeitsteuerung ist, dass die Zeit in gleichmäßige Intervalle diskretisiert wird, so dass alle Ereignisse, welche in einen Zeitschritt fallen, behandelt werden, als ob sie zum selben Zeitpunkt stattfinden. Die richtige Reihenfolge festzulegen, liegt damit in der Verantwortung des Modellierers. Verschiedene Simulationswerkzeuge mit Zeitsteuerung versuchen den Modellierer hierbei zu unterstützen, indem sie Möglichkeiten zur feineren Unterteilung des Zeittaktes Δt zur Verfügung stellen. So bietet SDML (Moss et al. 1998) benutzerdefinierte Zeitebenen, wodurch der grobe Zeitschritt ei-

nes Modells von beispielsweise einer Woche ergänzt werden kann um Tage und Stunden. Bestimmte Regeln des deklarativ spezifizierten Verhaltens der Agenten werden dann den unterschiedlichen Ebenen zugeordnet, wodurch erreicht wird, dass Agenten innerhalb eines Modell-Zeitschritts miteinander kommunizieren und zu einer Entscheidung kommen können. Ohne die verschiedenen Zeitebenen würde der Austausch von Nachrichten mehrere Zeitschritte benötigen, was in vielen Modellen nicht realistisch ist.

Ein weiteres Beispiel sind die hierarchisch geschachtelten Schwärme von Agenten in *Swarm* (Minar et al. 1996). Jeder Agentenschwarm ist mit seiner eigenen Ereignisliste versehen, auf der die in einem Zeitschritt auszuführenden Handlungen der Agenten eingetragen werden. Wird ein Schwarm aus mehreren anderen Schwärmen zusammengesetzt, sorgt die Simulationsumgebung automatisch für die korrekte Integration der zugehörigen Ereignislisten.

Die Verwendung der Ereignissteuerung macht solche Hilfsmittel überflüssig, da die Diskretisierung der Simulationszeit von den Abläufen im modellierten System selbst vorgegeben wird anstatt von einer künstlichen Taktrate. Dies ermöglicht darüber hinaus die natürliche Abbildung von Agenten und Umweltprozessen, welche auf unterschiedlichen Zeitskalen agieren. Die ereignisgesteuerte Simulation ist flexibler, genauer und effizienter als die zeitgesteuerte Simulation (Zaft und Zeigler 2002). Letztere kann darüber hinaus leicht mit Hilfe der Ereignissteuerung realisiert werden, indem sich Agenten in regelmäßigen Abständen zur Reaktivierung vormerken lassen. Dies wird in der Re-Implementation des Sugarscape-Modells in Abschnitt 5.4 demonstriert.

3.2.4. Ansätze zur formalen Definition

Im Unterschied zu traditionellen Simulationsmethoden wie der diskreten Simulation fehlt es bisher an einer vergleichbaren theoretischen Grundlage für die agentenbasierte Simulation. Dies lässt sich u. a. darauf zurückführen, dass die agentenbasierte Simulation noch ein verhältnismäßig junges Forschungsgebiet ist, dessen Wurzeln in verschiedenen Disziplinen liegen (vgl. Abschnitt 3.2.1). In den letzten Jahren ist sich die Forschungsgemeinde dieser Lücke bewusst geworden und hat vereinzelte Ansätze zur Formalisierung und Standardisierung hervorgebracht. Ein Beispiel für letzteres ist das sogenannte ODD-Protokoll für die Dokumentation von agentenbasierten Modellen (Grimm et al. 2006, 2010, 2013), welches anstrebt, die Vermittlung und Reproduzierbarkeit von Modellen zu verbessern und so ihre wissenschaftliche Glaubwürdigkeit zu erhöhen.

Die folgende Diskussion beschränkt sich auf diejenigen Ansätze zur Formalisierung, welche einen ereignisgesteuerten Zeitfortschritt verwenden oder zumindest andeuten. Zum einen lässt sich, wie in Abschnitt 2.1.2 dargelegt, die Zeitsteuerung als Spezialfall der Ereignissteuerung auffassen. Zum anderen kann formal bewiesen werden, dass sich ein Multiagentenmodell auf ein äquivalentes ereignisgesteuertes Modell abbilden lässt (Onggo 2010). Ereignissteuerung ist damit das allgemeinere Konzept.

3.2.4.1. DEVS-Erweiterungen

DEVS (*Discrete Event System Specification*) ist ein Formalismus zur Beschreibung ereignisdiskreter Systeme (Zeigler 1976, 1984; Zeigler et al. 2000), der zwischen atomaren und

gekoppelten Modellen unterscheidet und somit modulare, hierarchische Modellierung ermöglicht. Ein atomares Modell ist definiert durch eine Struktur

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle \tag{3.1}$$

aus einer Menge X von Eingabewerten (externe Ereignisse), einer Menge S von Zuständen, einer Menge Y von Ausgabewerten, einer internen (δ_{int}) und einer externen (δ_{ext}) Zustandsübergangsfunktion, einer Ausgabefunktion λ sowie einer Zeitfortschrittsfunktion ta. Zu jedem Zeitpunkt t befindet sich das Modell in einem Zustand $s \in S$. Nach Ablauf einer vorgegebenen Zeit $ta(s)$ produziert das Modell die Ausgabe $y = \lambda(s)$ und geht in den von der internen Zustandsübergangsfunktion bestimmten Folgezustand s' über. Dies kann als Auftreten eines internen Ereignisses interpretiert werden. Die einem Zustand von der Zeitfortschrittsfunktion zugeordnete Dauer ist in der Regel eine positive reelle Zahl; sie kann jedoch explizit die Werte 0 (sogenannter *transienter* Zustand) oder ∞ (*passiver* Zustand) annehmen. Tritt vor Ablauf der Zeit $ta(s)$ ein externes Ereignis ein, d. h. erhält das atomare Modell eine Eingabe x zum Zeitpunkt $e \leq ta(s)$, bestimmt die externe Zustandsübergangsfunktion den Folgezustand in Abhängigkeit von x, dem aktuellen Zustand s sowie der in diesem Zustand verbrachten Zeit e. Ein- und Ausgaben erreichen bzw. verlassen das Modell über spezielle Ein- und Ausgabekanäle, welche die Schnittstelle des Modells zur Außenwelt darstellen.

Ein gekoppeltes Modell besteht aus einer Menge von Komponenten (atomaren oder gekoppelten Modellen) und einer Spezifikation der Kopplung zwischen diesen Komponenten sowie den Ein- und Ausgabekanälen des gekoppelten Modells. DEVS unterstützt somit die hierarchische, modulare Konstruktion von Modellen. Der Zustand eines gekoppelten Modells ist die Summe der Zustände seiner Komponenten. Das Verhalten eines gekoppelten Modells ergibt sich aus dem Verhalten seiner Komponenten und deren Interaktion; es verfügt nicht über separate Zustandsübergangs-, Ausgabe- oder Zeitfortschrittsfunktionen. Der einzige Einfluss auf das Zusammenspiel der Komponenten erfolgt über eine sogenannte Selektor-Funktion, die bei zeitgleichen Ereignissen entscheidet, welches zuerst ausgeführt wird.

Um ein DEVS-Modell auszuführen, wird jedem atomaren Modell ein abstrakter Simulator zugeordnet, jedem gekoppelten Modell ein Koordinator. Der Koordinator ist für den Zeitfortschritt des gekoppelten Modells (und damit aller Komponenten) zuständig; jede Komponente liefert den Zeitpunkt des nächsten zukünftigen Ereignisses zurück und der Koordinator wählt daraus den unmittelbar bevorstehenden Zeitpunkt aus. Alle Komponenten, für die ein Ereignis zu diesem Zeitpunkt anliegt, werden benachrichtigt.

Übertragen auf die agentenbasierte Simulation können Agenten als atomare Modelle repräsentiert werden, während ein Multiagenten-Modell einem gekoppelten Modell in DEVS entspricht. Die externe Zustandsübergangsfunktion modelliert dann das reaktive Verhalten eines Agenten, die interne Zustandsübergangsfunktion bildet proaktives Verhalten ab. Letzteres kann durchaus deliberativ sein, indem die Menge der Zustände als Wissensbasis aufgefasst und beispielsweise in Ziele, Intentionen und Annahmen über die Umwelt unterteilt wird (Uhrmacher und Schattenberg 1998). Eine andere Möglichkeit besteht darin, Agenten als gekoppelte Modelle zu repräsentieren, deren Komponenten jeweils für einen Teil des Verhaltens zuständig sind (Müller 2009, S. 114).

Varianten des reinen DEVS-Formalismus wurden für verschiedene Spezialisierungen entwickelt, z. B. um die Integration von diskreten und kontinuierlichen Systemen (DEV& DESS, Zeigler et al. 2000, S. 203ff) zu erlauben oder die verteilte Simulation diskreter Systeme (Chow 1996, Zeigler et al. 2000, S. 89ff.) zu ermöglichen. Diese P-DEVS oder Parallel DEVS genannte Erweiterung führt die konfluente Zustandsübergangsfunktion δ_{con} ein, welche explizit für die Behandlung von gleichzeitig eintretenden internen und externen Ereignissen zuständig ist. Damit liegt die Verantwortung für die Reihenfolge der Ausführung beim Modell (bzw. Modellierer) und ist nicht vom verwendeten Simulator abhängig, was u. a. die Reproduzierbarkeit von Simulationsexperimenten verbessert.

Das ursprüngliche DEVS geht von der Annahme aus, dass die Struktur des zu modellierenden Systems statisch ist. Dies trifft für Multiagentensysteme und insbesondere für die agentenbasierte Simulation nicht zu. Das zu modellierende System ist in der Regel dynamisch, d. h. Agenten betreten oder verlassen das System (Modell) und die Interaktionsbeziehungen zwischen den Agenten können sich im Laufe der Simulation verändern. Uhrmacher (1996) trägt dieser Tatsache Rechnung mit dem Entwurf eines agentenorientierten DEVS (AgedDEVS). Obwohl dieser Ansatz für Testumgebungen für (intelligente) Agenten entwickelt wurde, lässt er sich leicht auf die agentenbasierte Simulation übertragen, schließlich handelt es sich bei beiden um Agenten in einer simulierten Umwelt (vgl. Abbildung 3.2). AgedDEVS und seine Weiterentwicklungen DynDEVS (Uhrmacher 2001) und ρ-DEVS (Uhrmacher et al. 2006) erlauben DEVS-Modellen ihre Struktur zu verändern, d. h. während der Simulation Modellkomponenten eines gekoppelten Modells zu entfernen, neue Komponenten hinzuzufügen und die Kopplung zwischen Komponenten zu modifizieren. Modelle mit variabler Struktur werden dabei als Folgen von Modellvarianten aufgefasst. Eine Strukturänderung bedeutet, dass ein Modell von einer Inkarnation in eine andere übergeht. Die spezielle Modellübergangsfunktion ρ_a, welche in Abhängigkeit vom aktuellen Zustand die neue Modell-Inkarnation bestimmt, sorgt dafür, dass die aktuellen Werte gemeinsamer Variablen erhalten bleiben und neue Variablen mit geeigneten Werten initialisiert werden.

Jean-Pierre Müller (2009) kombiniert P-DEVS und DynDEVS zu seinem M-DEVS genannten Ansatz. Er entkoppelt die „physikalischen“ Transitionen, d. h. zeitverbrauchenden Aktivitäten (modelliert wie in P-DEVS durch die externe, interne und konfluente Zustandsübergangsfunktionen), von der zeitverzugslosen Weiterleitung von Information (modelliert durch die logische Zustandsübergangsfunktion δ_{log} und zugehörige Ausgabefunktion λ_{log}) und strukturellen Änderungen (modelliert durch die strukturelle Ausgabefunktion λ_{str}). Die tatsächliche Durchführung struktureller Änderungen ist Aufgabe des Koordinators. Nach jeder Zustandsänderung eines (atomaren) Modells werden die daraus resultierenden logischen und strukturellen Einflüsse berechnet und als Nachrichten an die gekoppelten Modelle bzw. den Koordinator weitergeleitet. Letzterer führt die anliegenden strukturellen Änderungen (Erzeugung eines neuen Modells, Löschen eines bestehenden Modells, Aufheben oder Einfügen einer Kopplung zwischen zwei Modellen) aus bevor die benachrichtigten Modelle ihrerseits ihre logische Zustandsübergangsfunktion anwenden, was zu weiteren logischen und strukturellen Ausgaben führen kann. Die algorithmische Spezifikation des abstrakten M-DEVS-Koordinators sieht die iterative Wiederholung der letzten beiden Schritte vor, bis keine weiteren Ausgaben mehr abzuar-

beiten sind. Erst dann wird der Zeitpunkt des nächsten Ereignisses bestimmt.

Ein vergleichbarer Ansatz (Duboz et al. 2006) basiert auf DSDEVS (Barros 1997), einer alternativen Spezifikation für Modelle mit variabler Struktur, welche ein sogenanntes Ausführungsmodell (*executive model*) als Teil eines gekoppelten Modells vorsieht, das für die Durchführung struktureller Änderungen verantwortlich ist. Agenten sind als DSDEVS-Modelle definiert, mit dedizierten Input- und Output-Ports für Kommunikation mit anderen Agenten, mit der Umwelt und mit dem Simulator. Im Unterschied zu Müller (2009), der strikt auf der Ebene von DEVS bleibt, versuchen Duboz et al. (2006) explizit Agenten-Konzepte wie Wahrnehmung, proaktives und reaktives Verhalten sowie Autonomie auf DEVS-Elemente abzubilden, um Modellierern die Implementation zu erleichtern.

Die Diskrepanz zwischen den in Multiagentensystemen verwendeten Konzepten und dem in der Systemtheorie wurzelnden DEVS-Formalismus ist ein Nachteil der Benutzung von DEVS zur Formalisierung von agentenbasierter Simulation. Das von DEVS angebotene Abstraktionsniveau eines Systems mit Ein- und Ausgabekanälen und Zustandsübergangsfunktionen, welches über seine Kanäle mit anderen Systemen gekoppelt werden kann, ist niedriger als das eines Agenten, der seine Umgebung wahrnehmen, mit anderen Agenten kommunizieren und sich in der Umgebung bewegen kann. Alle Entitäten eines Modells, ob Agenten, passive Objekte oder Zellen eines Gitters müssen gleichermaßen als (atomare oder gekoppelte) DEVS repräsentiert und Interaktionen zwischen Entitäten als direkte Kopplung zwischen DEVS abgebildet werden. Sobald sich ein Agent von einer Zelle zu einer anderen bewegt oder mit einem anderen Agenten kommunizieren möchte, muss daher die Kopplungsstruktur des Modells verändert werden. Die Simulation eines solchen Modells ist aufwendig, da in der Regel eine große Zahl von atomaren Modellen inklusive Nachrichtenaustausch zwischen diesen zu realisieren sind (Bisgambiglia und Franceschini 2013).

Ein Vorteil von DEVS ist dagegen die klare Trennung von Modellen und Simulationsinfrastruktur. Diese ist in heutigen Werkzeugen für die agentenbasierte Simulation nicht gegeben:

> Most, if not all, existing MAS platforms produce simulation results which do not depend only on the model but on the way the model is implemented and the scheduling ordered. This ordering is at worst arbitrary and at best randomized. (Müller 2009, S. 111)

Durch geeignete algorithmische Spezifikation der abstrakten Simulatoren und Koordinatoren von DEVS kann eine eindeutige operationale Semantik festgelegt und damit die Implementationsgrundlage für eine Simulationsumgebung geliefert werden, welche die wesentlichen Eigenschaften für MAS wie reaktives and proaktives Verhalten, Nebenläufigkeit (d. h. zeitgleiche Ereignisse), zeitverzugslose Informationsweiterleitung und strukturelle Dynamik unabhängig von der Reihenfolge der zugehörigen Ereignisse ermöglicht (Müller 2009). Zusammenfassend lässt sich festhalten, dass DEVS gut für die Spezifikation eines Simulators für agentenbasierte Modelle geeignet ist, während es für die Unterstützung der Modellierung nicht das richtige Abstraktionsniveau bietet.

3.2.4.2. Referenzmodelle

Im Unterschied zu den beschriebenen DEVS-Erweiterungen, welche MAS-Konzepte in geeigneter Form in den DEVS-Formalismus zu integrieren suchen, streben Referenzmodelle an, ein klar definiertes Bezugssystem vorzugeben, das unabhängig von bestimmten Techniken oder Implementationen ist. Für die agentenbasierte Modellierung und Simulation bedeutet dies, dass ein vollständiges Referenzmodell die Schlüsselkonzepte agentenbasierter Modelle sowie deren Beziehungen und Abhängigkeiten identifizieren, eindeutige Definitionen für dieses Konzepte liefern und ein Ausführungsmodell bereitstellen muss, welches klar spezifiziert, was unter der Simulation eines agentenbasierten Modells zu verstehen ist (Siegfried et al. 2009).

Verschiedene Ansätze für ein Referenzmodell konzentrieren sich auf die Identifikation und Definition der Bestandteile agentenbasierter Modelle und vernachlässigen oder vereinfachen Simulationsaspekte, indem sie z. B. von einer schrittweisen Ausführung der Simulation mit synchroner Aktualisierung aller Agenten ausgehen, wodurch sie implizit annehmen, dass Handlungen keine bzw. die konstante Dauer von einem Zeitschritt besitzen (vgl. u. a. Scheutz und Schermerhorn 2006; Klügl 2007). Andere decken nur einen Teil der Konzepte ab, berücksichtigen dafür jedoch ein detailliertes Ausführungsmodell; (Helleboogh et al. 2007) ist ein Beispiel für ein Referenzmodell für dynamische Umgebungen.

Siegfried et al. (2009) präsentieren ein Referenzmodell, das ausdrücklich von der Art des Zeitfortschritts (regelmäßiges oder unregelmäßiges Intervall Δt) in der Simulation abstrahiert. Die Abarbeitung eines Ereignisses ist definiert als die Aktualisierung aller Agenten, welche durch das betreffende Ereignis aktiviert werden. Da Handlungen darüber hinaus explizit mit einer Dauer versehen sind, die unterschiedliche Werte für unterschiedliche Handlungen besitzen und nicht das Vielfache eines Zeitschritts sein müssen, erlaubt dies ereignisgesteuerten Zeitfortschritt mit asynchroner Aktualisierung. Ein agentenbasiertes Modell ist definiert als ein Tupel $ABM = (MW, ENT, EV)$ aus Modellwelt MW, Entitäten ENT und Ereignissen EV. Die Modellwelt bildet die gemeinsame Umwelt für alle Agenten; sie besteht aus einer Spezifikation der Simulationszeit (diskret oder kontinuierlich), der eigentlichen Umgebung, die als relativ passiv aufgefasst wird, und einer Menge von Regeln, welche die (Natur-)Gesetze der Modellwelt festlegen und ebenfalls für die Erkennung von Konflikten zuständig sind. Eine genauere Definition, wie dies umzusetzen ist, bleibt allerdings dem Anwender überlassen. Entitäten werden in Agenten und (passive) Objekte unterteilt. Von einer bestimmten Agenten-Architektur wird ausdrücklich abstrahiert; Agenten werden modelliert über ihre Aktionen, welche zu nebenläufigen Prozessen zusammengefasst werden können.

Während die Autoren die Verwendung des Referenzmodells anhand eines einfachen zeitgesteuerten Modells demonstrieren, steht eine vollständige Evaluation, die dessen generelle Anwendbarkeit auf agentenbasierte Modelle belegt, noch aus. Dies sollte auch einen Vergleich mit der traditionellen ereignisdiskreten Simulation beinhalten, um Gemeinsamkeiten und Unterschiede herauszuarbeiten.

Die Unterscheidung zwischen (aktiven) Agenten und (passiven) Objekten steht im Zentrum des sogenannten AOR-Ansatzes (Wagner 2004; Wagner et al. 2008). AOR ist die Abkürzung für *Agent-Object-Relationship* und kann als Verfeinerung des klassischen Entity-

Relationship-Modells angesehen werden. Ein AOR-Modell besteht aus einer Menge von Entitäten, einem Raummodell und einer Menge von Umgebungsregeln, welche die im Modell geltenden Kausalgesetzmäßigkeiten festlegen. Entitäten werden untergliedert in Agenten, Objekte, Nachrichten und Ereignisse; letztere umfassen die Aktionen der Agenten als spezielle „Handlungsereignisse“. Die Ausführungsemantik ist gegeben für einen abstrakten Simulator, bestehend aus einem Umgebungssimulator (der auch die allgemeine Ereignisliste verwaltet) und je einem Agentensimulator für die Agenten. Jeder Zeitschritt wird in fünf Teilschritte aufgeteilt: Nach Ermittlung der aktuellen Ereignisse wird zunächst die Umgebung aktualisiert, bevor der Umgebungssimulator jedem Agenten die ihn betreffenden Wahrnehmungsereignisse übermittelt. Die Agentensimulatoren aktualisieren daraufhin die zugehörigen Agenten und übermitteln die daraus resultierenden Handlungsereignisse an den Umgebungssimulator. Zuletzt wird die Simulationszeit um 1 erhöht. Dies realisiert eine Zeitsteuerung mit Zweiphasen-Aktualisierung (vgl. Abschnitt 2.1.3.2). Die Möglichkeit der Ereignissteuerung wird angedeutet (Wagner 2004, S. 183), jedoch nicht wirklich umgesetzt: Alle Handlungen besitzen die konstante Dauer 1 und die Aktualisierung aller Agenten verläuft synchron.

Der AOR-Ansatz geht über ein reines Referenzmodell hinaus, da er eine Modellierungssprache für die Spezifikation von agentenbasierten Modellen und Werkzeuge für die automatische Übersetzung einer solchen Spezifikation in ein ausführbares Modell besitzt. Als negativ zu bewerten ist die zugrundeliegende unsaubere Ontologie (Handlungen als Ereignisse, Ereignisse und Nachrichten als Modellentitäten), welche dem Anspruch der Autoren, „eine größere Unterstützung der Semantik eines Realsystems“ zu erlauben und „dadurch eine wirklichkeitsgetreuere Modellierung“ zu erleichtern (Wagner et al. 2008, S. 49), nicht gerecht wird.

3.2.5. Die agentenorientierte Weltsicht

Im folgenden wird abweichend von der in der agentenbasierten Simulation *de facto* etablierten Weltsicht, welche von einer zeitgesteuerten Ausführung des Modells ausgeht, eine der traditionellen diskreten Simulation näher stehende Weltsicht eingeführt. Diese verbindet die Konzepte der Multiagentensysteme mit einer ereignisgesteuerten Ausführung des Modells. Sie soll in Anlehnung an die Bezeichnungen der klassischen Weltsichten der diskreten Simulation (vgl. Abschnitt 3.1.1) als „agentenorientiert“ bezeichnet werden und kann als eine Variante der agentenbasierten Weltsicht aufgefasst werden.

Beide Weltsicht-Varianten verwenden die Agenten-Perspektive bei der Beschreibung des zu modellierenden Systems. Die aktiven Entitäten des Systems werden als autonome Agenten modelliert, welche miteinander und mit der gemeinsamen Umgebung interagieren. Das resultierende Modell bildet ein Multiagentensystem. Die Struktur eines solchen Modells ist daher in beiden Weltsicht-Varianten gleich; der Unterschied besteht in der Spezifikation des dynamischen Verhaltens. Dieses besteht aus den Handlungen der Agenten und eventueller Umweltprozesse, welche nebenläufig stattfinden.

In der vorherrschenden agentenbasierten Sicht muss ein Modellierer nach Identifikation der Aktionen die Reihenfolge festlegen, in der diese Aktionen in jedem Zeitschritt ausgeführt werden sollen (vgl. Abschnitt 3.2.2). Alle simulierten Ereignisse treten somit in einer vorherbestimmten Reihenfolge einmal pro Zeitschritt auf (Grimm und Railsback

2005, S. 109). Die Dauer der Aktionen ergibt sich implizit aus der Länge des Zeittaktes als $\leq \Delta t$. Über die Anzahl der in einem Zeitschritt auszuführenden Aktionen lässt sich deren Dauer weiter regulieren (Klügl 2001, S. 122).

Für manche Modelle ist dieser Ansatz weniger gut geeignet. Wenn die Zeitpunkte für Ereignisse nicht zu Beginn der Simulation vorherbestimmt sind, ist es besser, die Entitäten des Modells während der Simulation selbst entscheiden lassen, wann welche Ereignisse eintreten, wie es bei Ereignissteuerung der Fall ist. Solche Modelle und die ihnen zugrundeliegenden Systeme sind in verschiedenen Anwendungsbereichen zu finden: so z. B. Dominanz und Konkurrenz in der Ökologie, wo die Vertreibung eines Individuums aus einem Habitat eine Reihe weiterer Dominanztests und Vertreibungen in benachbarten Habitaten nach sich ziehen kann (Grimm und Railsback 2005, S. 112), chemische Reaktionen (Barnes und Chu 2010, S. 26ff), Auftragseingänge an der Börse (Jacobs et al. 2004; Daniel 2006; Boer et al. 2007) oder die komplexen, voneinander abhängigen Prozesse der Auftragsabwicklung in dem in Kapitel 6 beschriebenen Kurierdienst.

Die hier definierte agentenorientierte Weltsicht erlaubt die natürliche Abbildung solcher Vorgänge, indem sie einen ereignisgesteuerten Zeitfortschritt voraussetzt. Beim ereignisgesteuerten Ansatz wird der jeweils nächste Ereigniszeitpunkt benötigt, um die Simulationzeit entsprechend fortzuschreiben. Das Eintreten von Ereignissen muss demnach rechtzeitig im voraus bestimmt werden. Da zeitverbrauchende Aktivitäten auf Start- und Ende-Ereignisse abzubilden sind, muss für jede solche Aktivität spätestens zum Start der jeweilige Endzeitpunkt bekannt sein. Dies verlangt von einem Modellierer, explizit die Dauer von Aktionen festzulegen. Dazu gibt es die in Abschnitt 2.1.3.3 aufgeführten Möglichkeiten, deren Eignung vom jeweiligen Modellkontext abhängt: Aktionen können mit einer konstanten Dauer versehen werden, die Dauer kann anhand empirischer Daten oder (ggf. empirisch bestimmter) stochastischer Verteilungen ermittelt werden oder sie kann zustands-, zeit- oder aufwandsabhängig berechnet werden. Hierbei kann der Modellierer durch ein Simulationswerkzeug unterstützt werden, welches beispielsweise entsprechende Verteilungen oder die automatische Berechnung der Dauer einer Bewegung in der räumlichen Umgebung zur Verfügung stellt.

Ein Vorteil dieses Modellierungsstils ist, dass sich die Reihenfolge der Aktionsausführung generell automatisch ergibt. Bei Verwendung einer reellwertigen Zeitbasis können die Ereignis-Zeitpunkte exakt bestimmt werden, so dass gleichzeitig eintretende Ereignisse eine Ausnahme darstellen, anstatt wie bei der Zeitsteuerung die Regel zu sein. Da die Simulationszeit von Ereignis zu Ereignis voranschreitet und die Intervalle zwischen Ereignissen von den Abläufen im modellierten System vorgegeben werden, können Agenten und Umweltprozesse darüber hinaus auf unterschiedlichen Zeitskalen agieren.

Die agentenorientierte Weltsicht lässt sich wie folgt zusammenfassen:

- Das zu modellierende System wird als Multiagentensystem aufgefasst, d. h. die aktiven Systemelemente werden als Agenten modelliert, welche in einer gemeinsamen Umgebung interagieren. Wie bei der traditionellen diskreten Simulation wird zwischen Struktur und Verhalten des Systems unterschieden: Die Struktur ergibt sich aus den einzelnen Systemelementen und deren Beziehungen untereinander, das dynamische Verhalten aus den Handlungen der aktiven Systemelemente.

- Ein agentenorientiertes Modell bildet daher ebenfalls ein Multiagentensystem, be-

stehend aus den Agenten, deren Beziehungen und der Umgebung. Die Umgebung kann eine räumliche Ausprägung besitzen, über eine eigene Dynamik verfügen und zusätzliche (passive) Objekte wie z. B. Ressourcen enthalten.

- Agenten und ggf. die Umgebung handeln konzeptuell nebenläufig. Dies lässt sich abbilden, indem die möglichen Aktionen identifiziert und deren Dauer festgelegt wird. Die Reihenfolge ihrer (ggf. wiederholten) Ausführung ergibt sich dann automatisch. Die Simulation ist ereignisgesteuert, d. h. die Simulationssteuerung erfolgt in unregelmäßigen Zeitschritten mit asynchroner Aktualisierung der Agenten.

Im folgenden Kapitel 4 wird ein Konzept für ein Simulationswerkzeug entwickelt, das die agentenorientierte Weltsicht umsetzt. Besonderer Schwerpunkt des Werkzeugs ist es, die explizite Repräsentation einer räumlichen Umgebung sowie die Bewegung von Agenten im Raum zu unterstützen. Obwohl ein Großteil der Multiagentenmodelle den räumlichen Aspekt der Umgebung explizit berücksichtigen und mobile Agenten vorsehen (Davidsson et al. 2007), sind dies Bereiche, in denen bestehende Werkzeuge für agentenbasierte Modellierung Schwächen aufweisen.

4. Konzeption eines Frameworks für agentenorientierte Modellierung und Simulation mit expliziter Raumrepräsentation

Alles sollte so einfach wie möglich gemacht werden, aber nicht einfacher.
– Albert Einstein

In diesem Kapitel wird ein Konzept entwickelt für ein Werkzeug zur Modellierung und Simulation anhand der in Abschnitt 3.2.5 eingeführten agentenorientierten Weltsicht. Ein Schwerpunkt ist daher die Unterstützung des ereignisgesteuerten Zeitfortschritts. Dieser verlangt zwar nach einer aufwendigeren Simulationsinfrastruktur als eine reine Zeitsteuerung, ist jedoch wie in den Abschnitten 2.1.2 und 3.2.3 dargelegt flexibler und allgemeiner verwendbar. Um den Aufwand für Modellierer gering zu halten, macht das konzipierte Simulationswerkzeug die Ereignissteuerung der Agenten transparent.

Ein weiterer Schwerpunkt liegt auf der Unterstützung der expliziten Raumrepräsentation. Dies bedeutet nicht nur, geeignete Konstrukte zur Raummodellierung zu entwickeln, sondern auch, die für situierte Agenten benötigte Funktionalität bereitzustellen, wie beispielsweise Bewegung im Raum, Wahrnehmung der Umgebung oder die Manipulation von im Raum vorhandenen Objekten.

Eine Analyse existierender ABMS-Werkzeuge zeigt, dass Toolkits, die (auch) einen ereignisgesteuerten Zeitfortschritt unterstützen wie Repast, Swarm oder James II (Himmelspach und Uhrmacher 2007), keine oder kaum Funktionalität für die Implementation von Agenten und Umgebung bieten, während Toolkits, die solche Funktionalität zur Verfügung stellen wie NetLogo, SeSAm (Klügl 2001; Klügl et al. 2006), LEE (Menczer und Belew 1993) oder Sim_Agent (Sloman und Poli 1996), die Modellausführung auf Zeitsteuerung beschränken. Daraus lässt sich schließen,

> [...] there is a gap in the current design space for a toolkit which both provides/prescribes some structure for implementing agents but also provides a full Discrete Event scheduling implementation for the model's execution. (Theodoropoulos et al. 2009, S.85)

Das in dieser Arbeit konzipierte und implementierte Framework für agentenorientierte Modellierung und Simulation (FAMOS) schließt diese Lücke.

4.1. Anforderungsdefinition

Ein Framework für agentenorientierte Modellierung und Simulation muss sowohl die Erstellung des Computermodells als auch die Durchführung von Experimenten mit diesem Modell unterstützen, d. h. die Phasen Implementation und Simulation des Modellbildungszyklus (Page 1991, S. 11). Spezielle Anforderungen ergeben sich daher zum einen

aus der Struktur des Modells, welches ein Multiagentensystem bildet, und zum anderen aus der Simulation seines dynamischen Verhaltens, welche ereignisgesteuert erfolgt. Wie die paradigmatischen Simulationssprachen für die klassischen Weltsichten muss das hier vorgestellte Werkzeug die Grundbausteine für die Umsetzung der agentenorientierten Weltsicht zur Verfügung stellen.

Im folgenden werden die Anforderungen gemäß des Modellbildungszyklus in die Kategorien Modellierung, Simulation und Ergebnisanalyse bzw. Validierung unterschieden. Dabei werden nur diejenigen Anforderungen berücksichtigt, die sich aus der agentenorientierten Weltsicht ergeben. Allgemeine Anforderungen an Simulationsframeworks wie z. B. klare Struktur und vollständige Dokumentation mit Beispielen (vgl. Page et al. 2005b, S. 240f; Page et al. 2000, S. 33), gelten selbstverständlich ebenfalls für das zu entwickelnde Werkzeug.

4.1.1. Unterstützung der Modellierung

Die agentenorientierte Weltsicht ist insbesondere geeignet, um komplexe Systeme zu modellieren, die inhärent von Ereignissen gesteuert werden. Da diese in so verschiedenen Anwendungsgebieten wie Ökologie, Chemie und Wirtschaft zu finden sind, muss das Werkzeug explizit so konzipiert werden, dass es unabhängig von einer spezifischen Anwendungsdomäne ist. Zwar müssen Werkzeuge, welche mit Blick auf einen besonderen Modelltyp entworfen wurden, nicht unbedingt auf diesen Typ beschränkt sein, wie das Beispiel von NetLogo zeigt (Railsback et al. 2006), doch kann dies die Implementation anderer Modelle erschweren.

Domänen-Unabhängigkeit bedeutet, dass der Modellierer weder auf eine bestimmte Agenten-Architektur (siehe Abschnitt 2.3.4) festgelegt wird noch auf ein bestimmtes Raummodell wie z. B. die in der agentenbasierten Simulation immer noch vorherrschenden zweidimensionalen Gitter. Diese sind für viele Anwendungen nicht adäquat, da die abzubildenden räumlichen Strukturen selten mit regelmäßigen Gittern übereinstimmen, die Größe der Zellen Skalierungsprobleme nach sich ziehen kann und die Gitterstruktur die lokalen Interaktionsmöglichkeiten bestimmt (Bithell und Macmillan 2007).

Jedes agentenorientierte Modell bildet ein Multiagentensystem, d. h. eine Menge von Agenten, die in einer gemeinsamen Umgebung agieren (vgl. Abschnitt 3.2.5). Unterstützung der Modellierung bedeutet daher im Fall der agentenorientierten Simulation Unterstützung bei der Konstruktion eines Multiagentensystems. Der Anwender ist zu unterstützen bei der Modellierung (a) der Agenten, (b) der Umgebung und (c) der Beziehungen zwischen den Systemelementen, welche sich in Interaktionen zwischen Agenten (Kommunikation), Interaktionen zwischen Agenten und Umgebung (Wahrnehmung und Handlungen) und Organisationsstrukturen unterteilen lassen (siehe die Definition von MAS in Abschnitt 2.3.2).

Das Spektrum in agentenbasierten Modellen reicht von einfachen, reaktiven Agenten bis zu komplexen, adaptiven Agenten, welche ihr Verhalten ändern können. Dies ist ein weiterer Grund, das Werkzeug nicht auf eine Agenten-Architektur zu beschränken. Es könnte stattdessen austauschbare Verhaltensbausteine anbieten. Darüber hinaus kann der Anwender bei der Verhaltensbeschreibung der Agenten unterstützt werden, indem das Simulationswerkzeug Aktionsprimitive für die modell-unabhängigen Handlungen anbie-

tet, wie z. B. Wahrnehmen der Umwelt, Senden und Empfangen von Nachrichten (Kommunikation), Bewegen im Raum und Ändern von Attributen oder Objekten in der Umwelt.

Da organisatorische Beziehungen und Strukturen einen großen Einfluss auf das Verhalten des gesamten Systems ausüben können, ist es sinnvoll, sie explizit zu repräsentieren (vgl. Ferber und Gutknecht 1998, S. 129 und Jennings 2001, S. 38). Übertragen auf den Entwurf von Multiagentenmodellen bedeutet das, ein Simulationswerkzeug muss die flexible Definition und Verwaltung von Organisationsstrukturen unterstützen. Diese können variieren von Kooperationsbeziehungen unter Gleichrangigen (*peer-to-peer*) über Abhängigkeitsbeziehungen zwischen Kunde und Dienstanbieter (*client/server*) bis hin zu hierarchischen Beziehungen. Solche Hierarchien müssen nicht ausschließlich Führungskompetenzen ausdrücken, sondern können allgemeiner als Strukturprinzip aufgefasst werden: Hierarchie im Sinne des rekursiven Aufbaus eines Systems aus Teil-Systemen ist ein charakteristisches Merkmal vieler komplexer Systeme (Simon 1996, S. 184) und betrifft damit auch Multiagentensysteme und -modelle.

Im Unterschied zu anderen Multiagentensystemen, in denen die Umgebung als vorhanden vorausgesetzt werden kann, muss sie als Teil eines agentenbasierten Modells ausdrücklich mit entwickelt werden. Zwar ist die Art der Umgebung letztendlich abhängig vom jeweiligen Modell, doch kann ein Simulationswerkzeug die modell-unabhängigen Aspekte wie die zugrundeliegende Struktur und Basis-Bausteine vorgeben. Die Umgebung in MAS – und damit in Multiagentenmodellen – lässt sich aus folgenden Komponenten aufbauen:

- einer Infrastruktur für die Kommunikation der Agenten (Huhns und Stephens 1999, S. 81f);
- einem Raummodell, falls der Raum explizit repräsentiert wird (Ferber 1999, S. 11);
- einer Menge von (passiven) Objekten, die im Raum situiert sind (Ferber 1999, S. 11);
- einer eigenen Dynamik in Form von Prozessen, die nicht als Agenten aufgefasst werden (Russell und Norvig 1995, S. 47, Ferber und Müller 1996, S. 73).

Explizite Raummodellierung ist zentral für Systeme, in denen die Lokalität des Agentenverhaltens (begrenzter Wahrnehmungs- und Aktionsradius) die Systemdynamik beeinflusst. Der Raum stellt in diesen Fällen nicht einfach ein neutrales Medium dar, in welchem die Agenten interagieren, sondern spielt eine aktive und komplexe Rolle in deren Interaktionen. Ohne den Raum explizit zu modellieren, lässt sich diese Rolle nur schwer – wenn überhaupt – abbilden (Parunak et al. 2000, S. 28). Einer flexiblen Unterstützung der Raummodellierung kommt daher große Bedeutung zu.

Die Eigenschaften der Umgebung (siehe Abschnitt 2.3.1, Seite 35f), die in Agenten-Systemen im Allgemeinen nur berücksichtigt werden durch die Anforderungen, die sie an das Agenten-Design stellen, sind in diesem Fall auch für den Entwurf der Umgebung relevant. Ein Simulationswerkzeug sollte daher die flexible Kombination von Umgebungseigenschaften unterstützen, um eine möglichst umfassende Klasse von Umgebungen abbilden zu können.

Für die agentenorientierte Simulation mit expliziter Raumrepräsentation scheinen insbesondere die Eigenschaften interessant, die in Bezug zum Raummodell stehen und den Zugriff der Agenten auf den Raum beeinflussen. So kann sowohl der Wahrnehmungsbereich eines Agenten lokal beschränkt sein (unzugängliche Umgebung), als auch sein Aktionsradius (Grad der Kontrollierbarkeit). Die Umgebung kann darüber hinaus über eine eigene Dynamik verfügen, so dass Zustandsänderungen möglich sind, die nicht auf Handlungen von Agenten zurückzuführen sind (dynamische Umgebung).

4.1.2. Unterstützung der Simulation

Neben der Modellbildung muss ein Simulationswerkzeug die Simulation im engeren Sinne unterstützen, d. h. die Durchführung von Experimenten mit einem Modell. Das Werkzeug hat dafür die benötigte Simulationsinfrastruktur wie Simulationsuhr und Ablaufsteuerung zur Verfügung zu stellen, welche die (konzeptuelle) Nebenläufigkeit der aktiven Entitäten garantiert (Kreutzer 1986, S. 39f). Idealerweise erleichtert eine dedizierte Experimentierumgebung die Spezifikation und Ausführung von komplexen Experimenten.

Die agentenorientierte Weltsicht setzt einen ereignisgesteuerten Zeitfortschritt voraus, um das dynamische Verhalten eines Modells über die Zeit zu ermitteln. Simulationszeit schreitet von einem Ereigniszeitpunkt zum nächsten voran und zu jedem Zeitpunkt werden nur diejenigen Entitäten aktualisiert, die von dem aktuellen Ereignis betroffen sind. Aus der Verwendung der Ereignissteuerung folgt, dass das Simulationsframework es ermöglichen muss, (a) die Dauer von Aktivitäten der Agenten und Umgebung vor ihrem Ende zu bestimmen, (b) jeden Agenten sofort auf Veränderungen in der Umgebung reagieren zu lassen und (c) die Aktualisierung des Umgebungszustands trotz asynchroner Agenten-Aktualisierung effizient zu halten.

Wie in Abschnitt 2.1.3 dargelegt, existieren für die Bestimmung der Aktionsdauer mehrere Möglichkeiten, die sich in Aufwand und Eignung für bestimmte Modelle unterscheiden. Damit ein Modellierer hier umfassend unterstützt wird, sollte ein Simulationswerkzeug entsprechende Konstrukte anbieten wie beispielsweise stochastische Verteilungen oder die automatische Berechnung bestimmter Aktionsdauern.

Um auf Veränderungen in seinem Wahrnehmungsbereich sofort reagieren zu können, beobachtet ein Agent konzeptuell ständig die Umwelt. In der ereignisgesteuerten Simulation kann er jedoch nur in seinen (zeitverzugslosen) aktiven Phasen die Umgebung wahrnehmen, da Zeitverbrauch als programmtechnisch passive Phase realisiert wird. Falls während einer solchen Phase eine Veränderung in der Umwelt eintritt, muss der Agent dies sofort wahrnehmen und ggf. darauf reagieren können. Ein Simulationsframework für agentenorientierte Simulation muss für dieses Problem eine Lösung finden, ohne auf Zeitsteuerung zurückzugreifen.

Ein weiteres Problem ergibt sich aus der asynchronen Aktualisierung der aktiven Entitäten. Vorausgesetzt wird dabei, dass der Zustand des gesamten Systems zwischen Ereignissen konstant bleibt. Dies ist in der ereignisgesteuerten Simulation dadurch gewährleistet, dass die mit einem Ereignis verbundenen Zustandsänderungen per Definition zeitverzugslos ablaufen, während zeitverbrauchende Handlungen auf eine Folge von Ereignissen abgebildet werden. Für Modelle mit expliziter Raumrepräsentation stellt die Bewe-

gung im Raum die wesentliche zeitverbrauchende Handlung dar, die von einem Simulationswerkzeug unterstützt werden muss. Bewegung stellt insofern eine Herausforderung an die Ereignissteuerung dar, als sich die Position und damit der Zustand einer sich bewegenden Entität ständig verändert. Üblicherweise wird daher für die Simulation von mobilen Entitäten ein zeitgesteuerter Ansatz verwendet (Buss und Sánchez 2005, S. 992). Für die Umsetzung der agentenorientierten Weltsicht wird dagegen eine geeignete Abbildung auf die Ereignissteuerung benötigt.

4.1.3. Unterstützung der Analyse

Das in dieser Arbeit konzipierte Framework für agentenorientierte Modellierung und Simulation unterstützt die Erstellung von Multiagentenmodellen mit expliziter Raumrepräsentation und die Durchführung von Experimenten mit einem solchen Modell. Sein Funktionsumfang konzentriert sich auf die Phasen Implementation und Simulation des Modellbildungszyklus; Ergebnisanalyse und Validierung werden daher nur rudimentär unterstützt. Die Generierung von Simulationsergebnissen muss selbstverständlich gewährleistet sein. Eine Minimalanforderung an ein Simulationswerkzeug ist in diesem Zusammenhang die Aufzeichnung von Daten über einen Simulationslauf, so dass eine spätere Analyse der Ergebnisse mit Standard-Werkzeugen wie Statistikpaketen oder Tabellenkalkulationsprogrammen möglich ist. Für die Auswertung agentenbasierter Simulationen werden oft Daten über individuelle Merkmale einzelner Agenten benötigt. Die in der diskreten Simulation üblichen statistischen Datensammelobjekte für die Berechnung aggregierter Kennzahlen müssen daher geeignet erweitert werden, um die Aufzeichnung von Daten auf der Ebene einzelner Agenten zu ermöglichen.

Eine weitere Anforderung ergibt sich aus der besonderen Unterstützung der Raummodellierung. Um die räumliche Verteilung, Bewegung und Interaktion von Agenten zu untersuchen, ist eine animierte Darstellung des Agentenverhaltens in der räumlichen Umgebung, d. h. eine Visualisierung der Agenten in Raum und Zeit, hilfreich. Diese kann entweder simulationsbegleitend oder nachträglich anhand während des Simulationslaufs aufgezeichneter Daten erfolgen (Page et al. 2005b, S. 257). Animation ist in erster Linie ein Mittel zur Kommunikation über das Modell. Nicht-Experten kann so das Wesen des Modells auf einfache Weise verdeutlicht werden, während Domänenexperten tiefere Einblicke in dessen Abläufe gewinnen können (Law und Kelton 2000, S. 211).

Eine Animation der im Modell ablaufenden Prozesse hilft darüber hinaus bei der Verifikation und Validierung des Modells. Da in agentenbasierten Modellen Prozesse auf der Mikro-Ebene das Verhalten des Systems auf der Makro-Ebene erzeugen, kann eine Visualisierung der Agenten in ihrer Umgebung den Anwender bei der Erkennung von (in-)validen Mustern unterstützen. Dies ist gerade für Modelle mit einer expliziten räumlichen Ausprägung wichtig. Die Animation der Bewegung individueller Agenten im Raum hat beispielsweise bei vielen ökologischen Multiagentenmodellen dabei geholfen, Fehler in der Modell-Implementation, im konzeptuellen Modell oder den Eingabedaten rasch aufzuspüren (Ropella et al. 2002).

4.2. Designentscheidungen

Das in dieser Arbeit konzipierte Framework für agentenorientierte Modellierung und Simulation muss die im vorausgehenden Abschnitt 4.1 definierten Anforderungen umsetzen. Im folgenden werden generelle Designentscheidungen vorgestellt, welche die Entwicklung des Werkzeugs beeinflussen und bereits einen Teil der Anforderungen abdecken. Weitere Anforderungen an die Unterstützung der Modellbildung und Simulation werden detailliert in den entsprechenden Teilen des Konzepts behandelt (siehe die Abschnitte 4.3 bis 4.6), während die Minimalanforderungen an die Unterstützung der Analyse direkt umgesetzt werden (vgl. Abschnitt 5.2.5).

4.2.1. Integration eines bestehenden Simulationsframeworks

Für die Bereitstellung der Simulationsinfrastruktur wird die naheliegende Lösung gewählt, auf vorhandener Software für diskrete Simulation aufzubauen, so dass deren Infrastruktur direkt verwendet werden kann. Die Modellierung der Zeit in Multiagentenmodellen ist dann auf die Zeitrepräsentation des zugrundeliegenden Simulationswerkzeugs abzustimmen. Im Falle der von der agentenorientierten Weltsicht geforderten Ereignissteuerung bedeutet dies konkret, das dynamische Verhalten von Agenten und Umgebung auf Ereignisse abzubilden. Dies wird detailliert in Abschnitt 4.6 beschrieben. Die Anforderung, einen ereignisgesteuerten Zeitfortschritt zu realisieren, wird damit umgesetzt.

Integration eines bestehenden Simulationswerkzeugs setzt die Verwendung erweiterbarer Simulationssoftware voraus, wie es beispielsweise Simulationspakete sind, welche allgemeine Programmiersprachen um simulationsspezifische Konstrukte ergänzen (Page 1991, S. 159f). Das in dieser Arbeit konzipierte Werkzeug baut auf einem objektorientierten Simulationspaket für ereignis-diskrete Simulation auf, welches sowohl die ereignisorientierte als auch die prozessorientierte Weltsicht unterstützt (vgl. Abschnitt 5.1). Entitäten und Prozesse dieser klassischen Weltsichten können für die Modellierung von Objekten und Dynamik in der Umgebung übernommen werden. Sie bieten das richtige Abstraktionsniveau, da Umweltprozesse konzeptuell nicht mit Agenten gleichzusetzen sind. Hierauf wird in Abschnitt 4.4 näher eingegangen.

Die Integration eines bestehenden Simulationsframeworks erlaubt darüber hinaus auch die Wiederverwendung weiterer Komponenten wie stochastische Verteilungen, Warteschlangen und statistische Datensammlungsobjekte. Erstere können beispielsweise in der Verhaltensmodellierung der Agenten verwendet werden, um die Dauer von Handlungen festzulegen (siehe Abschnitt 4.6.1).

Eine eigene Experimentierumgebung wird nicht realisiert. Hier kann auf existierende Werkzeuge zurückgegriffen werden, die am Fachbereich Informatik der Universität Hamburg in den letzten Jahren entstanden und auf das integrierte Simulationsframework abgestimmt sind. Sowohl das im Rahmen der Dissertation von Björn Gehlsen (2004) entwickelte Optimierungssystem DISMO als auch das im Rahmen der Dissertation von Ralf Bachmann (2003) entwickelte System CoSim stellen eine Experimentierumgebung zur Verfügung. Während erstere die Verteilung von Simulationsläufen auf verschiedene Rechner erlaubt, unterstützt letztere die verteilte Ausführung eines aus Komponenten zusammengesetzten Modells. Aufbauend auf Konzepten beider Dissertationen und früherer

Arbeiten zu Experimentierumgebungen wie z. B. (Wittmann 1993) wurde darüber hinaus eine generische Experimentierumgebung auf Basis von Eclipse entwickelt (Czogalla et al. 2006), welche u. a. im Zuge der nachträglichen Validierung des in Kapitel 6 beschriebenen Kurierdienstmodells zum Einsatz kam (Denz 2013, Kap. 8).

4.2.2. Typ des Werkzeugs

Der Aufbau auf einem Simulationsframework legt nahe, das zu entwickelnde Werkzeug ebenfalls als Framework zu konzipieren. Es fällt damit in die Kategorie der Simulationspakete. Dies ist der Ansatz, den die meisten ABMS-Toolkits verfolgen (Railsback et al. 2006, S. 610); bekannte Beispiele sind Swarm, Repast und Mason. Wesentlicher Vorteil eines Simulationsframeworks ist dessen Flexibilität: Es bietet eine standardisierte Softwarearchitektur und die benötigte Simulationsfunktionalität ohne die Klasse der Modelle einzuschränken, die mit Hilfe des Frameworks implementiert werden kann.

Die Anwendung eines Simulationspakets setzt relativ detaillierte Programmierkenntnisse voraus; Zielgruppe sind somit Modellierer mit Programmiererfahrung.

4.2.3. Realisierung der Umgebung

Grundsätzlich ließe sich die Umgebung eines Multiagentenmodells als Agent implementieren. Für diesen Ansatz spricht, dass er einfach zu realisieren ist, da keine besonderen Konstrukte für die Modellierung der Umgebung bereitzustellen sind (Klügl 2001, S. 93). Die umgebungsinterne Dynamik kann als Verhalten eines Agenten ausgedrückt werden. Darüber hinaus besteht ein solches Modell ausschließlich aus gleichartigen Komponenten – den eigentlichen Agenten und dem Umgebungsagenten –, die auf gleiche Weise kommunizieren und von der Simulationsinfrastruktur ebenfalls gleich behandelt werden können.

Konzeptuell ist die Umgebung eines Multiagentensystems jedoch nicht als Agent aufzufassen, sondern bildet das Komplement zu den Agenten; das gemeinsame Substrat, in dem die Agenten handeln. Auch ist die Interaktion zwischen Agenten und Umgebung nicht mit der Kommunikation zwischen Agenten gleichzusetzen: Wahrnehmung über Sensoren und Handlung über Effektoren ist prinzipiell nicht auf den Austausch von Nachrichten beschränkt. Ein Modellierer sollte die Umgebung daher in anderer Form repräsentieren können. In dem in dieser Arbeit entwickelten Werkzeug für agentenorientierte Simulation wird die Umgebung daher als eigenständiges Objekt realisiert. Durch Integration (mindestens einer) der klassischen Weltsichten der diskreten Simulation stehen in diesem Werkzeug geeignete Konstrukte zur Verfügung, so dass die Umgebung nicht als Agent implementiert werden muss, um mit einem dynamischen Verhalten ausgestattet zu werden. Dies wird ausführlicher in Abschnitt 4.4.1 beschrieben.

4.2.4. Verzicht auf verteilte Simulation

Wie in Abschnitt 3.2.3 diskutiert, kann gegen die Ereignissteuerung der Einwand erhoben werden, die benötigte Ablaufsteuerung widerspreche dem grundlegenden Prinzip, dass Multiagentensysteme keiner zentralen Kontrollinstanz unterliegen (Davidsson 2001,

S. 100). Bei der verteilten Simulation eines ereignisgesteuerten Modells ersetzen Synchronisationsmechanismen die zentrale Ablaufsteuerung. Dies kommt dem MAS-Prinzip näher, ist jedoch wiederum aufwendiger zu realisieren als eine verteilte Implementation zeitgesteuerter Modelle, da Zeitanomalien, d. h. Verletzungen der Kausalitätsregel (siehe Abschnitt 2.1.2), verhindert werden müssen. Da sich die Lastverteilung wesentlich schwieriger gestaltet als bei einer Zeitsteuerung, wird zudem oft nur eine unwesentliche Beschleunigung der Simulation erreicht (vgl. z. B. Sonnenschein und Lorek 1995 für verteilte und Boulaire et al. 2013 für parallele Simulation). Die Ausführung auf einem Rechner gewährleistet außerdem eine bessere Reproduzierbarkeit von Simulationsläufen, da die unterschiedliche Netzlast keinen Einfluss auf das Ergebnis hat (Polani und Uthmann 2000).

Darüber hinaus sind Computer inzwischen so leistungsfähig, dass eine Verteilung aus Skalierungsgründen nicht mehr unbedingt notwendig wird. Als Beispiel sei ein Modell von Robert Axtell genannt, welches die Evolution von Firmengrößen auf Basis der gesamten US-Ökonomie abbildet und 120 Millionen Agenten enthält. Es wird bewusst auf nur einem Rechner mit entsprechend großem Arbeitsspeicher (256 GB) und Mehrkernprozessoren ausgeführt, da sich die bei einer Verteilung benötigte Kommunikation zwischen Rechnern als zu langsam herausgestellt hat (Axtell 2013).

Dies und die Tatsache, dass Ereignissteuerung zumindest die quasi-parallele Ausführung nebenläufiger Entitäten auf einem Prozessor gewährleistet, spricht dafür, auf eine Verteilung zu verzichten.

4.3. Repräsentation des Raumes

Wie in der Anforderungsdefinition (siehe Abschnitt 4.1) spezifiziert, muss ein Simulationswerkzeug, das nicht auf einen speziellen Anwendungsbereich festgelegt ist, die Modellierung des Raums möglichst flexibel unterstützen. Dem Raummodell kommt insofern große Bedeutung zu, als es die räumlichen Eigenschaften (Position, Form) von Agenten und Objekten sowie die raumbezogenen Aktionen der Agenten wie Wahrnehmung und Bewegung bestimmt.

Das im folgenden vorgestellte Raummodell versucht Flexibilität mit Einfachheit zu verbinden, um mit wenigen klaren Konzepten möglichst viele Anwendungsfälle abzudecken. Der Schwerpunkt liegt dabei – in Übereinstimmung mit der in Abschnitt 4.6 beschriebenen diskreten Zeitrepräsentation – auf einer Diskretisierung des Raums. Die in der agentenbasierten Simulation hier vorherrschenden zweidimensionalen regelmäßigen Gitter genügen zwar dem Kriterium der Einfachheit, sowohl im Konzept als auch in der Implementation. Sie sind jedoch als alleiniges Raummodell unzureichend, da sie beispielsweise keine angemessene Modellierung von (Verkehrs-)Netzen zulassen. Im Gegensatz dazu bilden Graphen eine natürliche Repräsentationsform beliebiger Netze und eignen sich auch zur Abbildung anderer komplexer Raumstrukturen. Daher wurden sie als Grundlage des diskreten Raummodells gewählt (siehe Abschnitt 4.3.1).

Bewegung im Raum ist in vielen Anwendungsbereichen ein zentrales Modellelement. So stellt sie z. B. in individuenbasierten ökologischen Modellen die elementare Methode dar, mit der Individuen auf Veränderungen in der Umgebung reagieren (Lamberson

2002), und kann allgemein als eine Form der Adaption in komplexen adaptiven Systemen gelten (Macy und Willer 2002, S. 146). Die überwiegende Mehrheit von Multiagentenmodellen mit expliziter Raumrepräsentation verwendet mobile Agenten (Davidsson et al. 2007). Da die Art der Bewegung auf das Raummodell abzustimmen ist, muss ein allgemeines Simulationswerkzeug auch hier flexible Unterstützung anbieten. Dies macht einen weiteren Schwerpunkt des in dieser Arbeit entwickelten Konzepts aus (Abschnitt 4.3.2).

4.3.1. Abstraktes Raummodell

Analog zur Repräsentation der Zeit in der Simulation kann auch der Raum als kontinuierlich oder diskret betrachtet werden. Im Unterschied zur Zeit wird sich diese Betrachtung bei der Modellierung realer Systeme in der Regel auf mehr als eine Dimension beziehen: Üblich ist die Berücksichtigung von zwei bzw. drei räumlichen Dimensionen, obwohl theoretisch n-dimensionale Raummodelle möglich sind. Um sowohl den kontinuierlichen als auch den diskreten Fall in beliebigen Dimensionen zu unterstützen, wird im folgenden ein Bezugssystem mit Koordinaten aus dem n-dimensionalen Punktraum $\mathbb{R}^n$ gewählt. Das Koordinatensystem sei darüber hinaus kartesisch, da dies geometrische Berechnungen vereinfacht.

Über dieses Bezugssystem lässt sich jedem Objekt mindestens ein Punkt im Raum zuordnen: seine Position. Die Genauigkeit der räumlichen Position ist zwar im diskreten Fall prinzipiell auf das den Punkt enthaltende Raumelement vergröbert, doch kann jedes Raumelement wiederum auf einen eindeutigen Referenzpunkt abgebildet werden. Im kontinuierlichen Raum muss zusätzlich die räumliche Ausdehnung der Objekte erfasst werden und ggf. ihre Ausrichtung, während im diskreten Raum die Form der Objekte implizit durch die Art der Diskretisierung bestimmt wird. Um sowohl kontinuierliche als auch diskrete Repräsentationsformen des Raums zuzulassen, wird als gemeinsame Schnittstelle ein abstraktes Raummodell eingeführt. Das abstrakte Raummodell setzt als allen konkreten Raummodellen gemeinsame Eigenschaften nur ein n-dimensionales Bezugssystem und auf Punkte zurückführbare räumliche Positionen voraus.

Die Positionen werden explizit als Objekte repräsentiert mit der Fähigkeit, einen Referenzpunkt zu liefern. Der Verweis auf das zugehörige Raumelement wie z. B. die Zelle eines Gitters, den Knoten eines Graphen oder einen Punkt im kontinuierlichen Raum wird im Positionsobjekt gekapselt. Agenten und situierte Objekte verwenden solche Positionsobjekte, um ihre räumliche Position zu referenzieren. Gegenüber der naheliegenden Lösung, Positionen mit Punkten gleichzusetzen und nicht gesondert zu repräsentieren, bietet dieser Ansatz zwei Vorteile. Zum einen ist ein effizienter Zugriff auf das jeweilige Raummodell gewährleistet, da Umrechnungen von Referenzpunkten auf Raumelemente entfallen. Zum anderen sind situierte Objekte und Agenten auf diese Weise von der konkreten Ausprägung des Raummodells weitgehend unabhängig, so dass das Raummodell verändert bzw. ausgetauscht werden kann, ohne zusätzliche Änderungen am Multiagentenmodell zu erfordern. Bildlich gesprochen wird mit den Positionsobjekten eine Schicht zwischen Raummodell und Agenten geschaffen, die den Zugriff auf die räumlichen Strukturen filtert (vgl. Abbildung 4.1).

Im folgenden wird ein detailliertes Konzept für ein diskretes Raummodell entwickelt, das auf gerichteten Graphen basiert. Die grundlegenden Ideen dieses Konzepts wurden

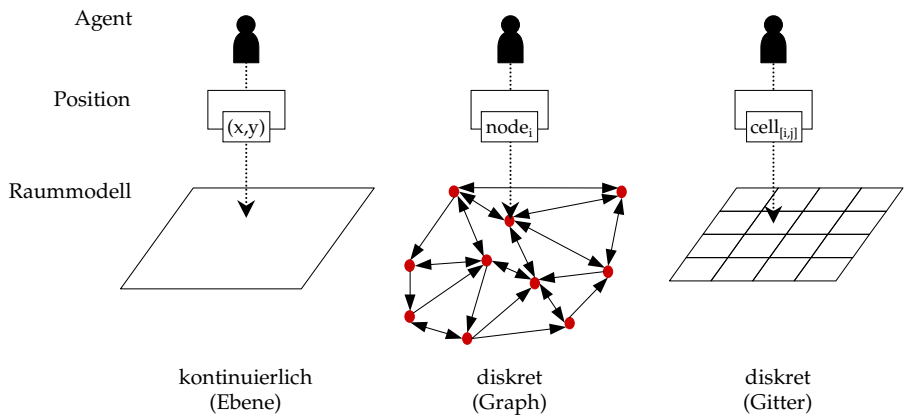

Abbildung 4.1.: Positionierung von Agenten im Raum, dargestellt am Beispiel verschiedener Raummodelle. Die Positionen kapseln den Verweis auf das konkrete Raumelement, so dass die Agenten unabhängig werden von der jeweiligen Realisierung des Raummodells.

bereits in (Meyer 2001) eingeführt. Ein kontinuierliches Raummodell wird im Rahmen dieser Arbeit nicht weiter verfolgt[1]. Dies ermöglicht Vereinfachungen bezüglich der geometrischen Repräsentation von Objekten und Agenten, die im kontinuierlichen Fall explizit mit zu modellieren wären. Darüber hinaus wird für die in Kapitel 6 beschriebene Simulationsstudie die besondere Unterstützung von Verkehrsnetzen benötigt. Die Modellierung solcher räumlichen Strukturen, welche die freie Bewegung einschränken, ist unverhältnismäßig aufwendig, wenn der Raum als n-dimensionales Kontinuum betrachtet wird: Für ein Straßennetz müssten beispielsweise alle nicht zum Netz gehörenden Bereiche explizit als Hindernisse im Raum abgebildet werden, um eine Bewegung außerhalb von Straßen zu verhindern.[2] Eine diskrete, graph-basierte Raumrepräsentation ist dagegen für die meisten Anwendungsfälle angemessen und kann – bei entsprechend fein gewählter Auflösung – den kontinuierlichen Raum hinreichend approximieren.

4.3.1.1. Gerichtete Graphen als Grundlage des diskreten Raummodells

Bei der Diskretisierung wird der Raum in einzelne Elemente zerlegt. Üblich ist eine Untergliederung in zusammenhängende Elemente, die das räumliche Kontinuum vollständig abdecken (*Tesselation*, siehe Abschnitt 2.2.2). Regelmäßige Tesselationen oder Gitter sind für die räumliche Modellierung in der agentenbasierten Simulation besonders interessant, weil der Abstand zwischen aneinandergrenzenden Zellen (genauer: den Zell-Mittelpunkten) identisch ist, so dass z. B. bei konstanter Geschwindigkeit die Dauer für

[1] Durch das geforderte n-dimensionale kartesische Bezugssystem und die von einem konkreten Raummodell abstrahierenden Positionen lässt sich das hier vorgestellte Konzept gut um ein kontinuierliches Raummodell erweitern. Im Rahmen einer Diplomarbeit (Czogalla und Matzen 2003) wurde dies auch umgesetzt.

[2] Dieses Problem gilt ebenso für Gitter: Batty und Jiang (2000, S. 59) verwenden beispielsweise eine spezielle *Constraint*-Variable $C(x_i y_i)$, um nicht zum Straßennetz gehörige Zellen zu blockieren.

die Bewegung von Zelle zu Zelle immer gleich bleibt.

Im Gegensatz zu Tesselationen überdecken Graphen prinzipiell den Raum nicht vollständig; rein mathematisch betrachtet, besitzen sie nicht einmal geometrische Eigenschaften, sondern repräsentieren allenfalls topologische Beziehungen. Ein Graph ist formal definiert als ein Tupel $G = (V, E)$ aus einer endlichen Menge V von Knoten (engl. *vertices*) und einer Menge E von Kanten (engl. *edges*). In ungerichteten Graphen ist die Kantenmenge eine Teilmenge aller zwei-elementigen Teilmengen von V, d. h. die Kanten sind ungeordnete Paare $\{v, w\}$. In gerichteten Graphen gilt dagegen $E \subseteq V \times V$, d. h. die Kanten sind geordnete Paare (v, w).[3] Die Kanten können beliebige Beziehungen zwischen den Knoten repräsentieren und bestimmen immer die Nachbarschaftsrelation: Zwei Knoten v und w sind benachbart (*adjazent*), wenn im Graphen eine ungerichtete Kante $\{v, w\}$ existiert. Diese Definition ist auf gerichtete Graphen übertragbar: v und w sind benachbart, wenn mindestens eine der gerichteten Kanten (v, w) oder (w, v) existiert. Die Nachbarschaftsrelation ist somit auch in gerichteten Graphen symmetrisch.

Ein Graph lässt sich auf einfache Weise im Raum positionieren, indem jedem Knoten ein Referenzpunkt zugeordnet wird. In solchen „geometrischen Graphen“ können Kanten als Linien (Geradenstücke oder Kurvensegmente) mit einer festen Länge betrachtet werden. Die Nachbarschaftsrelation drückt dann die topologische Nachbarschaft der geometrischen Knotenobjekte (Punkte) aus. Zur Abbildung von Verkehrsnetzen, in denen bestimmte Verbindungen nur in einer Richtung befahrbar sind, sind gerichtete Graphen besonders geeignet; ein Straßennetz mit Einbahnstraßen wird in Lehrbüchern der diskreten Mathematik (vgl. z. B. Aigner 2006, S. 115 und Biggs 2002, S. 227) als prägnantes Beispiel für einen gerichteten Graphen herangezogen. Die richtungsabhängige Benutzung von Kanten ließe sich in ungerichteten Graphen nur über zusätzliche Kantenattribute modellieren. Andererseits kann zu jedem ungerichteten Graphen ein äquivalenter gerichteter Graph konstruiert werden, indem jede ungerichtete Kante durch zwei gerichtete Kanten ersetzt wird. Daher wurden für das hier vorgestellte Konzept eines diskreten Raummodells gerichtete Graphen als allgemeinere Lösung gewählt.

Unter der plausiblen Annahme, dass nur aneinandergrenzende Elemente in Tesselationen benachbart sind, stellen Graphen auch gegenüber Tesselationen das allgemeinere diskrete Raummodell dar. Tesselationen lassen sich leicht auf Graphen abbilden, indem jedes Element auf einen Knoten und die Nachbarschaftsbeziehungen der Elemente auf die Kantenmenge abgebildet wird. Die Referenzpunkte, welche die Position der Knoten festlegen, können im Prinzip beliebig gewählt werden, solange sie eine eindeutige Zuordnung zu den Elementen erlauben. Im allgemeinen werden die Referenzpunkte daher innerhalb der zugehörigen Elemente liegen; bei einfachen konvexen Elementen eignen sich besonders die Schwerpunkte als Referenzpunkte. Da nur angrenzende Elemente als benachbart aufgefasst werden, ergibt sich immer ein in der jeweiligen Dimension kreuzungsfreier Graph; im zweidimensionalen Fall also ein planarer Graph. Graphen lassen dagegen beliebige Nachbarschaftsrelationen zwischen Knoten zu, so dass auch Überschneidungen von Kanten außerhalb von Knoten erlaubt sind.

Die Tatsache, dass zu jeder Tesselation ein dualer Graph existiert, wird als ein grundle-

[3] Mehrfachkanten sind in beiden Fällen ausgenommen; diese führen zu sogenannten Multigraphen, die hier nicht weiter betrachtet werden sollen.

gendes Prinzip des diskreten Raummodells ausgenutzt. Diese Beziehung zwischen räumlichen Partitionierungen und Graphen ist zwar naheliegend, wird jedoch selten explizit gemacht.[4] Eine Ausnahme stellt das Voronoi-Diagramm dar, eine besondere Art der Tesselation, dessen dualer Graph als Delaunay-Triangulation bekannt ist. Wie in Abschnitt 2.2.2.2 näher erläutert, ist das Voronoi-Diagramm einer gegebenen Menge von k Stützpunkten eine Unterteilung der Ebene in ebenfalls k Bereiche (Voronoi-Zellen), so dass jeder Punkt der Ebene demjenigen Stützpunkt zugeordnet wird, dem er am nächsten liegt. Der duale Graph verwendet die Stützpunkte als Knoten und verbindet diejenigen Stützpunkte durch Kanten, deren Voronoi-Zellen aneinandergrenzen. Die vorgegebenen Stützpunkte werden somit per Definition zu Referenzpunkten. Da sich Voronoi-Diagramm und Delaunay-Triangulation unter Erhalt ihrer Dualitätseigenschaft auf den n-dimensionalen Raum verallgemeinern lassen (Berg et al. 1997, S. 159), wurden sie für die Repräsentation unregelmäßiger Tesselationen in dem hier vorgestellten diskreten Raummodell ausgewählt.

Dieses Modell betrachtet den diskreten Raum als aus Raumelementen aufgebaut, die über Nachbarschaftsbeziehungen zusammenhängen. Die Raumelemente bestimmen die Möglichkeiten zur Positionierung von Objekten und Agenten im Raum, die Nachbarschaftsrelation beeinflusst Wahrnehmung und Bewegung von Agenten. Zur einheitlichen Beschreibung dieser räumlichen Struktur werden gerichtete Graphen verwendet, indem die Raumelemente auf Knoten und die Nachbarschaftsbeziehungen auf gerichtete Kanten abgebildet werden. In der Regel ist Nachbarschaft von Raumelementen eine symmetrische Eigenschaft: Wenn Element a ein Nachbar von Element b ist, gilt umgekehrt auch, dass b Nachbar von a ist. Daher sind zwei gerichtete Kanten zu erzeugen: die geordneten Paare (a, b) und (b, a). Die Verwendung gerichteter Graphen ermöglicht jedoch explizit auch asymmetrische Beziehungen zwischen Raumelementen, zwischen denen dann nur jeweils eine gerichtete Kante existiert.

In Übereinstimmung mit der Terminologie der Graphentheorie soll daher Nachbarschaft von Erreichbarkeit unterschieden werden. *Nachbarschaft* ist auch in gerichteten Graphen immer symmetrisch, *Erreichbarkeit* kann dagegen auch nicht symmetrisch sein: Ein Element (Knoten) b ist von a aus genau dann erreichbar, wenn die gerichtete Kante (a, b) existiert.[5] Sind zwei Elemente folglich nur durch eine einzige gerichtete Kante verbunden, so sind sie zwar benachbart, doch ist nur eines vom anderen aus erreichbar.

Der dem diskreten Raummodell zugrundeliegende gerichtete Graph muss somit nur die Erreichbarkeitsrelation der auf die Knoten abgebildeten Raumelemente abbilden, da sich aus der Erreichbarkeit direkt die Nachbarschaft ergibt. Für eine an die Raumstrukturen gebundene Bewegung von Agenten ist die Erreichbarkeit entscheidend, während die Wahrnehmung sich meist auf alle benachbarten Elemente erstreckt.

[4] Einen weiteren Ansatz, der diese Beziehung verwendet, beschreibt David O'Sullivan (2000) in seiner Dissertation: Er kombiniert Graphen mit irregulären Zellularautomaten für die Modellierung räumlicher Prozesse in Städten, um mittels graphentheoretischer Maße wie Zentralität und Konnektivität Aussagen über deren räumliche Struktur treffen zu können.

[5] Genau genommen ist die Definition von Erreichbarkeit in (gerichteten) Graphen nicht auf direkt benachbarte Knoten beschränkt, sondern bezieht sich auf beliebige Knoten: Ein Knoten v ist erreichbar von einem Knoten u, falls es einen (gerichteten) Weg P gibt mit Anfangsknoten u und Endknoten v (vgl. z. B. Aigner 2006, S. 113/116). Im folgenden soll mit Erreichbarkeit jedoch immer die direkte Erreichbarkeit über eine gemeinsame Kante gemeint sein.

Das diskrete Raummodell stützt sich im Einzelnen auf folgende Abstraktionen:

- Jedem Raumelement wird genau ein Referenzpunkt zugeordnet, über den es im kontinuierlichen kartesischen Bezugssystem lokalisiert ist. Von der bei Tesselationen vorhandenen geometrischen Form der Raumelemente wird abstrahiert: Die Positionierung von Objekten und Agenten erfolgt ausschließlich auf der Basis der Raumelemente, unterschiedliche Positionen innerhalb eines Raumelements werden nicht betrachtet. Dadurch werden die Raumelemente zu (Referenz-)Punkten aggregiert.[6] Dies liegt in der Natur der Diskretisierung, bei der die einzelnen diskreten Elemente als atomar angesehen werden. Die Berücksichtigung verschiedener Punkte innerhalb eines diskreten Raumelements würde dagegen kontinuierlichen oder zumindest weiter unterteilbaren Raum innerhalb eines Elements bedeuten.

- Ebenso wird von Form und Größe situierter Objekte und Agenten abstrahiert; sie werden statt dessen punktförmig repräsentiert. Dies vereinfacht die Modellierung erheblich und stellt zumindest für den diskreten Raum keine unzulässige Abstraktion dar, werden doch die einzelnen Raumelemente bereits auf Punkte abgebildet. Über die Angabe von Kapazitäten für Raumelemente ist zumindest eine implizite Modellierung der räumlichen Ausdehnung von Agenten und Objekten möglich. Bei einer Kapazität von 1 wird z. B. ein Agent als genauso groß wie das Raumelement angenommen, so dass zu jedem Zeitpunkt sich höchstens ein Agent in dem Element aufhalten kann. Dies ist die übliche Betrachtungsweise in Agentenumgebungen, die auf regelmäßigen Gittern beruhen wie z. B. in *Sugarscape* (Epstein und Axtell 1996), Schellings Segregationsmodell (Schelling 1978, S. 147–155) oder Hammond und Axelrods Modell der Evolution ethnozentrischen Verhaltens (Hammond und Axelrod 2006). Mit einer Kapazität von 0 könnten dagegen Raumelemente für Agenten „gesperrt“ werden.

- Ein gerichteter Graph erhält über seine mit Referenzpunkten versehenen Knoten eine bestimmte Form und Lage im Raum. Die Kanten besitzen daher neben ihrer Eigenschaft als Topologie-Repräsentanten eine zusätzliche geometrische Interpretation: Ihre Länge beschreibt den Abstand zwischen Anfangs- und Endknoten. Dieser könnte passend zum vorausgesetzten kartesischen Koordinatensystem generell als Euklidische Distanz gemessen werden, wodurch die Kanten zu Geradenstücken werden. Für eine größere Flexibilität beispielsweise in der Modellierung von Verkehrsnetzen ist es jedoch sinnvoll, von der Form der Kanten zu abstrahieren. Dies gelingt, indem die Kantenlänge explizit als ein Attribut der Kanten spezifiziert wird. Für die Bewegung der Agenten, die konzeptuell von Raumelement zu Raumelement stattfindet (siehe Abschnitt 4.3.2), wird nur die korrekte Entfernung zwischen benachbarten Knoten benötigt, nicht jedoch die tatsächliche Form der Kanten. Dadurch müssen gebogene Kanten nicht durch mehrere kurze, gerade Kanten approximiert werden, so dass für das Modell irrelevante Knoten vermieden werden können.

[6] Für Visualisierungen des modellierten Raums ist es sinnvoll, die geometrische Form der Raumelemente dennoch zu speichern, auch wenn sie für die Interaktion der Agenten mit dem Raummodell keine Bedeutung hat.

Um eine einfache Benutzung des graphbasierten diskreten Raummodells zu unterstützten, lässt sich die Abbildung einer Tesselation auf ihren dualen gerichteten Graphen – zumindest für die gebräuchlichsten Formen mit konvexen Raumelementen – automatisieren und so vor dem Anwender verbergen. Ein generisches Verfahren könnte z. B. die Schwerpunkte der Raumelemente berechnen und als Referenzpunkte der Knoten verwenden. Für je zwei in mehr als einem Punkt aneinandergrenzende Raumelemente sind jeweils zwei gerichtete Kanten zu erzeugen, deren Länge der Euklidischen Distanz zwischen den zugehörigen Referenzpunkten entspricht. Ein solches Verfahren setzt die weiter oben bereits erwähnte Annahme voraus, dass Nachbarschaft in Tesselationen auf angrenzende Elemente beschränkt und Erreichbarkeit immer symmetrisch ist.

Diese Annahme ist insofern plausibel, als beliebige Nachbarschaftsrelationen für Partitionierungen des Raums zwar möglich sind, jedoch der Anschauung widersprechen. Weil bei der Visualisierung einer Tesselation im Gegensatz zu einem Graphen üblicherweise nur die Raumelemente, nicht aber deren Beziehungen untereinander dargestellt werden, wird ein Betrachter implizit schließen, dass aneinandergrenzende Elemente auch benachbart und erreichbar sind. Dieses Problem ergibt sich bereits beim Standard-Gitter aus quadratischen Zellen, in dessen bildlicher Darstellung nicht sichtbar ist, ob die von-Neumann- oder die Moore-Nachbarschaft angewandt wird. Kompliziertere Formen der Diskretisierung des Raums mit beliebiger Nachbarschaft und insbesondere unterschiedlicher Erreichbarkeit zwischen Elementen sind vom Wesen her gerichtete Graphen und sollten daher auch explizit als solche repräsentiert werden.

4.3.1.2. Erweiterungen des Raummodells

Durch die Wahl eines ereignisgesteuerten Zeitfortschritts wird bei der Diskretisierung der Zeit eine Modellierung in unterschiedlicher Genauigkeit möglich. So können inaktive Perioden einzelner oder aller Entitäten „übersprungen“ werden, da nur Anfangs- und Endzeitpunkt repräsentiert werden müssen. Darüber hinaus kann jede Entität ihren eigenen Lebensrhythmus aufweisen und so gewissermaßen die Zeit mit ihrer eigenen Auflösung wahrnehmen. Analog zur Zeit sollte auch der Raum mit unterschiedlicher Genauigkeit diskretisierbar sein. Dies kann auf zwei verschiedenen Ebenen stattfinden:

1. Teile des Raums werden innerhalb eines Raummodells in unterschiedlich feiner Auflösung repräsentiert. Dies ergibt ein *heterogenes* Raummodell und ist mit dem vorgestellten Konzept bereits durchführbar: Sowohl Graphen als auch unregelmäßige Tesselationen erlauben es, den Detaillierungsgrad in beliebigen räumlichen Bereichen wie benötigt anzupassen.

2. Soll dagegen ein realer Raumauschnitt mehrfach auf unterschiedlichen Genauigkeitsstufen abgebildet werden, weil beispielsweise einzelne Entitäten unterschiedliche Anforderungen an das Raummodell stellen, so ergäbe das mehrere verschiedene Raummodelle, die in sich durchaus homogen strukturiert sein können. Da sich die Agenten eines Modells jedoch in der Regel in einem gemeinsamen Raum bewegen, müssen diese verschiedenen Raummodelle wieder zu einem Raummodell integriert werden. Eine relativ einfache und elegante Form der Integration führt zu einem *hierarchischen* Raummodell.

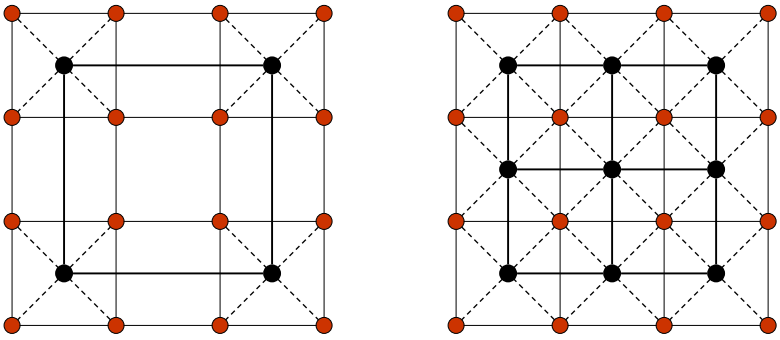

Abbildung 4.2.: Der Unterschied zwischen Mono-Hierarchie (links) und Poly-Hierarchie (rechts) am Beispiel eines regulären Gitters in zwei Hierarchie-Stufen. Nachbarschaft ist als durchgezogene Linie, Enthaltensein als gestrichelte Linie dargestellt. Da beide Relationen symmetrisch sind, wurde aus Gründen der Übersichtlichkeit auf die Darstellung der Orientierung (Pfeile) im gerichteten Graphen verzichtet.

Das bisher entwickelte Konzept eines graphbasierten diskreten Raummodells lässt sich leicht zu einem hierarchischen Raummodell erweitern, indem zusätzlich zur topologischen Beziehung „benachbart“ noch die ebenfalls topologische Beziehung „enthält“ bzw. „enthalten in“ gespeichert wird. In einer strengen Hierarchie (Mono-Hierarchie) ergibt sich eine zunehmende Verfeinerung der Diskretisierung so, dass ein Raumelement der Hierarchie-Stufe i in genau einem Raumelement der Stufe $i-1$ enthalten ist, während es selbst wiederum k Elemente der Stufe $i+1$ enthält. Die Anzahl k der Elemente, in die ein Raumelement auf der nächst-niedrigeren Hierarchie-Stufe unterteilt ist, muss dabei nicht über alle Elemente oder gar alle Hierarchie-Stufen gleich sein.

Eine Verallgemeinerung des monohierarchischen Prinzips erlaubt, dass ein Raumelement in mehreren Elementen der höheren Ebene enthalten sein kann (Poly-Hierarchie), d. h. dass das Raummodell der niedrigeren Hierarchie-Stufe nicht durch elementweise Verfeinerung konstruiert werden muss. In regelmäßigen Partitionierungen kann so eine engere Abstufung der einzelnen Raummodelle erreicht werden (vgl. Abbildung 4.2). Weitere Verallgemeinerungen führen zur Abkehr von der hierarchischen Struktur, indem die einzelnen Raummodelle nicht mehr zunehmende Verfeinerungen bilden müssen, sondern beliebige Diskretisierungen sein können. In jedem Fall ist die Enthaltensein-Relation explizit zu spezifizieren, damit die einzelnen Raummodelle – seien sie Stufen einer Hierarchie oder eher gleichrangige Komponenten – zu einem gemeinsamen Raummodell integriert werden können.

Diese Integration erfolgt am einfachsten durch Weiterleitung der Positionen situierter Objekte und Agenten an alle Komponenten. Jede Entität ist genau einer Raummodell-Komponente zugeordnet, deren Diskretisierung ihren Bedürfnissen entspricht und in der sie sich ausschließlich bewegt. Ihre Position bezieht sich daher auf ein Raumelement dieser Komponente. Über die Enthaltensein-Relation lässt sich die Anwesenheit der Entität jedoch problemlos in allen anderen Komponenten sichtbar machen, wobei die Entität in feiner aufgelösten Raum-Komponenten gleichzeitig an mehreren Raumelementen auftritt

(vgl. Abbildung 4.3, links). Auf diese Weise werden auch verschiedene räumliche Ausdehnungen einzelner Entitäten modellierbar, insbesondere bei Verwendung regelmäßiger Partitionierungen.

Da für die Modellierung der Bewegung im Raum nur die Erreichbarkeitsrelation der Raumelemente benötigt wird, stellen die Kanten des zugrundeliegenden gerichteten Graphen ausschließlich diese Relation dar. Das integrierte Raummodell basiert so weiterhin auf *einem* gerichteten Graphen, der allerdings in mehrere Komponenten zerfällt, also unzusammenhängend ist (vgl. Abbildung 4.3, rechts). Die Enthaltensein-Relation bildet praktisch einen zweiten gerichteten Graphen, der nicht für die Interaktion von Agenten mit ihrer räumlichen Umgebung zur Verfügung steht, sondern nur intern für die Weiterleitung der Positionen und damit die Aufrechterhaltung eines gemeinsamen Raummodells verwendet wird.

Bisher umfasst das graphbasierte diskrete Raummodell Informationen zu Geometrie (Referenzpunkte, Kantenlänge) und Topologie (Nachbarschaft, Erreichbarkeit, ggf. Enthaltensein) der räumlichen Strukturen. Agenten und Objekte lassen sich an Raumelementen positionieren, wobei ein Raumelement prinzipiell beliebig viele Agenten bzw. Objekte aufnehmen kann. Über die Angabe von Kapazitäten für Raumelemente könnten jedoch modellspezifische Beschränkungen durchgesetzt werden. Eine Erweiterung dieser Möglichkeit besteht darin, Raumelementen nicht nur Kapazitäten, sondern beliebige *Attribute* zuzuordnen, um weitere Eigenschaften festzulegen. Dieses Konzept stammt aus dem Bereich der Geo-Informationssysteme (GIS), in denen neben den rein raumbezogenen Angaben zur Geometrie und ggf. Topologie immer auch Attribute (Sachdaten, thematische Daten) verwaltet werden (siehe Abschnitt 2.2.2).

Ein Attribut ist zusammengesetzt aus einem eindeutigen Bezeichner, der zur Identifikation dient, und einem zugehörigen beliebigen Wert. Über solche Attribute lassen sich beispielsweise Ressourcen für Agenten modellieren, die eher Massengütern statt einzelnen Objekten entsprechen, wie die Ressource Zucker im *Sugarscape*-Modell (vgl. Abschnitt 5.4) oder sich ausbreitende Duftstoffe (Pheromone) in Ameisen-Modellen. Auch eher geographische Merkmale wie Oberflächenbeschaffenheit oder Niederschlagsmenge sind bei Bedarf leicht abbildbar.

Um die Verteilung von Attributen im Raum zu erleichtern, sind sogenannte *Attributgebiete* vorgesehen. Diese bestehen aus einer nichtleeren Menge von Positionen und einer ebenfalls nichtleeren Menge von Attributen, die den über die Positionen referenzierten Raumelementen zuzuordnen sind. Durch die Verwendung abstrakter Positionen statt konkreter Raumelemente werden Attributgebiete weitgehend unabhängig von einem speziellen Raummodell. Da die Menge der Attribute auf alle Raumelemente gleichermaßen angewendet wird, eignen sich die Attributgebiete allerdings nur für die Spezifikation von Raumbereichen, die identische Attribute aufweisen.

Die Verwendung von Attributen muss nicht auf Raumelemente beschränkt bleiben: Da deren Nachbarschafts- bzw. Erreichbarkeitsrelation als gerichtete Kanten des zugrundeliegenden Graphen explizit repräsentiert sind, können sie ebenfalls mit Attributen versehen werden. Wie auf Seite 93 beschrieben, wird zwar von der Form der Kanten abstrahiert, ihre Länge jedoch ausdrücklich als Attribut vermerkt, um bei der Bewegung der Agenten berücksichtigt zu werden. Weitere Attribute, die beispielsweise zusätzlich die

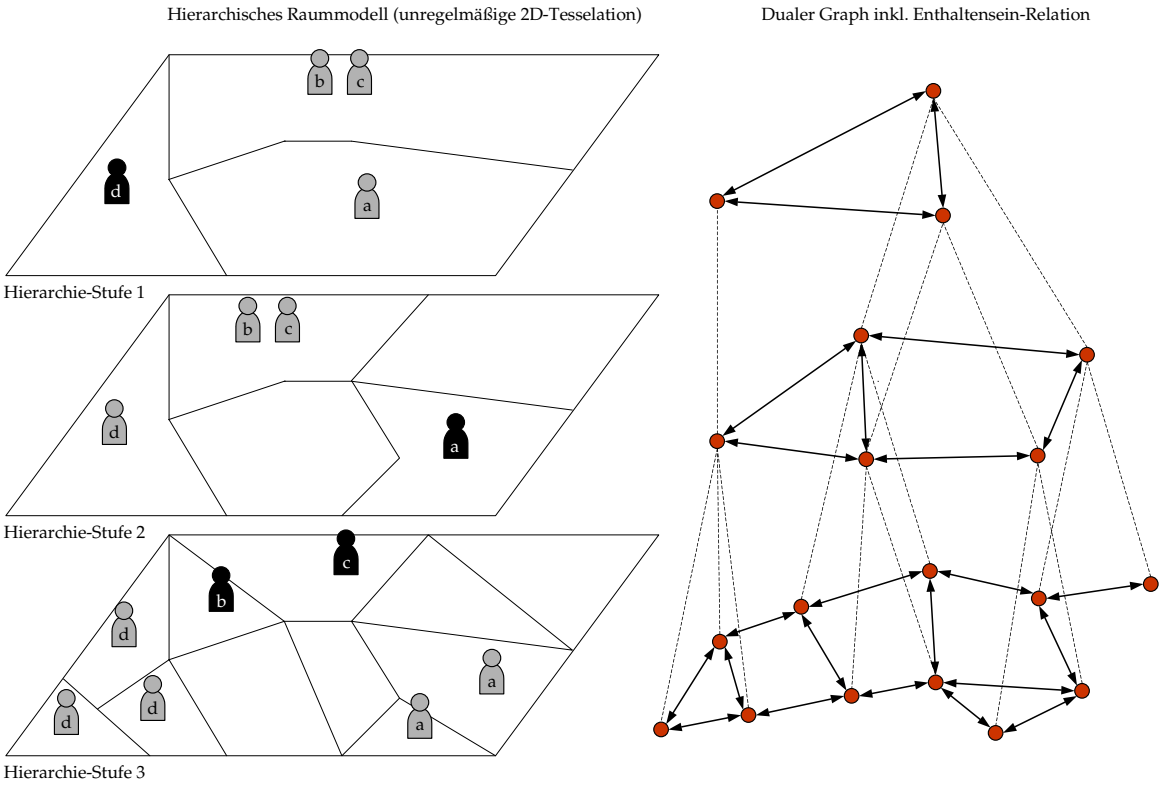

Abbildung 4.3.: Die Auswirkung der Positions-Weiterleitung an alle Hierarchie-Stufen am Beispiel eines einfachen monohierarchischen Raummodells. Ein Agent (schwarz dargestellt) ist zwar für seine Interaktion mit der Umwelt an eine Hierarchie-Stufe gebunden, jedoch auch in allen anderen Hierarchie-Stufen sichtbar (grau dargestellt), wobei er ggf. mehrfach innerhalb einer Stufe auftritt. Die gestrichelt gezeichneten ungerichteten Kanten im dualen Graphen geben die zur Weiterleitung benötigte Enthaltensein-Relation an. Der duale Graph selbst besteht aus drei Komponenten, je eine pro Hierarchie-Stufe.

Bewegung beeinflussen, können nach Bedarf hinzugefügt werden. Dies ist insbesondere bei der Modellierung von (Verkehrs-)Netzen nützlich, bei der der gerichtete Graph nicht eine interne Basis, sondern das eigentliche Raummodell darstellt und die Kanten daher eine reale Entsprechung besitzen. In diesem Zusammenhang lassen sich entsprechende Kantenattribute als Kantengewichte einsetzen, wie sie bei der Routensuche in gewichteten Graphen vorausgesetzt werden.

4.3.2. Unterstützung der Bewegung im Raum

Um in Agentenmodellen die Bewegung im Raum zu ermöglichen, muss ein Simulationswerkzeug mindestens Operationen zur Wahrnehmung und zum Wechsel der Position zur Verfügung stellen. Ein Agent bzw. dessen Entwickler kann darauf aufbauend eine beliebige Bewegungsstrategie entwerfen, z. B. zur Exploration des Raums oder zum Planen einer bestimmten Route. Da solche Strategien häufig nicht modellspezifisch sind, sondern gut wiederverwendbar, ist es sinnvoll, sie in ein allgemeines Werkzeug für Multiagentenmodelle, das den Raum explizit berücksichtigt, zu integrieren. So lassen sich die Strategien darüber hinaus gut auf das Raummodell abstimmen, was insbesondere für Strategien geboten ist, welche die räumliche Struktur beachten. Das im folgenden vorgestellte Konzept unterstützt die Bewegung im Raum auf möglichst abstrakter Ebene, so dass es flexibel einsetzbar ist. Der Schwerpunkt liegt dabei – dem im vorigen Abschnitt 4.3.1 entwickelten Raummodell entsprechend – auf Bewegungsstrategien für den diskreten Raum.

Physikalisch betrachtet bedeutet Bewegung die zeitabhängige Veränderung des Ortes. Ein Objekt bewegt sich von einer aktuellen Position a hin zu einer neuen Position b, indem es den Raum zwischen diesen Positionen überbrückt. Die dafür benötigte Zeit wird bestimmt von der Geschwindigkeit des Objekts. Bewegung lässt sich somit ausdrücken durch die drei Größen Zeit (t), Entfernung (zurückgelegte Strecke s) und Geschwindigkeit (v), zwischen denen folgende Beziehung gilt: $v = s/t$. Die Geschwindigkeit wird dabei als konstant angenommen.

Im Einklang mit der Diskretisierung von Raum und Zeit soll hier Bewegung immer als Bewegung von einer Position zu einer anderen Position aufgefasst werden, wobei Positionen wie beschrieben Punkte im Raum referenzieren. Damit lässt sich die Entfernung zwischen ihnen in jedem Fall als Euklidische Distanz bestimmen, sollte vom jeweiligen konkreten Raummodell jedoch unter Beachtung der räumlichen Struktur genauer berechnet werden können. Agenten, die sich autonom im Raum bewegen, besitzen zu jedem Zeitpunkt eine solche Position. Bewegung umfasst für sie die Wahl einer neuen Position und die anschließende Durchführung der eigentlichen Bewegung hin zu dieser Position. Ist die (konstante) Geschwindigkeit der Bewegung bekannt, kann der Ankunftszeitpunkt an der Zielposition im voraus berechnet werden, da die zurückzulegende Strecke ebenfalls bekannt ist. Dies ist eine Voraussetzung für die ereignisgesteuerte Simulation: Mit dem Beginn der Bewegung kann nun ein entsprechendes Ende-Ereignis vorgemerkt und die Bewegung somit auf den Verbrauch von Simulationszeit abgebildet werden.

Die Wahl einer neuen Position geschieht anhand einer Bewegungsstrategie, die ggf. abhängig von den aktuellen Zielen des Agenten ist. Sie muss in der Lage sein, ausgehend von der aktuellen Position eine neue, anzustrebende Position im Raum zu liefern. Im diskreten Raum wird dies in der Regel eine der erreichbaren Nachbarpositionen sein,

so dass die Struktur des Raumes berücksichtigt wird; es können aber beispielsweise auch alle wahrnehmbaren Positionen in die Auswahl einbezogen werden, wie im *Sugarscape*-Modell (siehe Abschnitt 5.4). Bei der Entscheidung über die neue Position können unterschiedlich komplexe Verfahren zum Einsatz kommen:

- Die neue Position wird *zufällig* aus der Menge der möglichen neuen Positionen gewählt. Diese Strategie wird häufig verwendet, wenn Agenten eine ihnen unbekannte Umgebung erkunden.
- Die neue Position wird anhand einer vorgegebenen *Bewertungsfunktion* bestimmt, indem z. B. die Position gewählt wird, die den höchsten Wert eines bestimmten Attributs aufweist. Agenten folgen auf diese Weise einem Gradientenfeld, in Modellen der Futtersuche bei Ameisen beispielsweise einer Pheromonspur. Diese Strategie benötigt somit zusätzliche Informationen in der Umgebung, die über die Attribute des Raummodells spezifizierbar sind.
- Bei der Bewegung im Raum wird eine bestimmte *Richtung* verfolgt. Die nächste Position ist dann so zu wählen, dass die Abweichung von der gegebenen Richtung möglichst gering ist.
- Die Bewegung erfolgt entlang einer vorher bestimmten *Route*. Eine solche Route besteht aus einer Folge von Positionen, die nacheinander aufgesucht werden. Die Wahl der neuen Position beschränkt sich dann auf die schrittweise Abarbeitung dieser Folge. In der Regel sind aufeinanderfolgende Positionen einer Route direkt erreichbar, d. h. im zugrundeliegenden Graphen durch eine gerichtete Kante verbunden, so dass die Route einen gerichteten Pfad darstellt.

Alle Verfahren können zusätzlich noch eine ausgezeichnete Zielposition berücksichtigen, die der Agent erreichen will. Ist dem Agenten dieses Ziel bekannt, wird es bei den beiden letzten Verfahren direkt in die Bestimmung der Richtung bzw. der Route eingehen.

Die Fähigkeit eines Agenten zur Bewegung im Raum ist mit diesem Konzept unabhängig vom konkreten Raummodell. Durch Auslagerung der Positionswahl in eine (beliebig austauschbare) Bewegungsstrategie ist die Durchführung der Bewegung auf einen Wechsel zwischen zwei gegebenen Positionen reduziert. Der entsprechende Zeitkonsum ergibt sich direkt aus der vom Agenten vorgegebenen Geschwindigkeit und der vom Raummodell berechneten Distanz zwischen den beiden Positionen. Daher kann das zugehörige Ereignis automatisch generiert und vorgemerkt werden. Indem einem Agenten bei der Ankunft an der neuen Position dieses Ereignis mitgeteilt wird, kann die Realisierung der Bewegung vor ihm verborgen werden. Ein mobiler Agent muss so nur noch eine seinen Bedürfnissen und ggf. dem verwendeten Raummodell angepasste Bewegungsstrategie spezifizieren, um sich im Raum bewegen zu können.

Die Ereignissteuerung (vgl. auch Abschnitt 4.6) stellt weitere Anforderungen an das Bewegungskonzept: Da Agenten potentiell Aktionen parallel ausführen, muss gewährleistet sein, dass sie während eines Bewegungsvorgangs beispielsweise gleichzeitig die Umgebung wahrnehmen, so dass sie auf Veränderungen sofort reagieren können. Damit der Umgebungszustand – und damit auch die aktuellen Positionen der Agenten – vor jedem Zugriff durch einen Agenten korrekt aktualisiert ist, ist u. a. die Bewegung als eine

Folge von atomaren Teilschritten zu realisieren. Für diskrete Raummodelle heißt das, längere vorausgeplante Routen müssen zerlegt werden in Abschnitte von einzelnen Raumelementen; Bewegung im Raum findet so von Raumelement zu Raumelement statt. Ein atomarer Bewegungsschritt kann eventuell unterbrochen werden, muss jedoch zu Ende ausgeführt werden, bevor die Richtung geändert werden kann. Dies ist darin begründet, dass Bewegung zumindest im diskreten Raummodell auf einem gerichteten Graphen stattfindet, für den nicht sichergestellt ist, dass zu jeder gerichteten Kante auch eine Kante in Gegenrichtung existiert. Bei entsprechend feiner Auflösung des Raummodells sollte diese Einschränkung jedoch nicht schwer wiegen.

4.4. Modellierung der Umgebung

In der agentenorientierten Simulation bildet jedes Modell ein Multiagentensystem, d. h. eine Menge von Agenten, die in einer gemeinsamen Umgebung agieren und zwischen denen vielfältige Beziehungen bestehen können. Diese Beziehungen lassen sich einteilen in Interaktion zwischen Agenten und Umgebung, Kommunikation zwischen Agenten und organisatorische Beziehungen. Kommunikation findet in der Regel als Nachrichtenaustausch statt, kann jedoch auch indirekt durch Veränderung der dann meist räumlichen Umgebung erfolgen. Da die Kommunikationsinfrastruktur üblicherweise der Umgebung zugeschrieben wird (Huhns und Stephens 1999, S. 81f), der Austausch von Nachrichten zwischen Agenten somit über die Umgebung läuft, kann Kommunikation als Spezialfall der Interaktion zwischen Agenten und Umgebung aufgefasst werden.

Das hier entwickelte Konzept legt diese Auffassung zugrunde und erweitert sie noch um die organisatorischen Beziehungen, indem diese ebenfalls von der Umgebung verwaltet werden. Gruppen als eine Form von Organisationsbeziehungen können so auch direkt als Mittel für die Kommunikation eingesetzt werden, um die Weiterleitung von Nachrichten an mehrere Empfänger gleichzeitig zu ermöglichen (*multicast* bzw. *broadcast*, wenn alle Agenten angesprochen werden). Damit ergibt sich eine klare Schnittstelle für die Agenten, die ausschließlich mit der Umgebung interagieren.

Neben der Kommunikationsinfrastruktur (Abschnitt 4.4.2) und den Organisationsstrukturen (Abschnitt 4.4.3) bildet in Multiagentenmodellen mit expliziter Raumrepräsentation das Raummodell eine zentrale Komponente der Umgebung. Es wurde im vorigen Unterkapitel 4.3 bereits ausführlich diskutiert. Im Zusammenhang mit dem Raummodell kann die Umgebung weitere Bestandteile aufweisen: eine Menge von situierten Objekten und eine eigene Dynamik, die agenten-unabhängige Zustandsänderungen ermöglicht. Hierfür sind spezielle Umgebungsprozesse vorgesehen (siehe Abschnitt 4.4.1).

Die in der räumlichen Umgebung positionierten Objekte können beliebiger Art sein. In der Regel sind sie passiv (Ferber 1999, S. 11) und dienen als Ressourcen für Agenten. Es ist daher nicht vorgesehen, dass sie sich selbst in der Umgebung bewegen oder in anderer Weise aktiv auf die Umgebung einwirken. Diese Möglichkeit ist jedoch nicht explizit ausgeschlossen: Da die Objekte beliebig sein können, ist dem Modellierer hier bewusst großer Spielraum gelassen. So können bei Bedarf auch aktive Objekte modelliert werden, indem beispielsweise Simulationsprozesse oder mit Ereignissen verknüpfte Entitäten verwen-

det werden.[7]. Weitere, nicht als diskrete Objekte modellierbare Merkmale der räumlichen Umgebung können über die Attribute des Raummodells ausgedrückt werden.

Aus der Sicht eines Agenten zeichnet sich die Umgebung durch bestimmte Eigenschaften aus, die mehr oder weniger hohe Anforderungen an den Agenten stellen (siehe Abschnitt 2.3.1, Seite 35f). Soweit solche Eigenschaften in einem Multiagentenmodell relevant sind, müssen sie beim Entwurf der Umgebung berücksichtigt werden und sollten daher vom verwendeten Simulationswerkzeug unterstützt werden. Hierfür sind folgende Möglichkeiten vorgesehen:

- Der Wahrnehmungsbereich eines Agenten kann lokal beschränkt sein (*unzugängliche* Umgebung), ebenso sein Aktionsradius (Grad der *Kontrollierbarkeit*), wobei die beiden Bereiche nicht identisch sein müssen. Durch die Angabe agentenspezifischer Werte für Wahrnehmungs- und/oder Aktionsradius kann die Lokalität beliebig begrenzt werden. Auf diese Interaktion zwischen Agent und Umgebung wird im Zusammenhang mit der Agentenmodellierung in Abschnitt 4.5.1 ausführlicher eingegangen.

- Ein einzelner Agent nimmt seine Umgebung als *dynamisch* wahr, wenn sich ihr Zustand während seiner Aktionsselektion ändern kann. In Multiagentensystemen, in denen mehrere Agenten nebenläufig handeln, ist dies der Regelfall. Zusätzliche Dynamik kann über die speziellen Umgebungsprozesse (Abschnitt 4.4.1) abgebildet werden.

- Durch die verwendete Diskretisierung in Zeit und Raum ist auch die Umgebung eher als *diskret* einzustufen. Über die Umweltdynamik und die beliebigen Attribute sind kontinuierliche Wertebereiche jedoch für ausgewählte Merkmale der diskreten Raumelemente modellierbar.

- Die Umgebung ist grundsätzlich *deterministisch*, d. h. jede Aktion hat ihren garantierten Effekt. Dies ist insofern keine wesentliche Einschränkung, als dynamische und unzugängliche Umgebungen einem Agenten zumindest nicht-deterministisch erscheinen können (Russell und Norvig 2009, S. 44). Ein modellspezifischer Nicht-Determinismus wäre jedoch realisierbar, indem die möglichen Aktionen eines Agenten nur mit einer gewissen Wahrscheinlichkeit wirksam werden.

Mit diesen Konzepten wird die Modellierung einer großen Bandbreite von Umgebungstypen unterstützt; sogar die komplexeste Klasse einer unzugänglichen, dynamischen, kontinuierlichen und nicht-deterministischen Umgebung (Wooldridge 2009, S. 25) ist mit geringen Abstrichen umsetzbar.

4.4.1. Umgebungsprozesse

Der Zustand der Umgebung in Multiagentensystemen – und damit auch in Multiagentenmodellen – wird durch Aktionen der in ihr handelnden Agenten verändert. Auf diese

[7] Ermöglicht wird diese Verbindung der agentenorientierten mit der prozess- und ereignisorientierten Simulation dadurch, dass auf einem Simulationspaket für die diskrete Simulation aufgebaut wird (vgl. Abschnitt 4.2)

Weise können Agenten sich gegenseitig beeinflussen. Eine dynamische Umgebung kann darüber hinaus jedoch auch selbst aktiv ihren Zustand manipulieren. Solche Aktivitäten entsprechen zeit- oder zustandsabhängigen Funktionen, die im Raum Attribute bzw. Attributgebiete ändern (z.B. Nebel, Signalausbreitung, Pheromon-Verdunstung) oder Objekte erzeugen und vernichten. Auf konzeptueller Ebene sind diese Vorgänge nicht mit Agenten gleichzusetzen, sondern deutlich unterhalb des Agenten-Niveaus anzusiedeln. Russell und Norvig sprechen im Zusammenhang mit einem sehr einfachen Umgebungssimulator von dynamischen Prozessen, die nicht als Agenten aufgefasst werden, und geben als Beispiel Regen an (1995, S. 47)[8]. Neben solchen Naturereignissen, die eher für ökologische Systeme relevant sind, können für soziale Systeme gesetzliche Regelungen und soziale Normen als Beispiele für Umgebungsprozesse genannt werden (Parunak 2000, S. 2).

Durch die Integration eines Simulationsframeworks, das zumindest die prozessorientierte Weltsicht unterstützt, stehen mit Simulationsprozessen geeignete Abstraktionen zur Modellierung dynamischer Umweltprozesse zur Verfügung. Alternativ könnten Konstrukte der anderen diskreten Weltsichten verwendet werden, wenngleich Prozesse die größte konzeptuelle Nähe aufweisen. Ein wesentlicher Vorteil dieses Ansatzes besteht darin, dass die Umgebung so über ein adäquat beschreibbares dynamisches Verhalten verfügt, ohne dass sie als spezieller Agent modelliert werden muss (siehe Abschnitt 4.2). Diese Hilfskonstruktion wird in Werkzeugen für die agentenbasierte Simulation gewählt, die Aktivitäten ausschließlich mit Agenten verbinden (vgl. z. B. Klügl 2001, S. 93 und S. 113).

Da die Umgebungsprozesse den Zustand der Umgebung (genauer gesagt: des modellierten Raums) verändern, benötigen sie Zugriff auf das interne Raummodell der Umgebung. Im Unterschied zu den situierten Objekten und Agenten lassen sie sich nicht genau einer Position im Raum zuordnen, wird doch ihr Einflussbereich in der Regel mehr als ein Raumelement umfassen. Es wäre möglich, sie analog zu der Vorgehensweise im hierarchischen Raummodell entsprechend mehrfach im Raum zu positionieren oder eine der Positionen ihres Einflussbereichs als Repräsentanten auszuwählen. Dagegen spricht, dass sie keine konkret wahrnehmbaren Objekte darstellen, sondern ihre Existenz nur indirekt über die Auswirkung ihres Einflusses sichtbar wird. Sie sollen daher nicht mit einer bestimmten Position im Raum verknüpft werden, sondern nur auf beliebige Ausschnitte des Raummodells zugreifen können. Für die Realisierung ist es somit sinnvoll, diese Prozesse direkt in der Umgebung zu verwalten und nicht an das Raummodell zu delegieren.

4.4.2. Kommunikationsinfrastruktur

Für die Kommunikation zwischen Agenten stellt die Umgebung die benötigte Infrastruktur zur Verfügung. Diese wird im Allgemeinen Protokolle verschiedener Abstraktionsstufen umfassen: Kommunikationsprotokolle für den Austausch einzelner Nachrichten und Interaktionsprotokolle für den Austausch strukturierter Nachrichtenfolgen, sogenannter Konversationen (Huhns und Stephens 1999, S. 79). Kommunikationsprotokolle werden üblicherweise in aufeinander aufbauenden Schichten definiert, wobei die unterste Schicht

[8] Dieses Beispiel ist in der aktuellen dritten Auflage (Russell und Norvig 2009) des erfolgreichen Lehrbuchs nicht mehr enthalten.

die Verbindungsmethode, die mittlere die Syntax der Nachricht und die obere die Semantik der übermittelten Information spezifiziert (Huhns und Stephens 1999, S. 86). Da in der agentenbasierten Simulation Art und Umfang der Kommunikation modellabhängig ist, kann die Infrastruktur nur den grundlegenden Dienst der Nachrichtenweiterleitung unterstützen und zwar in möglichst generischer Form, d. h. ein erweiterbares Nachrichtenformat vorgeben und sowohl Punkt-zu-Punkt- als auch Gruppen-Kommunikation zulassen.

Die Infrastruktur ist dann weder auf eine bestimmte Agenten-Kommunikationssprache festgelegt noch auf bestimmte Interaktionsprotokolle zur Koordination von Agenten. Mit dieser Reduktion auf das Wesentliche bietet ein Simulationswerkzeug dem Anwender die größtmögliche Flexibilität bei der Gestaltung der Agenten-Kommunikation, erhöht jedoch gleichzeitig dessen Aufwand bei der Definition und Umsetzung modellspezifischer Schnittstellen.

In dem hier entwickelten Konzept wird die Flexibilität höher bewertet als der damit einhergehende Aufwand auf Seiten des Modellierers. Daher beschränkt sich die Kommunikationsinfrastruktur im Wesentlichen auf die Funktionalität, Nachrichten an einzelne oder mehrere Empfänger weiterzuleiten. Ein direkter Nachrichtenaustausch zwischen Agenten ist nicht vorgesehen; jeder Agent interagiert wie weiter oben bereits dargelegt ausschließlich direkt mit der Umgebung. Bezüglich der Kommunikation bietet dieser Ansatz den zusätzlichen Vorteil, auch indirekten Nachrichtenaustausch über ein sogenanntes *Blackboard*, d. h. einen gemeinsamen, zentralen Informationspool, zu ermöglichen (Corkill 2003).

Eine Nachricht muss mindestens Sender, Empfänger und (beliebig komplexen) Inhalt spezifizieren. Als Empfänger sind einzelne Agenten oder Gruppen zugelassen. Die Kommunikation selbst beruht auf dem *Push*-Prinzip, d. h. die Initiative liegt beim Sender (Mühlhäuser 2006, S. 728), der eine Nachricht erzeugt und an die Umgebung schickt. Dort sorgt die Kommunikationsinfrastruktur dafür, dass der spezifizierte Empfänger die Nachricht erhält. In der Regel wird dies zeitverzugslos und korrekt erfolgen, doch ließen sich auch modellabhängig Störungen oder Verzögerungen in der Nachrichtenübermittlung realisieren, indem beispielsweise die Wahrscheinlichkeit für Störungsfreiheit oder eine stochastische Verteilung für die Dauer des Transports angegeben wird.

Die Kommunikation wird generell als asynchron angenommen, so dass ein Agent durch das Senden einer Nachricht nicht blockiert wird (Mühlhäuser 2006, S. 722). Um den Empfänger sofort über den Erhalt einer Nachricht zu informieren, können Nachrichten als „aktivierend“ gekennzeichnet werden, wodurch der entsprechende Agent seine aktuelle Tätigkeit unterbricht, um auf die Nachricht reagieren zu können. Diese Konstruktion wird im Zusammenhang mit der Modellierung der Agenten in Abschnitt 4.5.1 näher erläutert. Da ein Empfänger so nicht die Nachricht direkt vom Sender entgegennimmt, müssen Nachrichten bis zur Verarbeitung durch den Empfänger aufbewahrt werden können, und zwar möglichst in der Reihenfolge ihres Eingangs. Wenn man davon ausgeht, dass Gruppen die an sie gerichteten Nachrichten immer direkt an ihre Mitglieder weiterreichen, benötigen nur die Agenten einen Nachrichtenpuffer. Dieser kann entweder von der Kommunikationsinfrastruktur vorgehalten oder dem Agenten selbst zugeordnet werden. Zwar bietet die erste Lösung den Vorteil, dass so auch Nachrichten an noch nicht im System befindliche Agenten gesammelt werden können, doch entspricht die zweite Lösung

eher dem Konzept von Agenten als autonomen Akteuren und wird daher hier vorgezogen.

Voraussetzung für diese Art der Nachrichtenweiterleitung ist, dass die Empfänger der Kommunikationsinfrastruktur bekannt sind. Die Menge der Agenten als potentielle Empfänger kann sich jedoch dynamisch verändern, da Multiagentenmodelle – wie Multiagentensysteme generell – als offene Systeme konzipiert sind, so dass Agenten während der Simulation das System betreten und auch wieder verlassen können. Beim Eintritt in das System muss sich ein Agent daher bei der Umgebung anmelden und sich beim Verlassen wieder abmelden. Die Umgebung führt eine Liste aller angemeldeten Agenten. Gleiches gilt für die Agenten-Gruppen, die zur Laufzeit dynamisch erzeugt und gelöscht werden können (siehe den folgenden Abschnitt 4.4.3).

Eine weitere Voraussetzung ist, dass die Empfänger eindeutig identifizierbar sind. Da sowohl bei Agenten als auch bei Gruppen Namen zur Identifikation verwendet werden, muss die Kommunikationsinfrastruktur bei der Registrierung eines Agenten bzw. einer Gruppe sicherstellen, dass die Namen systemweit eindeutig sind. Auf Basis der Agenten- und Gruppen-Liste kann die Umgebung dann einen einfachen Verzeichnisdienst anbieten, über den ein anfragender Agent eine bestimmte Gruppe oder einen anderen Agenten lokalisieren kann. Allerdings wird in einer räumlich expliziten Umgebung das Auffinden anderer Agenten vornehmlich über die Wahrnehmung des lokalen Umgebungszustands geschehen (vgl. Abschnitt 4.5.1).

4.4.3. Organisationsstrukturen

In Multiagentensystemen können organisatorische Beziehungen und Strukturen einen großen Einfluss auf das Verhalten des gesamten Systems ausüben; es ist daher sinnvoll, sie explizit zu repräsentieren (vgl. Ferber und Gutknecht 1998, S. 129 und Jennings 2001, S. 38). Übertragen auf den Entwurf von Multiagentenmodellen bedeutet das, ein Simulationswerkzeug muss die flexible Definition und Verwaltung von Organisationsstrukturen unterstützen. Da Organisationen im Wesentlichen Gruppen von Entitäten beschreiben, die auf einer höheren Ebene jeweils als Einheit betrachtet werden können, stellen Gruppen eine elementare Form von Organisationsstrukturen dar.

Das im folgenden vorgestellte Konzept stützt sich auf das AGR-Modell (Ferber und Gutknecht 1998; Ferber et al. 2004) zur Beschreibung von Organisationen in Multiagentensystemen mittels Gruppen und Rollen. Eine Gruppe ist definiert als eine Menge von Agenten, während eine Rolle die abstrakte Repräsentation einer Funktion, eines Dienstes oder einer Kennzeichnung darstellt, die ein Agent innerhalb einer Gruppe einnimmt (Ferber und Gutknecht 1998, S. 130). Gruppen sind nicht disjunkt, so dass jeder Agent gleichzeitig Mitglied in verschiedenen Gruppen sein kann, und sie sind nicht statisch, so dass Agenten sie beliebig gründen, betreten und wieder verlassen können. Kommunikation zwischen Agenten wird auf Basis der gruppen-lokalen Rollen spezifiziert; dies ermöglicht die Verwendung unterschiedlicher Interaktionsprotokolle für einen Agenten, indem er entsprechende Rollen in verschiedenen Gruppen übernimmt (Ferber und Gutknecht 1998, S. 130).

Hier wird auf der allgemeinsten Bedeutung von Gruppen und Rollen aufgebaut: Eine Gruppe wird verstanden als eine Methode, eine Menge von Agenten zu kennzeich-

nen, und eine Rolle lässt sich auf die elementare Funktion „Gruppenmitglied“ reduzieren. Dieser Ansatz wird dahingehend erweitert, dass die für viele komplexe Systeme charakteristische hierarchische Struktur in das Gruppen-Konzept integriert wird, so dass eine Gruppe wiederum Mitglied einer anderen Gruppe sein kann.

Gruppen können unter beliebigen modellabhängigen Gesichtspunkten zusammengestellt und dazu verwendet werden, bestimmte Operationen auf alle Gruppenmitglieder anzuwenden. Da es sich bei den Gruppenmitgliedern letztendlich um autonome Agenten handelt, ist allerdings nicht garantiert, dass jedes Gruppenmitglied der Ausführung der angeforderten Operation zustimmt. Nur der Standardfall einer solchen Operation wird davon ausgenommen: Den Empfang einer Nachricht im Zuge einer *Multicast*-Kommunikation kann kein Agent ablehnen.

Die explizite Raumrepräsentation ermöglicht, den Raum als gruppenbildendes Kriterium heranzuziehen. In räumlich definierten Gruppen besteht das von allen Gruppenmitgliedern geteilte Merkmal darin, sich an einer von mehreren ausgezeichneten Positionen aufzuhalten. Solche räumlichen Gruppen können als Spezialfall von Attributgebieten (vgl. Abschnitt 4.3.1) aufgefasst werden. Sobald ein Agent den Bereich des Raummodells betritt, der als Gruppengebiet definiert ist, wird er automatisch Mitglied in dieser Gruppe. Verlässt er das Gruppengebiet wieder, so wird er ebenso automatisch wieder aus der Gruppe ausgetragen. Dieses Konzept räumlicher Gruppen wird intern für die Abbildung der Wahrnehmungsbereiche situierter Agenten verwendet (siehe Abschnitt 4.6).

Auf Basis der Gruppen ist eine einfache funktionale Referenzierung von Agenten möglich, indem diese den Zugriff auf ihre Mitglieder anhand bestimmter Kriterien erlauben. Sind Rollen für eine Gruppe definiert, so bilden sie natürliche Auswahlkriterien (Ferber 1999, S. 266), jedoch sind z. B. auch modellspezifische Bewertungsfunktionen denkbar, die – auf eine bestimmte Eigenschaft angewendet – den besten Agenten der Gruppe ermitteln.

Alle Gruppen werden von der Umgebung verwaltet, so dass sie sicher für die Kommunikation zur Verfügung stehen. Die Umgebung liefert auf Anfrage eine Liste aller derzeit definierten Gruppen oder eine Liste aller Gruppen, in denen ein bestimmter Agent Mitglied ist. Da Agenten ausschließlich mit der Umgebung interagieren, können Gruppen auch nur über die Umgebung dynamisch erzeugt und gelöscht werden.

4.5. Modellierung der Agenten

Wie in Abschnitt 3.1.3 diskutiert, können Agenten in der agentenorientierten Weltsicht konzeptuell sowohl als Erweiterung von Simulationsprozessen als auch von aktiven Objekten betrachtet werden. Da in der objektorientierten Weltsicht die Beschreibung des Verhaltens nicht beschränkt ist auf die linearen Abläufe der prozessorientierten Weltsicht, bieten aktive Objekte eine geeignetere Basis für die Entwicklung von Agenten. Letztendlich ist die verwendete Grundlage aus Sicht des Modellierers zweitrangig, solange ein Simulationswerkzeug Konstrukte auf dem Abstraktionsniveau von Agenten zur Verfügung stellt.

Diese Konstrukte müssen Möglichkeiten zur Beschreibung des Verhaltens und der Interaktionen eines Agenten mit der Umgebung bzw. anderen Agenten umfassen. Dabei sollte die Verhaltensbeschreibung möglichst allgemein unterstützt werden, so dass ein

Modellierer nicht auf eine bestimmte Agenten-Architektur (vgl. Abschnitt 2.3.4) festgelegt wird. In dem hier entwickelten Konzept wird ein Agent daher modular aufgebaut: Er besitzt ein austauschbares Verhalten (Abschnitt 4.5.2) und eine Menge von Fähigkeiten, die beliebig erweitert werden kann. Vordefiniert für alle Agenten sind die Fähigkeit zur Kommunikation, da dies in der Regel vorausgesetzt wird (Huhns und Stephens 1999, S. 83), und Fähigkeiten zur Interaktion mit einer räumlichen Umgebung wie Wahrnehmung oder Bewegung (siehe Abschnitt 4.5.1). Sowohl Verhalten als auch Fähigkeiten sind auf den ereignisgesteuerten Zeitfortschritt abgestimmt.

4.5.1. Interaktion mit der Umgebung

Wie im Zusammenhang mit der Modellierung der Umgebung in Abschnitt 4.4 bereits dargelegt, ist es sinnvoll, jegliche Interaktion eines Agenten mit anderen Komponenten des Multiagentenmodells auf Interaktion mit der Umgebung abzubilden. Für diese Interaktion mit der Umgebung ist ein Agent mit Sensoren und Effektoren ausgestattet, von deren Funktionsweise gewöhnlich abstrahiert wird, indem beispielsweise Wahrnehmung über Sensoren als Funktion definiert wird, die Umgebungszustände auf Perzepte abbildet (vgl. u. a. Ferber 1999, S. 190f und Wooldridge 2009, S. 38). Von einigen Ausnahmen[9] abgesehen, werden Sensoren auch in der agentenbasierten Simulation nicht explizit repräsentiert (Klügl 2001, S. 73) und Effektoren als eine Menge von möglichen Aktionen modelliert, aus der der Agent eine Auswahl treffen kann. Dieser Ansatz wird auch in dem hier beschriebenen Konzept verfolgt.

4.5.1.1. Wahrnehmung

Unter Wahrnehmung soll hier die lokal beschränkte Wahrnehmung des Zustands der räumlichen Umgebung verstanden werden. Damit sind konzeptuell nur situierte Agenten mit einem entsprechenden Sensor ausgestattet. Dieser wird implizit repräsentiert durch die Angabe seiner Reichweite, die den aktuellen Wahrnehmungsbereich des Agenten definiert. Als einfachste Lösung wird angenommen, dass der Sensor in alle Richtungen gleich funktioniert, d. h. die Sensorreichweite wird als Radius des kreisförmigen Wahrnehmungsbereichs interpretiert. Die Reichweite des Sensors kann modellabhängig für jeden Agenten individuell festgelegt werden.

Diese Art der Wahrnehmung ist unabhängig vom verwendeten Raummodell und lässt sich sowohl auf kontinuierliche als auch auf diskrete Modelle anwenden. Im diskreten Fall wird bei der Berechnung des Wahrnehmungsbereichs die Nachbarschaftsrelation zugrundegelegt, so dass auch Raumelemente, die eventuell nicht erreichbar sind, wahrgenommen werden können. Da der Abstand zwischen Raumelementen im zugrundeliegenden gerichteten Graphen explizit als Kantenlänge notiert ist, lässt sich leicht bestimmen, welche Raumelemente ausgehend von der aktuellen Position eines Agenten sichtbar sind.

[9] Das insbesondere als Testumgebung für Agenten-Verhaltensmodelle entwickelte System XRAPTOR sieht z. B. explizit modellierte Sensoren und Effektoren vor, deren Funktionalität eingeschränkt ist (Polani und Uthmann 2000, S. 113f). Damit sollen adaptive Verfahren untersucht werden können hinsichtlich ihrer Fähigkeiten, mit begrenzten Informationen und Handlungsmöglichkeiten umzugehen.

Die explizite Berücksichtigung der tatsächlichen Distanz bewirkt, dass die räumliche Struktur die Form des Wahrnehmungsbereichs beeinflusst: So wird der Bereich außer in regulären Gittern nicht exakt kreisförmig sein. Bereits die Moore-Nachbarschaft in zweidimensionalen Gittern aus quadratischen Zellen kann sich hier ggf. unerwartet auswirken, da die diagonal benachbarten Zellen weiter entfernt sind als die horizontal und vertikal benachbarten. Für Modelle, in denen eine eher abstrakte Auffassung des Raums vertreten wird, bei der es ausreicht, Abstände in diskreten Raumeinheiten zu messen, wird daher zusätzlich die Möglichkeit vorgesehen, die Sensorreichweite als Anzahl von Raumelementen zu spezifizieren. Damit wird der Wahrnehmungsbereich zumindest konzeptuell wieder kreisförmig; der Modellierer sollte sich jedoch bewusst machen, dass so im Prinzip ein der Raumstruktur angepasster variabler Sensor abgebildet wird, der in heterogenen Raummodellen abhängig von der aktuellen Position in verschiedene Richtungen unterschiedlich weit reicht.

Der Zustand der Umgebung ist definiert als die räumliche Verteilung der Attribute, Objekte und Agenten. Das von der Wahrnehmungsfunktion gelieferte Perzept entspricht dem Zustand des wahrnehmbaren Ausschnitts der Umgebung. Ein Agent erhält somit eine Liste aller wahrnehmbaren Agenten, Objekte und Attribute einschließlich ihrer Positionen. Bei diskreten Raummodellen kann unabhängig von der Existenz eines dort lokalisierten Merkmals auch eine Liste aller wahrnehmbaren Raumelemente, referenziert durch entsprechende Positionsobjekte, erstellt werden.

Ein Agent erhält damit Zugriff auf sämtliche Elemente innerhalb seines Wahrnehmungsbereichs, so dass sein (nicht explizit modellierter) Einflussbereich im Prinzip mit dem Wahrnehmungsbereich identisch ist. Andererseits wird ein Agent nicht daran gehindert, ausgewählte Positionen zur späteren Verfügung beispielsweise in einem internen Umweltmodell zu speichern und so seinen Einflussbereich beliebig zu erweitern. In dynamischen Umgebungen ist es jedoch wahrscheinlich, dass sich der Zustand ohne Einwirken des Agenten verändert, so dass ein Zugriff auf die außerhalb des aktuellen Wahrnehmungsbereichs liegenden Positionen mit einer gewissen Unsicherheit verbunden ist.

Aus dem Autonomiekonzept folgt, dass es in der Verantwortung eines jeden Agenten liegt, wann er seine Umgebung wahrnimmt. Bei zeitgesteuerter Simulation ist dies relativ unproblematisch, da zu jedem Zeitschritt alle Agenten einmal aktiviert werden[10] und damit die Möglichkeit haben, auf die Umgebung zuzugreifen. Für das hier konzipierte Simulationswerkzeug ist dagegen ein ereignisgesteuerter Ansatz für die Fortschaltung der Simulationszeit vorgesehen. Ein Vorteil dieses Ansatzes besteht darin, nicht mehr zu jedem Simulationszeitpunkt alle Agenten aktivieren zu müssen. Andererseits muss weiterhin gewährleistet bleiben, dass ein Agent auch während zeitverbrauchender Aktivitäten, in denen er programmtechnisch passiv ist, immer über ihn betreffende Änderungen in der Umgebung informiert sein kann.

Zu diesem Zweck sind Agenten mit der Fähigkeit ausgestattet, Signale aus der Umgebung zu empfangen. Hat sich im lokalen Wahrnehmungsbereich eines Agenten eine Änderung ergeben, sendet die Umgebung ein entsprechendes Signal. Dies bewirkt, dass der Agent aktiviert und dadurch in seiner aktuellen Handlung unterbrochen wird. Er kann

[10] Von den Problemen, die sich aus der konzeptuellen Gleichzeitigkeit bei sequentieller Ausführung auf einem Rechner ergeben, sei hier einmal abgesehen; diese wurden in Abschnitt 2.1.3 diskutiert.

feststellen, dass die Umgebung ihm ein Signal geschickt hat und daraufhin selbstständig den Zustand der Umgebung überprüfen. Ein solches Signal übermittelt also – im Gegensatz zu einer Nachricht von einem Agenten – keinen Inhalt, sondern nur einen Hinweis darauf, dass der Agent seine Aufmerksamkeit auf die Umgebung richten sollte. Diese Anpassung an die gewählte Ereignissteuerung wird in Abschnitt 4.6 näher erörtert.

4.5.1.2. Handlung

Analog zu den Sensoren werden auch die Effektoren eines Agenten nur implizit repräsentiert: als eine Menge von möglichen Aktionen, mit denen der Agent auf die Umgebung einwirken kann. Diese Aktionen sind ebenso wie die Wahrnehmung lokal beschränkt, beispielsweise auf die aktuelle Position des Agenten. Wie im vorausgehenden Unterabschnitt bereits beschrieben, wird der Einflussbereich eines Agenten nicht explizit modelliert, so dass er in der Regel mit dem Wahrnehmungsbereich übereinstimmt. Durch Aufbau eines internen Raummodells aus den bereits „entdeckten“ Positionen ist es einem Agenten jedoch prinzipiell möglich, seinen Einflussbereich beliebig zu vergrößern.

Da Art und Umfang der einem Agenten zur Verfügung stehenden Aktionen von Modell zu Modell variieren werden, kann ein allgemeines Simulationswerkzeug hier nur eine Auswahl von Basis-Aktionen unterstützen, die sich modellspezifisch ergänzen lassen. Neben dem Versenden von Nachrichten bilden in einer räumlichen Umgebung Aktionen zur Manipulation des lokalen Zustands solche Aktionsprimitive. Der Agent kann an spezifizierten Positionen Objekte ablegen bzw. aufnehmen und Attribute verändern. In einer objektorientierten Umsetzung dieses Konzepts lassen sich diese Aktionen auf entsprechende Methoden abbilden, die ein Umgebungsobjekt anbietet; Ferber bezeichnet dies als *spezialisierte Umwelt* (1999, S. 214).

Versteht man den möglichen Einfluss eines Agenten weitgehender, können sich die möglichen Handlungen eines Agenten auf eine Veränderung der Raumstruktur selbst oder der Umweltdynamik erstrecken. In diesem Fall benötigt ein Agent direkten Zugriff auf das Raummodell bzw. die Umgebungsprozesse. In Anwendungsgebieten wie Ökologie und Verkehr, in denen z. T. die menschlichen Eingriffe in die Umwelt modelliert werden, besteht potentiell Bedarf an solchen Aktionen.

Die direkte Manipulation von Zustandsattributen eines anderen Agenten widerspricht dagegen dem Autonomie-Konzept und ist nicht vorgesehen. Ein Agent hat hier nur Einfluss auf seine eigenen umweltrelevanten Eigenschaften wie z. B. seine aktuelle Position. Die Bewegung im Raum wird mit den in Abschnitt 4.3.2 vorgestellten verschiedenen Strategien besonders unterstützt.

Bewegung im Raum stellt eine zeitverbrauchende Handlung dar und ist auf entsprechende Ereignisse abzubilden. Da die einzelnen Schritte eines elementaren Bewegungsablaufs – Wahl der nächsten Position, Bestimmen der benötigten Zeit und Durchführen der Positionsänderung – modellunabhängig gleichbleiben, können sie in der Fähigkeit zur Bewegung gekapselt werden. Ein Agent, der mit dieser Fähigkeit ausgestattet ist, braucht so nur noch die passende Bewegungsstrategie zu spezifizieren und bei Bedarf das zur Verfügung stehende Aktionsprimitiv aufzurufen. Von der Umgebung erhält er ein entsprechendes Signal, sobald er eine neue Position erreicht hat. Damit ist die Bewegung für einen Agenten transparent auf den ereignisgesteuerten Zeitfortschritt abgestimmt; weite-

re Details dieses Konstrukts werden im Zusammenhang mit der gewählten Zeitrepräsentation (Abschnitt 4.6) erläutert.

4.5.1.3. Kommunikation

Kommunikation zwischen Agenten bedeutet im Kontext von Multiagentensystemen gewöhnlich den Austausch mehr oder weniger komplex strukturierter Nachrichten. Im allgemeineren Sinn von Kommunikation als „Verständigung untereinander“[11] kann die Definition auch auf die indirekte Form des Informationsaustausches über die gemeinsame Umwelt erweitert werden: Ein Agent veranlasst dabei eine Veränderung von Umgebungseigenschaften, welche nachfolgend von anderen Agenten wahrgenommen wird.

Auch wenn Kommunikation demnach prinzipiell einen Spezialfall der Interaktion mit der Umgebung darstellt, so dass der Empfang einer Nachricht als eine Form der Wahrnehmung und das Senden einer Nachricht entsprechend als eine bestimmte Aktion aufgefasst werden könnte, soll hier explizit zwischen Nachrichtenempfang und Wahrnehmung der Umgebung unterschieden werden. Dies erlaubt dem Agenten-Entwickler, bei Bedarf auf eine der beiden Fähigkeiten zu verzichten und so rein kommunikative bzw. rein situierte Multiagentensysteme (Ferber 1999, S. 11) zu realisieren.

Für die Kommunikation stehen einem Agenten Primitive zum Senden und Empfangen einer Nachricht zur Verfügung. Da die Nachrichtenweiterleitung wie in Abschnitt 4.4.2 beschrieben von der Kommunikationsinfrastruktur übernommen wird, wird ein Agent aktiv nur vom Nachrichten-Senden Gebrauch machen. Das allgemeine Nachrichtenformat legt als Bestandteile einer Nachricht nur Sender, Empfänger, beliebigen Inhalt, Sendezeitpunkt und ggf. eine Typ-Kennzeichnung fest. Es kann modellspezifisch erweitert werden, um z. B. KQML-konforme Nachrichten zu verwenden. Sender und Sendezeitpunkt lassen sich automatisch generieren, so dass ein Agent nur Empfänger, Inhalt und Typ spezifizieren muss, wenn er eine Nachricht verschicken möchte.

Jeder Agent besitzt einen Nachrichtenpuffer, in dem eingehende Nachrichten chronologisch sortiert abgelegt werden. Geschieht die Nachrichtenübertragung zeitverzugslos, kann hierfür auf den Sendezeitpunkt zurückgegriffen werden; sonst sollte der Nachricht der Empfangszeitpunkt als weiterer Zeitstempel hinzugefügt werden. Auf diese Weise ist sichergestellt, dass der Agent die Nachrichten in der korrekten Reihenfolge abarbeiten kann.

Der Empfang einer Nachricht ist in der Regel mit einem Simulationsereignis gleichzusetzen, da er mit einer Zustandsänderung des betreffenden Agenten einhergeht. Das bedeutet, dass der Agent aktiviert werden muss, sobald eine Nachricht eintrifft. Sind Sende- und Empfangszeitpunkt identisch, erfolgt so eine Synchronisation der beteiligten Agenten, auch wenn der Sender generell nicht blockiert wird (asynchroner Nachrichtenaustausch). Ist dies unnötig oder unerwünscht, können Nachrichten explizit als „nicht aktivierend“ gekennzeichnet werden. Der Empfänger wird dann frühestens bei seiner nächsten, aus einem anderen Grund vorgemerkten Aktivierung auf eine solche Nachricht reagieren.

[11] siehe Duden Bd. 5, Fremdwörterbuch

4.5.2. Modellierung des Verhaltens

Das Verhalten eines Agenten bestimmt, wie er die zur aktuellen Situation passende(n) Aktion(en) auswählt. Die aktuelle Situation ist gegeben durch den lokal wahrnehmbaren Zustand der Umgebung und seinen eigenen Zustand, der beliebig komplex strukturiert sein kann; die Bandbreite reicht hier von wenigen numerischen Attributwerten bis hin zu internen, nach mentalen Kategorien gegliederten Weltmodellen. Zur Spezifikation des Verhaltens wurden diverse Agenten-Architekturen entwickelt, die jeweils für unterschiedliche Typen von Umgebungen besonders geeignet erscheinen (vgl. Abschnitt 2.3.4).

Da in der agentenbasierten Simulation die Umgebung ebenso wie die Agenten Teil des Modells ist und somit in Abhängigkeit vom Modellzweck variieren wird, reicht es nicht aus, nur eine bestimmte Agenten-Architektur in einem generischen Simulationswerkzeug zu unterstützen. Besser wird hier von einzelnen Architekturen abstrahiert, indem das Verhaltensmodell in einer Komponente gekapselt wird, deren Schnittstelle eindeutig vorgegeben und möglichst minimal gehalten ist. Eine solche Modularisierung ist gängige Praxis beim Software-Entwurf (vgl. z. B. Pomberger und Pree 2004, S. 131ff und Züllighoven und Raasch 2006, S. 816f) und erlaubt in diesem Fall austauschbare Verhaltensbausteine für Agenten und damit Unabhängigkeit von einer konkreten Agenten-Architektur. Über zusammengesetzte Komponenten lassen sich auch nebenläufig aktive Verhaltensmodule erreichen, so dass ein Agent nicht auf einen einzelnen Verhaltensbaustein beschränkt ist.

Da die Frage, ob und wie ein Agent seinen internen Zustand repräsentiert, eng mit der Art der Verhaltensmodellierung gekoppelt ist, wird der Zustand als zur Verhaltenskomponente gehörig betrachtet. Die Schnittstelle ist so ausschließlich durch die Sensoren und Effektoren bestimmt: Die Verhaltenskomponente muss den lokal wahrnehmbaren Zustand der Umgebung und die erhaltenen Nachrichten verarbeiten können sowie alle geplanten oder gewählten Aktionsfolgen auf die dem Agenten zur Verfügung stehenden Aktionsprimitive abbilden.

Die einzige Forderung, die Multiagentenmodelle im Unterschied zu anderen Multiagentensystemen an die Architektur eines Agenten stellen, ist der Bezug zur Simulationszeit, den ein Verhaltensmodell herstellen muss. Es muss nicht nur festlegen, unter welchen zustandsabhängigen Bedingungen ein Agent seine Aktionen ausführt, sondern auch, zu welchen Zeitpunkten dies geschehen soll bzw. welche Zeitdauer die Durchführung einer Aktion in Anspruch nimmt. Zwar kann ein Simulationswerkzeug hier für die Basis-Aktionen wie Aufnehmen von Objekten, Bewegen im Raum oder Versenden von Nachrichten Vorgaben machen, doch sind die Aktionsdauern prinzipiell modellabhängig, so dass sie der Kontrolle des Modellierers unterliegen sollten.

In dem hier vorgestellten Konzept wird ein ereignisgesteuerter Fortschritt der Simulationszeit verwendet, wodurch auch die Modellierung des Agentenverhaltens von Ereignissen gesteuert wird. Diese Ereignisse können ihren Ursprung außerhalb des Agenten besitzen, so dass sie Änderungen in der Umgebung des Agenten oder den Empfang einer Nachricht markieren. Das Verhaltensmodell muss dann auf ein solches Ereignis geeignet reagieren. Genauso können Ereignisse jedoch auch interne Zustandsänderungen des Agenten betreffen: Indem ein Agent auf sich selbst bezogene Ereignisse generiert und vormerkt, lässt sich beispielsweise proaktives Verhalten modellieren.

Auf der Grundlage dieser abstrakten Agentenarchitektur lassen sich konkrete Verhal-

tensbausteine realisieren; im Idealfall unterstützt ein Simulationswerkzeug hier die Modellierung des Verhaltens auf mehreren Formalisierungsstufen, so dass sich beispielsweise eine graphische Verhaltensspezifikation ohne großen Aufwand in eine ablauffähige Verhaltenskomponente übersetzen lässt. In diesem Zusammenhang erscheinen Zustandsautomaten besonders geeignet: Sie sind ein in der Informatik etabliertes Modell, um das Verhalten von Systemen zu beschreiben, die auf Eingaben in Abhängigkeit ihres aktuellen Zustands reagieren, und besitzen eine anschauliche graphische Darstellung. Zwar sind sie ursprünglich nicht zeitbehaftet, so dass nur die Reihenfolge der Eingaben (Ereignisse) eine Rolle spielt, nicht deren Eintrittszeitpunkte, doch lassen sie sich leicht entsprechend erweitern. Zwei Formalismen, die dies leisten, sind die aus dem Kontext der diskreten Simulation stammenden *Event Graphs* (vgl. Schruben 1983; Law und Kelton 2000, S. 57ff), die sich auch für die Abbildung von Petrinetzen eignen (Schruben und Yucesan 1994), und die auf Harel (1987) zurückgehenden *Statecharts*, die im objektorientierten Entwurf als UML-Zustandsdiagramme breite Verwendung zur Verhaltensbeschreibung aktiver Objekte gefunden haben (siehe z. B. Fowler 2004, S. 107ff). Da Multiagentensysteme häufig objektorientiert implementiert werden, werden Zustandsdiagramme und damit die von ihnen repräsentierten Zustandsautomaten auch für die Verhaltensbeschreibung von Agenten verstärkt eingesetzt (vgl. z. B. Griss et al. 2003 für eine Erweiterung des Agentenframeworks JADE[12] und North et al. 2013 für die neueste Version von Repast). Auch zur Beschreibung von Interaktionsprotokollen werden Zustandsdiagramme herangezogen, vorrangig zur Definition von Nebenbedingungen (Odell et al. 2001, S. 135).

Zustandsdiagramme ermöglichen eine Verhaltensmodellierung auf subkognitiver bzw. reaktiver Ebene, da sie kein symbolisch repräsentiertes internes Weltmodell beinhalten und die Reaktion auf Ereignisse beschreiben (vgl. Abschnitt 2.3.4). Durch zustandsabhängige Vormerkung von internen Ereignissen lässt sich jedoch auch proaktives Verhalten hinlänglich abbilden. Ihre anschauliche graphische Spezifikation und die relativ einfache Übersetzbarkeit in ausführbaren Programmcode machen sie daher für viele Anwendungsfälle der agentenbasierten Simulation attraktiv. Darüber hinaus können sie gut auf die Ereignissteuerung abgestimmt werden: Zustände repräsentieren zeitverbrauchende Handlungen, während Transitionen als Zustandsveränderungen durch Ereignisse ausgelöst und zeitverzugslos ausgeführt werden. Aus diesen Gründen bieten sich Zustandsdiagramme als Methode zur Verhaltensbeschreibung innerhalb des hier konzipierten Simulationswerkzeugs zur agentenorientierten Simulation besonders an.

4.6. Repräsentation der Zeit

In dem hier vorgestellten Konzept wird ein ereignisgesteuerter Zeitfortschritt gewählt, um die in Abschnitt 3.2.5 eingeführte agentenorientierte Weltsicht umzusetzen. Damit grenzt sich diese Arbeit gegen die in der agentenbasierten Simulation vorherrschende Verwendung einer Zeitsteuerung ab (vgl. Abschnitte 3.2.2 und 3.2.3).

Ein ereignisgesteuerter Zeitfortschritt stellt gewisse Anforderungen an die Modellierung des dynamischen Verhaltens von Agenten und Umgebung. So ist die Dauer von Aktionen vorherzubestimmen, der Umgebungszustand trotz asynchroner Aktualisierung

[12] `http://jade.tilab.com` [22.1.2014]

der Agenten möglichst effizient aktuell zu halten und eine sofortige Reaktion von Agenten auf Veränderungen der Umgebung zu ermöglichen. Auf diese Probleme und ihre Lösungen im Rahmen des hier vorgestellten Konzepts ist zum Teil weiter oben bei der Beschreibung der Raum- bzw. Agentenmodellierung bereits hingewiesen worden. Hier sollen die gewählten Lösungen noch einmal im Zusammenhang diskutiert werden.

4.6.1. Bestimmung der Aktionsdauer

Die Ereignissteuerung beruht darauf, zeitverbrauchende Handlungen auf Start- und Ende-Ereignisse abzubilden, so dass deren Durchführung einer programmtechnisch passiven Phase entspricht. Damit ein Agent nach der Durchführung einer solchen Handlung wieder aktiviert werden kann, muss die Dauer der Aktion zu ihrem Beginn bekannt sein.[13] Wie in Abschnitt 2.1.3 dargelegt, existieren für die Bestimmung der Aktionsdauer mehrere Möglichkeiten, die sich in Aufwand und Eignung für bestimmte Modelle unterscheiden. Ein allgemeines Simulationswerkzeug für agentenorientierte Modelle kann hier nur grundlegende Unterstützung anbieten, da viele der zu simulierenden Aktionen modellabhängig sind. Zudem kann die Dauer einer Aktion von Modell zu Modell und innerhalb eines Modells von Agent zu Agent variieren. Dies betrifft insbesondere bei deliberativen Agenten auch die Zeit, die der Agent für die Entscheidung über die nächste auszuführende Aktion benötigt.

Für die modellunabhängigen Aktionsprimitive wie Senden einer Nachricht, Wahrnehmen der Umgebung und Manipulation von Attributen oder Objekten, welche das Simulationswerkzeug einem Agenten zur Verfügung stellt, kann dagegen idealerweise eine modell- bzw. agentenspezifische Dauer angegeben werden. Führt ein Agent dann ein solches Aktionsprimitiv aus, merkt die Simulationsinfrastruktur automatisch das zugehörige Reaktivierungsereignis vor. Alternativ ließen sich alle Aktionsprimitive zeitverzugslos durchführen, so dass die Zuordnung einer Dauer über entsprechend langes unbedingtes Warten abgebildet werden müsste.

Unabhängig davon, welche Alternative bei der Realisierung des Framework-Konzepts umgesetzt wird, wird die Bestimmung der Aktionsdauer in beiden Fällen in die Verantwortung des Modellierers gelegt. Ausgenommen ist hiervon nur die Bewegung im Raum, deren Dauer vom Simulationsframework berechnet wird. Die einfachste Lösung besteht für den Modellierer darin, Aktionen generell als zeitverzugslos aufzufassen oder mit einer konstanten Dauer > 0 zu versehen. Der Sonderfall, dass derselbe konstante Wert allen Aktionen zugeordnet wird, ergäbe bei synchron handelnden Agenten automatisch eine Zeitsteuerung mit der konstanten Aktionsdauer als Taktrate. Durch die Integration eines allgemeinen Simulationsframeworks stehen darüber hinaus Zufallszahlenströme und darauf aufbauende stochastische Verteilungen zur Verfügung, um Aktionen eine zufällig bestimmte Dauer zuzuordnen. Aufwendigere zustands- oder zeitabhängige Berechnungen sind dagegen zunächst nicht vorgesehen.

[13] Eine Ausnahme von dieser Regel stellt die Aktion „Warten auf …“ dar: Diese ist keine Aktivität im Sinne der diskreten Simulation, sondern eine sogenannte Verzögerung (vgl. Abschnitt 2.1.3). In diesem Fall muss der Agent nur im Rahmen seines Verhaltensmodell die Bedingung spezifizieren, auf deren Eintreten er wartet.

4.6.2. Sofortige Wahrnehmung von Veränderungen

Konzeptuell beobachtet ein Agent ständig die Umwelt, um auf Veränderungen in seinem Wahrnehmungsbereich sofort reagieren zu können. In der ereignisgesteuerten Simulation kann er jedoch nur in seinen (zeitverzugslosen) aktiven Phasen die Umgebung wahrnehmen, da Zeitverbrauch als programmtechnisch passive Phase realisiert wird. Eine Lösungsmöglichkeit besteht darin, einen Agenten trotz globaler Ereignissteuerung intern mit einem Zeittakt auszustatten, so dass er in regelmäßigen Abständen selbst seine aktuelle zeitverbrauchende Handlung unterbricht, um die Umgebung wahrzunehmen (vgl. z. B. Guessoum 2000). Zwar könnte so jeder Agent mit einem individuellen „Lebensrhythmus“ versehen werden, der zumindest eine quasi-kontinuierliche Umsetzung der ständigen Umweltbeobachtung erlaubt. Doch haftet dieser Lösung derselbe Nachteil der Ineffizienz an wie einer globalen Zeitsteuerung: Schließlich ist eine Wahrnehmung prinzipiell nur dann nötig, wenn auch eine Änderung eingetreten ist.

Effizienter ist daher die hier gewählte Variante, über spezielle Signale aus der Umgebung eine Aktivierung des Agenten zu erreichen. Dazu besitzen Agenten einen Kanal zum Empfang von Signalen, der Signale chronologisch geordnet zwischenspeichert. Für den Zeitpunkt des nächsten bevorstehenden Signals generiert der Agent jeweils automatisch ein Aktivierungsereignis, das er auf der globalen Ereignisliste vormerkt. Sobald sich im lokalen Wahrnehmungsbereich des Agenten eine Änderung ergeben hat, sendet die Umgebung ein entsprechendes Signal und bewirkt so, dass der Agent in der Ausführung seiner aktuellen Handlung unterbrochen wird. Das Signal selbst übermittelt keinerlei Informationen über die Art der Änderung, sondern dient dem Agenten nur als Hinweis, dass sich in der Umgebung etwas verändert hat. Diese Konstruktion stellt de facto einen Kompromiss zwischen Agenten-Autonomie und technischer Notwendigkeit dar.

Die Bestimmung der über eine Zustandsänderung zu informierenden Agenten lässt sich relativ effizient halten, indem die Umgebung die Wahrnehmungsbereiche aller situierten Agenten in Form von speziellen Attributgebieten führt. Diese sind bestimmt durch die aktuelle Position und Sensorreichweite eines Agenten und müssen nur bei einem Positionswechsel oder einer Veränderung der Reichweite neu berechnet werden. Tritt an einer Position im Raum eine Veränderung ein, weil beispielsweise ein Agent diese Position erreicht oder ein Umgebungsprozess einen Attributwert modifiziert, so müssen nur alle Wahrnehmungsbereiche bestimmt werden, die diese Position enthalten. Da jeder Wahrnehmungsbereich seinen zugehörigen Agenten als Attribut enthält, ist damit die Gruppe der zu informierenden Agenten bekannt.

4.6.3. Effiziente Aktualisierung des Umgebungszustands

Ein weiteres Prinzip der Ereignissteuerung ist die asynchrone Aktualisierung der aktiven Entitäten: Nur diejenige Entität berechnet ihren Zustand neu, die vom aktuellen Ereignis betroffen ist. Vorausgesetzt wird dabei, dass der Zustand des gesamten Systems zwischen Ereignissen konstant bleibt. Dies ist dadurch gewährleistet, dass die mit einem Ereignis verbundenen Zustandsänderungen per Definition zeitverzugslos ablaufen, während zeitverbrauchende Handlungen auf eine Folge von Ereignissen abgebildet werden. Die von der Aktion bewirkte Zustandsänderung fällt dabei üblicherweise mit dem Ende der Ak-

tion zusammen (Page 1991, S. 27). Änderungen, die eigentlich kontinuierlich über eine Zeitspanne stattfinden, müssen bei Bedarf in feinerer Auflösung diskretisiert werden.

Für das hier vorgestellte Konzept eines Simulationsframeworks betrifft dies die Bewegung im Raum, die einem mobilen Agenten als Fähigkeit zur Verfügung gestellt wird. Jede Bewegung muss auch bei längeren vorausgeplanten Routen in einzelnen „Schritten", d. h. im diskreten Raum von Raumelement zu Raumelement stattfinden. Die Ankunft an einem Raumelement bedeutet eine Zustandsänderung für den betreffenden Agenten und ist daher mit einem Ereignis verknüpft. Dieses wird automatisch generiert, indem die Umgebung dem Agenten nach Ablauf der intern berechneten Bewegungsdauer ein entsprechendes Signal sendet, das wiederum eine Reaktivierung des Agenten für den Ankunftszeitpunkt bewirkt. Darüber hinaus wird im Raummodell die Position des Agenten bereits nach der Hälfte der zurückgelegten Strecke aktualisiert. So gehört im diskreten Raummodell praktisch jede Kante des zugrundeliegenden Graphen halb zum Start- und halb zum Endknoten. Dies ist eine naheliegende Aufteilung und findet in (regelmäßigen) Gittern die anschauliche Entsprechung, dass exakt an dieser Stelle der Wechsel von einer Zelle zur nächsten erfolgt.

Jeder Agent, der eine Kante durchläuft, bleibt solange dem Startknoten zugeordnet, bis er die Mitte der Kante erreicht hat. Daraufhin wird er sofort dem Endknoten zugeordnet. Für einen anderen Agenten ist er ab diesem Zeitpunkt als bereits am Endknoten befindlich wahrnehmbar. Mit diesem Kunstgriff lässt sich die Aktualisierung der Umgebung effizient halten: Zu jedem Zeitpunkt ist sichergestellt, dass die Positionen (Knoten) der Agenten in der Umgebung ausreichend korrekt bekannt sind. Ein Agent, der sich noch auf der Kante vor einem Knoten bzw. bereits auf der Kante aus dem Knoten heraus befindet, gilt quasi als in „Rufnähe" des Knotens. Diese Ungenauigkeit wird dadurch aufgewogen, dass vor einem Zugriff eines Agenten auf die Umgebung deren Zustand nicht neu berechnet werden muss.

Das dynamische Verhalten der Umgebung selbst muss nicht speziell an die Ereignissteuerung angepasst werden, da es durch Simulationsprozesse oder andere Konstrukte der klassischen Weltsichten modelliert wird. Für solche Entitäten ist bereits gewährleistet, dass ihr Zustand zwischen Änderungen konstant ist und dass Änderungen zeitverzugslos stattfinden.

5. Realisierung und Test des Simulationsframeworks FAMOS

We become what we behold. We shape our tools
and then our tools shape us.
– Marshall McLuhan

Das *Framework für agentenorientierte Modellierung und Simulation* (FAMOS) setzt das in Kapitel 4 beschriebene Konzept um. FAMOS wurde in Zusammenarbeit mit Nicolas Denz (geb. Knaak) implementiert, der im Rahmen seiner Diplomarbeit (Knaak 2002) insbesondere die Konstrukte und Werkzeuge für die Verhaltensmodellierung der Agenten entwickelte.

Wie im Konzept vorgesehen, baut FAMOS auf vorhandener Simulationssoftware auf. Hierzu wurde das Simulationsframework DESMO-J ausgewählt, dessen Funktionalität und Integration mit FAMOS im folgenden Abschnitt 5.1 erläutert wird. Im Anschluss daran wird FAMOS im Detail vorgestellt (Abschnitt 5.2). Die Nutzung des Frameworks wird anhand zweier klassischer agentenbasierter Modelle demonstriert, die in FAMOS reimplementiert wurden: Schellings (1978) Segregationsmodell (Abschnitt 5.3) und Epstein und Axtells (1996) *Sugarscape*-Modell (Abschnitt 5.4). Über diese eher einfachen Modelle hinaus konnte FAMOS in einer umfangreichen, praxisrelevanten Anwendung eingesetzt und evaluiert werden (siehe Kapitel 6).

5.1. DESMO-J als zugrundeliegendes Simulationsframework

Die Integration vorhandener Simulationssoftware in ein Werkzeug für agentenbasierte Simulation hat den Vorteil, dass die reinen Simulationsaspekte nicht erneut implementiert werden müssen. Darüber hinaus eröffnet sich so die Möglichkeit, bereits bestehende Modellkomponenten in Multiagentenmodelle zu integrieren. Aufgrund der in Abschnitt 3.1.3 diskutierten konzeptuellen Nähe von (aktivem) Objekt und Agent eignen sich besonders objektorientierte Simulationssysteme zur Erweiterung um agentenbasierte Konstrukte. Mit dem am Fachbereich Informatik der Universität Hamburg entwickelten DESMO-J stand ein solches Simulationsframework zur Verfügung.

Verwendet – und im folgenden beschrieben – wurden die im Zeitraum der Entwicklung von FAMOS verfügbaren Versionen 1.4 bis 1.6, welche sich sowohl in Package-Struktur als auch Funktionsumfang von den neueren Versionen 2.*x* deutlich unterscheiden.[1] Die im Zuge der Entwicklung von FAMOS vorgenommenen Erweiterungen von DESMO-J (vgl. Abschnitt 5.1.2.1) haben wesentlich zu dem Versionssprung von 1.6 auf 2.0 beigetragen.

[1] vgl. `http://desmoj.sourceforge.net/version_history.html` [24.1.2014].

5.1.1. Überblick über DESMO-J

DESMO-J ist ein objektorientiertes Framework für zeitdiskrete Simulation (Page et al. 2000; Meyer et al. 2005b); die Abkürzung steht für **D**iscrete-**E**vent **S**imulation and **Mo**delling in **J**ava. Es ist als Simulationssoftware auf der Ebene der Simulationspakete einzuordnen (Page 1991, S. 159), da es die Basissprache Java um simulationsspezifische Konstrukte erweitert. In die Entwicklung von DESMO-J sind langjährige Erfahrungen mit der Realisierung von Simulationspaketen eingeflossen: Das 1989 in der Programmiersprache Modula-2 erstellte Paket DESMO (vgl. Page 1991, Kap. 7) bildet die konzeptionelle Grundlage. Dieses Paket wurde im Rahmen von studentischen Arbeiten erweitert und in verschiedene Programmiersprachen portiert, wobei die zuletzt erfolgten Portierungen nach C++ (Schniewind 1998) und Java (Lechler 1999; Claassen 1999) die Möglichkeiten der Objektorientierung gezielt nutzten.

Das Framework DESMO-J hat sich in der Lehre als didaktisches Hilfsmittel für die Vermittlung der wichtigsten Simulationskonzepte bewährt. Neben der klaren, durchdachten Struktur fördern die ausführliche Dokumentation und ein WWW-basiertes Tutorial (vgl. Neidhardt 2000 für dessen erste, Meyer et al. 2005a, Kap. 4 für dessen zweite Version) seine leichte Erlernbarkeit. Da zudem der Quelltext frei zugänglich ist, bietet DESMO-J eine gute Grundlage für Erweiterungen.

5.1.1.1. Unterstützung der Modellierung

DESMO-J unterstützt die ereignis- und die prozessorientierte Weltsicht der diskreten Simulation durch die Bereitstellung abstrakter Klassen für Ereignisse (`Event`), Entitäten (`Entity`) und Prozesse (`SimProcess`). Diese sogenannten Hot-Spot-Klassen (vgl. Pree 1997) sind durch Spezialisierung auf die modellspezifischen Anforderungen anzupassen. Dabei wird das Verhalten der Modellbestandteile spezifiziert, indem die jeweiligen abstrakten Methoden realisiert werden. Im ereignisorientierten Ansatz bedeutet dies, die Ereignisroutinen (Methode `eventRoutine()`) zu implementieren, im prozessorientierten Ansatz, den Lebenszyklus der Prozesse (Methode `lifeCycle()`) zu implementieren.

Auch die Kombination beider Weltsichten in einem Modell ist möglich, basierend auf der Vererbungsbeziehung zwischen Simulationsprozess und Entität. Dies ermöglicht nicht nur die gemeinsame Verwendung von Prozessen und Entitäten, sondern auch die Manipulation von Prozessen über Ereignisse, also die Behandlung von Prozessen als einfache Entitäten, und erhöht so die Flexibilität für den Modellierer. Vorteilhaft ist dies beispielsweise bei der Modellierung von unregelmäßig und/oder selten auftretenden Störfällen im prozessorientierten Ansatz: So kann in einem prozessorientierten Modell einer Fertigungsanlage der plötzliche Ausfall einer Maschine leicht über ein entsprechendes Ereignis realisiert werden.

Neben diesen zentralen Modellkomponenten, deren dynamisches Verhalten die Zustandsänderungen im Modell bewirkt, stellt DESMO-J eine Reihe weiterer, unterstützender Modellkomponenten zur Verfügung: Warteschlangen, Zufallszahlenströme mit verschiedenen Verteilungsfunktionen und Klassen für die Aufzeichnung und Auswertung von Simulationsergebnissen. Die Warteschlangen, die sowohl in der ereignisorientierten als auch in der prozessorientierten Weltsicht das elementare Hilfsmittel zur Synchronisation

nebenläufig agierender Entitäten darstellen, sind mit diversen Methoden zum Einfügen bzw. Entnehmen von Entitäten ausgestattet und generieren typische statistische Größen zur Leistungsbeurteilung.

Aufbauend auf Warteschlangen werden höhere Synchronisationskonstrukte bereitgestellt, mit denen Wechselwirkungen zwischen Prozessen auf einer höheren, problemorientierteren Ebene modelliert werden können. Dies sind Mechanismen für Ressourcenwettbewerb, Produzenten/Konsumenten-Beziehungen, bedingtes Warten und direkte Prozesskooperation. Bei den beiden ersten Konstrukten tritt eine Synchronisation erst bei Engpässen durch überhöhte Nachfrage ein und kann mehr als zwei Prozesse gleichzeitig betreffen. Ähnliches gilt auch für das bedingte Warten: Eine Synchronisation erfolgt nur, wenn die Bedingung, auf die gewartet wird, nicht erfüllt ist, und ein Prozess kann sich mit beliebig vielen anderen Prozessen koordinieren. Im Gegensatz dazu findet bei der direkten Prozesskooperation ein sogenanntes Rendezvous zwischen genau zwei Prozessen statt, bei dem gemeinsame Aktivitäten ausgeführt werden können. Einer der beiden Prozesse agiert dabei als aktiver Partner (*Master*), der andere als passiver (*Slave*); es handelt sich also um eine asymmetrische Beziehung.

Mit Hilfe der Konstrukte für bedingtes Warten, direkte Prozesskooperation und Ressourcen lassen sich darüber hinaus der aktivitätsorientierte bzw. transaktionsorientierte Modellierungsstil nachbilden. Zusätzlich zu den beschriebenen Mechanismen steht mit der Unterbrechung von Prozessen eine sehr einfache Form der direkten Prozesskommunikation zur Verfügung. Hierbei wird dem (programmtechnisch inaktiven) unterbrochenen Prozess die Ursache für die Unterbrechung übermittelt, jedoch nur in Form einer in einem Objekt gekapselten ganzzahligen Konstante (`InterruptCode`). Für jede unterschiedlich zu behandelnde Unterbrechungsursache ist im Modell vorher ein eindeutiger InterruptCode zu definieren. Außerdem ist die Behandlung von Unterbrechungen explizit in den Lebenszyklus der betroffenen Prozesse aufzunehmen.

Unabhängig von der zugrundegelegten Weltsicht unterstützt DESMO-J eine rudimentäre hierarchische Modellierung, indem Modelle als Komponenten in anderen Modellen wiederverwendet werden können. Dies wird erreicht über die nach dem Kompositum-Entwurfsmuster (Gamma et al. 1994, S. 163ff) gestaltete Beziehung zwischen Modellkomponente und Modell. Jedes Modell ist aus Komponenten zusammengesetzt, wobei Entitäten, Prozesse, Warteschlangen und statistische Verteilungen die Basis-Komponenten bilden.

5.1.1.2. Unterstützung der Simulation

In DESMO-J wird eine strikte Trennung von Modell und Experiment verfolgt. Die Simulationsinfrastruktur (Scheduler, Ereignisliste, Simulationsuhr) wird dabei im Experiment gekapselt, ist für Modellkomponenten aber bei Bedarf über das zugehörige Modell zugreifbar. Dies wird durch eine dynamische Verknüpfung zwischen Modell- und Experiment-Objekt erreicht, die zu Beginn der Simulation vorzunehmen ist. Erzeugung und Steuerung eines Experiments liegen ebenfalls in der Verantwortung des Anwenders, der dazu vordefinierte Methoden der Klasse `Experiment` in bestimmter Reihenfolge aufrufen muss. Die für stochastische Modelle wichtige Durchführung mehrerer Simulationsläufe mit unterschiedlichen Startwerten für die Zufallszahlenströme lässt sich programmtech-

nisch am einfachsten über eine Schleife realisieren; dabei wird pro Simulationslauf ein Experiment-Objekt benötigt. Alternativ unterstützt DESMO-J die Anwendung des Batch-Mittelwert-Verfahrens, indem es zulässt, Experimente zu unterbrechen. Während einer solchen Unterbrechung können dann die bisher generierten Ergebnisse ausgegeben und die statistischen Zähler zurückgesetzt werden. Eine graphische Oberfläche für die Durchführung von Simulationsexperimenten ist nicht Bestandteil von DESMO-J.

Die Fortschreibung der Simulationszeit geschieht ereignis-gesteuert, wobei die Vormerkung eines Prozesses zur (erneuten) Aktivierung als internes Ereignis betrachtet wird. Die auf der Ereignisliste eingetragene Ereignisnotiz enthält neben dem Zeitstempel entweder Verweise auf eine Entität in Verbindung mit einem Ereignis oder auf einen Simulationsprozess. Auf diese Weise kann der Scheduler entscheiden, ob die Ereignisroutine aufzurufen oder die Lebenszyklus-Methode eines Prozesses zu aktivieren ist. Die Ereignisliste ist aufsteigend nach Ereigniszeitpunkten sortiert. Ein neues Ereignis wird in der Regel direkt vor das Ereignis mit dem nächstgrößeren Zeitpunkt eingeordnet, so dass zeitgleiche Ereignisse in der Reihenfolge ihres Eintrags vorliegen. Abweichungen von dieser Regel sind möglich, indem explizit das Einsortieren direkt hinter (Methode `scheduleAfter()`) oder vor (Methode `scheduleBefore()`) einem bereits vorgemerkten Ereignis angefordert wird. Über die Angabe des speziellen Simulationszeitpunkts `NOW` ist auch ein verdrängendes Vormerken realisierbar.

Die für die prozessorientierte Weltsicht benötigte Nebenläufigkeit der Prozesse wird – da Java nicht über einen Koroutinen-Mechanismus verfügt – durch Threads realisiert. Jeder Simulationsprozess ist mit einem eigenen Thread verknüpft, innerhalb dessen seine `lifeCycle()`-Methode ausgeführt wird. Auf diese Weise wird der Lebenszyklus unterbrechbar, so dass die Kontrolle während der simulationszeitkonsumierenden Phasen an andere Prozesse abgegeben werden kann. Um die Reihenfolge der Ausführung gemäß der Einträge auf der Ereignisliste sicherzustellen, muss eine zusätzliche Thread-Steuerung implementiert werden, die einem Koroutinen-Mechanismus entspricht. Hierfür wird das Monitor-Konstrukt von Java (durch `synchronized` spezifizierte kritische Abschnitte) in Verbindung mit den Methoden `wait()` und `notify()` verwendet und in den Methoden zum Vormerken und Passivieren eines Simulationsprozesses gekapselt (Page et al. 2000, S. 153ff). Fasst man den Scheduler selbst ebenfalls als einen Prozess auf, der in seinen aktiven Phasen die Simulationszeit aktualisiert und die Behandlung des nächsten Ereignisses anstößt, so ist auf diese Weise zu jedem Zeitpunkt immer nur genau ein Prozess aktiv. Auf die Problematik von möglichen Verklemmungen, insbesondere bei den höheren Synchronisationskonstrukten, soll hier nicht weiter eingegangen werden; dieses Thema wird ausführlich in (Claassen 2001) behandelt.

5.1.1.3. Unterstützung der Analyse

Der Schwerpunkt von DESMO-J liegt auf der Unterstützung der Modellierung, wie es bei Simulationssoftware auf Sprachebene üblich ist (vgl. Page 1991, S. 158f). Wie schon für die Durchführung von Experimenten wird auch für die Analyse der Ergebnisse nur Basis-Funktionalität bereitgestellt. In diesem Fall beschränkt sich DESMO-J auf die Sammlung und Berechnung grundlegender statistischer Kennzahlen über das Verhalten der Modellkomponenten. Die Erstellung dieser Statistiken erfolgt automatisch für alle modellunab-

hängigen Komponenten (Warteschlangen, Verteilungen). Für die Berechnung zeitgewichteter und nicht zeitgewichteter Statistiken über charakteristische Größen modellspezifischer Komponenten wie Entitäten und Prozesse stellt DESMO-J spezielle Datensammelobjekte zur Verfügung. Diese können durch Anwendung des Beobachter-Entwurfsmusters (Gamma et al. 1994, S. 293ff) ebenfalls automatisch arbeiten – unter der Voraussetzung, dass der Anwender einen entsprechenden Wertlieferanten spezifiziert – oder müssen im Verlauf der Simulation explizit mit aktuellen Werten versorgt werden, indem ihre Methode `update()` aufgerufen wird.

Die berechneten Kennzahlen werden im Ergebnisreport ausgegeben. Zusätzlich werden simulationsbegleitend drei Dateien generiert für die Ablaufverfolgung (*Trace*), die Protokollierung der Manipulationen von Ereignisliste und Warteschlangen (*Debug*) sowie die Aufzeichnung von Fehlermeldungen (*Error*). Dies dient der Unterstützung bei Verifikation und Validierung des Modells. Trace und Debug müssen dabei nicht über den gesamten Simulationslauf erfolgen, sondern können zu Beginn eines Experimentes auf bestimmte Zeiträume eingeschränkt werden.

Die Aufzeichnung der Daten geschieht über ein internes Nachrichtensystem, das als Teil der Simulationsinfrastruktur im Experiment gekapselt ist. Ein Nachrichtenverteiler leitet die ankommenden Nachrichten an die zuständigen Empfänger weiter; diese Empfänger müssen sich dazu für eine oder mehrere Klassen von Nachrichten beim Verteiler registrieren. Unterschieden werden einfache Nachrichten (`Message`), die im Wesentlichen eine Zeichenkette transportieren und in DESMO-J für die Ausgabe von Fehler-, Debug- und Trace-Meldungen verwendet werden, und strukturierte Nachrichten (`Reporter`), welche die gesammelten Informationen über ein zugeordnetes reportfähiges Objekt wie z. B. eine Warteschlange für den Ergebnisreport enthalten. Vordefinierte Empfänger sind die vier Ausgabekanäle für den Report und die Protokolldateien.

Die Ausgabe erfolgt jeweils im HTML-Format, das eine einfache tabellarische Darstellung ermöglicht. Nachteilig ist, dass dieses Format nicht direkt in Datenbanken oder Tabellenkalkulationsprogramme eingelesen werden kann. Die Ausgabe in beliebige Dateien ist aber bereits im Framework angelegt und muss vom Anwender nur noch für seine spezifischen Anforderungen implementiert werden. Eine Visualisierung von Ergebnissen, beispielsweise Zeitreihen von Zustandsgrößen, wird von DESMO-J nicht geboten.

5.1.1.4. Bewertung

Als Grundlage für ein Werkzeug zur agentenorientierten Simulation ist ein leistungsfähiges Simulationspaket wie DESMO-J, das die diskrete Simulation unterstützt, gut geeignet. Wie bereits in Kapitel 4.2 erörtert, bietet sich die Verwendung solcher Simulationssoftware an, da deren allgemeine Simulationsfunktionalität direkt übernommen werden kann. Darüber hinaus lassen sich so auch vorhandene Modellkomponenten in agentenbasierte Modelle integrieren. Der mit einem universell einsetzbaren Simulationspaket einhergehende Verzicht auf einen spezifischen Anwendungsbereich bedeutet zwar einen größeren Aufwand für den Anwender, auf der anderen Seite aber größere Flexibilität (Page 1991, S. 158ff).

Die Eignung von Java als Basissprache für Simulationspakete wird von Page, Lechler und Claassen (2000, S. 23ff) sowie Page, Knaak und Meyer (2005a, S. 144ff) ausführ-

lich diskutiert. Als Vorteile sind die Plattformunabhängigkeit, der auf die wesentlichen objektorientierten Konzepte beschränkte Sprachumfang, die Robustheit und nicht zuletzt die weite Verbreitung zu nennen, die sich auch in der Verfügbarkeit zahlreicher unterstützender Bibliotheken niederschlägt. Diese Argumente sprechen ebenso für den Einsatz von Java bei der Entwicklung von Multiagentenmodellen.

Demgegenüber stehen Defizite in der Laufzeiteffizienz der interpretierten Sprache, die bei der Ausführung komplexer Simulationsmodelle spürbar werden können. Ein Performanzproblem stellt insbesondere die Verwendung von Threads für Simulationsprozesse dar: Aufgrund des Verwaltungsaufwands zur Erzeugung, Steuerung und Vernichtung der Threads benötigt ein prozessorientiertes Modell in DESMO-J etwa doppelt so viel Rechenzeit wie ein äquivalentes ereignisorientiertes Modell (Page et al. 2000, S. 162). Dies ist einer der Gründe, warum in FAMOS die Agenten nicht als Erweiterung von Simulationsprozessen, sondern von Entitäten implementiert sind – auch wenn ein Agent konzeptuell als spezieller Simulationsprozess aufgefasst werden kann (vgl. Abschnitt 3.1.3).

DESMO-J besitzt eine ausgereifte Struktur, die bereits in vielen Anwendungen erprobt worden ist. Dies sowie die umfangreiche Dokumentation und Verfügbarkeit des Quelltextes begründen die leichte Erweiterbarkeit. Weitere Vorteile sind die Unterstützung mehrerer Weltsichten im Rahmen der Modellierung, die explizite Trennung von Modell und Experiment und die ereignisgesteuerte Fortschreibung der Simulationszeit, die allgemeiner ist als der zeitgesteuerte Ansatz (siehe Abschnitt 2.1). Einzig eine adäquate Strategie für die Behandlung zeitgleicher Ereignisse – wie beispielsweise eine zufällige Bearbeitungsreihenfolge – ist noch hinzuzufügen.

Für ein Simulationspaket typische Schwächen weist DESMO-J bei der Unterstützung der Experimentdurchführung und der Ergebnisanalyse auf; so fehlt bisher eine graphische Benutzungsoberfläche. Da die Beobachtungsgrößen in den traditionellen Anwendungsgebieten diskreter Simulation in der Regel auf Systemebene angesiedelt sind, konzentriert sich die Statistikfunktionalität auf die Berechnung aggregierter Kennzahlen. Zum Beispiel werden in einem Bedien-/Wartesystem üblicherweise die mittlere Verweilzeit aller Kunden oder die mittlere Auslastung der Bediener betrachtet, während individuelle Merkmale einzelner Kunden oder Bediener keine Rolle spielen. Für mikroskopische Fragestellungen sind sie dagegen relevant, so dass für die agentenbasierte Simulation entsprechende Datensammelobjekte zu ergänzen sind.

5.1.2. Integration mit FAMOS

Dieser Abschnitt gibt einen kurzen Überblick, auf welche Weise das Framework für agentenorientierte Modellierung und Simulation (FAMOS) auf DESMO-J aufbaut. Eine ausführliche Beschreibung der internen Struktur von FAMOS wird im folgenden Unterkapitel 5.2 geleistet. Der Schwerpunkt in diesem Abschnitt liegt auf den verwendeten Klassen von DESMO-J und den zusätzlichen Erweiterungen, die im Zuge der Entwicklung von FAMOS durchgeführt wurden. Funktionalität, die sowohl für die agentenorientierte als auch für andere Weltsichten der diskreten Simulation nutzbar ist, wurde direkt in DESMO-J integriert. Dies betrifft eine rudimentäre graphische Oberfläche für die Durchführung von Experimenten und die simulationsbegleitende Visualisierung von Beobachtungsgrößen, eine Variante der Ereignisliste mit zufälliger Abarbeitung gleichzeitiger Ereignisse, Klas-

sen zur Repräsentation physikalischer Größen mit Einheiten und die Möglichkeit, Simulationszeit mit wahrer Zeit zu verknüpfen.

5.1.2.1. Erweiterungen von DESMO-J

In vielen Anwendungsbereichen werden physikalische Größen für Attribute von modellierten Entitäten benötigt; so zum Beispiel in der Verkehrssimulation für aktuelle und maximale Geschwindigkeiten von Fahrzeugen oder in biologischen Modellen für das aktuelle Gewicht (Biomasse) von Individuen einer Population. Aus Mangel an geeigneten Datentypen werden hierfür oft reelle Zahlen eingesetzt und die verwendete Einheit implizit vorausgesetzt. Um eine Umrechnung zwischen verschiedenen Größen zu ermöglichen und gleichzeitig die Kompatibilität von Modellkomponenten zu erhöhen, ist es notwendig, diese Größen mit Einheiten zu versehen. Das Paket `desmoj.app` stellt solche speziellen Datentypen für häufig benötigte physikalische Größen zur Verfügung und kann um weitere anwendungsspezifische Datentypen erweitert werden. Die bisher unterstützten Einheiten sind in einer nicht-instantiierbaren Hilfsklasse namens `Units` zusammengefasst, die auch Konstanten für die Umrechnung und Bezeichner für die Einheiten exportiert.

Das Framework FAMOS verwendet die Datentypen für Länge und Geschwindigkeit, um Distanzen im Raummodell bzw. die Dauer einer Bewegung zu berechnen. Letzteres benötigt zusätzlich eine Abbildung der wahren Zeit auf die Simulationszeit, die mindestens der Simulationszeit eine Zeiteinheit zuordnet. Die in DESMO-J integrierte Klasse `TimeConverter` nimmt mittels der vom Anwender spezifizierten Abbildung die Umrechnung zwischen den verschiedenen Zeiten vor, wobei sowohl relative Zeitintervalle (Zeitdauern) als auch absolute Zeitpunkte konvertierbar sind. Da in DESMO-J die Klasse `Experiment` – gemäß dem Entwurfsmuster Fassade (Gamma et al. 1994, S. 185ff) – als einzige Schnittstelle zur Simulationsinfrastruktur dient, ist auch die Zeitumrechnung nur über diese Schnittstelle zugänglich; die Klasse `Experiment` wurde um entsprechende Methoden erweitert.

Ebenso wurde die Simulationsinfrastruktur ergänzt um die Möglichkeit, zeitgleiche Ereignisse in zufälliger Reihenfolge abzuarbeiten. Dies ist die verbreitetste (weil einfachste) Strategie, um Artefakte zu vermeiden, die sich aus einer immer gleichen Aktualisierungsreihenfolge ergeben könnten (vgl. Abschnitt 2.1.3). Auch in diesem Fall ist die betreffende Klasse `RandomizedEventVector`, die diese Strategie kapselt, nur über die Experiment-Klasse benutzbar, indem deren Methode `randomizeConcurrentEvents()` aufgerufen wird.

Darüber hinaus wurde DESMO-J um eine prototypische graphische Benutzungsoberfläche erweitert (Paket `desmoj.gui`). Diese dient dazu, das Starten von Experimenten zu vereinfachen und das simulationsbegleitende Beobachten modellspezifischer Zustandsgrößen zu ermöglichen. Die in DESMO-J integrierte Funktionalität beschränkt sich bei letzterem auf die Darstellung des zeitlichen Verlaufs numerischer Größen; in FAMOS wurden zusätzlich spezielle graphische Beobachter zur Visualisierung der räumlichen Dynamik in agentenbasierten Modellen realisiert (vgl. Abschnitt 5.2.5). Einen Screenshot zeigt Abbildung 5.17 auf Seite 167.

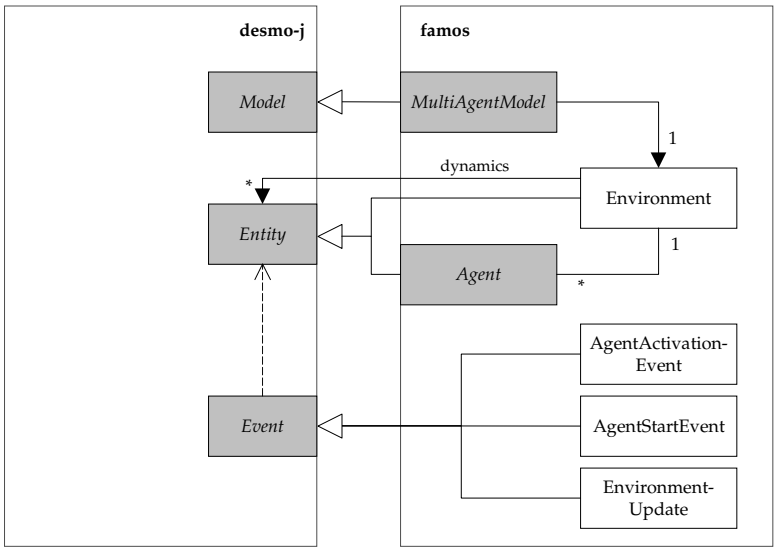

Abbildung 5.1.: Die Schnittstelle zwischen DESMO-J und FAMOS im Bereich der Modellierung. Hot-Spot-Klassen sind grau hervorgehoben.

5.1.2.2. Schnittstelle zu FAMOS

Im Bereich der Modellierung baut FAMOS ausschließlich auf den spezifizierten Hot-Spot-Klassen von DESMO-J auf, so dass eine wohldefinierte Schnittstelle existiert. Einige der abgeleiteten Klassen dienen wiederum als *hot spots* in FAMOS wie in Abbildung 5.1 dargestellt.

Die Klasse `MultiAgentModel` erweitert die von DESMO-J bereitgestellte Modell-Klasse um die für ein Multiagentensystem benötigte Umgebung (Klasse `Environment`). Diese beinhaltet die Kommunikationsinfrastruktur für die Agenten sowie ggf. ein Raummodell und verwaltet eine Liste aller Agenten. Da sowohl das Raummodell als auch Art und Anzahl der Agenten modellspezifisch sind, können diese Modellkomponenten nicht bereits vom Framework für die Klasse `MultiAgentModel` vordefiniert werden. Sie müssen vom Anwender in seiner Spezialisierung dieser Hot-Spot-Klasse ergänzt werden durch Implementation der von `desmoj.Model` geerbten abstrakten Methoden `init()` und `doInitialSchedules()` sowie entsprechender Konstruktoren.

Als technische Grundlage für die Modellierung von Agenten und Umgebung dienen die ereignisorientierten Konstrukte `Entity` und `Event`. Auch wenn Agenten konzeptuell – wie in Kapitel 3.1.3 diskutiert – als Erweiterung von Simulationsprozessen aufgefasst werden können, ist zum einen aus Effizienzgründen von einer Spezialisierung der Klasse `SimProcess` abzuraten; der durch die Verwendung von Threads für die Koroutinensteuerung verursachte Verwaltungsaufwand kann insbesondere bei Modellen mit einer großen Anzahl von Agenten erheblich sein. Zum anderen ist der Prozessen inhärente lineare Ablauf ihrer Aktivitäten nur schlecht geeignet, um reaktives Verhalten von Agenten zu modellieren. Zwar steht mit dem Interrupt-Konstrukt eine Möglichkeit zur Verfügung,

den Prozessablauf zu unterbrechen, doch ist dieses für Ausnahmefälle ausgelegt (vgl. Abschnitt 5.1.1) und somit für die Abbildung der fortlaufenden Reaktion eines Agenten auf Umweltereignisse unpassend.

Die Implementation auf Basis der ereignisorientierten Konstrukte ist dagegen sowohl technisch einfacher zu realisieren als auch in der Ausführung effizienter. Vor dem Anwender bleibt sie verborgen: Die Verhaltensbeschreibung erfolgt dem Autonomie-Konzept entsprechend ganz aus Sicht des Agenten und wird in einem speziellen Verhaltensobjekt gekapselt (siehe Abschnitt 5.2.3). Auf diese Weise ist es möglich, unterschiedliche Verhaltensmodelle in FAMOS zu unterstützen.

Für die Modellierung von Umweltdynamik wird dagegen direkt auf die ereignis- und prozessorientierte Weltsicht zurückgegriffen. Vorgänge, die aktiv den Zustand der Umgebung manipulieren wie die Ausbreitung von Pheromonen oder die Regeneration von Ressourcen, müssen nicht durch Agenten abgebildet werden, sondern können als Simulationsprozesse oder einfache Entitäten realisiert werden. Die Klasse `Environment` referenziert hier die gemeinsame Oberklasse `Entity`, so dass es dem Anwender überlassen bleibt, ob er dynamische Prozesse in der Umgebung ereignis- bzw. prozessorientiert implementiert oder sich doch für Agenten entscheidet.

Das in DESMO-J für die Datensammlung und -ausgabe realisierte Nachrichtensystem wurde evaluiert im Hinblick auf seine Eignung, als Grundlage der Kommunikationsinfrastruktur für Agenten zu dienen. Zwar ist es einer Spezialisierung gegenüber offen, doch verhindern die folgenden Punkte eine konzeptuell und implementationstechnisch überzeugende Nutzung:

- Das Nachrichtensystem ist in DESMO-J Bestandteil eines Experiments, nicht eines Modells. Die Kommunikationsinfrastruktur für Agenten stellt dagegen eine zentrale Modellkomponente dar, die von einem speziellen Experiment unabhängig sein sollte. Zwar ist es technisch möglich, die Kommunikationsinfrastruktur dennoch auf dem Nachrichtensystem aufzubauen, doch sind dazu Konstruktionen nötig, die einem sauberen Framework-Entwurf zuwiderlaufen: Entweder wird es in die Verantwortung des Anwenders gelegt, ob die Kommunikationsinfrastruktur korrekt initialisiert wird, oder es muss in die von DESMO-J vorgegebene Architektur eingegriffen werden, indem der Ablauf bei der Verknüpfung von Modell und Experiment verändert wird.

  Da in DESMO-J die Simulationsinfrastruktur einschließlich des Nachrichtensystems für ein Modell erst nach dessen Verknüpfung mit einem Experiment zur Verfügung steht, ist die Integration eines Nachrichten-Verteilers für Agenten frühestens innerhalb der `connectToExperiment()`-Methode des Modells möglich, mit der die Verknüpfung durchgeführt wird. Im Anschluss an die Verknüpfung von Modell und Experiment wird in dieser Methode noch die Initialisierung des Modells angestoßen, indem die abstrakt deklarierte Methode `init()` aufgerufen wird, die in anwendungsspezifischen konkreten Modellklassen zu realisieren ist. Die Einbindung der Kommunikationsinfrastruktur in das Nachrichtensystem sollte vor der Modellinitialisierung erfolgen, damit sie bei der Initialisierung von Modellkomponenten (Agenten) bereits zur Verfügung steht. Deshalb muss entweder die Methode `connectToExperiment()` entsprechend verändert werden (in DESMO-J selbst

oder in FAMOS in der abgeleiteten Klasse `MultiAgentModel`) oder es muss dem Anwender überlassen werden, die Einbindung selbst vorzunehmen (durch Aufruf einer bestimmten Methode von `MultiAgentModel` als erste Anweisung in seiner Implementation der `init()`-Methode bzw. durch Verwenden einer spezialisierten Experiment-Klasse).

Beide Vorgehensweisen wurden als inakzeptabel bewertet und daher verworfen.

- In DESMO-J beruht die Weiterleitung der Nachrichten auf Nachrichtentypen (Klassen), nicht expliziten Empfängern. Ein Empfänger meldet sich beim Nachrichtenverteiler für eine oder mehrere Nachrichten-Klassen an und erhält dann jede beim Verteiler eingehende Nachricht der entsprechenden Klasse(n) zugesendet. Für den Nachrichtenaustausch zwischen Agenten ist dagegen die Möglichkeit, bestimmte Empfänger zu adressieren, unerlässlich. Die Weiterleitung von Nachrichten auf Basis von Klassen ist dennoch für die Agenten-Kommunikation nutzbar, indem ein spezieller Nachrichtenverteiler konstruiert wird, der sich als Empfänger für die Oberklasse aller Agenten-Nachrichten beim DESMO-J-Nachrichtenverteiler anmeldet und die Weiterleitung der Nachrichten an die richtigen Agenten selbst übernimmt. Die Verwendung eines solchen speziellen Nachrichtenverteilers ist aus den unter Punkt 1 genannten Gründen jedoch abzulehnen.

- Das Nachrichtensystem ist in DESMO-J nicht nur für einfache Nachrichten, sondern auch für die komplexeren Reporter ausgelegt, die gesammelte Daten für den Ergebnisreport aufbereiten. Die von Empfängern zu implementierende Schnittstelle umfasst daher den Empfang sowohl von Nachrichten als auch von Reporter-Objekten. Da letztere für die Agenten-Kommunikation keine Rolle spielen, müsste der spezielle Agenten-Nachrichtenverteiler die nicht benötigte Methode leer implementieren. Auch dies zeigt, dass sich das Nachrichtensystem vom Konzept her nicht als Ausgangspunkt der Agentenkommunikation eignet.

Aus diesen Gründen wurde die Kommunikationsinfrastruktur für Agenten in FAMOS eigenständig realisiert, wie in Abschnitt 5.2.2 genauer beschrieben.

Die Simulationsinfrastruktur von DESMO-J einschließlich der Trennung von Modell und Experiment wird in FAMOS direkt wiederverwendet, indem für die Simulation agentenbasierter Modelle ebenfalls die Klasse `Experiment` eingesetzt wird. Diese wurde nur um einige Funktionen erweitert, wie im vorigen Abschnitt beschrieben. Die Durchführung von Experimenten wird in FAMOS zusätzlich unterstützt durch die Bereitstellung eines generischen, von `SimProcess` abgeleiteten Ankunftsprozesses, mit dem u. a. Agenten aus empirischen Daten erzeugt werden können. Für die simulationsbegleitende Beobachtung von Agenten sowie die Aufzeichnung von Daten wurden darüber hinaus die graphische Benutzungsoberfläche entsprechend ergänzt und spezielle Datensammelobjekte mit zugehöriger Reportfunktionalität für Attribute einzelner Entitäten geschaffen (siehe Abschnitt 5.2.5).

5.2. Das Framework FAMOS

Das Framework FAMOS setzt das in Kapitel 4 beschriebene Konzept objektorientiert in Java um. Die Programmiersprache Java bietet u. a. Plattformunabhängigkeit, eine Grundlage für verteilte bzw. parallele Ausführung von Modellen und einen großen Anwenderkreis. Auch oder gerade zur Implementation von Multiagentensystemen hat sich Java etabliert; viele der aktuell verfügbaren Agenten-Konstruktionswerkzeuge sind ganz oder teilweise in Java realisiert (vgl. z. B. Mangina 2002; Bordini et al. 2005). Dasselbe gilt für die verfügbaren Werkzeuge für agentenbasierte Simulation (vgl. z. B. Tobias und Hofmann 2004; LePage et al. 2012).

Wie in Abschnitt 4.2 dargelegt, werden Multiagentenmodelle in FAMOS nicht verteilt ausgeführt. Durch die vorhandenen Konstrukte zur Simulation gleichzeitig aktiver Entitäten auf einem sequentiellen Rechner ist jedoch die inhärente Nebenläufigkeit von Agenten und Umgebung weitestgehend sichergestellt (siehe Abschnitte 4.6 und 5.2.4). Eine Verteilung einzelner Modellkomponenten auf verschiedene Rechner ist durch den Einsatz zusätzlicher Software zumindest möglich: Das im Rahmen der Dissertation von Ralf Bachmann entwickelte Werkzeug *CoSim* erlaubt die verteilte Ausführung von in beliebigen Programmiersprachen implementierten Modellkomponenten; seine Anwendbarkeit auf DESMO-J wurde bereits demonstriert (Bachmann 2003, Abschnitt 9.2).

Die objektorientierte Realisierung von FAMOS stützt sich auf verschiedene Entwurfsmuster, wie es dem Stand der Technik bei der Konstruktion von Frameworks entspricht (vgl. Züllighoven und Raasch 2006, S. 815). Auf sie wird bei der Beschreibung ihres jeweiligen Einsatzes Bezug genommen. Die Darstellung folgt dem Prinzip, für einen potentiellen Anwender Wesentliches in den Vordergrund zu stellen und so vorrangig die Hot-Spot-Klassen und die zur direkten Verwendung vorgesehenen Klassen zu erläutern. Der vollständige Quelltext sowie die *API*-Dokumentation des Pakets FAMOS findet sich in (Meyer 2013).

5.2.1. Repräsentation des Raums

Das in Abschnitt 4.3 beschriebene Konzept eines abstrakten Raummodells ist in FAMOS weitgehend realisiert. Der Schwerpunkt liegt wie vorgesehen auf der diskreten Repräsentation des Raums: Neben den grundlegenden gerichteten Graphen wurden hier die besonders häufig in der agentenbasierten Simulation eingesetzten zweidimensionalen Gitter in verschiedenen Ausprägungen umgesetzt. Zwei konkrete kontinuierliche Raummodelle wurden prototypisch implementiert, um die Machbarkeit – zusätzlich zur erfolgreichen Umsetzung in (Czogalla und Matzen 2003) – zu demonstrieren. Beide bilden ein zweidimensionales räumliches Kontinuum (Ebene) ab, wobei einmal linienförmige Hindernisse in der Ebene berücksichtigt werden.[2]

Das als Erweiterung des diskreten Raummodells entwickelte hierarchische Konzept ist bisher nicht realisiert. Attribute und Attributgebiete werden dagegen unterstützt, da sie für die interne Verwaltung der lokalen Wahrnehmungsbereiche der Agenten verwendet

[2] Die Implementation der Klasse `ObstructedPlane` wurde mit geringen Anpassungen aus einer Klassenbibliothek (Meyer 1998) übernommen, die im Rahmen des Forschungsprojekts MOBILE (Hilty et al. 1998) entwickelt wurde.

werden. Eine implizite Modellierung der räumlichen Ausdehnung situierter Objekte und Agenten ist möglich, indem Kapazitäten als Attribute spezifiziert werden. Eine besondere Berücksichtigung von Kapazitäten findet jedoch nicht statt, so dass die bisher realisierten Bewegungsstrategien von unbeschränktem Zugang zu erreichbaren Raumelementen ausgehen. Sie werden im Unterabschnitt 5.2.1.2 im Einzelnen vorgestellt.

Die gemeinsame Schnittstelle für diskrete und kontinuierliche Raummodelle wird von der abstrakten Klasse `Space` gebildet. Sie führt das n-dimensionale Bezugssystem und die auf Punkte abbildbaren räumlichen Positionen ein. Punkte werden als Objekte der Klasse `Point` repräsentiert. Um Koordinaten aus dem n-dimensionalen Punktraum $\mathbb{R}^n$ berücksichtigen zu können, speichern `Point`-Objekte ihre Koordinaten in einem Array der Länge n. Die gewünschte Dimension ist bei der Erzeugung eines Punktes zu spezifizieren. Der Standardfall des zwei- oder dreidimensionalen Raums wird mit entsprechenden Konstruktoren und Methoden besonders unterstützt (siehe Abbildung 5.2).

Räumliche Positionen werden durch die abstrakte Klasse `Position` dargestellt. Sie besitzen die Fähigkeit, ihren Referenzpunkt zu liefern (Methode `getPoint()`) und das Raummodell, auf das sie verweisen (Methode `getSpace()`). Durch Spezialisierung der Klasse `Position` wird erreicht, dass Positionen darüber hinaus einen Verweis auf das zugehörige Raumelement kapseln können. Für Positionen in gerichteten Graphen (Klasse `GraphPosition`) ist dies ein Knoten, für Positionen in Tesselationen (Klasse `GridPosition`) dagegen eine Zelle des Gitters. Bei der Klasse `PointPosition` verschmelzen Referenzpunkt und Raumelement: Sie dient zur Abbildung punktförmiger Positionen in den prototypisch umgesetzten kontinuierlichen Raummodellen.

Zur Bestimmung des lokal wahrnehmbaren Ausschnitts der Umgebung und die für einen Bewegungsvorgang benötigte Zeit muss das Raummodell in der Lage sein, die Distanz zwischen zwei Positionen zu berechnen. Bei Voraussetzung eines kartesischen Koordinatensystems ergibt sich im kontinuierlichen Raum der Euklidische Abstand[3] zwischen den beiden Referenzpunkten. Ist dagegen eine räumliche Struktur vorhanden wie bei der Ebene mit Hindernissen oder generell im diskreten Raum, so muss diese bei der Distanzberechnung berücksichtigt werden. Die einzelnen konkreten Raummodelle implementieren daher die in der Klasse `Space` abstrakt definierte Methode `getDistance()` entsprechend: In einem gerichteten Graphen (Klasse `Graph`) wird der kürzeste Weg zwischen den gegebenen Positionen bestimmt und dessen Länge zurückgegeben, während in einem zweidimensionalen regulären Gitter aus quadratischen Zellen (Klasse `RectangularGrid`) die Manhattan-Metrik[4] eingesetzt wird. Die Distanz wird nicht als einfache Zahl, sondern als einheitenbehaftete Größe behandelt und daher als Objekt der Klasse `Length` zurückgegeben. Die dabei verwendete Standard-Einheit ist Meter (m), sie kann jedoch modellspezifisch angepasst werden über die Methode `setUnit()`.

Das Raummodell verwaltet alle raumbezogenen Informationen; neben seiner eigenen Struktur betrifft dies auch die situierten Agenten und Objekte sowie die Attribute, mit denen Eigenschaften von Raumelementen beschrieben werden können. Auf diese Weise ist der aktuelle Zustand der Umgebung, der hier als räumliche Verteilung von Agenten, Ob-

[3] Der Euklidische Abstand zwischen zwei Punkten misst die „Luftlinie“ und ist im zweidimensionalen Raum definiert als $d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

[4] Die Manhattan-Distanz zwischen zwei Punkten misst den rechtwinkligen Abstand und ist im zweidimensionalen Raum definiert als $d((x_1, y_1), (x_2, y_2)) = \mid x_1 - x_2 \mid + \mid y_1 - y_2 \mid$.

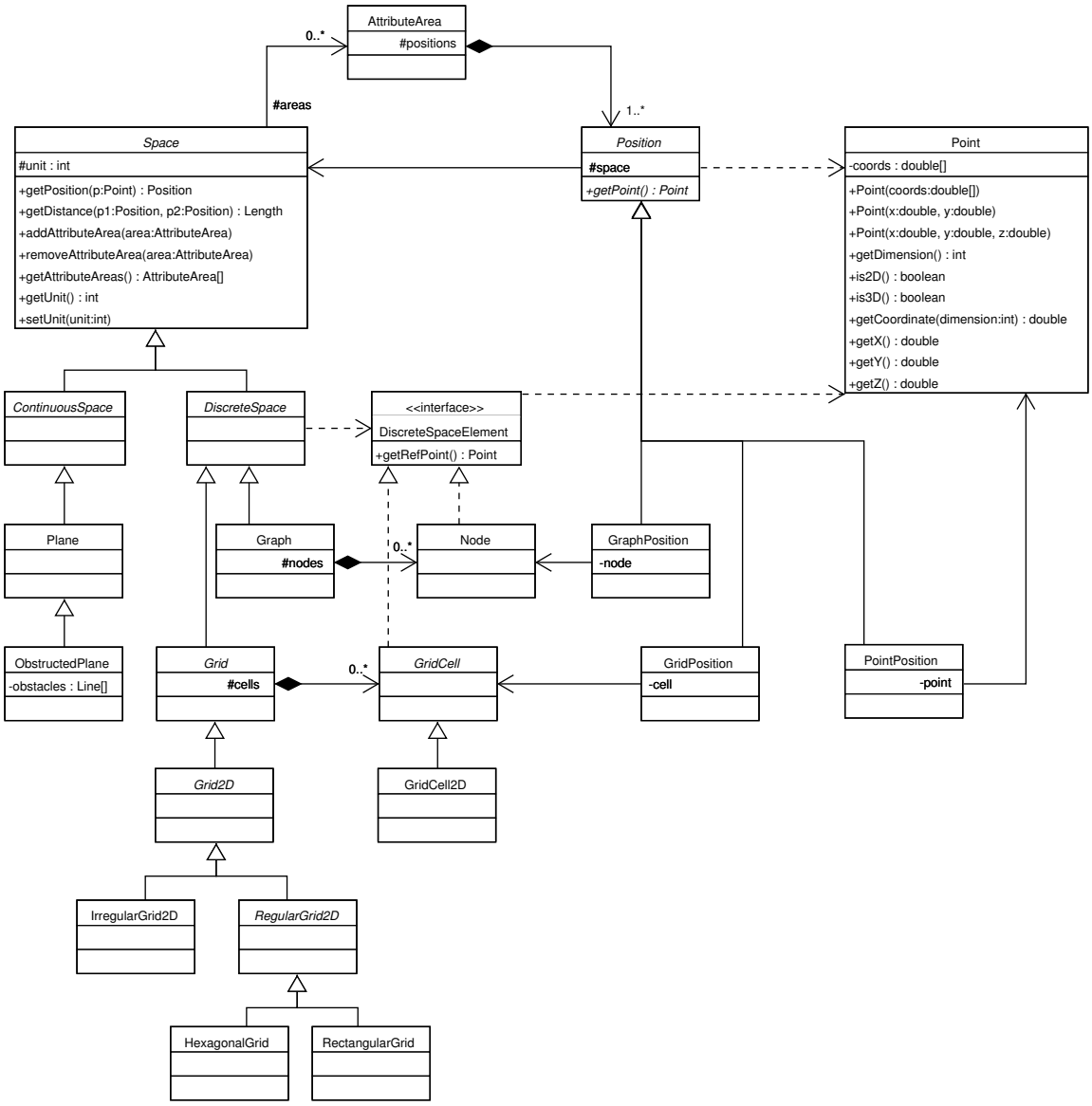

Abbildung 5.2.: Die Realisierung des abstrakten Raummodells in FAMOS durch die Klassen `Space`, `Position` und `Point`. Attributgebiete werden für Positionen definiert und basieren so direkt auf dem abstrakten Raummodell. Die derzeit umgesetzten konkreten Raummodelle sind einschließlich ihrer Klassenhierarchie zusätzlich im Überblick dargestellt.

jekten und Attributen definiert ist (siehe Abschnitt 4.5.1, Seite 107), effizient zu ermitteln. Da im diskreten Fall sowohl Attribute als auch situierte Objekte und Agenten über ihre Positionen an einzelne Raumelemente geknüpft sind, verwalten diese die entsprechenden Referenzen direkt und leiten die Information auf Anfrage an das enthaltende Raummodell weiter. Das Raummodell bildet die Schnittstelle zur Umgebung und bietet eine Reihe von Methoden, um Agenten, Objekte oder Attribute zu manipulieren. Diese beziehen sich immer auf einzelne Positionen.

Um die Verteilung von Attributen im Raum zu erleichtern, sind sogenannte Attributgebiete als Erweiterung des Raummodells vorgesehen. Sie spezifizieren eine Liste von Attributen für eine Menge von Positionen. Durch die Verwendung von Positionen zur Definition des Gebietes lassen sich Attributgebiete bei Bedarf leicht zwischen verschiedenen konkreten Raummodellen austauschen. Vorausgesetzt, sie bilden einen übereinstimmenden Ausschnitt des (realen) Raums ab, so dass die Positions-Referenzpunkte in beiden Modellen enthalten sind. Attributgebiete werden in FAMOS explizit als Objekte der Klasse `AttributeArea` repräsentiert und stehen im Multiagentenmodell für Agenten oder Umgebungsprozesse zur Verfügung. Sowohl Attribute als auch Positionen können ggf. verändert werden.

5.2.1.1. Das graphbasierte diskrete Raummodell

In FAMOS werden gerichtete Graphen (Klasse `Graph`) und Tesselationen (Gitter, Klasse `Grid`) als Diskretisierungen des Raums unterstützt. Wie im Konzept begründet (Abschnitt 4.3.1.1), existiert zu jeder Tesselation ein dualer Graph, so dass sich Graphen als Grundlage des diskreten Raummodells anbieten. Die auf den einzelnen Raumelementen definierte Nachbarschafts- und Erreichbarkeitsrelation wird daher grundsätzlich für jedes diskrete Raummodell als Kantenmenge eines gerichteten Graphen repräsentiert, während die Raumelemente selbst auf die Knoten des Graphen abgebildet werden. Daher besitzen sowohl `Grid`- als auch `GridCell`-Objekte eine Referenz auf ihr jeweiliges Gegenstück: `Graph` bzw. `Node` (siehe Abbildung 5.3).

Da Graphen eigenständig verwendbar sind, ist es notwendig, die an Raumelemente geknüpften raumbezogenen Daten wie Referenzpunkt, Attribute, situierte Objekte und Agenten in Knoten-Objekten zu speichern. Es ist außerdem hinreichend, sie *ausschließlich* dort zu speichern, da Gitterzellen über ihre Referenz auf den dualen Knoten Zugriff auf diese Daten erhalten. Somit reichen `GridCell`-Objekte entsprechende Methodenaufrufe (`getAgents()` usw.) direkt an ihr duales `Node`-Objekt weiter.

Die Nachbarschaftsrelation wird auf Basis der Raumelemente verwaltet: Jeder Knoten besitzt eine Liste der hinführenden und der abgehenden Kanten, während den gerichteten Kanten Start- und Endknoten bekannt sind. Diese leicht redundante Datenstruktur erlaubt eine effiziente Suche nach kürzesten Wegen, die in Graphen bei der Berechnung der Distanz zwischen zwei Positionen durchgeführt wird. Die Klasse `Graph` bedient sich dazu der Routensuchstrategie `GraphPathDijkstra`, die auch für die Bewegung von Agenten zur Verfügung steht (vgl. Abschnitt 5.2.1.2).

Die explizite Repräsentation von Kanten durch Objekte der Klasse `Edge` erlaubt, Kanten beliebige Attribute zuzuordnen. Neben der obligatorischen Kantenlänge, die für die Berechnung kürzester Wege benötigt wird, können modellspezifische Merkmale beispiels-

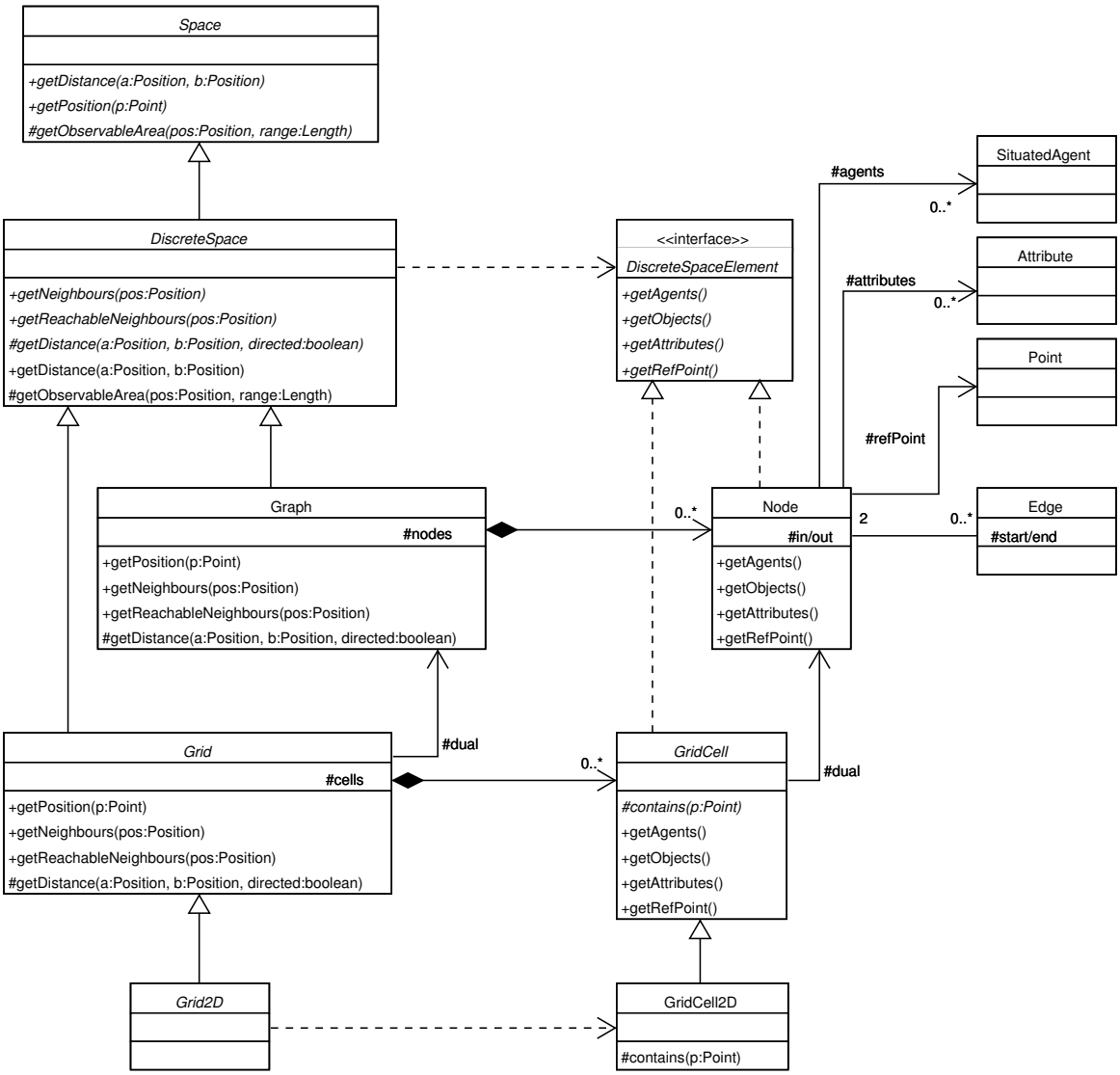

Abbildung 5.3.: Der Zusammenhang zwischen Graph und Grid

weise die Bewegung von Agenten beeinflussen, indem sie in eine passende Bewegungsstrategie eingehen.

Um einen gerichteten Graphen zu erzeugen, wird mindestens eine Liste von Kanten mit Angaben über Start- und Endknoten sowie Kantenlänge benötigt. Jeder Knoten muss dabei mit einem Referenzpunkt versehen sein. Größere Graphen auf diese Weise „manuell“ zu erzeugen, ist praktisch nicht zumutbar. Deshalb verfügt FAMOS über eine XML-Schnittstelle, welche die automatische Generierung eines Graphen aus einer als Textdatei vorliegenden XML-Spezifikation ermöglicht (vgl. Abschnitt 5.2.4.3).

Der Gittern zugrundeliegende Graph wird dagegen immer automatisch konstruiert. Für unregelmäßige Tesselationen, die in FAMOS als Voronoi-Diagramme realisiert sind, ist der duale Graph durch die Delaunay-Triangulation gegeben. In der bisherigen Implementation werden mit der Klasse `IrregularGrid2D` nur zweidimensionale unregelmäßige Tesselationen unterstützt. Sie lassen sich einfach spezifizieren durch Angabe der die Voronoi-Zellen definierenden Punktmenge. Die Berechnung von Voronoi-Diagramm und dualem Delaunay-Graph stützt sich auf ein Java-*Applet* der Fernuniversität Hagen[5], dessen Quelltext frei zugänglich ist.

In regelmäßigen Tesselationen hängt die Nachbarschaftsrelation und damit der duale Graph von der Form der Zellen ab, da in der Regel nur in mindestens einer Kante aneinandergrenzende Zellen als benachbart aufgefasst werden. Eine Ausnahme stellt hier die Moore-Nachbarschaft in regulären zweidimensionalen Gittern aus quadratischen Zellen dar, die auch über einen Eckpunkt angrenzende Zellen in die Nachbarschaft aufnimmt. Sind Zellenform und Definition von „Aneinandergrenzen“ bekannt, lässt sich der duale Graph problemlos konstruieren. In FAMOS sind derzeit die in der agentenbasierten Simulation am häufigsten anzutreffenden zweidimensionalen regulären Gitter realisiert: die Klasse `RectangularGrid` für rechteckige Zellen und die Klasse `HexagonalGrid` für hexagonale Zellen. Für beide sind die üblichen periodischen Randformen möglich, so dass sich statt eines ebenen Gitters ein Torus ergibt. Darüber hinaus werden sie in FAMOS dahingehend speziell unterstützt, dass sie möglichst einfach zu benutzen und möglichst effizient implementiert sind. Ersteres zeigt sich an einer Reihe von unterschiedlichen Konstruktoren, deren einfachster nur die Angabe der Zellenanzahl in x- und y-Dimension erwartet, um z. B. bei `RectangularGrid` ein Gitter mit quadratischen Zellen der Einheitsgröße und Von-Neumann-Nachbarschaft mit festen Grenzen zu erzeugen. Letzteres bedeutet u. a. , dass Distanz-Berechnung und Bestimmung des lokalen Wahrnehmungsbereichs intern auf dem Zellen-Array durchgeführt wird, statt – wie in allgemeinen Gittern (Oberklasse `Grid`) vorgesehen – an den dualen Graphen delegiert zu werden.

Die Standard-Nachbarschaftsformen in 2D-Gittern mit quadratischen Zellen sind Von-Neumann- und Moore-Nachbarschaft. Beide lassen sich unterschiedlich streng anwenden bei der Berechnung des Wahrnehmungsbereichs eines Agenten. Umfasst dessen Sensorreichweite mehr als die direkt benachbarten Zellen, so stellt sich die Frage, welche der an diese Zellen angrenzenden Zellen zu berücksichtigen sind. Wird die Nachbarschaft ausschließlich auf die Ausgangszelle bezogen, so werden beispielsweise bei der

[5] `http://www.pi6.fernuni-hagen.de/GeomLab/VoroGlide` [24.1.2014]. Das am Geometrie-Labor der Fernuniversität Hagen entwickelte *Applet* verwendet ein inkrementelles Verfahren, um Voronoi-Diagramm und Delaunay-Triangulation zu berechnen. Dessen Quelltext wurde für FAMOS leicht angepasst.

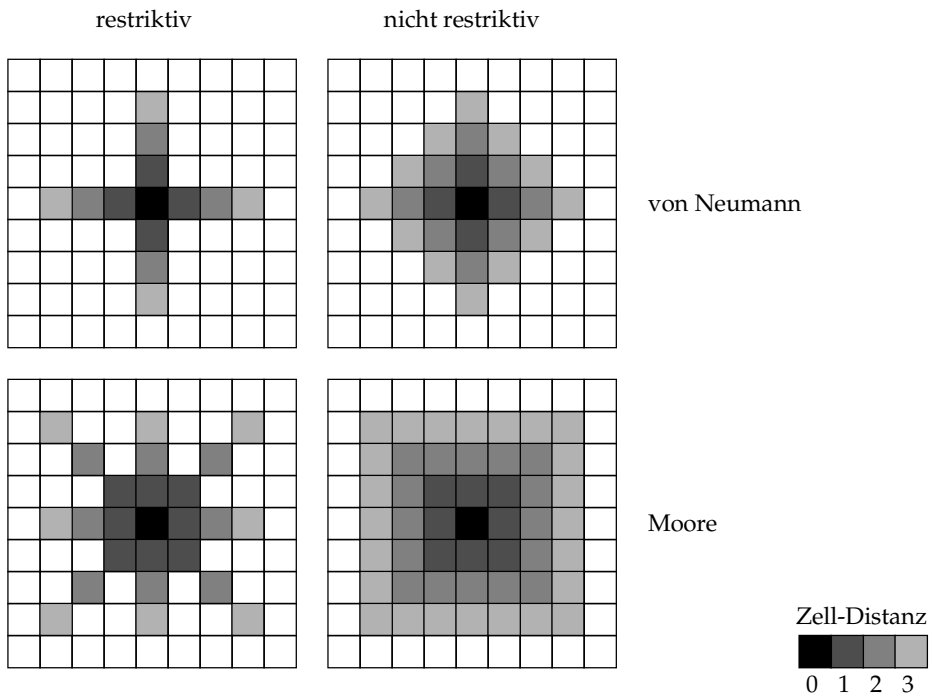

Abbildung 5.4.: Die Auswirkung von restriktiv und nicht restriktiv interpretierten Nachbarschaftsformen auf die Bestimmung des wahrnehmbaren Bereichs in zweidimensionalen regulären Gittern aus quadratischen Zellen.

Von-Neumann-Nachbarschaft nur horizontal und vertikal zu dieser Zelle liegende Zellen in den Wahrnehmungsbereich einbezogen (siehe Abbildung 5.4, links oben). Sonst ist der Wahrnehmungsbereich schrittweise um diejenigen Zellen zu erweitern, die Nachbarn der bisher bestimmten Zellen sind (siehe Abbildung 5.4, rechts). Für größtmögliche Flexibilität bei der Benutzung bietet FAMOS beide Verfahren an; das gewünschte kann über den Schalter `restrictive` in der Klasse `RectangularGrid` gesetzt werden (Methode `setRestrictive()`). Als Standard wird das nicht-restriktive Verfahren eingesetzt.

5.2.1.2. Unterstützung der Bewegung im Raum

Die Bewegung im Raum wird in FAMOS als Fähigkeit eines Agenten in einem Objekt der Klasse `Movement` gekapselt. Situierte Agenten mit dieser Fähigkeit werden von der abstrakten Klasse `MobileAgent` repräsentiert, die mit der Methode `move()` eine Schnittstelle zur Ausführung eines atomaren Bewegungsvorgangs bietet, indem intern die gleichnamige Methode des `Movement`-Objekts aufgerufen wird. Verhaltensmodelle können sich dieser Methode bedienen, um die Bewegung des Agenten zu veranlassen.

Ein Bewegungsvorgang besteht konzeptuell immer aus den beiden Teilvorgängen Wahl der nächsten Position und Durchführung des Positionswechsels (siehe Abschnitt 4.3.2), wobei momentan nur der Positionswechsel selbst Simulationszeit verbraucht.

Wahl der nächsten Position Die Wahl der nächsten Position wird an eine austauschbare Bewegungsstrategie delegiert, die ausgehend von der aktuellen Position eine Folgeposition bestimmt, wobei sie ggf. Ziele des Agenten und die Struktur des Raummodells berücksichtigt. Diese Situation ist ein typischer Anwendungsfall des Strategie-Entwurfsmusters, das die austauschbare Verwendung unterschiedlicher Algorithmus-Varianten ermöglicht (Gamma et al. 1994, S. 315ff). Die gemeinsame Schnittstelle für den Zugriff auf die einzelnen konkreten Strategien wird hier durch das *Interface* `MovementStrategy` definiert: Die Methode `chooseNextPosition()` liefert eine Position zurück, die vom `Movement`-Objekt als anzusteuernde Position benutzt wird; weitere Methoden erlauben den Zugriff auf die aktuelle Position und eine optionale Zielposition. Um die Implementation einer konkreten Bewegungsstrategie zu erleichtern, bietet die abstrakte Klasse `AbstractMovementStrategy` ein Grundgerüst[6] an, das gemeinsame Funktionalität wie die Verwaltung der aktuellen und angestrebten Position zur Verfügung stellt (siehe Abbildung 5.5).

Da der Schwerpunkt in Entwurf und Realisierung von FAMOS auf der Unterstützung des diskreten Raums liegt, sind alle angebotenen konkreten Bewegungsstrategien auf das diskrete Raummodell abgestimmt. Durch dessen Fundierung auf einem gerichteten Graphen sind sie jedoch unabhängig von einer konkreten Ausprägung des Raummodells und funktionieren gleichermaßen für Tesselationen wie für Graphen. In der vorliegenden Implementation berücksichtigen alle Bewegungsstrategien die räumliche Struktur, indem sie nur die von der aktuellen Position aus erreichbaren Positionen in die Auswahl der Folgepositionen einbeziehen. Dieses Verfahren ist in der Regel aufwendiger und enger mit dem Raummodell verbunden als die Auswahl aus einer beliebigen Teilmenge von existierenden Positionen, weshalb es sich für eine Umsetzung innerhalb des Frameworks anbietet. Abgeleitete Klassen lassen sich jedoch leicht an die spezifischen Bedürfnisse eines Modells anpassen, wie in den Beispiel-Modellen demonstriert wird (vgl. Abschnitte 5.3 und 5.4).

Umgesetzt wurden Varianten aller im Konzept genannten Strategien (Abschnitt 4.3.2): Agenten können sich zufällig im Raum bewegen (Klasse `RandomWalk`), einem durch entsprechende Umgebungsattribute abgebildeten Gradientenfeld folgen (Klasse `GradientTrace`), eine bestimmte Richtung einhalten (Klasse `MoveInDirection`) oder eine vorab geplante Route nutzen (Klasse `MoveAlongPath`). Während die beiden erstgenannten Bewegungsstrategien die optionale Zielposition ignorieren, wird diese bei der Strategie `MoveAlongPath` verwendet, um den Endpunkt der zu generierenden Route zu markieren, und kann in der Strategie `MoveInDirection` eingesetzt werden, um die Richtung vorzugeben. Da letztere Strategie in der derzeitigen Implementation auf den am häufigsten benötigten zweidimensionalen Raum beschränkt ist, wird die Richtung sonst als Winkel $-180° \leq \alpha \leq 180°$ spezifiziert, wobei $0°$ mit der Richtung der zunehmenden y-Koordinate („Norden“) identifiziert wird. Als Folgeposition wird jeweils diejenige erreichbare Position gewählt, welche die geringste Abweichung von der gegebenen Richtung ermöglicht, wobei versucht wird, die in vorigen Schritten erfolgte Abweichung auszugleichen.

[6] Die Namensgebung dieser Klasse folgt der in Java üblichen Konvention, die der Bezeichnung des implementierten *Interface* das Wort „Abstract“ voranstellt (Bloch 2008, S. 94).

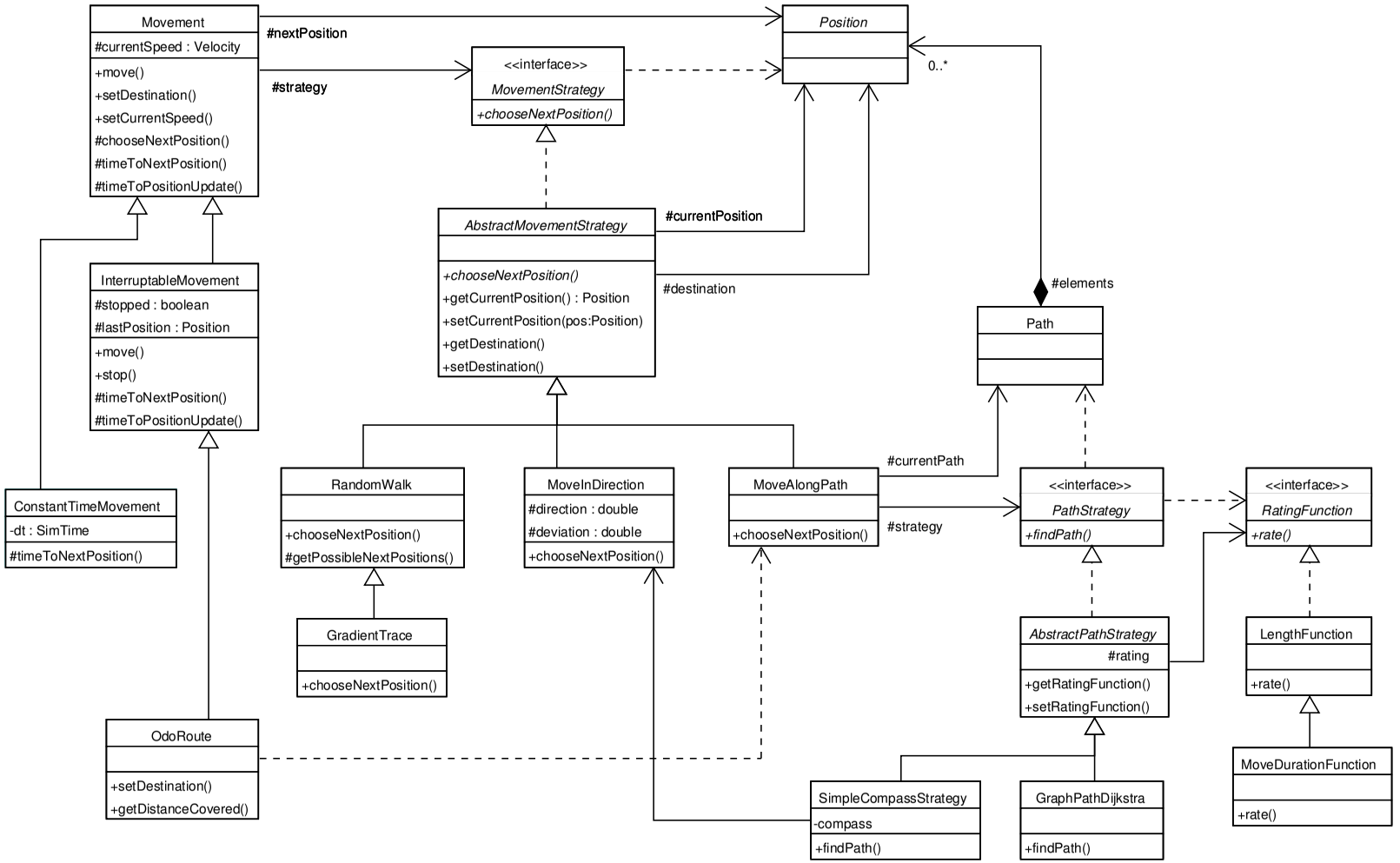

Abbildung 5.5.: Framework zur Realisierung der Bewegung im Raum. Agenten mit dieser Fähigkeit spezifizieren eine passende Bewegungsstrategie (`MovementStrategy`) und delegieren die Durchführung der Bewegung an ein Objekt der Klasse `Movement`, indem sie dessen `move()`-Methode aufrufen.

Die Bewegungsstrategie `MoveAlongPath` befähigt einen Agenten, ausgehend von seiner aktuellen Position eine Route zu einer Zielposition zu bestimmen und sich anschließend entlang dieser Route fortzubewegen. Solange der Agent noch nicht am Ziel angekommen ist, bewirkt ein Aufruf der Methode `chooseNextPosition()` die Rückgabe der in der Route auf die aktuelle Position folgenden Position. Ist die Route vollständig abgearbeitet, d. h. befindet sich der Agent bereits am Ziel, wird diese Zielposition zurückgeliefert, so dass erst dann eine weitere Bewegung erfolgt, wenn der Agent eine neue Zielposition spezifiziert. Eine Route (Klasse `Path`) ist als eine Folge von Positionen definiert, bei der unter Berücksichtigung der Raumstruktur eine Nachfolger-Position von ihrem jeweiligen Vorgänger direkt erreichbar ist. Im zugrundeliegenden gerichteten Graphen existiert somit zwischen je zwei von benachbarten Positionen referenzierten Knoten eine gerichtete Kante, so dass die Route einen gerichteten Weg oder Pfad bildet. Diese Annahme ist jedoch nicht zwingend: Ein `Path`-Objekt kann eine beliebige Folge von Positionen beschreiben, damit modellabhängig beispielsweise auch eine Folge von „Sprüngen“ zu weiter entfernten Positionen abgebildet werden kann.

Die Art der Route wird von der verwendeten Suchstrategie festgelegt. Hier kann ebenfalls das Strategie-Entwurfsmuster mit Gewinn eingesetzt werden: Die Bewegungsstrategie `MoveAlongPath` delegiert die Routensuche an einen Algorithmus, der die Schnittstelle `PathStrategy` implementiert und daher in der Lage ist, eine Route zwischen zwei gegebenen Positionen zu bestimmen (Methode `findPath()`). Dazu wird er sich ggf. einer Bewertungsfunktion (`RatingFunction`) bedienen, um anhand von räumlichen Attributen aus einer Menge von möglichen Positionen die zur Aufnahme in die Route am ehesten geeignete zu finden. Ein bekanntes Beispiel für die Anwendung einer solchen Bewertungsfunktion ist die Routensuche in Graphen; so verwendet der Algorithmus von Dijkstra zur Bestimmung von kürzesten Wegen in Graphen (Dijkstra 1959) die Kantenlänge, um zu entscheiden, welche Kanten durchlaufen und damit welche Knoten in den kürzesten Weg aufgenommen werden sollen. Durch einfachen Austausch der Bewertungsfunktion lässt sich mit demselben Algorithmus z. B. auch eine Suche des schnellsten Weges realisieren, was in Verkehrssimulationen oft von Bedeutung ist.

In der Klasse `GraphPathDijkstra` ist eine Variante des Originals implementiert, welche, statt von einer gegebenen Startposition Wege zu allen anderen Positionen zu berechnen, nur den einen – je nach Bewertungsfunktion besten – Weg von der Start- zu einer ebenfalls gegebenen Ziel-Position bestimmt. Als Voreinstellung wird der kürzeste Weg gesucht (Bewertungsfunktion `LengthFunction`), mit `MoveDurationFunction` steht aber auch eine Möglichkeit zur Bestimmung schnellster Wege zur Verfügung. Dass diese Routensuche für Graphen definiert ist, bedeutet in FAMOS keine Einschränkung ihrer Verwendbarkeit, da das diskrete Raummodell graphbasiert ist und so auch zu beliebigen Tesselationen immer der duale Graph vorliegt. Zwar ist damit nicht sichergestellt, dass die Routensuche immer ein Ergebnis liefert: Die Voraussetzung für die erfolgreiche Anwendung des Dijkstra-Algorithmus ist ein stark zusammenhängender gerichteter Graph, d. h. ein Graph, in dem zwischen je zwei Knoten ein (gerichteter) Weg existiert. Doch gilt dies sowohl für Graphen, die reale Verkehrsnetze abbilden, als auch für die quasi als Standard-Raummodell eingesetzten regelmäßigen Gitter; die Strategie `GraphPathDijkstra` kann daher einen breiten Anwendungsbereich erwarten.

Eine weiterer für die agentenbasierte Simulation interessanter Algorithmus zur Routen-

suche ist in der Klasse `SimpleCompassStrategy` realisiert: Ausgehend von einer Startposition versucht diese Strategie, einen Weg zu der gegebenen Zielposition so zu bestimmen, dass er – unter Berücksichtigung der räumlichen Struktur – möglichst gering von der Luftlinie zwischen beiden Positionen abweicht. Dieses Verfahren erscheint besonders für Agenten mit beschränkten Ressourcen geeignet, da außer der bekannten Zielposition nur lokale Informationen in die Entscheidung über die nächste Position einfließen. Der hier implementierte Algorithmus verwendet Backtracking, um aus möglichen „Sackgassen" herauszunavigieren. Dennoch ist nicht gewährleistet, dass die `SimpleCompassStrategy` immer einen Weg zwischen zwei Positionen findet, geschweige denn, dass sie einen kürzesten Weg zurückliefert. Modelle, die diese Strategie verwenden möchten, sind somit auf Graphen angewiesen, die ihre Anwendung dahingehend unterstützen, dass von jeder Position sicher eine Route zu jeder anderen Position gefunden wird. Für Delaunay-Triangulationen, die dualen Graphen der in FAMOS für unregelmäßige Tesselationen eingesetzten Voronoi-Diagramme, konnte diese Eigenschaft bewiesen werden (Kranakis et al. 1999). Damit steht in FAMOS zumindest ein direkt nutzbares konkretes Raummodell zur Verfügung; bei allen anderen Raummodellen liegt es in der Verantwortung des Modellierers, deren Eignung für diese Strategie herzustellen.

Durchführung der Bewegung Nach der Auswahl einer neuen Position anhand der spezifizierten Bewegungsstrategie erfolgt die Durchführung der Bewegung. In FAMOS wird generell angenommen, dass dies Simulationszeit verbraucht. Die elementaren Schritte des Bewegungsalgorithmus umfassen daher die Berechnung des Zeitpunkts, an dem der Agent die neue Position erreicht, die Vormerkung eines entsprechenden Ereignisses für diesen Zeitpunkt und die Aktualisierung der internen Variablen. Geht man von einer konstanten Geschwindigkeit während des Bewegungsvorgangs aus, lässt sich dessen Dauer leicht aus der im Attribut `currentSpeed` gespeicherten Geschwindigkeit des Agenten und der zurückzulegenden Entfernung berechnen; für die Bestimmung der Entfernung kann auf das Raummodell zurückgegriffen werden. Der eigentliche Positionswechsel des Agenten innerhalb des Raummodells wird von der Umgebung durchgeführt. Wie im Konzept (Abschnitt 4.6) begründet, erfolgt dieser Wechsel in Annäherung an einen zu jedem Zeitpunkt korrekten Umgebungszustand, sobald der Agent die Hälfte der Strecke zwischen den beiden Positionen zurückgelegt hat. Der Zeitpunkt der Aktualisierung ergibt sich dann – konstante Geschwindigkeit vorausgesetzt – direkt aus dem Ankunftszeitpunkt.

Da ein Bewegungsvorgang als „atomare" im Sinne von nicht abbrechbare Handlung aufgefasst wird, könnte das `Movement`-Objekt nach der Bestimmung von neuer Position und Ankunftszeitpunkt alle weiteren Schritte an die Umgebung delegieren. Um größtmögliche Flexibilität zu erreichen, so dass beispielsweise auch Bewegungen mit zunehmender Geschwindigkeit möglich sind oder Unterbrechungen, die einem Halt auf der Strecke entsprechen, wurden sowohl die Berechnung des Ankunftszeitpunkts als auch die des Aktualisierungszeitpunkts als Einschubmethoden realisiert (vgl. Abbildung 5.6). Gemäß dem Entwurfsmuster Schablonenmethode (Gamma et al. 1994, S. 325ff) gibt die `move()`-Methode die Abfolge der einzelnen Schritte vor, während `timeToNextPosition()` und `timeToPositionUpdate()` in der Klasse `Movement` mit dem beschriebe-

nen Standardverhalten versehen sind. Diese Einschubmethoden können in abgeleiteten Klassen erweitert oder ersetzt werden, um die Berechnung der Zeitpunkte an andere Bedingungen anzupassen.

```
public void move() {
   // choose next position
   nextPosition = strategy.chooseNextPosition();

   // determine duration for move
   SimTime duration = timeToNextPosition();
   // determine duration for update position
   SimTime updateDuration = timeToPositionUpdate(duration);

   // request position update by environment
   requestPositionUpdate(updateDuration);
   // request PositionReached signal for agent
   // at simulation time: current time + duration
   requestPositionNotification(duration);

   // update internal position
   strategy.setCurrentPosition(nextPosition);
   nextPosition = null;
}
```

Abbildung 5.6.: Der in FAMOS verwendete Algorithmus zur Durchführung eines Bewegungsvorgangs; implementiert in der Methode `move()` der Klasse `Movement()`. Die Methode ist als Schablonenmethode realisiert, mit den aufgerufenen Methoden `timeToNextPosition()` und `timeToPositionUpdate()` als Einschubmethoden, deren Verhalten in abgeleiteten Klassen überschrieben werden kann.

Die `move()`-Methode teilt der Umgebung daher beide Zeitpunkte mit über die Methoden `requestPositionUpdate()` und `requestPositionNotification()`, die sie in entsprechende interne Ereignisse umwandelt. Zum Zeitpunkt der Aktualisierung veranlasst die Umgebung den Positionswechsel im Raum, während sie zum Zeitpunkt der Ankunft dem Agenten ein Signal der Klasse `PositionReached` sendet. Dieser wird dadurch aktiviert und kann auf den Ortswechsel reagieren, indem er beispielsweise zunächst die Umgebung wahrnimmt, bevor er sich zu einer anderen Position weiterbewegt.

Ein Bewegungsvorgang kann in FAMOS zwar nicht abgebrochen, jedoch unterbrochen werden, um einem Agenten beispielsweise die zeitverbrauchende sofortige Reaktion auf eine Nachricht zu ermöglichen. Die Klasse `InterruptableMovement` ist zu diesem Zweck mit einer Methode `stop()` versehen, die den aktuellen Bewegungsvorgang auf unbestimmte Zeit unterbricht, indem sie das bei der Umgebung für den Ankunftszeitpunkt angeforderte `PositionReached`-Signal annulliert. Falls der Zeitpunkt zur Aktualisierung noch nicht eingetreten ist, wird auch das zugehörige interne Umgebungsereignis gelöscht. Ein erneuter Aufruf von `move()` setzt anschließend die unterbrochene Bewegung fort, wobei Ankunfts- und ggf. Aktualisierungszeitpunkt neu berechnet werden.

Zur zielgerichteten Bewegung eines Agenten anhand der Strategie `MoveAlongPath` steht die spezialisierte Klasse `OdoRoute` zur Verfügung, die zusätzlich die Länge der zurückgelegten Strecke aufzeichnet[7]. Durch Aufruf der Methode `setDestination()` legt ein Agent eine Zielposition fest, woraufhin die `OdoRoute` den Streckenzähler auf 0 setzt und mit Hilfe der zu Beginn festgelegten Suchstrategie die günstigste Route von der aktuellen Position zur Zielposition bestimmt. Wiederholter Aufruf von `move()` bewegt den Agenten entlang der Route, bis er das Ziel erreicht hat. Die einzelnen zurückgelegten Teilstrecken werden dabei automatisch aufsummiert; der aktuelle Stand kann über die Methode `getDistanceCovered()` jederzeit abgefragt werden. Insbesondere wird die bereits auf einer gerichteten Kante zurückgelegte Strecke bei einer Unterbrechung der Bewegung (geerbte Methode `stop()`) in der Summe berücksichtigt, so dass ein Agent diesen Wert bei Bedarf in seine Bewertung der aktuellen Situation einbeziehen kann. In dem in Kapitel 6 beschriebenen Modell eines Stadtkurierdienstes machen die als Agenten modellierten Kuriere von dieser Funktionalität Gebrauch, wenn sie ein während der Fahrt eintreffendes Auftragsangebot bewerten.

Für Modelle, die eine konstante Zeitdauer für jeden Bewegungsvorgang annehmen wie z. B. das zeitgesteuerte *Sugarscape*-Modell (siehe Abschnitt 5.4), ist die in FAMOS unterstützte exakte Berechnung der tatsächlich benötigten Dauer entbehrlich. Zudem kann sie in diesem Zusammenhang korrekt nur in regulären Gittern angewandt werden, in denen die Abstände zwischen benachbarten Raumelementen identisch sind. Um die Modellierung solcher Bewegungsvorgänge zu erleichtern, stellt FAMOS die von `Movement` abgeleitete Klasse `ConstantTimeMovement` zur Verfügung. Sie liefert bei der Bestimmung der Bewegungsdauer einen spezifizierten konstanten Wert Δt zurück, unabhängig von der tatsächlichen Entfernung zwischen der aktuellen und der über die Bewegungsstrategie gewählten nächsten Position.

5.2.2. Modellierung der Umgebung

In FAMOS repräsentiert die Klasse `Environment` die Umgebung eines Multiagentensystems. Da jedes MAS aus einer Menge von Agenten in einer gemeinsamen Umgebung besteht, verfügt jedes Multiagentenmodell in FAMOS über genau ein `Environment`-Objekt (siehe Abbildung 5.7). Diese Beziehung herzustellen und Zugriff auf die Umgebung zu gewährleisten ist die einzige Funktionalität der Hot-Spot-Klasse `MultiAgentModel`; sie ist unabhängig von einer speziellen Anwendung.

Agenten interagieren in FAMOS ausschließlich mit der und über die Umgebung, sie müssen sich daher bei der Umgebung anmelden (Methode `enter()`), wenn sie das System betreten, und wieder abmelden (Methode `exit()`), wenn sie das System verlassen. Besitzt die Umgebung eine räumliche Ausprägung, dargestellt durch ein entsprechendes Objekt vom Typ `Space`, können situierte Agenten der Umgebung eine Anfangsposition mitteilen (Methode `enter(SituatedAgent, Position)`). Die Umgebung verwaltet eine Liste aller angemeldeten Agenten. Diese Aufgabe kann sie nicht wie die Verwaltung der raumbezogenen Daten an das Raummodell delegieren, da so auch Umgebungen ohne

[7] Der Name der Klasse nimmt auf diese Funktionalität Bezug, indem er einen Kilometerzähler (engl. *odometer*) mit der abzufahrenden Route verbindet.

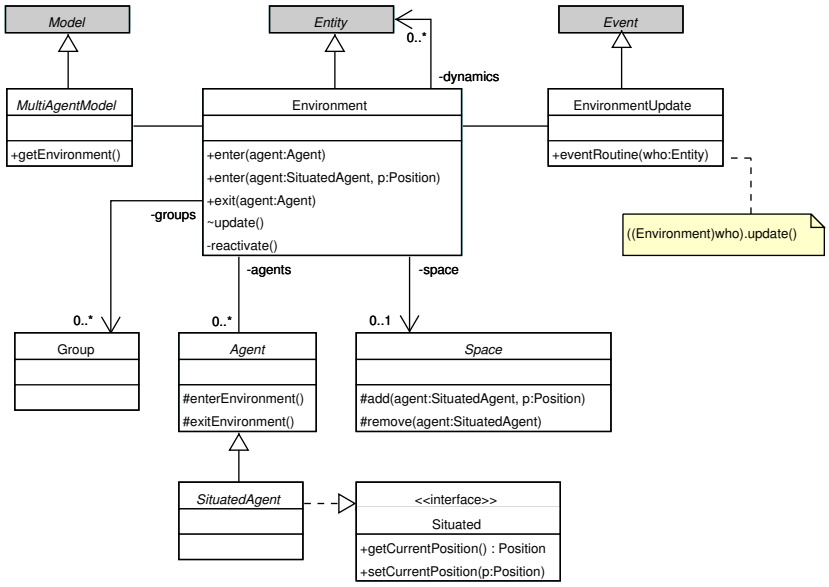

Abbildung 5.7.: Modellierung der Umgebung in FAMOS. Die grau unterlegten Klassen sind Hot-Spot-Klassen des zugrundeliegenden Simulationsframeworks DESMO-J.

explizite Raumrepräsentation möglich sind. Desweiteren führt die Umgebung eine Liste aller modellspezifischen Gruppen, die zur Kommunikation bzw. allgemeiner als Organisationsstrukturen dienen. Sie werden weiter unten detailliert beschrieben.

Da die Umgebung den Agenten als Schnittstelle zu allen Systemressourcen dient, muss sie ihnen entsprechende Zugriffsmöglichkeiten bieten. Die Klasse `Environment` stellt hierfür eine Reihe von Methoden zur Verfügung: von der Kommunikation zwischen Agenten über die Verwaltung von Gruppen bis hin zur Wahrnehmung und Manipulation des Umgebungszustands. Letztere werden im Zusammenhang mit der Agentenmodellierung in Abschnitt 5.2.3 erläutert. Mit der Klasse `Environment` ist damit eine sogenannte *spezialisierte Umgebung* (Ferber 1999, S. 214f) realisiert. Softwaretechnisch gesehen übernimmt `Environment` die Rolle der Fassade im gleichnamigen Entwurfsmuster (Gamma et al. 1994, S. 185ff).

In Anpassung an die Ereignissteuerung ist die Umgebung als aktive Entität (Spezialisierung von `desmoj.Entity`) implementiert. Dadurch kann sie sich selbst bei Bedarf zur Aktivierung vormerken, wobei sie als Ereignis ein Objekt der nur intern zugreifbaren Klasse `EnvironmentUpdate` benutzt. Deren Methode `eventRoutine()` ruft dann die `update()`-Methode der Umgebung auf (siehe Abbildung 5.7). Intern führt die Umgebung eine Ereignisliste, die Anforderungen zur Aktualisierung in zeitlich aufsteigender Reihenfolge zwischenspeichert. Anhand der Einträge auf dieser internen Ereignisliste entscheidet die Umgebung über den nächsten Aktivierungszeitpunkt. Diese technisch bedingte Konstruktion dient zur Abbildung zeitverbrauchender Handlungen in der Umgebung und bleibt vor dem Benutzer verborgen. In der derzeitigen Implementation wird sie ausschließlich im Zusammenhang mit der von FAMOS besonders unterstützten Be-

wegung im Raum verwendet. Alle anderen Handlungen von Agenten in der Umgebung finden zeitverzugslos statt.

5.2.2.1. Umgebungsprozesse

Die Dynamik der Umgebung wird konzeptuell auf Prozesse in der Umgebung zurückgeführt. Diese können in FAMOS durch Integration des Simulationsframeworks DESMO-J direkt als Simulationsprozesse modelliert werden. Da DESMO-J neben der prozessorientierten auch die ereignisorientierte Weltsicht unterstützt, verwendet die Klasse `Environment` die allgemeinere Klasse `desmoj.Entity`, um auf Umgebungsdynamik Bezug zu nehmen. Dadurch lassen sich die Umgebungsprozesse bei Bedarf auch als mit Ereignissen verknüpfte Entitäten abbilden.

Die Umgebungsprozesse werden direkt vom Umgebungsobjekt verwaltet, statt an das Raummodell delegiert zu werden. Dies ist v. a. darin begründet, dass diese Prozesse zwar die räumliche Verteilung von Attributen, Objekten und Agenten manipulieren können und damit Zugriff auf das Raummodell besitzen, selbst jedoch nicht mit einer bestimmten Position im Raum verknüpft sind. Darüber hinaus lassen sie sich so auch in Umgebungen ohne explizite Raumrepräsentation einsetzen, z. B. um automatisch generierte Nachrichten zu verschicken oder Organisationsstrukturen zu verändern.

5.2.2.2. Kommunikationsinfrastruktur

Die Kommunikation zwischen Agenten ist in FAMOS an die Umgebung gebunden. Deren Kommunikationsinfrastruktur unterstützt sowohl Punkt-zu-Punkt- als auch Gruppen-Kommunikation, indem die für die Modellierung von Organisationsstrukturen vorgesehenen Gruppen auch als Nachrichtenempfänger fungieren. Agenten schicken eine Nachricht nicht direkt an den Empfänger, sondern an die Umgebung. Dazu stellt die Klasse `Environment` die Methode `send()` zur Verfügung (siehe Abbildung 5.8). Um deren Benutzung zu vereinfachen, besitzen Agenten eine gleichnamige interne Methode, die automatisch den Aufruf an die Umgebungsmethode weiterleitet. Die Umgebung sorgt dann dafür, dass die Nachricht den spezifizierten Empfänger erreicht.

In der vorliegenden Implementation erfolgt der Nachrichtentransport zeitverzugslos und immer korrekt, indem einfach die `receive()`-Methode des Empfängers aufgerufen wird. Eine umfangreichere Kommunikationsinfrastruktur mit zeitverzögerten oder fehleranfälligen Kanälen ließe sich jedoch leicht ergänzen.

5.2.2.3. Organisationsstrukturen

Für die Modellierung der Organisationsstrukturen eines Multiagentensystems stellt FAMOS das Konzept der Gruppe (Klasse `Group`) bereit. Gruppen werden durch einen modellweit eindeutigen Namen bezeichnet. Um hierarchische Strukturen zu ermöglichen, können Gruppen wiederum Mitglied anderer Gruppen sein. Dies wird in Anlehnung an das Kompositum-Entwurfsmuster (Gamma et al. 1994, S. 163ff) realisiert über eine Schnittstelle `GroupMember`, die ein Gruppenmitglied repräsentiert. Gruppen akzeptieren nur solche Objekte als Mitglieder, die diese Schnittstelle implementieren. An erster

Stelle stehen hier Agenten, weshalb bereits die allgemeinste Klasse `Agent` die Schnittstelle `GroupMember` implementiert. Indem die Klasse `Group` selbst ebenfalls die Schnittstelle implementiert, können Gruppen als Mitglieder übergeordneter Gruppen fungieren.

Eine Gruppe fasst eine Menge von Agenten oder anderen Gruppenmitgliedern nach beliebigen, modellspezifischen Kriterien zu einer Einheit zusammen, indem sie die Anwendung von generischen Operationen auf alle Gruppenmitglieder ermöglicht. Die Gruppe leitet dabei die angeforderte Operation an alle Mitglieder weiter, während die atomaren Gruppenmitglieder wie Agenten sie auf sich selbst anwenden. Um das Autonomie-Konzept nicht zu verletzen, steht es einem Agenten frei, ob und wie er auf die Anforderung reagiert, eine bestimmte Operation anzuwenden. Sein Verhalten ist dazu in der Methode `apply()` von `GroupMember` zu spezifizieren. Diese erhält als Parameter ein die Schnittstelle `Visitor`[8] implementierendes Objekt, das die gewünschte Operation kapselt. In Abhängigkeit vom Typ des Objekts kann der Agent dann die Ausführung der Operation akzeptieren oder ablehnen.

Den FAMOS-internen Anwendungsfall dieser Funktionalität bildet die Operation „Empfang eines Signals“, die sowohl von Agenten bei der *Multicast*-Kommunikation eingesetzt wird als auch von der Umgebung, wenn Agenten über Veränderungen innerhalb ihres Wahrnehmungsbereichs informiert werden (siehe Abschnitt 4.6.2). Da Agenten den Empfang einer Nachricht – in FAMOS allgemeiner den Empfang eines Signals – im Gegensatz zur Ausführung anderer Operationen nicht ablehnen können, wird dieser Fall besonders behandelt: Sowohl Agenten als auch Gruppen implementierten die Schnittstelle `SignalReceiver`, wobei eine Gruppe bei Aufruf der Methode `receive()` das empfangene Signal an alle `SignalReceiver` unter ihren Mitgliedern weiterleitet. Dadurch können Gruppen als Adressaten externer Signale eingesetzt werden.

Aufgrund der potentiell hierarchischen Struktur der Gruppen und der Tatsache, dass ein Agent gleichzeitig Mitglied in beliebig vielen (Unter-)Gruppen sein kann, kann bei dieser Form der Weiterleitung ein Signal einen Empfänger mehrfach erreichen. Ist ein Modell darauf angewiesen, dass jedes Signal exakt einmal empfangen wird, muss vor dem Versenden des entsprechenden Signals über die Methode `flatten()` eine hierarchielose Kopie der betreffenden Gruppe angefordert werden. Diese enthält jedes atomare Gruppenmitglied der Gruppe bzw. einer ihrer rekursiv enthaltenen Untergruppen genau einmal. Da Gruppen sich dynamisch verändern können und die hierarchielose Gruppenversion aus Aufwandsgründen intern nicht ständig mitgeführt wird, ist eine solche Kopie jedesmal neu anzufordern.

Um Gruppenmitglieder über die elementare Funktion „Mitglied“ hinaus als Träger spezieller Funktionen kennzeichnen zu können, wird das Konzept der Rollen innerhalb einer Gruppe unterstützt (vgl. Ferber und Gutknecht 1998; Ferber et al. 2004). Dies übernimmt die von `Group` abgeleitete Klasse `RoleGroup`. Mitglieder einer solchen Gruppe können beliebig viele, durch eindeutige Namen identifizierte Rollen annehmen, wobei eine Rolle gleichzeitig von mehreren Gruppenmitgliedern übernommen werden kann. Durch die Möglichkeit, Gruppen hierarchisch zu strukturieren, lassen sich Rollen leicht als Unter-

[8] Der Name der Schnittstelle wurde in Anlehnung an das gleichnamige Entwurfsmuster (Gamma et al. 1994, S. 331) gewählt, das dieser Realisierung zugrundeliegt. Da `Visitor` auch außerhalb des Gruppenkontextes nutzbar ist, übernimmt ihre einzige Methode `visit()` als Parameter Objekte des allgemeinsten Typs `Object`, statt auf Objekte des Typs `GroupMember` beschränkt zu sein.

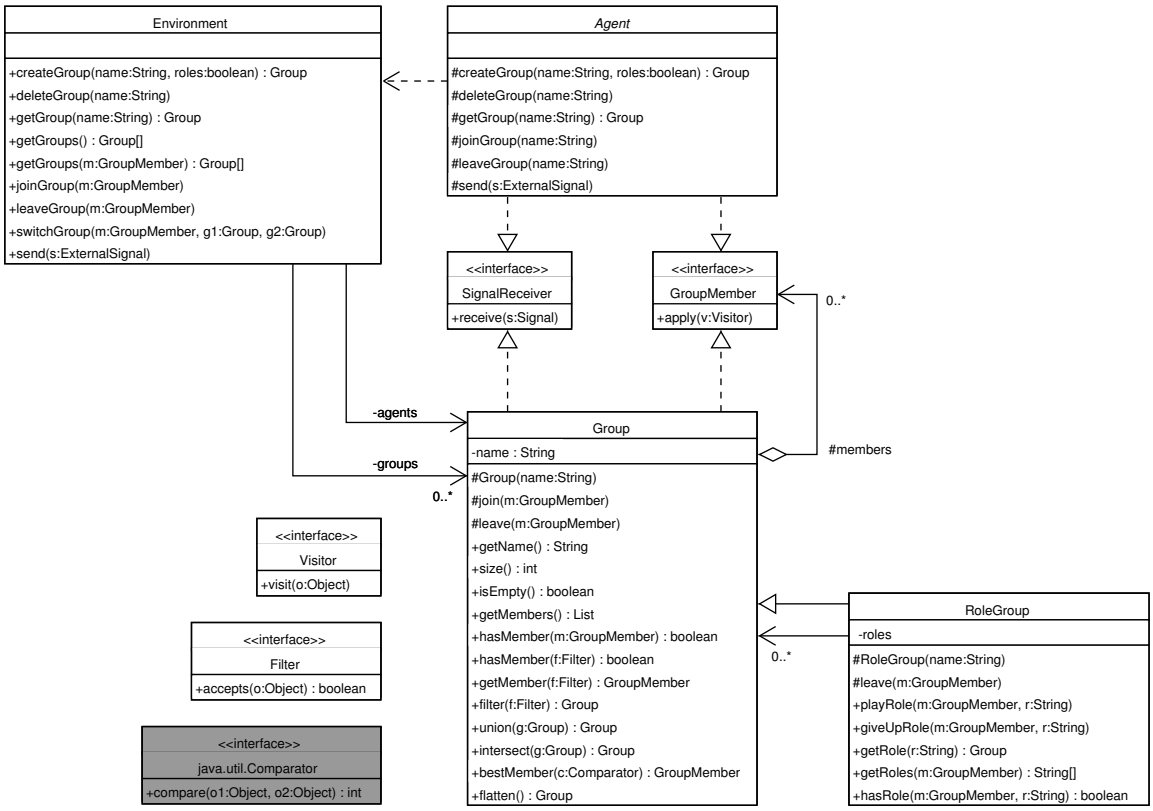

Abbildung 5.8.: Framework zur Modellierung von Organisationsstrukturen. Für die Klassen `Environment` und `Agent` sind nur die relevanten Methoden angegeben. Die grau unterlegte Klasse `Comparator` ist Teil der Java-Standardbibliothek.

gruppen realisieren: Für jede mit der Methode `playRole()` angeforderte und noch nicht vorhandene Rolle erzeugt die `RoleGroup` dynamisch eine Untergruppe.

Rollen ermöglichen es, einen Agenten nicht nur über seinen Namen, sondern auch über eine seiner Funktionen anzusprechen. Auf Anfrage liefert eine `RoleGroup` alle Gruppenmitglieder, welche die gesuchte Rolle gerade ausüben (Methode `getRole()`). Die funktionale Referenzierung von Gruppenmitgliedern wird zusätzlich unterstützt, ohne die explizite Modellierung einzelner Rollen zu erfordern: Die Methode `bestMember()` ermittelt das Gruppenmitglied, das ein als `Comparator`-Objekt repräsentiertes Kriterium am besten erfüllt, während über die Angabe einer Filterbedingung (Schnittstelle `Filter`) eine Teilmenge von Mitgliedern bestimmt werden kann (Methode `filter()`). Weitere Mengenoperationen erlauben die Bildung von Schnittmengen (Methode `intersect()`) und Vereinigungsmengen (Methode `union()`) bezüglich zweier Gruppen und damit die nachträgliche Verknüpfung funktional referenzierter Teilgruppen.

5.2.3. Modellierung der Agenten

Agenten werden in FAMOS auf der Grundlage der ereignisorientierten Konstrukte `Entity` und `Event` realisiert. Die Entscheidung gegen die Verwendung von Simulationsprozessen als Basis ist in der höheren Ausführungseffizienz und der größeren Flexibilität bei der Beschreibung des Verhaltens begründet (siehe Abschnitt 5.1.2.2). Im Unterschied zur ereignisorientierten Weltsicht erfolgt die Verhaltensbeschreibung ganz aus der Sicht des autonomen Agenten, so dass Agenten in FAMOS praktisch auf der objektorientierten Variante der Ereignisorientierung – und somit auf aktiven Objekten – aufbauen. Der ereignisorientierte Ansatz dient lediglich als technische Plattform und bleibt vor dem Modellierer verborgen.

Abbildung 5.9 gibt einen Überblick über den Aufbau eines Agenten in FAMOS. Die Klasse `Agent` ist von `desmoj.Entity` abgeleitet und erweitert diese Klasse zunächst um die Fähigkeit zur Kommunikation. Das dynamische Verhalten eines Agenten wird in einem Objekt der Klasse `Behaviour` gekapselt, was verschiedene, austauschbare Verhaltensbeschreibungen ermöglicht. Da FAMOS einen ereignisgesteuerten Zeitfortschritt verwendet, wird auch ein Agent durch Ereignisse gesteuert, die er jedoch, dem Autonomieprinzip folgend, selbstverantwortlich generiert. Zu diesem Zweck besitzt jeder Agent eine interne „Ereignisliste“ (`Schedule`), die eintreffende Signale in zeitlich aufsteigender Reihenfolge zwischenspeichert bis zu ihrer Bearbeitung durch das Verhaltensobjekt. Erhält der Agent ein Signal, fügt er es automatisch in diese Liste ein und erzeugt für den Zeitpunkt des nächsten bevorstehenden Signals ein Aktivierungsereignis (`AgentActivationEvent`), das er auf der globalen Ereignisliste der Simulationsinfrastruktur vormerkt. Eintritt dieses Ereignisses bewirkt, dass die Ereignisroutine die `activate()`-Methode des Agenten aufruft, die wiederum alle für den aktuellen Zeitpunkt gültigen Signale zur Bearbeitung an das Verhaltensobjekt weiterleitet.

Durch diese intern vorgenommene Abbildung von Signalen auf ein spezielles Aktivierungsereignis ist ein Agent in FAMOS mit dem ereignisgesteuerten Zeitfortschritt verbunden. Gleichzeitig ist sein Verhalten auf einer höheren Abstraktionsebene beschreibbar als das einer Entität der ereignisorientierten Weltsicht. Signale dienen als Informationsträger, die dem Agenten den Eintritt bestimmter Ereignisse melden, d. h. Veränderungen seines eigenen Zustands oder des Zustands seiner Umwelt. Die Verhaltensbeschreibung arbeitet ausschließlich auf diesen Signalen und kann bei jeder Aktivierung in Abhängigkeit der Signaltypen und des aktuellen Agentenzustands eine oder mehrere Aktionen auswählen.

Die typisierten Signale sind als Unterklassen einer abstrakten Klasse `Signal` realisiert. Es wird unterschieden zwischen internen Signalen (`InternalSignal`) und externen Signalen (`ExternalSignal`). Eine Reihe von Signaltypen ist in FAMOS bereits vordefiniert; Signale mit modellspezifischer Bedeutung lassen sich durch Spezialisierung der entsprechenden Oberklasse erhalten. Zwar ist der Aufwand, für jeden Signaltyp eine eigene Klasse zu schreiben, für den Anwender höher als bei Verwendung einer einzigen Signalklasse, die Typ und Parameter eines bestimmten Signals als Attribute besitzt. Doch bietet dieses Vorgehen den Vorteil der Typsicherheit bezüglich der Signalparameter, die bei einer einzigen Signalklasse nur als Liste der Java-Basisklasse `Object` realisierbar wären. Darüber hinaus können Signalhierarchien (vgl. Douglass 1999, S. 313f) verwendet werden. Diese ermöglichen bei Bedarf die Gleichbehandlung mehrerer spezieller Signale, indem eine

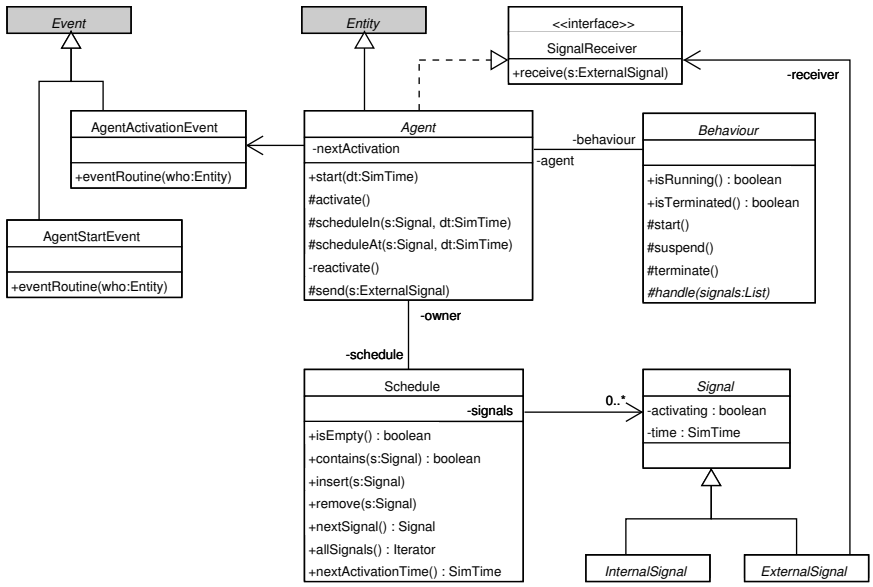

Abbildung 5.9.: Realisierung von Agenten in FAMOS. Die grau unterlegten Klassen sind die ereignisorientierten Konstrukte von DESMO-J, die als technische Grundlage dienen.

Reaktion an deren gemeinsame Oberklasse gebunden wird.

Interne Signale dienen dem Agenten zur Vormerkung von Handlungsabsichten, so dass mit ihnen proaktives Verhalten ausgedrückt werden kann. Der Agent fügt sie mit den Methoden `scheduleIn()` bzw. `scheduleAt()` für einen relativ bzw. absolut angegebenen Zeitpunkt in seine interne Ereignisliste ein. Externe Signale dienen der Interaktion mit der Umgebung im weitesten Sinne, d. h. sie schließen auch die Kommunikation mit anderen Agenten ein. Dies gelingt, indem Nachrichten als spezielle externe Signale behandelt werden. Weitere externe Signale erhält der Agent von der Umgebung, wenn sich in seinem lokalen Wahrnehmungsbereich eine Zustandsänderung ergeben hat. Diese Interaktion zwischen Agent und Umgebung wird im folgenden Abschnitt dargelegt.

5.2.3.1. Interaktion mit der Umgebung

Die Interaktion mit der Umgebung umfasst in FAMOS die Kommunikation zwischen Agenten, die Wahrnehmung des aktuellen Umgebungszustands und die Handlung in der Umgebung, wobei die Bewegung im Raum besonders unterstützt wird. Nur die Kommunikation kann auch unabhängig von einer expliziten Repräsentation des Raums stattfinden, weshalb sie bereits von der abstrakten Basisklasse `Agent` unterstützt wird. Durch Spezialisierung dieser Klasse können Agenten eines rein-kommunikativen MAS (Ferber 1999, S. 11) modelliert werden.

Alle weiteren Interaktionsformen sind an eine räumliche Umgebung gebunden, in der Agenten zu jedem Zeitpunkt eine bestimmte Position einnehmen. Für solche situierten Agenten stellt FAMOS die Klasse `SituatedAgent` zur Verfügung. Sie implementiert die

Schnittstelle `Situated`, welche die Fähigkeit zur Positionierung im Raum beschreibt. Damit einhergehend ist die Sensorreichweite zu spezifizieren, da situierte Agenten in der Lage sind, ihre räumliche Umgebung wahrzunehmen. Die Fähigkeit zur Bewegung im Raum erhalten Agenten der Klasse `MobileAgent` durch ein Objekt der Klasse `Movement`, das sich einer der in Abschnitt 5.2.1.2 beschriebenen Bewegungsstrategien bedient.

Kommunikation Die Kommunikation zwischen Agenten wird abgebildet auf das Senden und Empfangen externer Signale unter Vermittlung der Umgebung. Ein Agent erhält ein externes Signal, indem die Umgebung seine Methode `receive()` aufruft, welche die Schnittstelle `SignalReceiver` realisiert. Diese Schnittstelle implementieren in FAMOS auch Gruppen, um so *Multicast*-Kommunikation zu ermöglichen (vgl. die Beschreibung der Organisationsstrukturen im Abschnitt 5.2.2.3). Für das Versenden steht dem Agenten die Methode `send()` zur Verfügung, die ihrerseits die gleichnamige Methode der Umgebung aufruft. Dort wird das Signal an den spezifizierten Empfänger weitergeleitet. Da Signale neben ihrem durch die Klasse repräsentierten Typ keinen weiteren Inhalt transportieren, wurde zur Modellierung getypter Nachrichten die Klasse `Message` von `ExternalSignal` abgeleitet. Sie besitzt einen als Zeichenkette angegebenen Typ und einen beliebigen Inhalt, der als `Object` übergeben wird. Außerdem wird der Absender explizit vermerkt. Diese Klasse kann dazu verwendet werden, komplexer strukturierte Nachrichten umzusetzen, die einer standardisierten Agenten-Kommunikationssprache wie KQML oder FIPA-ACL folgen.

In der Regel bewirkt ein Signal eine Aktivierung des Agenten, so dass der zeitverzugslose Nachrichtenaustausch praktisch eine Synchronisierung der beteiligten Agenten bedeutet. Um auch zeitlich entzerrte Kommunikationsformen abbilden zu können, bei denen der Empfänger die Signale zu selbst gewählten Zeitpunkten behandelt, besitzen die Signalklassen in FAMOS ein Flag `activating`. Für jedes Signalobjekt kann so festgelegt werden, ob es lediglich auf die interne Ereignisliste eingetragen wird (Wert `false`) oder außerdem den Agenten durch Vormerken des Aktivierungsereignisses reaktiviert (Wert `true`).

Wahrnehmung Der Sensor eines Agenten zur Wahrnehmung des aktuellen Umgebungszustands wird in FAMOS durch die Methode `observe()` der Klasse `SituatedAgent` repräsentiert. Seine Reichweite muss explizit spezifiziert werden, entweder als metrische Länge (Objekt der Klasse `Length`) oder – bei Verwendung eines diskreten Raummodells – als diskrete Länge, d. h. Anzahl von Raumelementen (Objekt der von `Length` abgeleiteten Klasse `DiscreteLength`). Ein Aufruf von `observe()` liefert den Zustand des lokal wahrnehmbaren Ausschnitts der Umgebung als Listen von Positionen, Agenten, Attributen und Objekten, gekapselt in einem Objekt der Klasse `EnvironmentState`. Diese bietet neben dem Zugriff auf die gesamten Listen auch selektivere Auswahl-Methoden an, so dass ein Agent beispielsweise gezielt die Objekte an einer bestimmten Position abfragen kann.

Bestimmt wird der Umgebungszustand von der Umgebung unter Rückgriff auf das jeweilige Raummodell. Die Methode `observe()` ruft intern die Methode `getLocal-`

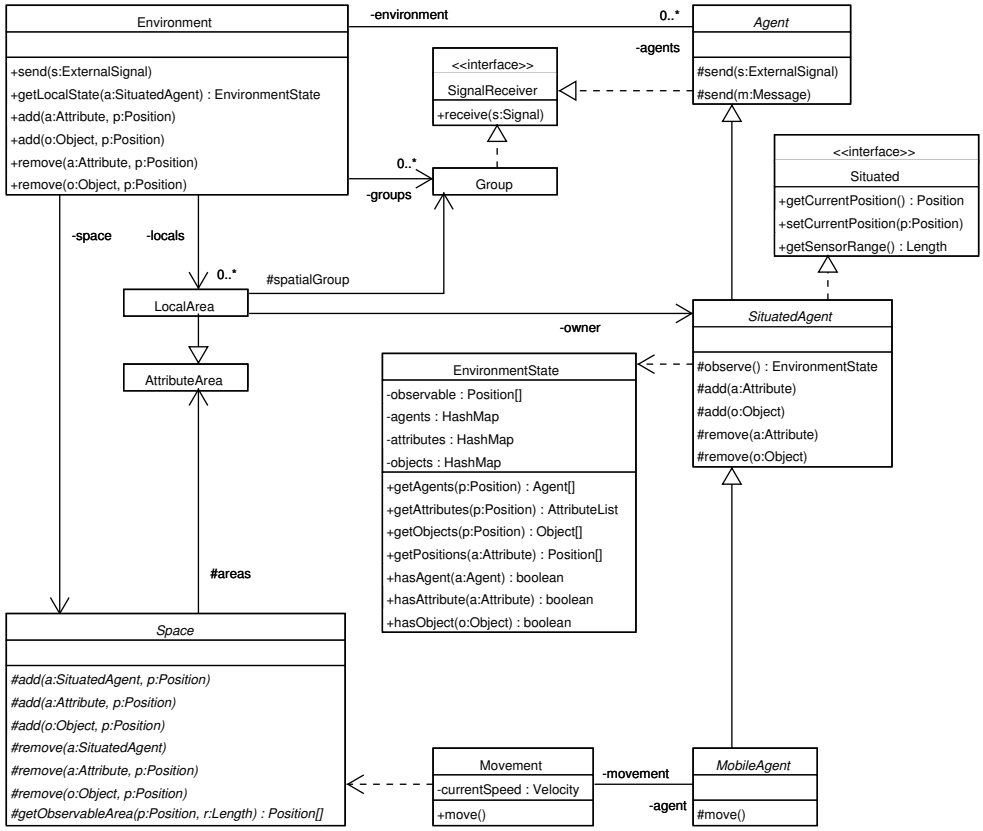

Abbildung 5.10.: Framework zur Interaktion zwischen Agent und Umgebung. Aus Gründen der Übersichtlichkeit ist nur eine Auswahl relevanter Methoden dargestellt; auf die Standard-Zugriffsmethoden für Attribute wurde generell verzichtet.

`State()` des `Environment`-Objekts auf. Diese lässt im Prinzip[9] vom Raummodell den Wahrnehmungsbereich des Agenten anhand seiner aktuellen Position und Sensorreichweite berechnen (Methode `getObservableArea()`) und generiert aus der resultierenden Liste von Positionen den Umgebungszustand.

In der derzeitigen Implementation wird ein „Rundum-Sensor“ vorausgesetzt, so dass dessen Reichweite vom Raummodell als Radius interpretiert wird. Die Wahrnehmung erfolgt zeitverzugslos und ohne Störungen.

Handlung Die Effektoren eines Agenten werden analog zum Sensor durch Methoden abgebildet. Vordefiniert sind grundlegende Aktionen zur Veränderung der Umgebung: Mit

[9] Aus Effizienzgründen wird der Wahrnehmungsbereich nicht bei jedem Aufruf von `observe()` bzw. `getLocalState()` neu bestimmt, sondern für jeden Agenten im `Environment`-Objekt aktuell vorgehalten. Nur bei Änderungen – durch Wechsel der Position bzw. Verändern der Sensorreichweite – muss der Wahrnehmungsbereich aktualisiert werden.

den Methoden `add()` und `remove()` kann ein situierter Agent an seiner aktuellen Position Attribute manipulieren oder Objekte aufnehmen bzw. ablegen. Die Methodenaufrufe werden direkt an das `Environment`-Objekt weitergeleitet, das sie wiederum an das Raummodell delegiert. Zusätzlich informiert die Umgebung alle situierten Agenten, in deren Wahrnehmungsbereich die Änderung stattgefunden hat, durch ein Signal der Klasse `Alert`. Die betroffenen Agenten werden daraufhin aktiviert und können bei Bedarf sofort auf die Veränderung reagieren. Wird diese durch die Ereignissteuerung bedingte Benachrichtigung der Agenten in einem Modell nicht benötigt, kann sie mit der Methode `setNoAlerts()` der Klasse `Environment` abgeschaltet werden.[10]

Situierte Agenten können ihre Position nur mit der Methode `setCurrentPosition()` explizit verändern. Eine komfortablere und umfangreichere Unterstützung der Bewegung im Raum steht in FAMOS für Agenten der Klasse `MobileAgent` zur Verfügung. Ihnen ist ein Objekt der Klasse `Movement` zugeordnet, das die Fähigkeit zur Bewegung anhand einer Bewegungsstrategie kapselt. Ein von `MobileAgent` abgeleiteter, modellspezifischer Agent muss nur eine Strategie und eine Geschwindigkeit vorgeben und kann sich anschließend durch sukzessive Aufrufe der Methode `move()` im Raum bewegen. Nach Ablauf der für einen Bewegungsvorgang benötigten Zeit wird er von der Umgebung über ein Signal der Klasse `PositionReached` automatisch über die Ankunft an der neuen Position informiert.

5.2.3.2. Modellierung des Verhaltens

Das Verhalten eines Agenten wird in FAMOS durch Erweiterung der abstrakten Klasse `Behaviour` beschrieben, welche als Schnittstelle die Behandlung von Signalen festlegt (abstrakte Methode `handle()`). Auf diese Weise können beliebige Verhaltensmodelle integriert werden. Umgesetzt sind in FAMOS derzeit vier Verhaltensbausteine, welche die Modellierung des Verhaltens auf unterschiedlich hohem Niveau ermöglichen. Sie wurden von Nicolas Denz (geb. Knaak) im Rahmen seiner Diplomarbeit entwickelt, deren Schwerpunkt auf der Unterstützung der Verhaltensmodellierung liegt. Im folgenden werden sie kurz vorgestellt; für eine ausführlichere Beschreibung der Implementation sei auf die Diplomarbeit verwiesen (Knaak 2002).

Ereignisorientierte Modellierung Die Klasse `SimpleBehaviour` ermöglicht eine einfache Beschreibung des Agentenverhaltens in Reaktion auf empfangene Signale. Dazu sind vom Anwender im Wesentlichen die beiden abstrakten Methoden `initialActions()` und `process()` zu implementieren. Erstere definiert Aktionen, die einmalig beim Start des Agenten ausgeführt werden, letztere realisiert die Reaktion des Agenten auf ein empfangenes Signal, beispielsweise in Form von *if-then*-Regeln. Diese Art der Modellierung ist insofern als ereignisorientiert einzustufen, als für jedes den Agenten betreffende Signal (Ereignis) die zugehörigen Aktionen zu spezifizieren sind. Da dies ausschließlich aus Sicht des Agenten erfolgt, ergibt sich so die objektorientierte Variante der Ereignisorientierung, in der Entitäten die auf sie bezogenen Ereignisroutinen zusammenfassen (vgl. z. B. Spaniol und Hoff 1995).

[10] Dies spart sowohl den Aufwand zur Bestimmung der zu informierenden Agenten als auch die resultierende Menge an Aktivierungen.

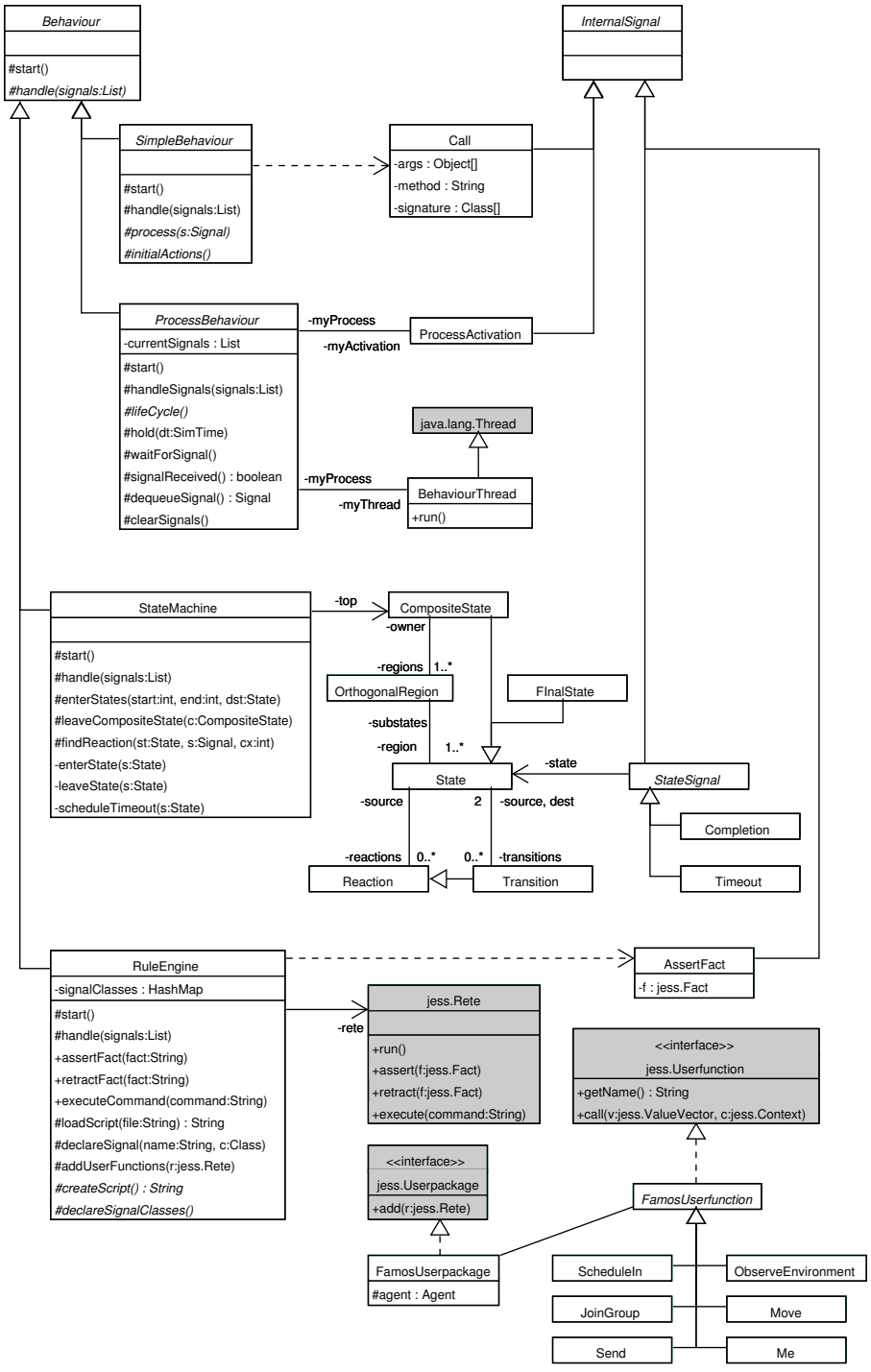

Abbildung 5.11.: Die in FAMOS realisierten Bausteine zur Verhaltensmodellierung.

Da für einen Agenten während einer Aktivierungsphase möglicherweise mehrere Signale vorliegen, wird die `process()`-Methode ggf. mehrmals nacheinander für je ein Signal aufgerufen. Dies geschieht in der Methode `handle()`, welche die zu bearbeitende Liste der Signale vom Agenten übergeben bekommt. Auf diese Weise ist zwar die Reihenfolge der Signalbearbeitung durch die Reihenfolge des Signalempfangs bestimmt und kann nicht modellspezifisch angepasst werden, doch bietet die Methode `process()` so eine übersichtliche Schnittstelle.

Die Klasse `SimpleBehaviour` eignet sich hauptsächlich für die Implementation einfacher reaktiver Agenten, wie sie in den Testbeispielen vorkommen (vgl. Abschnitte 5.3 und 5.4). Proaktives Verhalten kann jedoch durch das Vormerken interner Signale für bestimmte Zeitpunkte mit den agenteninternen Methoden `scheduleIn()` oder `scheduleAt()` erreicht werden. Eine Besonderheit stellen hier Signale vom Typ `Call` dar, die Methodenaufrufe am Agenten repräsentieren. Diese werden in `handle()` abgefangen und mittels Introspektion der Agentenklasse direkt ausgeführt. Auf diese Weise lässt sich auch die in NetLogo, Swarm und verwandten Frameworks vorgesehene Verhaltensmodellierung per Vormerkung von Methodenaufrufen nachbilden.

Prozessorientierte Modellierung Mit der Klasse `ProcessBehaviour` kann das Verhalten eines Agenten als Lebenszyklus eines Simulationsprozesses beschrieben werden. Dazu ist die `lifeCycle()`-Methode zu implementieren. `ProcessBehaviour` eignet sich daher besonders für eher lineare, proaktive Verhaltensweisen, die gelegentlich von asynchron auftretenden Signalen unterbrochen werden.

In der Implementation ähnelt ein Verhaltensprozess zwar einem Simulationsprozess in DESMO-J, doch sind die Mechanismen zur Prozess-Synchronisation an das Konzept autonomer, kommunizierender Agenten angepasst. So interagiert ein Agent mit `ProcessBehaviour` weiterhin ausschließlich über Signale mit seiner Umwelt statt über den Aufruf einer Aktivierungsmethode. Für zeitverbrauchende Aktivitäten ist neben dem unbedingten Warten für eine bestimmte Zeitdauer (Methode `hold()`) mit der Methode `waitForSignal()` auch die Form des bedingten Wartens realisiert, in welcher der Prozess die Ende-Bedingung selbst spezifiziert (vgl. Abschnitt 3.1.1.3). Dieses Konstrukt ist mit dem Agentenkonzept besser vereinbar als die Variante `passivate()`, bei der ein Prozess auf eine Reaktivierung durch andere Prozesse angewiesen ist.

In jeder Aktiverungsphase stehen einem `ProcessBehaviour` alle für diesen Zeitpunkt eingetroffenen Signale in einem Puffer zur Verfügung. Mit der Methode `dequeueSignal()` kann der Verhaltensprozess ein Signal nach dem anderen aus dem Puffer entnehmen. Varianten dieser Methode erlauben die Angabe einer bestimmten Signalklasse oder Bedingung, so dass die Reihenfolge der Bearbeitung modellabhängig angepasst werden kann. Zum Überprüfen, ob der Signalpuffer ein (bestimmtes) Signal enthält, dient die Methode `signalReceived()`, die ebenfalls in mehreren Variationen angeboten wird. Nach der Bearbeitung ausgewählter Signale kann der Agent den Signalpuffer über `clearSignals()` entleeren, so dass ggf. vorhandene unberücksichtigte Signale nicht in die nächste Aktivierungsphase übertragen werden.

Zustandsorientierte Modellierung Zur Verhaltensmodellierung mit endlichen Zustandsautomaten unterstützt FAMOS ausführbare Zustandsdiagramme (*Statecharts*). Diese lassen sich alternativ implementieren durch Fallunterscheidungen, Vererbungshierarchien entsprechend dem Zustands-Entwurfsmuster (Gamma et al. 1994, S. 305ff) oder ein Automatenframework, das die Elemente von Zustandsdiagrammen als Klassen zur Verfügung stellt. In FAMOS wurde letzteres gewählt, da es die explizite Trennung der Daten eines bestimmten Statecharts vom Ausführungsmechanismus erlaubt.

Zentraler Bestandteil ist die von `Behaviour` abgeleitete Klasse `StateMachine`, die in der Methode `handle()` den Interpreter zur Ausführung eines Zustandsdiagramms kapselt. Ein solches Diagramm besteht aus einem gerichteten Graphen mit Objekten der Klasse `State` als Knoten und `Transition`-Objekten als Kanten. Hierarchische und nebenläufige Zustände werden ermöglicht über die Klasse `CompositeState`, welche `State`-Objekte in ggf. mehreren orthogonalen Regionen (Klasse `OrthogonalRegion`) als Unterzustände enthält (siehe Abbildung 5.11). Die Klasse `StateMachine` besitzt über die Referenz auf den Wurzelzustand des Diagramms (`top`) Zugriff auf alle Zustände und Transitionen.

Die von einem Agenten empfangenen und während eines Aktivierungszyklusses an die Methode `handle()` übergebenen Signale bearbeitet das `StateMachine`-Objekt sequentiell. Für jedes Signal durchsucht es die aktuelle Konfiguration, d. h. die momentan aktiven Zustände, nach einer passenden Reaktion oder Transition und führt diese aus. Der dabei verwendete Algorithmus basiert auf einem Verfahren, das die UML-Ausführungssemantik umsetzt (Köhler 1999), und erweitert es um die korrekte Ausführung orthogonaler Regionen mit mehreren Wiedereintrittszuständen und sogenannten Inter-Level-Transitionen, d. h. Transitionen, die direkt einen Unterzustand eines zusammengesetzten Zustands betreten bzw. verlassen. Darüber hinaus wurde die Performanz des Verfahrens verbessert, insbesondere in Bezug auf dessen Speicherbedarf (Knaak 2004).

Die Konstruktion eines Zustandsdiagramms wird in FAMOS durch einen Programmgenerator unterstützt, der textuell oder graphisch repräsentierte Statecharts in ablauffähigen Quelltext übersetzt und im folgenden Abschnitt 5.2.3.3 beschrieben wird.

Deklarative Modellierung Eine deklarative Verhaltensbeschreibung auf der Grundlage regelbasierter Produktionssysteme erlaubt die explizite Trennung von Wissen über das Verhalten des Agenten – Regeln, wie Wahrnehmungen auf Aktionen abzubilden sind – und den Kontrollinformationen, welche die Ausführung der Regeln steuern. Diese sind Teil des Regelinterpreters. Ein Modellierer kann daher das Verhalten eines Agenten auf hohem Abstraktionsniveau beschreiben und ist im Idealfall unabhängig von einem konkreten Ausführungsmechanismus.[11]

In FAMOS wird die regelbasierte Modellierung als Kombination objektorientierter und deklarativer Methoden durch Einbindung der *Java Expert System Shell (Jess)*[12] unter-

[11] Letzteres kann in der Simulation jedoch auch nachteilig sein, wenn es auf die Reproduzierbarkeit von Experimenten ankommt: Da der Regelinterpreter beispielsweise die Reihenfolge festlegt, in der mehrere gleichzeitig anwendbare Regeln auszuführen sind, kann das simulierte Verhalten des Agenten durch den verwendeten Interpreter beeinflusst werden.

[12] `http://www.jessrules.com/jess` [24.1.2014]. Eine Lizenz ist für akademische Anwendungen frei erhältlich.

stützt. *Jess* umfasst einen als Java-Klasse gekapselten Regelinterpreter mit symbolischer Wissens- und Regelbasis, der auf Grundlage des effizienten *Rete*-Algorithmus (Forgy 1982) arbeitet. Die Deklaration von Fakten und Regeln geschieht über Java-Anweisungen oder zur Laufzeit geladene Skripte einer Lisp-ähnlichen deklarativen Sprache. Diese Sprache ermöglicht neben der Manipulation symbolischer Fakten auch die Erzeugung beliebiger Java-Objekte sowie den Aufruf von Methoden.

Ein deklaratives Verhaltensmodell wird in FAMOS durch die Klasse `RuleEngine` gesteuert, die eine Instanz des Regelinterpreters `jess.Rete` integriert (siehe Abbildung 5.11 auf Seite 147). Vom Benutzer sind zwei abstrakte Methoden zu implementieren, die Methode `declareSignalClasses()`, um dem Regelinterpreter die vom Agenten zu behandelnden Signaltypen bekannt zu machen, und die Methode `createScript()`, um das Agentenverhalten als Skript zu spezifizieren. Das Skript kann dabei entweder direkt im Programmtext erzeugt oder mit der Methode `loadScript()` aus einer Datei eingelesen werden. Für die Definition der Regeln ist die Skriptsprache über die Implementation der Schnittstelle `Userfunction` um Funktionen erweitert worden, die Zugriff auf die wichtigsten Methoden des Agenten erlauben. Neben einer Referenz auf den Agenten selbst (`Me`) sind dies Befehle zum Vormerken und Senden von Signalen (`ScheduleIn`, `Send`), zur Gruppenverwaltung (u. a. `JoinGroup`) und zur Interaktion mit der Umwelt (z. B. `ObserveEnvironment`, `Move`). Diese sind in der Klasse `FamosUserpackage` zusammengefasst und werde bei der Initialisierung des regelbasierten Agenten automatisch geladen. Eine Liste aller simulationsspezifischen Erweiterungen der *Jess*-Sprache findet sich in Anhang A.1.

Die Ausführung des regelbasierten Verhaltens ergibt sich ereignisgesteuert wie folgt: Bei jeder Aktivierung des Agenten werden die bis zum aktuellen Zeitpunkt empfangenen Signale an die Methode `handle()` weitergeleitet. Diese fügt sie zunächst als Fakten in die Wissenbasis ein, bevor sie mit `run()` einen Aktivierungszyklus des Regelinterpreters startet. Hier werden aufgrund der vorhandenen Fakten so lange Regeln ausgeführt, bis keine weitere anwendbar ist.

5.2.3.3. Werkzeuge zur Verhaltensmodellierung

Die direkte Implementierung der vorgestellten Verhaltensmodelle ist zwar möglich, jedoch nur für die einfachen ereignis- und prozessorientierten Varianten mit vertretbarem Aufwand durchführbar. Deshalb wird in FAMOS insbesondere die zustandsorientierte Modellierung bereits in der Entwurfsphase unterstützt: zum einen durch eine XML-basierte Skriptsprache zur Beschreibung regelbasierter oder durch Zustandsdiagramme spezifizierter Agenten und zum anderen durch einen graphischen Editor für hierarchische Zustandsdiagramme. Mit Hilfe eines Codegenerators können sowohl Skripte als auch Diagramme in Quelltext entsprechender Agentenklassen übersetzt werden, so dass die Implementierungsphase weitgehend automatisiert wird.

XML-basierte Skriptsprache Die Verwendung von speziellen Skriptsprachen ist in der Simulation üblich, um die Beschreibung von Modellen auf einer anwendungsnahen Ebene zu ermöglichen. Auch wenn sich hier eine graphische Modellerstellung inzwischen als

Standard etabliert hat, bieten einige Simulationssysteme zusätzlich Sprachen für die Spezifikation von Modellen und z. T. auch Experimenten an. Auf diese Weise erfüllen sie die Forderung, mehrere Arten von Modellbeschreibungen zu unterstützen (Page 1991, S. 166). Als Beispiel sei das an der Universität Passau entwickelte System SIMPLEX (Schmidt 2000) genannt, das sowohl über eine eigene Modellbeschreibungssprache (*Model Description Language, MDL*) als auch eine Experimentbeschreibungssprache (*Experiment Description Language, EDL*; Wittmann 1993) verfügt. Das vergleichbare MOBILE-System (Hilty et al. 1998) kombiniert dagegen Modell- und Experimentbeschreibung in einer gemeinsamen Skriptsprache (MOBILE *Script Language, MSL*; Mügge und Meyer 1996).

Eine spezielle Skriptsprache bereitzustellen, um die Spezifikation (von Teilen) eines Multiagentenmodells zu erleichtern, ist auch bei Werkzeugen für die agentenbasierte Simulation anzutreffen: So wird das Framework *Swarm* durch die *Multi-Agent Modelling Language* (MAML; Gulyás et al. 1999) ergänzt, während die neueste Repast-Version die von NetLogo und ähnlichen Logo-Dialekten inspirierte Sprache ReLogo anbietet (Ozik et al. 2013). In Testumgebungen für Agentenarchitekturen ist der Einsatz von Skriptsprachen verständlicherweise auf die Spezifikation von Experimenten beschränkt; ein Beispiel ist die *Scenario Specification Language* (SSL) des *TouringMachines*-Simulators (Ferguson 1992, S. 129f).

FAMOS bietet im Rahmen der Verhaltensmodellierung zwei Skriptsprachen an: eine einfache Sprache zur Deklaration modellspezifischer Signaltypen, um die Implementation von Signalhierarchien zu erleichtern, und eine komplexere Sprache für die Spezifikation von Agenten mit zustands- bzw. regelbasiertem Verhalten. Beide basieren auf der Metasprache XML (*Extensible Markup Language*)[13], welche die Definition eigener Beschreibungssprachen ermöglicht. Eine XML-basierte Sprache besteht aus einer Menge hierarchisch strukturierter Elemente, die durch anwendungsspezifische Bezeichner (*tags*) begrenzt sind und ggf. zusätzliche Attribute aufweisen. Die Grammatik einer solchen Sprache wird entweder in Form einer sogenannten DTD (*Document Type Definition*) oder eines XML-Schemas angegeben. Schemata haben gegenüber DTDs den Vorteil, selbst wiederum XML-konform zu sein, weshalb sie meist der älteren DTD vorgezogen werden.

Aufgrund ihrer regelmäßigen Struktur und der Eigenschaft, in Verbindung mit ihrer Grammatik selbst-beschreibend zu sein, lassen sich XML-Dokumente gut automatisch verarbeiten. Daher existieren inzwischen zahlreiche generische Parser für verschiedene Programmiersprachen, u. a. Java. Damit entfällt beim Einsatz von XML die aufwendige Implementation eines eigenen Parsers; es ist lediglich die Grammatik der Sprache zu spezifizieren und dem generischen Parser zusammen mit dem zu verarbeitenden Skript zu übergeben. Darüber hinaus vereinfacht die regelmäßige Struktur, Skripte automatisch zu generieren. Dies wird im FAMOS-Diagrammeditor genutzt, der aus der graphischen Repräsentation eines Zustandsdiagramms das zugehörige XML-Skript erzeugt. Auf diese Weise lassen sich auch graphisch definierte Zustandsdiagramme vom Codegenerator in Quelltext übersetzen.

Beim Entwurf der Skriptsprachen für FAMOS stand ihre einfache Verwendbarkeit und damit ihre – trotz der von XML vorgegebenen starren Struktur – ausreichende Lesbarkeit im Vordergrund. Da die während der Implementierungsphase verfügbaren Parser noch

[13] `http://www.w3c.org/XML` [24.1.2014].

keine ausreichende Unterstützung für XML-Schemata boten, wurde zur Definition auf das DTD-Format zurückgegriffen; die formalen Definitionen finden sich in Anhang A.2.1.

Die Skriptsprache zur Spezifikation von modellspezifischen Signalklassen besteht im Wesentlichen aus den Elementen `<signal>` zur Deklaration einer Signalklasse und dem eingebetteten Tag `<param>` zur Deklaration eines Signalparameters. Beide Elemente werden über zusätzliche Attribute genauer spezifiziert, so dient z. B. das erforderliche Attribut `name` zur Angabe des Signal- bzw. Parameternamens. Abbildung 5.12 (oben) zeigt ein Beispiel eines derartigen Skripts sowie die vom Codegenerator erzeugten zugehörigen Signalklassen mit den erforderlichen Methoden (unten).

Die Sprache für die Spezifikation deklarativer bzw. auf Zustandsdiagrammen basierender Verhaltensmodelle für Agenten benötigt eine größere Ausdruckskraft. Sie umfasst daher eine größere Anzahl z. T. hierarchisch strukturierter Elemente. Diese dienen in der Mehrheit zur Beschreibung von hierarchischen Zustandsdiagrammen (vgl. Abbildung 5.13). Regelbasierte Verhaltensmodellierung durch XML-Skripte wird in FAMOS bisher nur rudimentär unterstützt; im Wesentlichen dient die Skriptsprache hier zur Zusammenfassung der Java- und *Jess*-spezifischen Codebestandteile einer Agentenklasse. Eine Erläuterung aller zur Verfügung stehenden Tags befindet sich in Anhang A.2.2.

Graphischer Editor Für den Entwurf von Zustandsdiagrammen stellt FAMOS einen eigenen graphischen Editor bereit. Obwohl gängige Werkzeuge für den objektorientierten Entwurf (sogenannte *CASE-Tools*) diese Funktionalität bereits bieten, hat die Neuentwicklung eines Zustandsdiagrammeditors verschiedene Vorteile. So lässt sich die Benutzerführung besser an die Modellierung simulierter Agenten anpassen und zudem einfacher gestalten, als dies in allgemeinen CASE-Tools möglich ist. Diese bieten neben dem Editieren von Zustandsdiagrammen zahlreiche weitere Funktionen und Diagrammtypen an, die für die Verhaltensmodellierung von Agenten nicht notwendig sind und damit einen unerwünschten *Overhead* darstellen. Das für eine mögliche Integration in FAMOS evaluierte UML-Modellierungswerkzeug ArgoUML[14], dessen Quelltext frei verfügbar ist, berücksichtigte zum Zeitpunkt der Entwicklung darüber hinaus einige wichtige Elemente von Zustandsdiagrammen wie Zeitereignisse, aufgeschobene Ereignisse und orthogonale Regionen nicht. Ein speziell für FAMOS entwickelter Diagrammeditor stellt zudem sicher, dass die manuelle Erstellung von Agentenskripten mit der graphischen Modellierung kombiniert werden kann, da bestehende Teile des Frameworks wie die XML-Skriptsprache und die Signalklassen im Rahmen des Editors verwendbar sind.

Deklarationen der benötigten Signale sowie die Oberklasse des durch das Zustandsdiagramm modellierten Agenten sind zu Beginn einer Sitzung mit dem graphischen Editor zu spezifizieren. Wie aus Abbildung 5.14 zu erkennen ist, gleicht die Benutzungsoberfläche denjenigen vorhandener Statechart-Editoren. Das Zustandsdiagramm wird auf der Zeichenfläche im Hauptfenster aus Zustands- und Transitionselementen zusammengesetzt, die über Schaltflächen am linken Rand erreichbar sind. Ein Doppelklick auf eines dieser Elemente öffnet ein Dialogfenster zum Editieren der Eigenschaften wie Zustands- bzw. Transitionsnamen, Eintritts- und Austrittsaktionen sowie Bedingungen (Abbildung 5.14 rechts). Um sowohl die Konzepterstellungs- als auch die Implementationsphase des

[14] `http://argouml.tigris.org` [24.1.2014].

```
<signalset class="FactoryModel" package="factoryModel">

   <import path="desmoj.SimTime" />

   <signal name="TransportRequest" internal="FALSE">
       <param name="start" type="Station" />
       <param name="dest" type="Station" />
       <param name="time" type="SimTime" />
   </signal>

   <signal name="Emergency" internal="FALSE">
       <param name="level" type="int" />
   </signal>

   <signal name="DestinationReached" internal="TRUE">
       <param name="station" type="Station" />
   </signal>

</signalset>
```

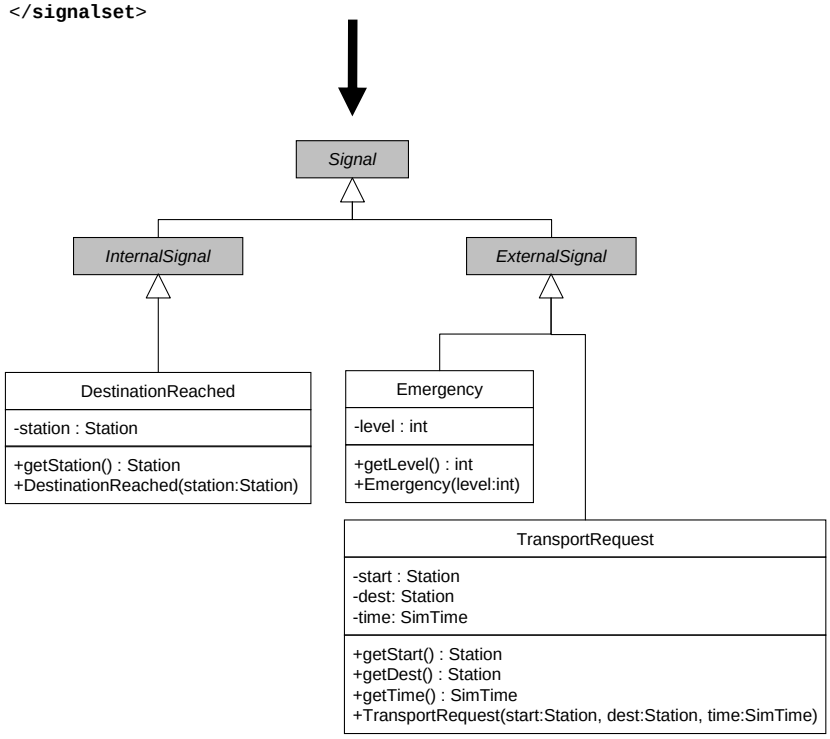

Abbildung 5.12.: Deklaration von modellspezifischen Signalklassen mittels XML-basierter Skriptsprache (oben) und daraus generierte Klassen (unten). Die grau unterlegten Klassen sind Bestandteil von FAMOS.

```
<agent name="TransportRobot" extends="BasicRobot" package="factory_model">

  <signals file="c:\dateien\xml\FactoryModelSignals.xml"/>

  <fsm>

    <top initial="Warten"/>
      <state name="Warten"/>
      <composite name="Aktiv" initial="ZumStartFahren" history="NO_HISTORY">
         <state name="ZumStartFahren"/>
         <state name="ZumZielFahren"/>
      <composite>
      <state name="ZurBasisFahren"/>
      <final name="Ausgeschaltet"/>
    </top>

    <in state="Warten">
      <entry>setzeStatusLampe(GRUEN)</entry>
      <on sig="SelbstTest"><effect>testAusfuehren()</effect></on>
    </in>

    <in state="Aktiv">
      <entry>setzeStatusLampe(ROT)</entry>
      <exit>setzeStatusLampe(GELB)</exit>
      <on sig="Transport"><effect>ablehnen()</effect></on>
      <defer sig="SelbstTest"/>
    </in>

    ... further state declarations

    <transition sig="Transport" src="Warten" dst="Aktiv">
      <effect>speichern($start, $ziel)</effect>
    </transition>

    <transition sig="Notfall" src="Aktiv" dst="ZurBasisFahren">
      <guard>stufe &gt; 2</guard>
    </transition>

    ... further transition declarations

  </fsm>
</agent>
```

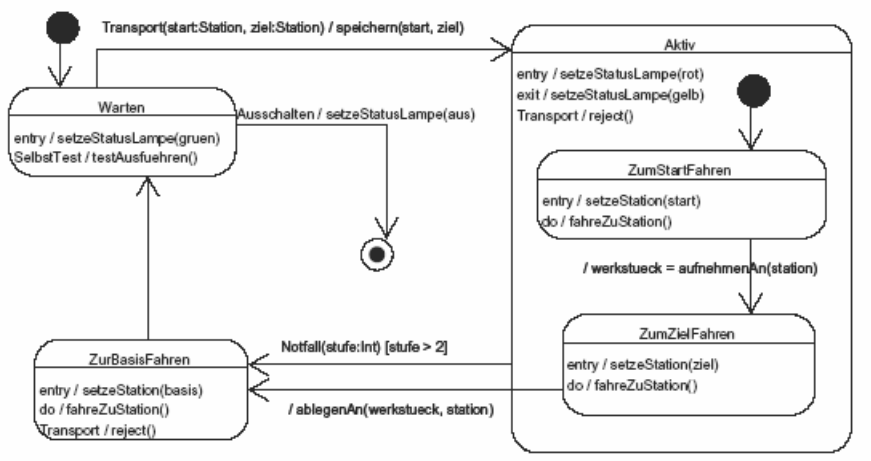

Abbildung 5.13.: Deklaration eines hierarchischen Zustandsdiagramms mittels XML-basierter Skriptsprache (oben). Das so spezifizierte Diagramm (unten) modelliert das Verhalten eines einfachen Transportroboters in einer Fabrik (Knaak 2002, S. 67 u. 101).

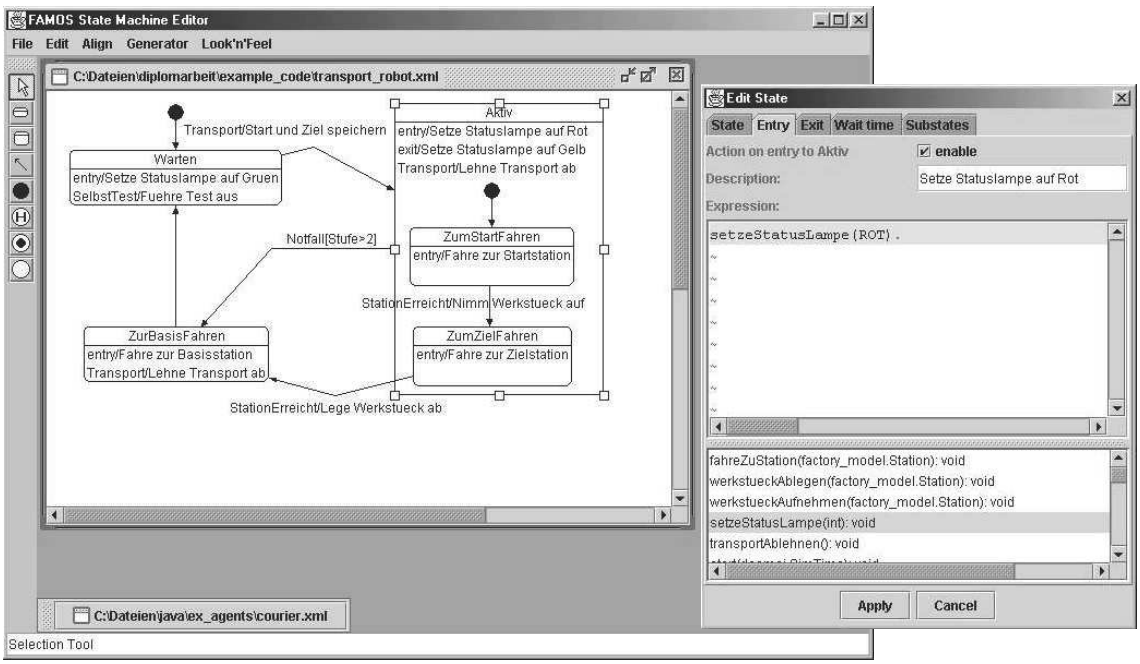

Abbildung 5.14.: Benutzungsoberfläche des graphischen Editors für Zustandsdiagramme mit Hauptfenster (links) und Dialog zum Editieren von Zustandseigenschaften (rechts).

Modellentwurfs zu unterstützen, können alle Beschriftungen zunächst rein beschreibend im Eingabefeld `Description` angegeben werden, bevor sie auf Implementationsebene als Java-Anweisungen im darunterliegenden Feld `Expression` spezifiziert werden. Im unteren Teilbereich des Dialogfensters werden dabei geeignete Sensor- und Effektormethoden der Oberklasse des Agenten zur Auswahl aufgelistet, welche der Editor durch Introspektion dieser Klasse ermittelt.

Das im graphischen Editor entworfene Zustandsdiagramm kann für eine spätere Bearbeitung gespeichert werden. Dies erfolgt in der oben beschriebenen XML-basierten Skriptsprache, wobei die graphische Repräsentation des Diagramms getrennt in einem weiteren XML-Dialekt abgelegt wird. Außerdem ermöglicht der Editor den direkten Aufruf des Codegenerators sowie die Vorab-Anzeige des generierten Agentenskripts bzw. Quellcodes.

Der graphische Editor wurde prototypisch in Java unter Verwendung der *Swing*-Bibliotheken implementiert und baut auf JHotDraw[15] auf, einem frei verfügbaren Framework für die Erstellung strukturierter graphischer Editoren. Es unterstützt nicht nur die Erzeugung von benutzerdefinierten geometrischen Formen und deren Verknüpfung, sondern insbesondere die Definition von Randbedingungen für das Verhalten der graphischen Elemente. So lassen sich beispielsweise die UML-Diagrammen zugrundeliegenden Regeln, welche Elemente wie verknüpft werden können, gut realisieren. Der graphische Editor für Zustandsdiagramme verwendet diese Randbedingungen, um die syntaktische Korrektheit der gezeichneten Diagramme weitgehend sicherzustellen.

[15] `http://www.jhotdraw.org` [24.1.2014]

5.2.4. Unterstützung der Simulation

Neben der Modellbildung unterstützt FAMOS die Simulation im engeren Sinne, d. h. die Durchführung von Experimenten mit einem Modell. Die hierfür benötigte Simulationsinfrastruktur wie Simulationsuhr und Ablaufsteuerung wird von dem zugrundeliegenden Simulationsframework DESMO-J geliefert. Die Integration von DESMO-J ermöglicht darüber hinaus, alle in DESMO-J vordefinierten Modellkomponenten wie stochastische Verteilungen, Warteschlangen und statistische Datensammlungsobjekte in agentenbasierten Modellen wiederzuverwenden (vgl. Abschnitt 5.2.4.2).

Die Modellierung der Zeit in FAMOS ist – wie im Konzept in Abschnitt 4.6 beschrieben – auf den ereignisgesteuerten Zeitfortschritt abgestimmt, indem das dynamische Verhalten von Agenten und Umgebung intern auf Ereignisse abgebildet wird (Abschnitt 5.2.4.1).

5.2.4.1. Repräsentation der Zeit

Wie das zugrundeliegende Simulationswerkzeug DESMO-J verwendet auch FAMOS Ereignissteuerung, um das dynamische Verhalten des Modells in der Simulationszeit fortzuschreiben. Simulationszeit ist somit in unregelmäßige Intervalle diskretisiert – nur diejenigen Zeitpunkte werden betrachtet, zu denen Zustandsänderungen im System (Ereignisse) auftreten – und zu jedem Zeitpunkt werden nur diejenigen Entitäten aktualisiert, die vom aktuellen Ereignis betroffen sind. Alle zeitverbrauchenden Handlungen werden auf Start- und Ende-Ereignisse abgebildet; konzeptuell aktive Phasen einer Entität sind programmtechnisch passive Phasen zwischen zwei Aktivierungen.

In Anpassung an die Ereignissteuerung sind Agenten und Umgebung in FAMOS auf der Grundlage von aktiven Entitäten realisiert, welche sich über spezielle Ereignisse (`AgentActivationEvent` bzw. `EnvironmentUpdateEvent`) selbst zur Aktivierung vormerken. Zu diesem Zweck verfügen sowohl jeder Agent als auch die Umgebung über eine interne Ereignisliste, in der alle eintreffenden Signale (Meldungen über zukünftige Ereignisse) in zeitlich aufsteigender Reihenfolge zwischengespeichert werden. Diese Signale wurden entweder vom Agenten bzw. der Umgebung selbst generiert (interne Signale), z. B. um das Ende einer Handlung anzuzeigen, oder von außen, beispielsweise von anderen Agenten, erhalten (externe Signale). Anhand der Einträge auf ihrer internen Ereignisliste entscheiden Agenten und Umgebung über den nächsten Aktivierungszeitpunkt, für den automatisch das entsprechende Aktivierungsereignis auf der globalen Ereignisliste der Simulationsinfrastruktur eingetragen wird. Dies ist ausführlicher in den vorhergehenden Abschnitten über die Modellierung von Agenten (5.2.3) und Umgebung (5.2.2) beschrieben.

Damit das entsprechende Signal für das Ende einer Handlung zu ihrem Beginn generiert werden kann, muss ihre Dauer bekannt sein. Ebenso gilt es, den Umgebungszustand trotz asynchroner Aktualisierung der Agenten möglichst effizient aktuell zu halten und eine sofortige Reaktion von Agenten auf Veränderungen der Umgebung zu ermöglichen.

Bestimmung der Aktionsdauer Um die Dauer einer Handlung vorherzubestimmen, gibt es verschiedene Möglichkeiten (vgl.Abschnitt 2.1.3), welche je nach Modellkontext unterschiedlich gut geeignet sind. Eine Handlung kann mit einem (a) konstanten oder (b) aus

empirischen Daten vorliegenden Wert als Dauer versehen werden, die Dauer kann (c) zufällig, einer bestimmten stochastischen Verteilung folgend gewählt werden, (d) zustands- oder zeitabhängig berechnet oder (e) abgeschätzt werden anhand des Rechenaufwands, der zur Durchführung der Aktion nötig ist.

Für die von FAMOS zur Verfügung gestellten modell-unabhängigen Handlungen eines Agenten wurden die folgenden Optionen implementiert:

- Senden einer Nachricht (Methode `send()` der Klasse `Agent`): Die versandte Nachricht wird von der Umgebung sofort an den Empfänger weitergeleitet. Damit ist der Nachrichtenverkehr in FAMOS zeitverzugslos. Dies entspricht Option (a) mit einem konstanten Wert von 0.

- Wahrnehmung der Umgebung (Methode `observe()` der Klasse `SituatedAgent`): Der Zustand des vom Agenten wahrnehmbaren Ausschnitts der Umgebung wird sofort zurückgeliefert, d. h. Wahrnehmung ist wie Nachrichtenaustausch zeitverzugslos.

- Veränderung von Attributen oder Objekten in der Umgebung (Methoden `add()` und `remove()` der Klasse `SituatedAgent`): Ebenfalls zeitverzugslos, die entsprechenden Änderungen am Umgebungszustand werden sofort wirksam.

- Bewegung im Raum (Methode `move()` der Klasse `MobileAgent`): Option (d), d. h. die Dauer wird jeweils berechnet. Dies ist möglich, da Geschwindigkeit und zurückzulegende Distanz bekannt sind: Geschwindigkeit ist eine Eigenschaft des Agenten, Distanz ist abhängig vom Raummodell und wird daher von diesem berechnet. Unter der Annahme, dass die Geschwindigkeit während des Bewegungsvorgangs von einer Position zur nächsten konstant bleibt, ergibt sich die Dauer als Quotient aus der zurückgelegten Distanz und der aktuellen Geschwindigkeit. Abweichungen von diesem elementaren Ablauf wie z. B. variable Geschwindigkeit oder Unterbrechung eines Bewegungsvorgangs sind möglich und z. T. in FAMOS auch bereits realisiert wie in Abschnitt 5.2.1.2 über die Unterstützung der Bewegung im Raum näher erläutert.

Abgesehen von der Bewegung im Raum arbeiten somit alle modell-unabhängigen Aktionen in FAMOS zeitverzugslos. Ein konstanter oder entfernungsabhängiger Faktor ließe sich jedoch beispielsweise als Parameter der Umgebung realisieren.

Für modellspezifische Aktionen wie z. B. die Übernahme von Transportaufträgen im Kurierdienstmodell (siehe Kapitel 6) bleibt es dem Modellierer überlassen, welche der Optionen gewählt wird. Aktionen mit einer konstanten Dauer ≥ 0 zu versehen, stellt dabei die einfachste Lösung dar. Durch die Integration von DESMO-J stehen darüber hinaus stochastische Verteilungen zur Verfügung, um Aktionen eine zufällig bestimmte Dauer zuzuordnen. Aufwendigere zustands- oder zeitabhängige Berechnungen werden von FAMOS dagegen nicht weiter unterstützt.

Zwei Verhaltensbausteine liefern Konstrukte für bedingtes Warten, d. h. Warten für unbestimmte Zeit auf das Eintreffen eines Signals. Für die prozessorientierte Modellierung

des Agentenverhaltens (Klasse `ProcessBehaviour`) existiert mit der Methode `waitForSignal()` eine solche Möglichkeit, den Prozessablauf zu unterbrechen. In der Verhaltensmodellierung mit ausführbaren Zustandsdiagrammen (Klasse `StateMachine`) ist das Warten auf ein bestimmtes Signal der Standardfall, um Transitionen zu aktivieren und somit von einem Zustand in den nächsten überzugehen. Um einem Zustand eine bestimmte Dauer zuzuordnen, steht ein spezielles Zeitereignis (Klasse `TimeOut`) zur Verfügung.

Sofortige Wahrnehmung von Veränderungen Um einem Agenten auch während einer programmtechnisch passiven Phase die sofortige Reaktion auf Veränderungen in der Umgebung zu ermöglichen, dreht FAMOS, wie im Konzept in Abschnitt 4.6 beschrieben, das Prinzip der ständigen Beobachtung der Umgebung durch einen Agenten um und lässt stattdessen die Umgebung den Agenten auf Veränderungen aufmerksam machen. Dies geschieht über ein spezielles Signal vom Typ `Alert`, welches die Umgebung an alle Agenten sendet, in deren Wahrnehmungsbereich die Veränderung stattgefunden hat. Das Eintreffen eines Signals sorgt dafür, dass der Agent in seiner aktuellen Handlung unterbrochen und zur Reaktivierung vorgemerkt wird.

Ähnliches gilt für Nachrichten (Klasse `Message`). Diese sind als Spezialisierung der Klasse `ExternalSignal` realisiert und werden somit wie alle anderen Signale auf der internen Ereignisliste eines Agenten zwischengespeichert. Der Erhalt einer Nachricht bewirkt dadurch automatisch eine Reaktivierung des Empfängers für den Zeitpunkt des Nachrichtenempfangs.

5.2.4.2. Integration eines bestehenden Simulationspakets

Für die Simulationsinfrastruktur stützt sich FAMOS auf DESMO-J. Durch die Realisierung von Multiagentenmodellen als spezielle Modelle (`Multiagentmodel` ist eine Spezialisierung von `desmoj.Model`) kann DESMO-Js Experiment-Klasse unverändert für die Simulation von Multiagentenmodellen eingesetzt werden. Damit werden Scheduler, Ereignisliste und Simulationsuhr von DESMO-J bereitgestellt.

Nebenläufigkeit von Agenten und Umwelt In jedem Multiagentensystem handeln Agenten und ggf. deren Umgebung nebenläufig. Die konzeptuelle Nebenläufigkeit aktiver Entitäten wird in der ereignisgesteuerten Simulation von der Ablaufsteuerung sichergestellt, indem der Scheduler für die korrekte Sortierung der Ereignisliste sorgt und die Simulationsuhr immer auf den Zeitpunkt des nächsten Ereignisses fortschaltet. Bei Gleichzeitigkeit von Ereignissen wird (mehr oder weniger) automatisch eine geeignete Serialisierung gefunden (vgl. Kapitel 2.1.3). In DESMO-J stehen hierzu ein Prioritätsmechanismus und – mit den für FAMOS vorgenommenen Erweiterungen – eine zufällige Abarbeitungsreihenfolge zur Verfügung.

Mit der technischen Realisierung von Agenten und Umwelt als aktive Entitäten und den im vorigen Abschnitt beschriebenen Anpassungen ist in FAMOS garantiert, dass sowohl Agenten als auch Umgebung jeweils für den korrekten Zeitpunkt der nächsten Aktivierung auf der globalen Ereignisliste vorgemerkt sind. Damit ergibt sich automatisch die konzeptuelle Nebenläufigkeit von Agenten und dynamischer Umgebung.

Die Behandlung gleichzeitiger Ereignisse, welche sich aus dem Verhalten eines einzelnen Agenten ergeben, ist dagegen in die Verantwortung des Modellierers gelegt. Jeder Agent ist nur genau einmal auf der globalen Ereignisliste eingetragen. Bei seiner Aktivierung werden alle bis zu diesem Zeitpunkt eingetroffenen Signale zur Verarbeitung an den Verhaltensbaustein weitergeleitet. Dieser übernimmt dann die Kontrolle über die Reihenfolge der Verarbeitung.

Trennung von Modell und Experiment Die klare Trennung von Modellen und Experimenten ist eine zentrale Anforderung an Simulationssysteme (Page 1991, S. 167). Sie ermöglicht es, verschiedene Experimente mit einem Modell durchzuführen und ggf. auch ein Experiment mit verschiedenen Modellen durchzuführen, beispielsweise um konkurrierende Modelle [16] zu vergleichen. In DESMO-J ist dies umgesetzt, indem Modell und Experiment als eigenständige Objekte konzipiert sind, welche erst zur Laufzeit dynamisch verknüpft werden. Dies ermöglicht es u. a., verschiedene Experimente mit demselben Modell auszuführen. Jedes Experiment besitzt standardmäßig vier Ausgabekanäle für die Aufzeichnung von Daten (*Report*, *Debug*), Ablaufprotokoll (*Trace*) und Fehlermeldungen (*Error*).

DESMO-J unterstützt darüber hinaus Experimentreihen mit einer experimentweiten Kontrolle aller Zufallszahlenströme. Dies ist nützlich für alle Modelle mit stochastischen Komponenten, da so die Durchführung mehrerer Simulationsläufe mit unterschiedlichen Startwerten für den Zufallszahlengenerator vereinfacht wird.

Wiederverwendung weiterer Komponenten Mit der Integration von DESMO-J stehen in FAMOS alle in DESMO-J vordefinierten Modellkomponenten zur Verfügung. Neben aktiven Entitäten und Prozessen, welche für die Abbildung von dynamischem Verhalten unterhalb des Agentenlevels verwendet werden können und in FAMOS für die Modellierung der Umweltdynamik vorgesehen sind (vgl. Abschnitt 5.2.2), umfasst dies die *Blackbox*-Komponenten Warteschlangen, Zufallszahlenströme und statistische Datensammlungsobjekte.

Die technische Realisierung der Agentenklassen als Spezialisierung der Klasse `Entity` erlaubt es, Warteschlangen bei Bedarf auch auf Agenten anzuwenden. Von größerem Nutzen für Multiagentenmodelle sind jedoch die auf einem Zufallszahlengenerator basierenden stochastischen Verteilungen. Diese können u. a. verwendet werden, um Agentenattribute zu initialisieren oder die Dauer von modellspezifischen Handlungen eines Agenten zu bestimmen. Die von DESMO-J zur Verfügung gestellten statistischen Datensammelobjekte liefern vornehmlich aggregierte Daten für beliebige numerische Modellgrößen. Um Daten über individuelle Merkmale einzelner Agenten aufzeichnen zu können, wurde FAMOS um einen entsprechenden Datensammler ergänzt (vgl. Abschnitt 5.2.5).

[16] Konkurrierende Modelle bilden das gleiche Realsystem ab, beruhen jedoch auf unterschiedlichen, einander ausschließenden Hypothesen (Zeigler 1984, S. 13).

5.2.4.3. Import-Schnittstellen

Die agentenbasierte Simulation ist insbesondere für die Modellierung heterogener Entitäten mit individuellem Verhalten geeignet. Für die Beschreibung dieser Entitäten werden empirische Daten benötigt. Handelt es sich um ein Modell mit räumlicher Ausprägung, kommt der Bedarf für räumliche Daten hinzu. Um die Konfiguration der Modellbestandteile zu unterstützen, stellt FAMOS daher geeignete Schnittstellen für den Import der benötigten Daten zur Verfügung.

XML-Schnittstelle für räumliche Daten Räumliche Daten werden vorwiegend in Geographischen Informationssystemen verwaltet, bearbeitet und analysiert. Es ist daher für ein Simulationswerkzeug wie FAMOS, das explizit eine flexible räumliche Modellierung unterstützt, von großer Bedeutung, den Import von GIS-Daten zu ermöglichen. Zwar lassen sich einfache grid-basierte Raummodelle relativ problemlos durch wenige Parameter spezifizieren und so im Modell direkt konstruieren, doch graph-basierte Raummodelle sind in der Regel so komplex, dass sie aus einer externen Quelle eingelesen werden müssen.

Ein Problem, das sich in diesem Zusammenhang stellt, ist das Format der einzulesenden Daten. Es existieren Dutzende von verschiedenen GIS mit jeweils proprietären Datenformaten, die nicht alle von FAMOS unterstützt werden können. Statt sich auf eines der marktführenden GIS zu beschränken, wurde in FAMOS mit der Entwicklung eines eigenen XML-basierten Datenformats eine allgemeinere Lösung gewählt.

Der Import geographischer Daten ist prototypisch auf Basis eines eigenen XML-Schemas[17] realisiert. Dies erlaubt die Definition eigener Datentypen, einschließlich Vererbungs- und Substitutionsbeziehungen. Darüber hinaus können XML-Dokumente, die einem XML-Schema folgen, automatisch auf Gültigkeit geprüft werden. Das FAMOS-XML-Schema bildet die Struktur des abstrakten Raummodells nach, beschränkt sich jedoch aus dem oben genannten Grund auf Graphen. Das Wurzelelement `<Space>` enthält die Beschreibung des Graphen (Element `<Graph>`), eine optionale Liste von Attributgebieten (Element `<AttributeArea>`) sowie eine ebenfalls optionale Liste von im Raum positionierten Objekten (Element `<SituatedObject>`). Die formale Definition des Schemas sowie eine Erläuterung aller enthaltenen Elemente ist in Anhang A.3 gegeben.

Für den Import eines als XML-Dokument vorliegenden Raummodells greift FAMOS auf das sogenannte *Data Binding* zurück. Diese Methode erzeugt aus den Daten innerhalb des XML-Dokuments direkt die entsprechenden Objekte einer Programmiersprache, ohne Umweg über ein die XML-Struktur repräsentierendes Dokumentmodell wie es die beiden alternativen Ansätze zum Einlesen von XML-Dateien – SAX (Simple API for XML) und DOM (Document Object Model) – verwenden (vgl. z. B. McLaughlin und Edelson 2007). Data Binding erlaubt somit die Manipulation der Daten auf der Ebene des Anwendungsprogramms anstatt auf der niedrigeren Ebene des spezifischen XML-Datenformats.

Der Transfer von XML-Repräsentation in programmiersprachliche Repräsentation der Daten (Objekt-Erzeugung) wird *Unmarshalling*, der umgekehrte Vorgang *Marshalling* genannt. Für Java existieren verschiedene Werkzeuge, welche diese Funktionalität bereitstellen. FAMOS verwendet Castor[18], eines der wenigen frei verfügbaren Data-Binding-

[17] `http://www.w3c.org/XML/Schema` [25.1.2014].
[18] `http://castor.codehaus.org/1.2/index.html` [25.1.2014].

Frameworks, welche zum Implementationszeitpunkt der Import-Schnittstelle zur Verfügung standen. Castor ist heute ein Open-Source-Projekt mit einer breiten Anwenderbasis. Es erlaubt flexibles Marshalling und Unmarshalling mittels einer sogenannten *Mapping*-Datei, in der – wiederum in einem XML-Dialekt – die Abbildung von Java-Klassen auf XML-Strukturen detailliert spezifiziert werden kann. Dies ist notwendig, wenn die verwendeten Klassen nicht *Java-Bean* konform sind, so dass z. B. nicht automatisch auf die zu Attributen gehörenden Methoden geschlossen werden kann. Das für FAMOS erstellte *Mapping* findet sich in Anhang A.3.3.

Das Aufkommen von GML (*Geography Markup Language*) bestätigt die Entscheidung für eine XML-Schnittstelle zum Import geographischer Daten. GML ist ein XML-Dialekt zur einheitlichen Repräsentation von Geo-Daten mit dem Zweck, Austausch und Speicherung geographischer Daten über das Internet zu erleichtern. Durch die Verwendung von XML ist ein sowohl hersteller- als auch implementations-unabhängiges Format gewährleistet. GML wird vom Open Geospatial Consortium (OGC)[19] festgelegt und ist seit 2007 eine offizielle Internationale Norm der ISO (ISO 2007). Viele GIS-Produkte bieten eine Unterstützung von GML an. Durch Anpassung des FAMOS-XML-Schemas zu einer Spezialisierung von GML (siehe Anhang A.3.4) ergibt sich somit eine standardisierte Schnittstelle zu GIS.

Schnittstelle für empirische Eingabedaten Temporäre Entitäten, die das modellierte System zu bestimmten Zeitpunkten betreten und nach einiger Zeit wieder verlassen, sind Bestandteil der meisten ereignisdiskreten Modelle. Ankunftszeitpunkte und ggf. Attribute dieser Entitäten werden oft als zufällig angenommen und mittels stochastischer Verteilungen modelliert. Entitäten können dann während des Simulationslaufs direkt innerhalb des Modells erzeugt werden. Das Framework DESMO-J unterstützt die Erzeugung temporärer Entitäten beispielsweise durch einen speziellen Ankunftsprozess (Klasse `ArrivalProcess`), der eine Folge von Simulationsprozessen an stochastisch bestimmten Simulationszeitpunkten aktiviert. Die Simulationsprozesse selbst werden mit einer vom Modellierer zu implementierenden Fabrikmethode generiert (Claassen 2001).

FAMOS verallgemeinert diesen Ankunftsprozess in der Klasse `ExternalObjectArrival` zu einem Prozess, der aus einer Datenquelle (*Interface* `DataSource`) beliebige Objekte einliest. Dies ermöglicht es, empirische Daten für die Erzeugung der temporären Entitäten zu verwenden. Darüber hinaus ist der Ankunftsprozess nicht mehr auf Simulationsprozesse beschränkt, sondern kann diverse Entitätstypen von einfachen Objekten bis hin zu Agenten umfassen. Der Modellierer hat zu diesem Zweck drei als abstrakt in `ExternalObjectArrival` deklarierte Methoden zu überschreiben: `createObject()` erzeugt aus einer von der Datenquelle ausgelesenen Eigenschaftsliste eine neue Entität, während `getArrivalTime()` deren Ankunftszeitpunkt im System bestimmt, ggf. unter Benutzung beliebiger Attribute der Entität. Der Ankunftsprozess wartet anschließend bis zu diesem Simulationszeitpunkt, bevor er über die Methode `scheduleObject()` die neue Entität auf benutzerdefinierte Weise das Modell betreten lässt.

Für das Einlesen empirischer Daten stehen zwei verschiedene Datenquellen zur Verfügung. Die Klasse `FileDataSource` importiert ASCII-Dateien, in denen jede Zeile die

[19] `http://www.opengeospatial.org/` [25.1.2014]

Attributwerte genau einer Entität enthält, während die Kopfzeile die Attributbezeichner definiert. Die Klasse `DOM_DataSource` dient zum Einlesen von XML-Dateien mit Hilfe des DOM-API[20]. Die Spezifikation jeder Entität in einer solchen Datei muss durch das Tag `<entity>` eingeschlossen sein. Eingebettete Tags und Attribute werden als Bezeichner und Werte beliebiger, möglicherweise hierarchisch strukturierter Eigenschaften der Entität interpretiert. Beide Datenquellen erzeugen nicht direkt Entitäten, sondern Eigenschaftslisten (Objekte der Klasse `PropertyList`), die wiederum wie oben beschrieben vom Ankunftsprozess verarbeitet werden.

5.2.5. Unterstützung der Analyse

Der Schwerpunkt von FAMOS liegt in der Unterstützung der Modellierung, wobei die explizite Raumrepräsentation und der modulare Aufbau von Agenten hervorzuheben sind. Für die Durchführung von Experimenten greift FAMOS auf die Simulationsinfrastruktur von DESMO-J zurück und erweitert sie, wie oben beschrieben, um eine graphische Benutzungsoberfläche. Die Unterstützung der Analyse beschränkt sich zunächst darauf, die von DESMO-J gebotenen Datensammlungs- und Statistikfunktionen um Möglichkeiten zur Aufzeichnung individueller Eigenschaften von Agenten zu ergänzen. Auf diese Weise lassen sich Simulationsergebnisse so exportieren, dass sie mit spezialisierten Werkzeugen wie Tabellenkalkulationsprogrammen oder Statistikpaketen analysierbar werden. Darüber hinaus integriert FAMOS eine graphische Darstellung der räumlichen Umgebung in die Benutzungsoberfläche. Diese zeigt eine simulationsbegleitende Animation der Agentenbewegungen.

5.2.5.1. Aufzeichnung von Daten

Für die Auswertung agentenbasierter Simulationen werden oft Daten über individuelle Merkmale einzelner Agenten benötigt. Die von DESMO-J zur Verfügung gestellten statistischen Datensammelobjekte für die Berechnung aggregierter Kennzahlen werden in FAMOS daher um ein generisches Datensammelobjekt für Attribute einzelner Entitäten ergänzt, das eine tabellarische Auflistung der gesammelten Werte u. a. im Ergebnisreport ermöglicht. Die Klasse `IndividualObserver` kann für eine beliebige Anzahl von zu beobachtenden Entitäten eines Typs eine Reihe von Merkmalen aufzeichnen; die Tabelle der Werte enthält pro Entität eine Zeile und pro Merkmal eine Spalte. Darüber hinaus lassen sich für numerische Merkmale automatisch grundlegende statistische Kenngrößen (Minimum, Maximum, arithmetisches Mittel und Standardabweichung) berechnen und als abschließende Zeilen in die Tabelle aufnehmen. Anhand der Methode `setDisplayMode()` wird der Umfang der Tabelle konfiguriert: Sie kann alternativ nur die individuellen Merkmale, nur die aggregierten Kennzahlen oder beides enthalten.

Typ und aufzuzeichnende Merkmale werden modellabhängig definiert, indem ein konkreter `IndividualObserver` implementiert und instantiiert wird. Im Simulationslauf werden mit der Methode `register()` die zu beobachtenden Entitäten beim Beobachter

[20] Diese Programmierschnittstelle ermöglicht es, ein XML-Dokument zu parsen und in eine baumartige Datenstruktur, das sogenannte *Document Object Model*, zu überführen, welche ausgelesen und manipuliert werden kann (McLaughlin und Edelson 2007, S. 91ff).

angemeldet. Sie überprüft, ob deren Klasse dem festgelegten Typ entspricht und weist Entitäten eines nicht kompatiblen Typs mit einer Warnung zurück, um mögliche Konflikte beim Zugriff auf nicht vorhandene Merkmale zu vermeiden. Die Klasse `IndividualObserver` ist durch die Ableitung von `desmoj.Reportable` in das Reportsystem von DESMO-J eingebunden . Sie liefert auf die Anfrage `createReporter()` ein Objekt vom Typ `IndividualReporter`, das wiederum die aufgezeichneten Werte in Form einer Tabelle aufbereitet.

Reporter-Objekte können in DESMO-J über das interne Nachrichtensystem an beliebige Ausgabekanäle weitergeleitet werden. Standardmäßig wird diese Funktionalität für die Erstellung des Ergebnisreports verwendet, wobei die über `getEntries()` abgefragten Einträge eines Reporters im HTML-Format ausgegeben werden. In FAMOS ist diese Export-Schnittstelle erweitert worden um Ausgabekanäle, die ASCII-Dateien mit Tabulatoren als Trennzeichen erzeugen bzw. eine Anbindung an SQL-fähige Datenbanken erreichen. Letztere nutzt die von Java zur Verfügung gestellte Programmierschnittstelle für Datenbankzugriffe (*JDBC API*)[21] und ist derzeit für die Datenbanksysteme *mySQL* und *MS Access* realisiert.

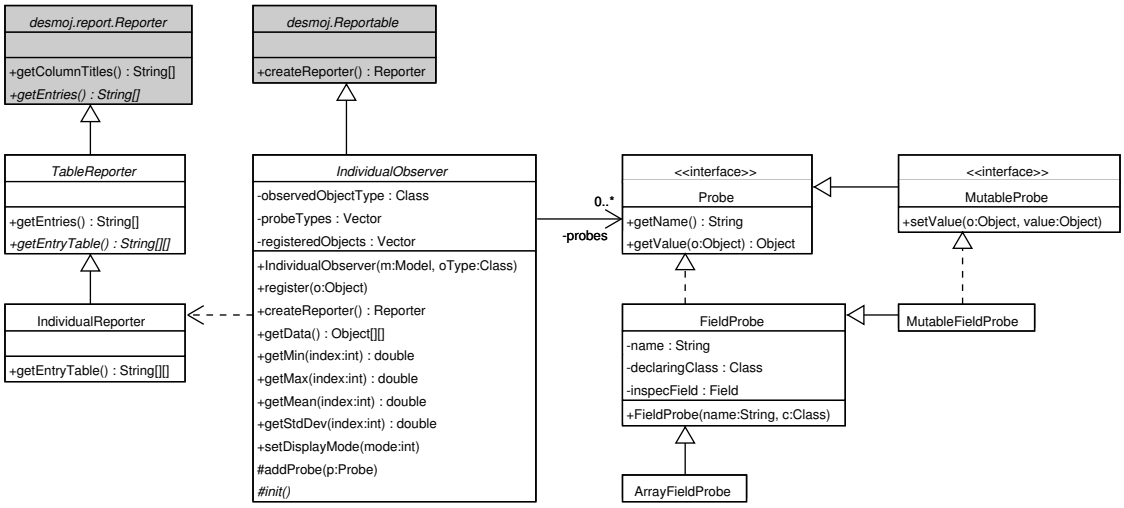

Abbildung 5.15.: Datensammlung auf mikroskopischer Ebene in FAMOS

Die Funktionalität des `IndividualObserver`, beliebige Attribute der beobachteten Objekte auszulesen, stützt sich auf das Konzept der Sonde (*Probe*). Es wurde aus dem Simulationsframework *Swarm* entlehnt, in dem Sonden dem lesenden und schreibenden Zugriff auf Zustandsvariablen von Agenten dienen (Minar et al. 1996, S. 6). Einem `IndividualObserver`-Objekt werden die zu beobachtenden Merkmale in Form solcher Sonden übergeben (Methode `addProbe()`). Die Schnittstelle `Probe` definiert zunächst ausschließlich lesenden Zugriff über die Methode `getValue()`, der in der Schnittstelle `MutableProbe` durch schreibenden Zugriff ergänzt wird (vgl. Abbildung 5.15). Um die

[21] `http://www.oracle.com/technetwork/java/overview-141217.html` [28.1.2014].

Schnittstellen so flexibel wie möglich zu gestalten, wird den Zugriffsmethoden das zu inspizierende Objekt als Parameter übergeben.

Umgesetzt sind in FAMOS Sonden für einzelne Objekt-Attribute (Klassen `FieldProbe` und `MutableFieldProbe`), welche per Introspektion der im Konstruktor spezifizierten Klasse Zugriff auf ein in dieser Klasse oder eine ihrer Oberklassen definiertes Attribut erhalten. Sie werden nicht nur im Zusammenhang mit der Datensammlung, sondern – wie in *Swarm* und vergleichbaren Frameworks – auch zur interaktiven Inspektion von Agenten im Rahmen der simulationsbegleitenden Visualisierung verwendet.

5.2.5.2. Visualisierung

Eine simulationsbegleitende Visualisierung kann insbesondere für raumbezogene Modelle die Validierung und Ergebnisanalyse wesentlich unterstützen (vgl. z. B. Ropella et al. 2002). In FAMOS übernimmt ein Objekt der Klasse `EnvironmentView` diese Aufgabe für die Visualisierung der räumlichen Umgebung. Es wird als graphischer Beobachter (Oberklasse `GraphicalObserver`) in die Benutzungsoberfläche integriert, um die DESMO-J erweitert wurde (vgl. Abschnitt 5.1.2). Da die Visualisierung wie die gesamte graphische Benutzungsoberfläche bisher nur prototypisch umgesetzt ist, abstrahiert die folgende Darstellung von den Details der Implementation und beschränkt sich auf die grundlegenden Entwurfsentscheidungen. Das Klassendiagramm in Abbildung 5.16 zeigt daher nur die für das Verständnis relevanten Klassen, Methoden und Beziehungen.

Aufgebaut ist die Darstellung der räumlichen Umgebung in Anlehnung an das Layer-Prinzip Geographischer Informationssysteme aus mehreren Schichten: Zuunterst eine Darstellung des Raummodells (Klasse `SpaceView`), darüber beliebige Ebenen mit Objekten (noch nicht realisiert) und als oberstes die Darstellung der Agenten (Klasse `AgentPlane`). Jede Darstellungsebene lässt sich einzeln aus- und einblenden über entsprechende Auswahlschalter am rechten oberen Fensterrand (vgl. Abbildung 5.17).

Ein `EnvironmentView` benötigt eine Referenz auf die darzustellende Umgebung (Objekt der Klasse `Environment`). Über diese erhält er Zugriff auf das Raummodell und die situierten Agenten und Objekte, um deren graphische Darstellungen automatisch konstruieren zu können. Da die Visualisierung eines Raummodells eng mit demselben verknüpft ist, verfügen in FAMOS die Raummodelle über die Fähigkeit, eine passende graphische Darstellung zurückzuliefern. Dies ist über die abstrakte Methode `getSpaceView()` in der abstrakten Oberklasse `Space` festgelegt. Implementiert ist sie derzeit für Graphen (Klasse `Graph`) und zweidimensionale Gitter (Klasse `Grid2D`). Da Gitterzellen ihre geometrische Form als Folge von Punkten speichern, kann eine gemeinsame Visualisierung für alle zweidimensionalen Gitter realisiert werden.

Über eine Farbkodierung kann die räumliche Verteilung eines numerischen Attributs innerhalb eines `SpaceView` visualisiert werden. Hierzu ist die gewünschte Basisfarbe, die Anzahl der Farbstufen und das darzustellende Attribut bei der Konstruktion des `SpaceView` anzugeben. Die im Raummodell vorliegenden Attributwerte werden daraufhin in die spezifizierte Anzahl Klassen sortiert und auf eine von der Basisfarbe nach Weiß abgestufte Farbskala abgebildet. Diejenigen Raumelemente, die das darzustellende Attribut besitzen, werden entsprechend eingefärbt gezeichnet. Diese Form der Visualisierung ließe sich auf mehrere Attribute erweitern, indem jede Attributverteilung in einem eige-

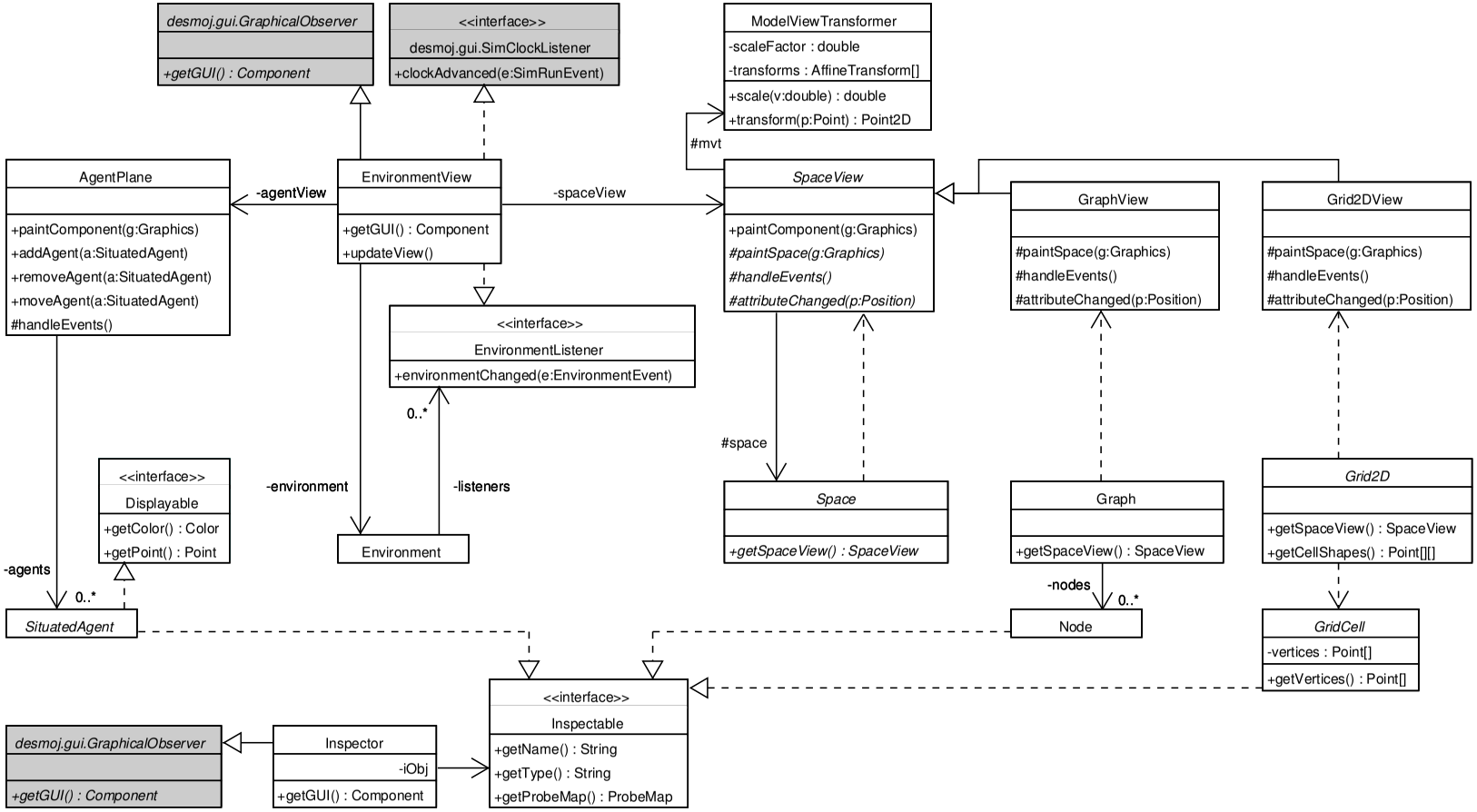

Abbildung 5.16.: Framework zur Visualisierung der räumlichen Umgebung. Die Klasse `EnvironmentView` stellt das Raummodell und die situierten Agenten in zwei übereinanderliegenden Schichten dar (`SpaceView` bzw. `AgentPlane`). Agenten und Raumelemente können durch Anklicken inspiziert werden (Popup-Fenster `Inspector`). Die grau unterlegten Klassen bilden die Schnittstelle zur graphischen Benutzungsoberfläche (Paket `desmoj.gui`).

nen Layer dargestellt wird.

Die Umrechnung der Koordinaten des räumlichen Bezugssystems in das für die graphische Darstellung benötigte Bezugssystem geschieht über ein Objekt der Klasse `ModelViewTransformer`. Dieser verwendet dazu eine Reihe von affinen Transformationen, u. a. Translation für die Verschiebung des Ursprungs, der bei 2D-Grafik in Java generell für die linke obere Fensterecke definiert ist, und Skalierung, um die Ausdehnung des Raummodells an die Fenstergröße anzupassen. Das Raummodell legt somit die für alle anderen Layer des `EnvironmentView` ebenfalls zu verwendenden affinen Transformationen fest.

Derzeit ist als einzige weitere Darstellungsschicht ein Layer für situierte Agenten (Klasse `AgentPlane`) realisiert. Dieser führt eine Liste aller in der Umgebung angemeldeten Agenten und zeichnet für jeden Agenten einen ausgefüllten Kreis mit dessen Position als Mittelpunkt. Die Größe des Kreises ist abhängig vom Skalierungsfaktor des Raummodells. Die zu verwendende Farbe wird vom Agenten vorgegeben; Standard ist rot. Dies kann modellspezifisch verändert werden, indem die Methode `getColor()` überschrieben wird: Im Segregationsmodell (siehe den folgenden Abschnitt 5.3) definieren die Agenten ihre Gruppenzugehörigkeit beispielsweise über die Farben blau und grün.

Veränderungen in der räumlichen Umgebung, die im Laufe der Simulation auftreten, werden der graphischen Darstellung auf Basis des bekannten Beobachtermusters mitgeteilt. Ein `EnvironmentView` meldet sich dazu als Beobachter vom Typ `EnvironmentListener` an der zugehörigen Umgebung an. Diese teilt ihren Beobachtern eine Änderung in Form eines speziellen Ereignisses (Klasse `EnvironmentEvent`, in Abbildung 5.16 nicht dargestellt) mit; berücksichtigt werden derzeit die Veränderung von Attributwerten, Positionswechsel von Agenten und das Betreten bzw. Verlassen der Umgebung durch einen Agenten. Die agentenspezifischen Ereignisse leitet der `EnvironmentView` an seine `AgentPlane` weiter, indem er die entsprechenden Methoden `moveAgent()`, `addAgent()` bzw. `removeAgent()` aufruft. Die Veränderung eines Attributwerts wird dagegen an den `SpaceView` übermittelt (Methode `attributeChanged()`).

Jeden Fortschritt der Simulationszeit erfährt der `EnvironmentView` über ein Ereignis, das Beobachtern der Simulationsuhr mitgeteilt wird (Schnittstelle `SimClockListener`). Daraufhin veranlasst er, dass alle Layer neu gezeichnet werden (Methode `updateView()`). Auf diese Weise ergibt sich eine Animation der Bewegung von Agenten während der Durchführung eines Experiments.

Zusätzlich zu dieser simulationsbegleitenden Visualisierung der räumlichen Umgebung kann der aktuelle Zustand von Agenten und Raumelementen interaktiv abgefragt werden. Situierte Agenten sowie Knoten und Gitterzellen implementieren zu diesem Zweck die Schnittstelle `Inspectable`, die in der Methode `getProbeMap()` eine Liste von Sonden liefert, die Zugriff auf Attribute der Klassen ermöglichen. Diese werden von einem `Inspector`-Objekt in einem Popup-Fenster tabellarisch angezeigt (siehe Abbildung 5.17). Die Methoden `getName()` und `getType()` dienen zur Beschriftung des Fensters. Ein solches Fenster öffnet sich, wenn ein Benutzer die graphische Repräsentation eines Agenten oder Raumelements im `EnvironmentView` mit der Maus anklickt. `SpaceView` und `AgentPlane` definieren diese Reaktion auf Mausklicks in ihren jeweiligen Verarbeitungsroutinen für interaktive Ereignisse (Methode `handleEvents()`).

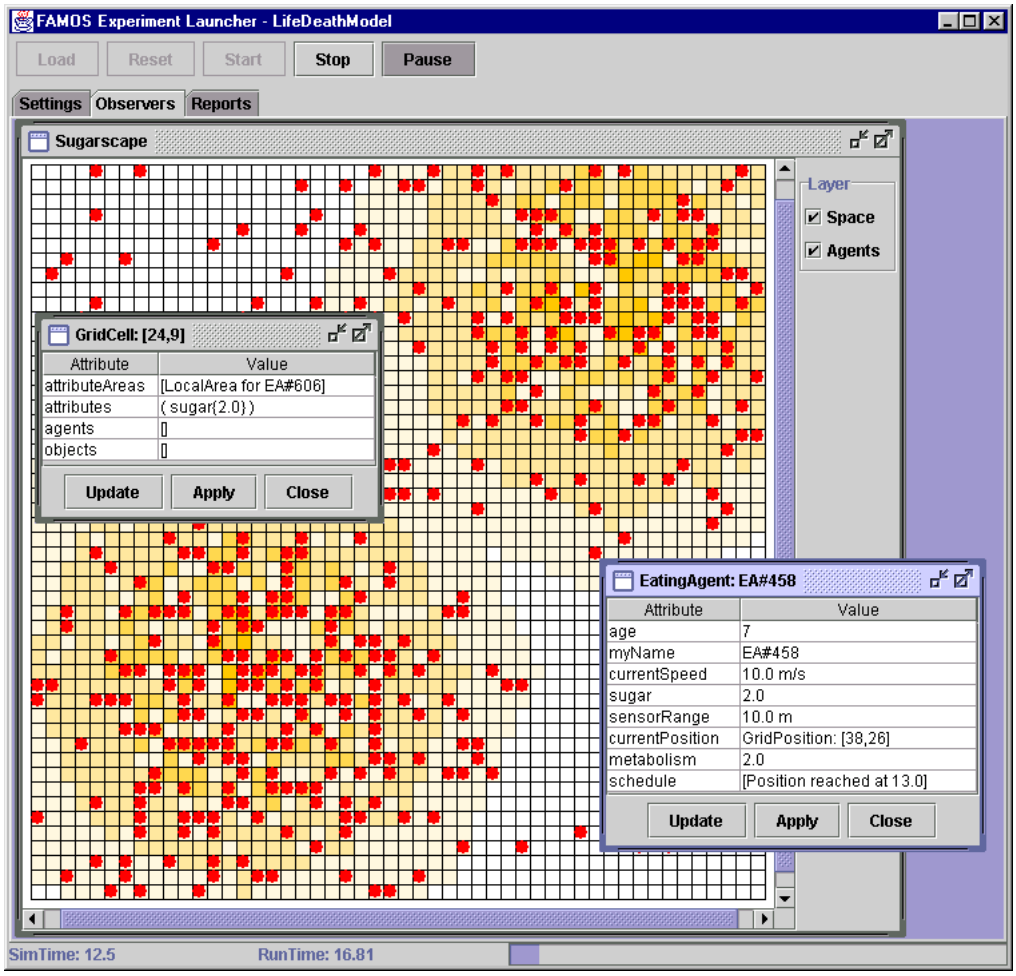

Abbildung 5.17.: Visualisierung der räumlichen Umgebung am Beispiel des *Sugarscape*-Modells. Das als Raummodell verwendete regelmäßige Gitter zeigt eine farbkodierte Attributwertverteilung. Die Agenten sind als rote Kreise dargestellt. Für einen Agenten und eine Gitterzelle ist jeweils ein Inspektions-Fenster geöffnet.

5.3. Testbeispiel 1: Schellings Segregationsmodell

Ein einfaches Modell, welches die Emergenz von Makrophänomenen auf der Grundlage individueller Entscheidungen behandelt, ist das von Thomas Schelling entwickelte Segregationsmodell (Schelling 1978, S. 147–155). Es zeigt, wie sich bei der räumlichen Anordnung zweier unterschiedlicher Gruppen von Akteuren getrennte Bereiche ergeben trotz relativ großer Toleranz aller Beteiligten gegenüber der jeweils anderen Gruppe. Angewandt auf Städte in den USA, verdeutlicht es den Effekt der Rassentrennung (Segregation) zwischen Weißen und Schwarzen. Dieses Modell, das Schelling bereits 1969 erstmals beschrieben hat, kann als eine der ersten agentenbasierten Simulationen bezeichnet werden (vgl. z. B. Axelrod 1997, S. 215; Macal und North 2006) – auch wenn das Originalmodell ohne Computerunterstützung auskommt.

Es wurde als erstes Testbeispiel ausgewählt, weil sich mit diesem einfachen Modell bereits wesentliche Konzepte von FAMOS darstellen lassen, insbesondere die Flexibilität des Raummodells.

5.3.1. Beschreibung des Modells

In dem von Schelling als *self-forming neighborhood model* (Schelling 1978, S. 147) bezeichneten Modell werden zwei homogene Gruppen von Individuen unterschieden, die sich in einer gemeinsamen räumlichen Umgebung befinden. Diese Umgebung besteht aus einem zweidimensionalen Gitter aus quadratischen Zellen, in dem sich pro Gitterzelle höchstens ein Individuum aufhalten kann. Das Gitter hat feste Grenzen, eine Moore-Nachbarschaft und sollte mindestens Schachbrettgröße aufweisen, d. h. 8×8 Zellen umfassen. Das Verhalten der Individuen wird beeinflusst durch die räumlich benachbarten Individuen: Wenn der Anteil an Nachbarn gleicher Art (interpretiert als Rasse, Konfession, Geschlecht o. ä.) unter einen vorher festgelegten Schwellwert sinkt, ist das Individuum mit seiner aktuellen Position unzufrieden und wechselt zu einer anderen Position. Im Originalmodell ist dies die nächstgelegene Gitterzelle, in welcher der Bedarf des Individuums erfüllt, es also zufrieden ist. Diese Regel wird in Computermodellen meist ersetzt durch zufällige Wahl einer freien Position, um die Implementation zu vereinfachen (vgl. u. a. Epstein und Axtell 1996, S. 166; Axelrod 1997, S. 215; Resnick 2000, S. 85).

Da Schelling ein manuell ausführbares Modell beschreibt, geht er explizit auf die Durchführung von Experimenten ein. Modellparameter sind Anzahl und Anfangspositionen der Individuen sowie deren jeweilige Toleranz-Schwellwerte, die zumindest für alle Individuen einer Art identisch sind. Diese Einschränkung auf homogene Gruppen erleichtert die manuelle Ausführung. Als Startwerte schlägt Schelling gleiche Anzahlen von Individuen vor mit einem „moderaten“ Schwellwert von etwas mehr als einem Drittel. Die Individuen können zufällig über das Gitter verteilt werden oder wie im regelmäßig abwechselnden Schachbrettmuster mit kleinen, zufällig bestimmten Störungen[22]. Ausgehend von der Anfangsverteilung sind jeweils die unzufriedenen Individuen zu bestimmen und an

[22] Wenn die vier Eckzellen freigelassen werden, ist das reine Schachbrettmuster für Toleranz-Schwellwerte bis 0.5 eine stabile Konfiguration und eignet sich daher nicht, um die Dynamik des Segregationsprozesses zu zeigen (Schelling 1978, S. 148f).

andere Positionen zu bewegen, bis sich auf der Ebene des Systems ein stabiles Muster ergibt oder Zyklen bzw. chaotisches Verhalten erkennbar sind. Für die Reihenfolge, in der die einzelnen Individuen zu aktualisieren sind, führt Schelling mehrere Möglichkeiten an und bemerkt ausdrücklich, dass die Ergebnisse von der gewählten Reihenfolge unabhängig sind (Schelling 1978, S. 148).

5.3.2. Implementation in FAMOS

Die in FAMOS implementierte Version[23] des Segregationsmodells besteht aus einer Menge von einfachen, rein reaktiven Agenten und einer Umgebung ohne eigenes dynamisches Verhalten. Der Raum wird durch ein zweidimensionales Gitter repräsentiert, wobei im Gegensatz zum oben beschriebenen Originalmodell zwischen verschiedenen Ausprägungen (`RectangularGrid` für quadratische Zellen, `HexagonalGrid` für hexagonale Zellen oder `IrregularGrid2D` für unregelmäßige Zellen) gewählt werden kann. Die Implementation der Agenten ist unabhängig vom verwendeten speziellen Raummodell, da nur auf die Schnittstelle der abstrakten Oberklasse `Grid2D` zugegriffen wird.

Um in FAMOS ein lauffähiges Modell zu erhalten, sind zumindest die Hot-Spot-Klassen `MultiAgentModel`, `Agent` bzw. eine der beiden Unterklassen sowie eine Verhaltensbeschreibung (Oberklasse `Behaviour`) zu realisieren. Für die Benutzung der graphischen Experimentierumgebung ist zusätzlich eine konkrete Klasse von `MASModelFactory` abzuleiten.

Als Ausgangsklasse für die Agenten des Segregationsmodells wurde `MobileAgent` gewählt, da die Agenten sich im Raum bewegen können. Alle Agenten sind im Prinzip identisch; die beiden Typen unterscheiden sich nur in einem Attribut, das in dieser Implementation durch ihre Farbe (blau oder grün) abgebildet wird. Ein Agent reagiert ausschließlich auf den lokalen Zustand der Umgebung. Sein Wahrnehmungsbereich ist dabei auf die direkt benachbarten Zellen beschränkt, d. h. auf acht Zellen bei Nutzung von `RectangularGrid`, sechs Zellen bei `HexagonalGrid` und einer variablen Anzahl bei `IrregularGrid2D`. Der Sensorradius ist als diskrete Länge (ein Raumelement) spezifiziert, was die flexible Austauschbarkeit des Raummodells vereinfacht. Interaktion zwischen Agenten findet nur indirekt statt, indem ein Agent durch das Aufsuchen einer neuen Position den Umgebungszustand verändert. Das Verhalten eines solchen Agenten lässt sich daher adäquat als Spezialisierung von `SimpleBehaviour` beschreiben. Die einzige Reaktion eines Agenten auf ein ihn betreffendes Ereignis besteht darin, seine Zufriedenheit zu überprüfen und ggf. zufällig eine neue Position zu wählen und sich dahin zu bewegen. Der Schwellwert für Zufriedenheit – im Originalmodell 0.375, d. h. drei von acht Nachbarn gleicher Farbe – ist als Modellparameter realisiert, so dass leicht mit unterschiedlichen Werten experimentiert werden kann.

Da das Modell von der tatsächlichen Bewegung der Agenten abstrahiert und die zeitliche Dauer eines Positionswechsels irrelevant ist, kann ein Agent direkt von einer Position zur nächsten „springen“. Hierfür wurde eine eigene Bewegungsstrategie (Klasse `SegregationStrategy`) von `RandomWalk` abgeleitet, um mehr als nur die direkt benachbarten Zellen in die zufällige Auswahl der neuen Position einbeziehen zu können.

[23] Der vollständige Quelltext steht als Teil des gesamten FAMOS-Pakets in (Meyer 2013) zur Verfügung (Unterpaket `famos.examples.schelling`).

Besonders berücksichtigt werden muss hierbei, dass im Modell zu jedem Zeitpunkt höchstens ein Agent pro Gitterzelle zugelassen ist. Kapazitätsbeschränkungen dieser Art werden in FAMOS bisher nicht unterstützt, so dass sie mit Hilfe eines speziellen Attributs, mit dem Agenten eine Position als reserviert markieren können, nachgebildet wurden.

Die Dauer eines Sprungs wird als beliebiger konstanter Wert ≥ 0 spezifiziert; sie auf 0 zu setzen hieße, die Aktivität der Bewegung als zeitverzugslos aufzufassen und Weggang und Ankunft zu einem Ereignis zusammenzufassen. Für Bewegungen mit konstanter Dauer stellt FAMOS die spezielle Klasse `ConstantTimeMovement` zur Verfügung, die unverändert eingesetzt wurde.

Auf diese Weise ergibt sich quasi eine Zeitsteuerung mit der konstanten Bewegungsdauer als Taktrate, jedoch ohne die damit einhergehende Notwendigkeit, zu jedem Zeitschritt alle Agenten ihre Zufriedenheit überprüfen zu lassen. Die für eine effiziente Ereignissteuerung erforderlichen Anpassungen in FAMOS (vgl. Abschnitt 5.2.4.1) bedeuten, dass jeder Agent sowohl bei der Ankunft an einer neuen Position als auch bei Umgebungsänderungen innerhalb seines Wahrnehmungsbereichs entsprechende Signale der Umgebung erhält, auf die er dann angemessen reagieren kann.

Neben Art und Größe des Gitters und dem Schwellwert für Zufriedenheit kann die Anzahl der Agenten pro Typ als Modellparameter spezifiziert werden. Es ist darüber hinaus möglich, die Agenten unterschiedlich tolerant zu machen, indem sowohl eine untere als auch eine obere Grenze des Zufriedenheitsschwellwerts angegeben wird. Das Modell berechnet dann gleichverteilt für jeden Agenten einen Schwellwert in dem spezifizierten Intervall. Die Anfangsposition der Agenten wird zufällig bestimmt.

Abbildung 5.18 zeigt Ergebnisse je eines Simulationslaufs mit den drei unterschiedlichen Gitterformen. Die linke Spalte stellt den zufälligen Anfangszustand dar, die rechte Spalte den zugehörigen Endzustand. Abgesehen von der Art des Raummodells wurden alle weiteren Parameter gleich gewählt: Das Gitter besteht aus 400 Zellen (20×20 bei den beiden regelmäßigen Gittern), es ist mit 140 blauen und 140 grünen Agenten besetzt, deren Toleranzgrenze einheitlich bei dem Originalwert von 0.375 liegt. Die Dauer für einen Positionswechsel beträgt 1.0 Simulationszeiteinheiten.

5.3.3. Diskussion

Schellings Segregationsmodell kann als Beispiel für die untere Grenze des Spektrums der agentenbasierten Simulation dienen. Zwar besitzt die Umgebung eine räumliche Ausprägung und die Agenten können sich im Raum bewegen, doch lassen sie sich als rein reaktiv klassifizieren. Ihr Verhalten ist mit dem einfachsten Verhaltensbaustein, dem ereignisorientierten `SimpleBehaviour`, leicht zu realisieren. Obwohl das Segregationsmodell daher bei weitem nicht die Funktionalität von FAMOS ausnutzt, eignet es sich gut, die Vorteile der flexiblen Zeit- und Raumrepräsentation zu demonstrieren.

Die Schnittstelle des Raummodells ist in FAMOS so konzipiert, dass ein Benutzer die Agenten seines Modells praktisch unabhängig von den konkret verwendeten Klassen implementieren kann (siehe Abschnitt 4.3 und 5.2.1). Dieser Anspruch wird im Segregationsmodell für zweidimensionale Gitter eingelöst: Im Verhalten eines Agenten wird bei der Prüfung auf Zufriedenheit ausschließlich die raumunabhängige Schnittstelle benötigt, welche die Beschreibung des lokalen Umweltzustands (Klasse `EnvironmentState`) zur

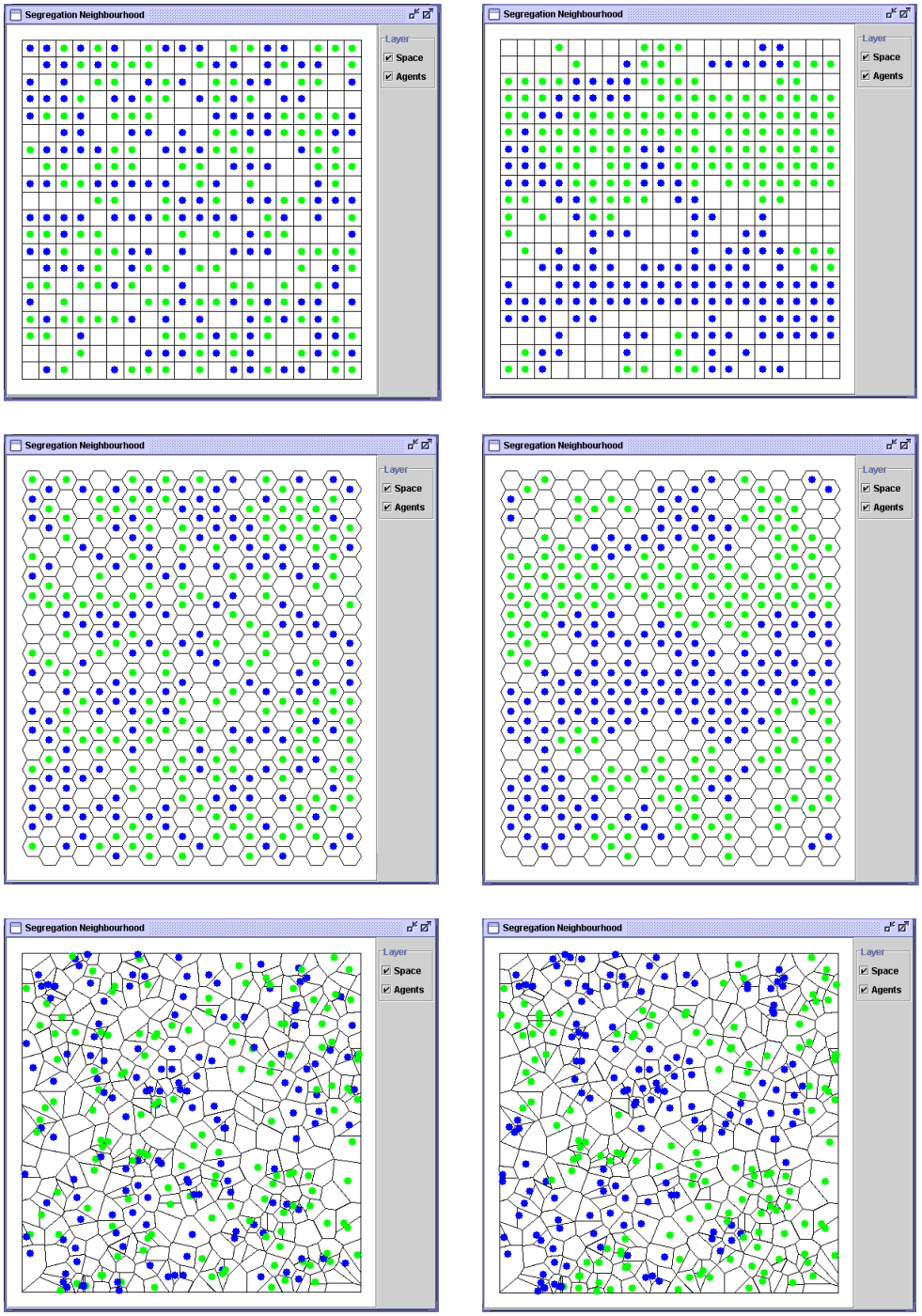

Abbildung 5.18.: Screenshots des Segregationsmodells aus Simulationsläufen mit den drei verschiedenen Gitterformen `RectangularGrid` (oben), `HexagonalGrid` und `IrregularGrid2D` (unten). Die linke Spalte zeigt den jeweiligen Anfangszustand, die rechte Spalte den zugehörigen Endzustand.

Verfügung stellt. Über deren Methode `getAgents()` erhält der Agent Zugriff auf seine direkten Nachbarn und kann so den Anteil der Agenten gleicher Farbe bestimmen. Auch die Bewegung im Raum ist aus Sicht des Agenten unabhängig von einer konkreten Raumklasse, da er nur seine Methode `move()` aufrufen muss. Diese in `MobileAgent` implementierte Methode vereinfacht den Aufruf der gleichnamigen Methode des entsprechenden `Movement`-Objekts, das die Fähigkeit zur Bewegung kapselt.

Die beiden einzigen Klassen, die direkter mit dem Raummodell arbeiten, sind die Modellklasse und die spezielle Bewegungsstrategie. Beide sind jedoch nur auf die abstrakte Oberklasse `Grid2D` angewiesen, um zufällige Anfangspositionen für die Agenten zu generieren bzw. die möglichen freien Zellen zur zufälligen Auswahl einer neuen Position zu bestimmen. Ausschließlich beim Erzeugen des über einen Modellparameter spezifizierten Gitters muss im Modell einmal auf die konkreten Unterklassen `RectangularGrid`, `HexagonalGrid` und `IrregularGrid2D` Bezug genommen werden.

Das mit dem Segregationsmodell abgebildete System weist typische Merkmale eines zeitdiskreten Systems auf und eignet sich daher gut für die von FAMOS propagierte ereignisgesteuerte Simulation: Nach der Initialisierungsphase, in der alle Agenten einmal testen müssen, ob sie mit ihrer aktuellen Position zufrieden sind, ist eine erneute Überprüfung nur nötig, wenn sich der lokale Zustand der Umgebung verändert hat. Bewirkt werden solche Zustandsänderungen ausschließlich durch Positionswechsel von Agenten, wobei Weggang von und Ankunft an einer Position jeweils ein Ereignis darstellen.

Da jeder Agent in FAMOS ein Signal (`Alert`) erhält, wenn in seinem Wahrnehmungsbereich eine Veränderung aufgetreten ist, lässt sich eine Ereignissteuerung für das Segregationsmodell konsequent umsetzen. In der Verhaltensbeschreibung des Agenten muss dazu nur die entsprechende Reaktion auf dieses Signal – Testen der Zufriedenheit und ggf. Bewegen zu einer neuen Position – implementiert werden. Auch das Ereignis „Ankunft an einer neuen Position“ wird von FAMOS automatisch generiert und dem betreffenden Agenten über ein `PositionReached`-Signal mitgeteilt.

Obwohl sich das Segregationsmodell somit für den ereignisgesteuerten Zeitfortschritt anbietet, lässt es sich in FAMOS mit geringfügig höherem Aufwand auch zeitgesteuert realisieren (siehe Abbildung 5.19). Eine saubere (da mit dem Autonomie-Konzept konforme) Lösung besteht darin, dass sich jeder Agent in regelmäßigen Abständen ein internes Signal (`LookSignal`) sendet und nur auf dieses reagiert. Die Reaktion umfasst dann neben dem Überprüfen der lokalen Situation und der eventuellen Wahl einer neuen Position zusätzlich das erneute Senden des internen Signals in Δt Zeiteinheiten. Die von der Umgebung automatisch generierten Signale werden in diesem Fall nicht benötigt und können daher aus Effizienzgründen über die Methode `setNoAlerts()` der Klasse `Environment` abgeschaltet werden.

Wird die Taktrate Δt größer gewählt als die konstante Dauer eines Bewegungsvorgangs, so erhält man praktisch ein zweiphasiges Aktualisierungsschema (siehe Abschnitt 2.1.3): Die durch Bewegungen verursachten Veränderungen der Umgebung sind immer abgeschlossen, bevor alle Agenten wieder auf den Zustand der Umgebung reagieren.

Ein Vergleich der ereignisgesteuerten mit der zeitgesteuerten Variante des Modells unter sonst gleichen Bedingungen ergab im Durchschnitt eine um 20% geringere Rechenzeit

```
// ereignis-gesteuertes Verhalten

protected void initialActions() {
   // enter the environment
   getEnvironment().enter(SegregationAgent.this,
                          getCurrentPosition());
   ...
}


protected void process(Signal s) {
   // react only on signals from the environment
   if (s instanceof Alert) {
      // check neighbourhood
      if (!isContent()) {
         move(); // move to a new position
      }
      // otherwise just stay
   }
}
```

```
// zeit-gesteuertes Verhalten

protected void initialActions() {
   // enter the environment
   getEnvironment().enter(SegregationAgent.this,
                          getCurrentPosition());
   ...
   // schedule first action
   scheduleIn(lookSignal, dt);
}

protected void process(Signal s) {
   // react only on own special signal
   if (s instanceof LookSignal) {
      // check neighbourhood
      if (!isContent()) {
         move(); // move to a new position
      }
      // otherwise just stay
   }
   // schedule next look signal
   scheduleIn(lookSignal, dt);
}
```

Abbildung 5.19.: Gegenüberstellung der ereignisgesteuerten (links) mit der zeitgesteuerten Variante (rechts) des Agentenverhaltens im Segregationsmodell. Gezeigt werden die relevanten Ausschnitte aus den beiden Methoden `initialActions()` und `process()`, die bei Ableitung von `SimpleBehaviour` zu implementieren sind.

für das ereignisgesteuerte Modell[24]. Durchgeführt wurden pro Modell 10 Simulationsläufe mit dem Hexagonal-Grid von 20×20 Zellen, je 140 blauen und grünen Agenten mit der Standard-Toleranzgrenze 0.375, einer Bewegungsdauer von 1 und einer Simulationsdauer von 50 Simulationszeiteinheiten. Die Taktrate des zeitgesteuerten Modells wurde auf 2 Zeiteinheiten gesetzt. Da es sich bei den Rechenzeiten um geringe Werte von unter 20 Sekunden handelt, macht sich der Geschwindigkeitsvorteil der ereignisgesteuerten Variante allerdings kaum bemerkbar.

Im Zusammenhang mit der Zeitrepräsentation wurde außerdem überprüft, ob die Reihenfolge bei der Aktivierung der Agenten tatsächlich ohne Auswirkung auf das Modellergebnis bleibt, wie von Schelling angegeben (Schelling 1978, S. 148). Dazu wurden mit beiden Modellvarianten Simulationsläufe mit bzw. ohne zufälliger Abarbeitung zeitgleicher Ereignisse durchgeführt – mit dem bestätigenden Ergebnis, dass sich weder ein anderes Modellverhalten noch abweichende Rechenzeiten ergeben. Die zufällige Bearbeitung lässt sich wie in Abschnitt 5.1.2 beschrieben über die Methode `randomizeConcurrentEvents()` der Experiment-Klasse einstellen; für das Segregationsmodell wird sie im graphischen Experimentstarter über einen Experimentparameter gesteuert.

[24] Dabei wurde das ereignisgesteuerte Agentenverhalten vorher dahingehend optimiert, dass nur einmal pro Ereigniszeitpunkt reagiert wird. Dies ist nützlich, da ein Agent in FAMOS für *jede* Veränderung innerhalb seines Wahrnehmungsbereichs ein Signal erhält, so dass mehrere zeitgleiche Signale vorkommen können.

5.4. Testbeispiel 2: Sugarscape

Ein im Kontext der agentenbasierten Simulation sehr bekanntes Modell ist das sogenannte *Sugarscape*-Modell von Epstein und Axtell (1996), das eine künstliche Gesellschaft abbildet. Das Grundmodell besteht aus einer Menge von relativ einfachen Agenten und einer dynamischen räumlichen Umgebung, in der die Agenten sich auf der Suche nach erneuerbaren Ressourcen bewegen. Es bildet damit praktisch ein einfaches Ökosystem ab. In seinen zunehmend komplexeren Ausbaustufen dient das Modell zur Untersuchung von sozialwissenschaftlichen und ökonomischen Fragestellungen; im Vordergrund steht – analog zum Segregationsmodell – die Emergenz von Makrophänomenen wie sozialen Netzwerken, Handel und Märkten sowie der Evolution kultureller Unterschiede basierend auf der lokalen Interaktion individueller Akteure.

Dieses Modell ist mehr oder weniger umfassend in diversen Softwaresystemen für agentenbasierte Simulation nachimplementiert worden (z. B. Bousquet 2001; StarLogo 2001; Bigbee et al. 2007; Repast 2008; Li und Wilensky 2009) und kann damit fast als Referenzmodell gelten. In FAMOS wurde nur die einfachste Version der künstlichen Gesellschaft umgesetzt, da sich die komplexeren Varianten im Wesentlichen nur durch die zusätzlichen Regeln unterscheiden, die das Verhalten der Agenten spezifizieren. Die Unterstützung von Raummodell, Bewegung im Raum und Umweltdynamik lässt sich bereits am Grundmodell angemessen zeigen.

5.4.1. Beschreibung des Modells

Im Sugarscape-Grundmodell (Epstein und Axtell 1996, S. 21–32) wird eine Population von Agenten betrachtet, die in einer als *sugarscape* bezeichneten Landschaft leben. Diese Landschaft besteht aus einem zweidimensionalen Gitter von 50×50 quadratischen Zellen, die in unterschiedlicher Menge eine nachwachsende Ressource (Zucker) bereitstellen. Die Zucker-Verteilung bildet zwei diagonal versetzte „Hügel“, die in mehreren Stufen von 0 bis zum Maximalwert 4 ansteigen. Um Randeffekte auszuschließen, besitzt das Gitter sowohl in horizontaler als auch in vertikaler Richtung periodische Grenzen, so dass sich ein Torus ergibt.

Das Nachwachsen der Ressource wird durch folgende einfache Regel bestimmt: Zucker regeneriert sich an jeder Position mit einer Rate von α Einheiten pro Zeitintervall bis zur vordefinierten Kapazität der Position (*growback rule* G_α). Da diese Regel für jede Zelle des Gitters in gleicher Weise gilt, stellt die Sugarscape-Landschaft einen einfachen Zellularautomaten dar (Epstein und Axtell 1996, S. 19). Das Modell ist dementsprechend zeitgesteuert; wobei explizit eine zufällige Reihenfolge bei der Aktualisierung der Agenten gefordert wird (Epstein und Axtell 1996, S. 26).[25]

Die einzelnen Agenten besitzen prinzipiell dasselbe Verhalten, unterscheiden sich aber bezüglich ihrer „genetischen“ Ausstattung, welche die Attribute Stoffwechsel (gleichverteilt zwischen 1 und 4 Zuckereinheiten pro Zeitschritt) und Wahrnehmungsradius (gleichverteilt zwischen 1 und 6 Zellen) umfasst. Der wahrnehmbare Ausschnitt der Umgebung

[25] Eine ereignisgesteuerte Variante, die ohne zufällig bestimmte Aktualisierungsreihenfolge auskommt, ließe sich erreichen, indem die Zeitdauern der Aktivitäten (Regeneration und Bewegung) nicht konstant gewählt werden, sondern beispielsweise einer Exponentialverteilung folgen (Lawson und Park 2000).

ist eingeschränkt auf die vom Radius vorgegebene Anzahl Zellen in die vier Hauptrichtungen des Gitters; der Verzicht auf diagonale Wahrnehmung soll eine begrenzte Rationalität der Agenten modellieren (Epstein und Axtell 1996, S. 24).

Die Agenten ernähren sich von der Ressource Zucker und bewegen sich im Raum, um Zucker zu finden. Sobald ein Agent eine Zelle betritt, sammelt er die gesamte verfügbare Menge an Zucker ein und verbraucht eine von seinem Stoffwechsel diktierte Menge. Alles, was darüber hinaus geht, wird zum aktuellen Zuckervorrat des Agenten hinzugefügt. Hat ein Agent weniger Zucker zur Verfügung als er verbraucht, stirbt er und wird aus der Umgebung entfernt.

Die Bewegung der Agenten wird von einer Form der Gradientensuche bestimmt: Aus den sichtbaren Zellen wählt ein Agent die nächstgelegene freie Zelle mit dem größten Zuckerbestand aus und begibt sich direkt dorthin (*movement rule M*). Auch in diesem Modell können Agenten daher – wenn auch eingeschränkt – von Zelle zu Zelle springen, statt einen Weg vom Start zum Ziel über benachbarte Zellen suchen zu müssen. Zu jedem Zeitpunkt darf eine Zelle von höchstens einem Agenten belegt sein.

Modellparameter sind die Zucker-Regenerationsrate α und die Anzahl der Agenten; alle anderen Werte sind wie oben beschrieben festgelegt. Die Anfangspositionen der Agenten werden zufällig bestimmt, ebenso ihr anfänglicher Zuckervorrat (gleichverteilt zwischen 5 und 25, Epstein und Axtell 1996, S. 33). Der Zuckerbestand jeder Zelle wird mit dem jeweils maximalen Wert initialisiert. Vorgestellt werden zwei Experimente mit $\alpha = 1$ bzw. $\alpha = \infty$ und jeweils 400 Agenten.

5.4.2. Implementation in FAMOS

Die in FAMOS implementierte Version[26] des Sugarscape-Modells besteht aus einer Menge von einfachen proaktiven Agenten und einer räumlichen Umgebung mit eigener Dynamik. Der Raum wird durch ein zweidimensionales Gitter mit quadratischen Zellen (`RectangularGrid`), periodischen Randformen und von-Neumann-Nachbarschaft repräsentiert. Da die Agenten auch bei Sensorradien > 1 keinerlei diagonal gelegene Zellen wahrnehmen dürfen, muss diese Nachbarschaft bei der Berechnung des lokal wahrnehmbaren Bereichs restriktiv interpretiert werden (vgl. Abschnitt 5.2.1).

Die Ressource Zucker wird als räumlich verteiltes Attribut abgebildet, d. h. denjenigen Zellen des Gitters, die eine Zuckerkapazität > 0 aufweisen, wird während der Initialisierung des Modells ein entsprechendes Attribut (Name *sugar*, Wert zwischen 1 und 4) zugewiesen. Die Zuweisung geschieht über eine Ausbreitungsfunktion, mit der die Stufen der Zuckerlandschaft festgelegt werden, und zwei Attributverteiler, welche die Ausbreitungsfunktion anwenden, um die beiden „Hügel“ zu erzeugen. Sowohl Ausbreitungsfunktion (Klasse `famos.util.DiffusionFunction.Step`) als auch Attributverteiler (Klasse `famos.space.AttributeDiffusor`) werden von FAMOS zur Verfügung gestellt und konnten unverändert eingesetzt werden. Da Epstein und Axtell keine exakte Spezifikation ihrer Zuckerlandschaft liefern, mussten die benötigten Daten anhand von Abbildungen geschätzt werden (Epstein und Axtell 1996, S. 22) .

[26] Der vollständige Quelltext steht als Teil des gesamten FAMOS-Pakets in (Meyer 2013) zur Verfügung (Unterpaket `famos.examples.sugarscape`).

Das Nachwachsen der Ressource Zucker – definiert durch die Regel G_α – stellt die Umgebungsdynamik dar und lässt sich adäquat als einfacher Prozess abbilden; die Klasse `GrowBack` ist daher direkt von `desmoj.SimProcess` abgeleitet. Der Lebenszyklus dieses Umgebungsprozesses besteht darin, in einer Endlosschleife für Δt Zeiteinheiten zu warten (Methode `hold()`) und anschließend den Zuckerbestand entsprechend der Regenerationsrate α und der jeweiligen Zell-Kapazität wieder aufzufüllen. Da fast ein Viertel des Gitters keinen Zucker bereitstellt und die lokale Regel zur Aktualisierung des Zuckerbestands unabhängig von den Zuständen der benachbarten Zellen ist, ist es weder notwendig noch effizient, jede Zelle des Gitters mit einem eigenen `GrowBack`-Prozess auszustatten, wie es dem Konzept eines Zellularautomaten entsprechen würde. Statt dessen reicht ein Prozess für alle betroffenen Gitterzellen aus. Der Zugriff auf diese Gitterzellen wird in FAMOS erleichtert durch die Möglichkeit, vom Raummodell die Verteilung eines bestimmten Attributs in Form einer Liste aus Position-Attribut-Paaren anzufordern (Methode `getAttributeDistribution()` der Oberklasse `Space`).

Als Ausgangsklasse für die Agenten des Sugarscape-Modells wurde `MobileAgent` gewählt, da die Agenten sich im Raum bewegen können. Alle Agenten zeigen grundsätzlich dasselbe Verhalten; Unterschiede ergeben sich nur aufgrund ihrer heterogenen „genetischen“ Attribute, die sie dieselbe lokale Situation verschieden bewerten lassen. Interaktion zwischen Agenten findet im Grundmodell nicht statt. Die Agenten beeinflussen sich gegenseitig nur indirekt über die Umgebung, indem ihre jeweilige Position für andere Agenten gesperrt ist und ihr Zuckerverbrauch die Attributverteilung zeitweilig lokal verändert. Zur Verhaltensbeschreibung reicht wiederum der einfachste Verhaltensbaustein `SimpleBehaviour` aus – zum Vergleich wurde parallel auch eine prozessorientierte Variante basierend auf `ProcessBehaviour` implementiert (vgl. Abbildung 5.20).

In beiden Fällen reagieren die Agenten auf das `PositionReached`-Signal, das sie automatisch von der Umgebung bei Ankunft an einer neuen Position erhalten. Die Bewegung eines Agenten ist als `ConstantTimeMovement` realisiert in Verbindung mit einer Gradientensuche. Da die Agenten ihren gesamten Wahrnehmungsbereich in die Auswahl der nächsten Position einbeziehen statt wie in FAMOS vorgesehen nur die direkt benachbarten Raumelemente, wurde eine eigene Klasse `SugarSearch` von `GradientTrace` abgeleitet, welche die Einschubmethode `getPossibleNextPositions()` überschreibt. Die benötigte Kapazitätsbeschränkung von höchstens einem Agent pro Gitterzelle wird wie im Segregationsmodell über die Verwendung eines speziellen Markierungs-Attributs erreicht.

Die Zeitsteuerung des Originalmodells wird nachgebildet durch das unbedingte Warten (Methode `hold()`) im GrowBack-Prozess und die Aktivierung durch das `PositionReached`-Signal bei den Agenten. Die konstante Bewegungsdauer entspricht dem Zeitschritt Δt; auf diese Weise ist ein spezielles Zeitschritt-Signal für die Agenten überflüssig. Darüber hinaus kann auch auf die Aufmerksamkeitssignale (`Alert`) aus der Umgebung verzichtet werden, denn die Agenten müssen nicht direkt auf Veränderungen in der Umgebung reagieren. Sie entscheiden einfach in regelmäßigen Abständen, welche der aktuell sichtbaren freien Positionen zum Erreichen ihres Ziels (Überleben) am besten geeignet ist. In diesem Sinne können sie als proaktiv bezeichnet werden.

Abbildung 5.21 zeigt Ausschnitte eines Simulationslaufs mit dem Sugarscape-Modell. Die Modellparameter Regenerationsrate α und Anzahl Agenten wurden auf die von Ep-

```
// ereignisorientierte Verhaltensbeschreibung

protected void initialActions() {
   // enter the environment
   getEnvironment().enter(agent,
                          getCurrentPosition());
   ...
   move();
}

protected void process(Signal s) {
   // react only on PositionReached signals
   if (s instanceof PositionReached) {
      ...
      collectSugar();
      metabolize();
      if (sugar > 0.0)
         move();
      else
         ((LifeDeathModel)getModel()).die(agent);
   }
}
```

```
// prozeßorientierte Verhaltensbeschreibung

protected void lifeCycle() {
   // enter the environment
   getEnvironment().enter(agent,
                          getCurrentPosition());
   ...

   // try to stay alive
   while (sugar > 0.0) {
      move();
      waitForSignal(PositionReached.class);
      clearSignals();
      ...
      collectSugar();
      metabolize();
   }

   // starved to death
   ((LifeDeathModel)getModel()).die(agent);
}
```

Abbildung 5.20.: Gegenüberstellung der ereignisorientierten (links) mit der prozessorientierten Variante (rechts) des Agentenverhaltens im Sugarscape-Modell. Gezeigt werden die relevanten Ausschnitte aus den Methoden `initialActions()` und `process()` für die Unterklasse von `SimpleBehaviour` bzw. aus der Methode `lifeCycle()` für die Unterklasse von `ProcessBehaviour`.

stein und Axtell vorgeschlagenen Werte 1.0 bzw. 400, die Taktrate Δt auf 1.0 gesetzt. Simuliert wurden 500 Zeitschritte.

5.4.3. Diskussion

Das Sugarscape-Modell ist bereits in seiner Grundversion komplexer als das im vorigen Abschnitt diskutierte Segregationsmodell. Dies betrifft vor allem die räumliche Umgebung mit Eigendynamik, deren Umsetzung in FAMOS besonders unterstützt wird. Zwar stellt FAMOS keine speziellen Klassen für Zellularautomaten zur Verfügung, doch lassen sich diese einfach abbilden durch die Kombination eines Gitters mit entsprechenden Umgebungsprozessen für die lokalen Regeln. Gleichzeitig bietet diese Vorgehensweise einen Gewinn an Flexibilität: So ist es dem Modellierer freigestellt, welche Zellen mit welchen Regeln verknüpft werden. Im Sugarscape-Modell ist beispielsweise die Realisierung der lokalen Regel als ein einziger Umgebungsprozess angemessen – und darüber hinaus auch effizienter –, da die Zucker-Regeneration im Unterschied zu üblichen Regeln in Zellularautomaten nicht die Zustände benachbarter Zellen einbezieht und alle Zellen mit Kapazität 0 diese Regel nie anwenden. Alternativ wäre in FAMOS auch eine ereignisorientierte Umsetzung möglich.

Für die statische Repräsentation des Raums wird nur ein zweidimensionales Gitter mit quadratischen Zellen benötigt, was aufgrund seiner einfachen Umsetzbarkeit immer noch das Standard-Raummodell in der agentenbasierten Simulation darstellt. Die Stärken des graphbasierten Raummodells von FAMOS kommen deshalb hier weniger zum Tragen. Jedoch stellt das Sugarscape-Modell einige spezielle Anforderungen, die zumindest die um-

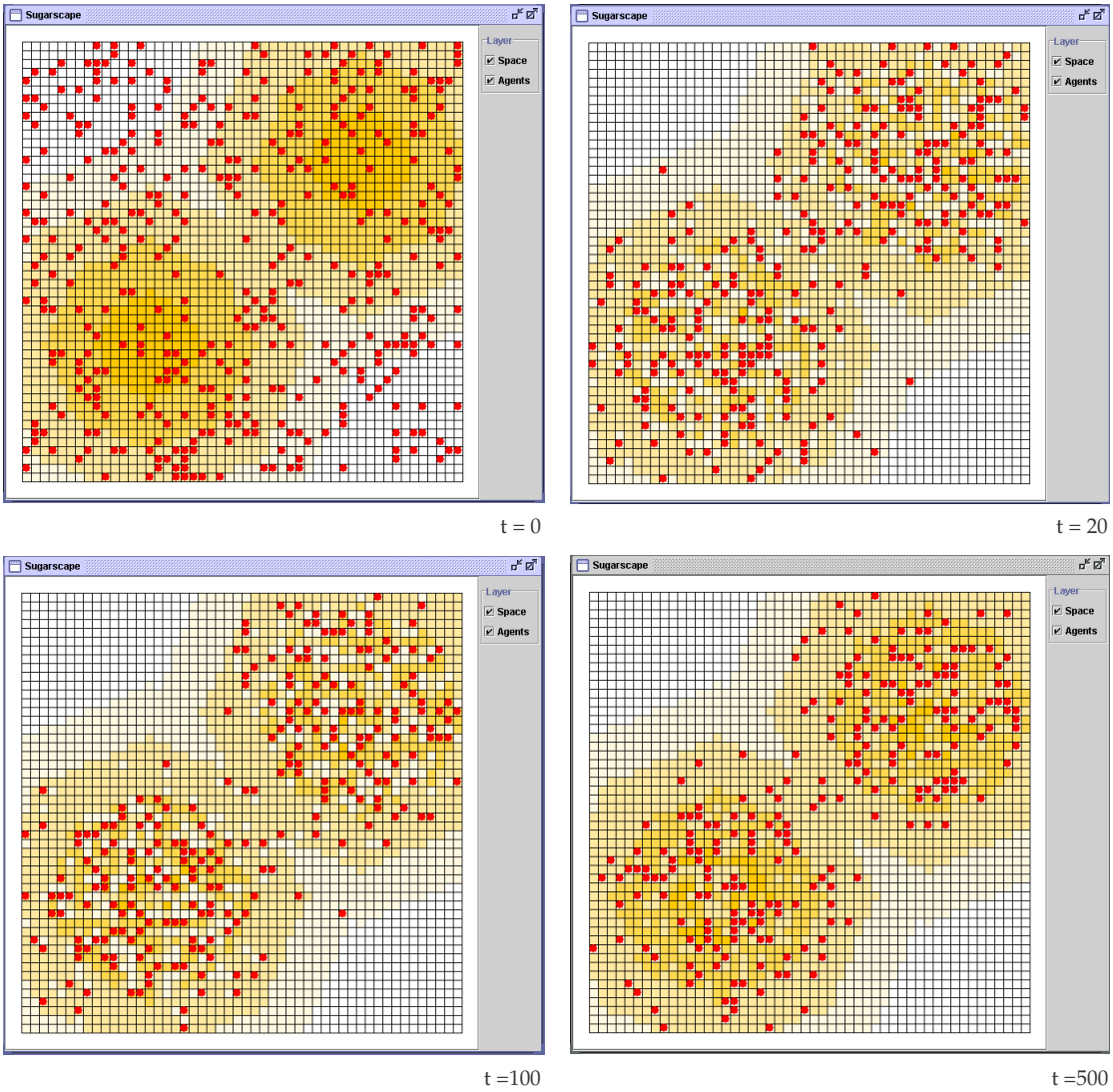

Abbildung 5.21.: Screenshots des Sugarscape-Modells aus einem Simulationslauf mit 400 Agenten, Zucker-Regenerationsrate $\alpha = 1.0$ und Zeitschritt $\Delta t = 1.0$. Der Anfangszustand (oben links) ist gekennzeichnet durch eine zufällige Verteilung der Agenten und maximale Zuckerwerte in der räumlichen Umgebung.

fangreiche Funktionalität der Klasse `RectangularGrid` gut illustrieren.

Eine Besonderheit dieser Klasse ist die Möglichkeit, beliebige Nachbarschaftsrelationen zu spezifizieren und diese auch nach Instantiierung eines Objekts zu verändern. Diese Fähigkeit hat sich als nützlich erwiesen bei der anfänglichen Verteilung des Zucker-Attributs im Raum. Da die hierfür gut geeigneten Attributverteiler (`AttributeDiffusor`) eine Moore-Nachbarschaft benötigen, um die aus kreisförmigen Terrassen aufgebauten Zuckerhügel zu generieren, die Wahrnehmung der Agenten jedoch auf eine Von-Neumann-Nachbarschaft beschränkt ist, erzeugt das Modell zunächst ein Gitter mit Moore-Nachbarschaft. Nach der Initialisierung der Zuckerverteilung wird diese Nachbarschaftsrelation dann zu einer von-Neumann-Nachbarschaft abgeändert. Gleiches gilt für die Randform des Gitters: Laut Modellspezifikation setzt die Attributverteilung feste Grenzen voraus, während die Agenten sich auf einem Torus bewegen, also quasi über die Gittergrenzen hinaus. Dies stellt kein Problem dar, denn in FAMOS können die Randformen aller zweidimensionaler Gitter mit regelmäßigen Zellen (Oberklasse `RegularGrid2D`) für jede Dimension ebenfalls nachträglich verändert werden (Methode `setWrap()`).

Das Verhaltensrepertoire der Agenten ist zumindest im Basis-Modell so eingeschränkt, dass es sich wiederum mit dem einfachsten Baustein `SimpleBehaviour` zufriedenstellend beschreiben lässt. Da die Agenten jedoch eher proaktiv als reaktiv handeln, ist in diesem Fall die Verwendung der prozessorientierten Verhaltensbeschreibung (`Process-Behaviour`) angemessen. Diese ist übersichtlicher – es wird nur eine Methode anstatt zwei in der ereignisorientierten Variante benötigt – und kompakter, wie aus der Gegenüberstellung in Abbildung 5.20 auf Seite 177 erkennbar ist. Der einzige Nachteil besteht in dem etwas erhöhten Rechenzeitbedarf, der analog zu den Simulationsprozessen in DESMO-J aus dem Einsatz von Threads resultiert. Ein direkter Vergleich der beiden Verhaltensvarianten ergab im Mittel eine um 18% höhere Rechenzeit, wenn die Agenten des Modells mit `ProcessBehaviour`-Bausteinen versehen wurden. Durchgeführt wurden jeweils 10 Simulationsläufe mit 400 Agenten, der Regenerationsrate $\alpha = 1.0$, einer Taktrate von 1.0 und einer Simulationsdauer von 500 Zeitschritten. Da sich die absoluten Zahlen im Bereich von 3–4 Minuten bewegen, ist eine Erhöhung um 18% praktisch vernachlässigbar; v. a. verglichen mit der für DESMO-J berichteten Verdoppelung der Rechenzeit bei Verwendung der prozessorientierten Weltsicht (Page et al. 2000, S. 162).

Die von den Agenten als Bewegungsstrategie eingesetzte Gradientensuche wird von FAMOS bereits zur Verfügung gestellt. Sie kann jedoch nicht direkt übernommen werden, da die im Sugarscape-Modell vorgesehene Möglichkeit, in einem Bewegungsvorgang mehrere Raumelemente zu überspringen, nicht mit der von FAMOS propagierten realistischeren Bewegung zwischen benachbarten Raumelementen konform ist. Andererseits ist das Framework hier explizit erweiterbar angelegt: Durch Überschreiben der Einschub-Methode `getPossibleNextPositions()` ist eine Anpassung der Funktionalität für die Strategien `RandomWalk` und `GradientTrace` leicht zu realisieren.

Die vom Sugarscape-Modell verwendete Zeitsteuerung lässt sich im ereignisgesteuerten FAMOS problemlos nachbilden, ohne dass ein besonderes Aktivierungssignal benötigt wird, das in Δt-Zeitabständen an alle beteiligten Akteure gesendet wird. Der für die Zucker-Regeneration verantwortliche Umgebungsprozess erreicht über das aus der prozessorientierten Weltsicht stammende unbedingte Warten für Δt-Zeiteinheiten (Methode `hold()`), dass er von der Simulationsinfrastruktur nach Ablauf dieser Zeitspanne auto-

matisch reaktiviert wird. Für die Agenten kann dagegen das – in FAMOS ebenfalls automatisch generierte – `PositionReached`-Signal die Rolle des regelmäßigen Reaktivierungssignals übernehmen, indem die konstante Dauer einer Bewegung mit der Taktrate gleichgesetzt wird. Durch die Erweiterung von DESMO-J um eine Ereignisliste mit zufälliger Abarbeitung zeitgleicher Ereignisse (vgl. Abschnitt 5.1.2) steht eine zufällige Reihenfolge bei der Aktualisierung der Agenten zur Verfügung, wie sie von (Epstein und Axtell 1996, S. 26) explizit gefordert wird.

Zu beachten ist hier jedoch, dass die Aktualisierung der Umgebung (genauer: die Reaktivierung des Umgebungsprozesses) stets vor oder nach allen Agenten-Aktualisierungen stattfindet. Dies wird von Epstein und Axtell implizit vorausgesetzt; aus der Spezifikation der in einem Simulationslauf auszuführenden Regeln als geordnetes Paar (E, A) mit E als Menge der Umgebungsregeln und A als Menge der Agentenregeln (Epstein und Axtell 1996, S. 26) kann geschlossen werden, dass in jedem Zeitschritt zuerst die Regeln der Umgebung und anschließend – für jeden Agenten – die Regeln der Agenten ausgeführt werden. Eine detaillierte Beschreibung des für das Sugarscape-Modell implementierten Simulators wird von Epstein und Axtell nicht geleistet[27].

Um den Umgebungsprozess von der zufälligen Reihenfolge bei der Abarbeitung zeitgleicher Ereignisse auszunehmen, wird er in der FAMOS-Version des Sugarscape-Modells zeitversetzt zu den Agenten gestartet: Der Umgebungsprozess beginnt seinen Lebenszyklus zum Startzeitpunkt der Simulation $t_0 = 0.0$, während die Agenten zum Zeitpunkt $t = \Delta t/2$ starten. Da die Taktrate Δt für alle Modellentitäten identisch ist, bleibt auf diese Weise gewährleistet, dass der Umgebungsprozess immer vor den Agenten reaktiviert wird.

[27] In einem Anhang (Epstein und Axtell 1996, S. 179–181) geben die Autoren einen Überblick über die objektorientierte Implementation des Modells, allerdings ohne näher auf die simulationstechnischen Aspekte einzugehen.

6. Anwendung: Simulationsstudie zu Stadtkurierdiensten

Designing a multi-agent system consists primarily in analysing and solving a very large number of problems.
– Jacques Ferber

Im Rahmen eines Forschungsprojekts am Fachbereich Informatik der Universität Hamburg konnte das in dieser Arbeit entwickelte Framework für Agentenorientierte Modellierung und Simulation (FAMOS) in einer praxisrelevanten Anwendung eingesetzt und evaluiert werden. Das Projekt „Nachhaltige Logistikkonzepte für Stadtkurierdienste“ wurde vom 1.04.2000 bis 30.09.2003 von der Behörde für Wissenschaft und Forschung der Freien und Hansestadt Hamburg aus dem Forschungsfond „Ökologie und Nachhaltige Entwicklung“ gefördert. Gegenstand des Projekts war es, die betriebswirtschaftlichen, ökologischen und sozialen Auswirkungen möglicher neuer Organisationsformen für Stadtkurierdienste im voraus abzuschätzen. Hierzu bietet sich die Simulation auf Basis eines agentenbasierten Modells an, da die Arbeitsabläufe innerhalb eines Kurierdienstes durch geringe zentrale Kontrolle sowie hohe Autonomie und Eigenverantwortlichkeit der beteiligten Akteure geprägt sind. Zudem ist es notwendig, die Bewegung der Kuriere auf dem städtischen Verkehrsnetz explizit im Modell abzubilden, da diese das Verhalten der Kuriere entscheidend beeinflusst.

Im folgenden soll zunächst in das Umfeld der Simulationsstudie eingeführt werden (Abschnitt 6.1), bevor in Abschnitt 6.2 das Modell des *Status quo* eines typischen Kurierdienstes detailliert vorgestellt wird. Dieses auch als *Basismodell* bezeichnete Modell diente als Referenz für den Vergleich mit den alternativen Logistikstrategien *Hub and Shuttle* (Abschnitt 6.3) bzw. *Innen/Außen* (Abschnitt 6.4). Um die Beschreibung der Simulationsstudie zu vervollständigen, wird in Abschnitt 6.5 kurz auf die erzielten Ergebnisse eingegangen, auch wenn diese für die Evaluation des Frameworks FAMOS (Abschnitt 6.6) nicht relevant sind. Eine ausführliche Diskussion der Ergebnisse findet sich im Abschlussbericht des Projekts (Knaak et al. 2004).

Teile dieses Kapitels sind in z.T. leicht modifizierter Form bereits an anderer Stelle veröffentlicht worden (Deecke et al. 2000; Knaak et al. 2002a, b, 2004). Dies betrifft die Einführung in den Gegenstandsbereich, die Beschreibung der Modellvarianten, sowie die Darstellung der Ergebnisse. In diesen Publikationen lag der Schwerpunkt auf Problemstellung und Durchführung der Simulationsstudie, während hier Einsatz und Bewertung des Modellierungswerkzeugs FAMOS im Vordergrund stehen.

6.1. Gegenstand der Simulationsstudie

6.1.1. Problemfeld

Stadtkurierdienste sind in vielen Großstädten ein wichtiger Bestandteil des Güter- und Wirtschaftsverkehrs. Beispielsweise arbeiteten Ende der neunziger Jahre in Hamburg über 100 Kurier-, Express- und Paketdienste mit insgesamt ca. 3500 Kurieren. Deren tägliche Kilometerleistung weist allein im Hamburger Stadtgebiet eine sechsstellige Größenordnung auf.

Kurierdienste zeichnen sich dadurch aus, dass jede Sendung durchgehend von einer einzelnen Person, dem Kurier, begleitet wird. Die Folge ist ein hoher Verkehrsaufwand pro Sendung, der organisatorisch prinzipiell durch Sendungsbündelung reduziert werden könnte. Mit der Einführung entsprechender Logistikkonzepte würde sich die organisatorische Struktur der Kurierdienste derjenigen der Expressdienste annähern. Eine solche Optimierung der Tourenplanung wäre unter Umweltgesichtspunkten grundsätzlich zu befürworten, weil dadurch Verkehr reduziert würde. Jedoch sind bei der Reorganisation von Stadtkurierdiensten einige Besonderheiten zu beachten:

- Neben motorisierten Kurieren gibt es Fahrradkuriere, die keine Schadstoffemissionen und praktisch keinen Energieverbrauch verursachen. Sie beanspruchen zudem weniger Straßenfläche zum Fahren und Parken und kommen näher an die Abhol- und Lieferpunkte heran. Aus ökologischer Sicht wäre ein vermehrter Einsatz von Fahrradkurieren zu befürworten. Logistikkonzepte, die auf Sendungsbündelung beruhen, könnten jedoch den PKW-Anteil tendenziell erhöhen, da beim Fahrrad das Bündelungspotential aufgrund von Gewicht und Volumen der Sendungen begrenzt ist.

- Kuriere sind in der Regel selbstständige Unternehmer, die eine Monatspauschale an ihre Zentrale für die Vermittlung von Aufträgen entrichten. Die Vermittlung und Vergabe von Aufträgen ist daher ein sensibler Vorgang, der jahrelang bewährten Regeln folgt, an die sich die Zentrale und alle Kuriere halten. Die Regeln sorgen dafür, dass kein Kurier bevorzugt oder benachteiligt wird, und stellen gleichzeitig die Servicequalität für den Kunden sicher. Jedes alternative Logistikkonzept muss daher ebenfalls eine gerechte Auftragsverteilung garantieren.

- Eine Besonderheit der Kurierdienstleistung ist die Tatsache, dass die Sendung auf dem schnellsten Wege befördert wird. Das bedeutet in der Praxis, dass der Kurier, der den Auftrag übernommen hat, z.B. innerhalb von 30 Minuten beim Versender eintrifft und von dort ohne wesentliche Umwege zum Empfänger fährt. Sendungsbündelung, die ja normalerweise mit Zeitverzögerungen einhergeht, darf diese Besonderheit der Kurierdienstleistung nicht wesentlich gefährden.

- Die Servicequalität besteht für den Kunden nicht immer nur in der Schnelligkeit oder Pünktlichkeit, sondern in einigen Fällen auch im eigentlichen „Kurierprinzip“, also der persönlichen Begleitung der Sendung vom Abhol- zum Lieferpunkt. Hier spielen Zuverlässigkeitsfragen eine Rolle. Es kommt sogar vor, dass Kunden einen

bestimmten Kurier wünschen, etwa weil sich der Kurier auf dem Gelände des Absenders oder Empfängers schon auskennt oder weil sie eine besonders wichtige Sendung nur einer bewährten Person anvertrauen wollen. Alternative Logistikkonzepte dürfen diese Art der Servicequalität nicht ausschließen.

Seit einigen Jahren deuten sich Sättigungstendenzen der Stadtkuriermärkte an, die einen Druck in Richtung Rationalisierung und Erschließung weiterer Marktsegmente ausüben. Die Situation wird dadurch verschärft, dass ein Kernbereich des Stadtkuriergeschäfts in Hamburg, Beförderungsaufträge aus und für Medienbetriebe, durch die weiter fortschreitende Digitalisierung von Medienprodukten und die Verbesserung elektronischer Übertragungsmedien in größeren Teilen wegfallen wird.

Die schlechter werdende Verkehrssituation in Städten stellt eine weitere Restriktion für Stadtkurierdienste dar. Längere Fahrzeiten durch höhere Verkehrsdichten und längere Suchzeiten und -fahrten für das Abstellen der Fahrzeuge bei Abhol- und Zustellvorgängen beeinträchtigen die operative Funktionalität und betriebswirtschaftliche Effizienz. Die Kurierfahrer sind davon besonders betroffen, da sie in der Regel neben einer Anfahrtpauschale ausschließlich für die zurückgelegten Distanzen bezahlt werden und nicht für den benötigten Zeitaufwand.

Obwohl bei weitem noch nicht von einer Krise der Stadtkuriere gesprochen werden kann, deuten sich doch Probleme, Grenzen und Umwandlungsprozesse auf den städtischen Transportmärkten an, die mittelfristig die bisher erfolgreichen Kurierdienste in ihrer Ertragskraft oder sogar ihrer Existenz bedrohen können. Eine der Hauptursachen dafür liegt in der bisherigen Organisationsform mit ihren vergleichsweise hohen Kostenstrukturen, welche die Stadtkurierdienste auf bestimmte Marktsegmente des städtischen Eiltransports beschränkt. Ein weiterer Grund ist die mangelnde wirtschaftliche Innovationsfähigkeit der klein- bis mittelständischen Kurierdienstunternehmen. Die geringe Kapitalkraft beschränkt die meisten Stadtkurierdienste auf Verbesserungen, die entweder keine zusätzlichen Kosten verursachen, oder bei denen höhere Kosten sich schnell durch höhere Erträge amortisieren.

6.1.2. Ziele der Simulationsstudie

Ziel der hier vorgestellten Studie ist es, mögliche neue Organisationsformen für Stadtkurierdienste auf ihre ökologische Effizienz bei gleichzeitiger Wettbewerbsfähigkeit und sozialer Verträglichkeit hin zu evaluieren. Als Methode wird die agentenbasierte Simulation gewählt. Sie erscheint für die Modellierung von Stadtkurierdiensten besonders angemessen, da deren Arbeitsabläufe durch geringe zentrale Kontrolle und hohe Autonomie und Eigenverantwortung der beteiligten Akteure geprägt sind.

Der Gedanke liegt nahe, Kurierdienste nach dem Vorbild der Expressdienste zu organisieren, also das Kurierprinzip zugunsten einer auf Umschlagpunkten basierenden Logistik aufzugeben und eine zentrale Routenplanung vorzusehen. Aufgrund der im vorangehenden Abschnitt beschriebenen Besonderheiten der Kurierdienste ist jedoch zu vermuten, dass eine Reorganisation, welche die spezielle Qualität der Dienstleistung und auch der Funktionsweise eines Kurierdienstes nicht beachtet, entscheidende Vorteile der heutigen Organisationsform aufheben würde.

Bestehende Kurierdienste sind erstaunlich gut in der Lage, den latenten Zielkonflikt zwischen den Zielen

- Minimierung des Verkehrsaufwandes für eine gegebene Auftragsmenge,
- gerechte Auftragsverteilung innerhalb der Kurierflotte und
- Sicherstellung der Servicequalität

auszutarieren, wie eine Systemanalyse verschiedener Hamburger Kurierdienstunternehmen gezeigt hat.[1] Die Kuriere optimieren ihre individuellen Routen recht gut, indem sie mehrere (im Normalfall bis zu fünf) Aufträge überlappend oder verschachtelt abarbeiten und bevorzugt Aufträge annehmen, die sich räumlich und zeitlich gut in ihren aktuellen Plan einfügen. Das Verfahren zur Auftragsvergabe hält die Auslastung der Kuriere, die Effizienz in der Routenplanung und die erreichte Servicequalität in einem von allen Beteiligten akzeptierten Gleichgewicht.[2]

Ausgangspunkt der Simualationsstudie ist daher ein Modell der bei den betrachteten Stadtkurierdiensten vorherrschenden Organisationsform, der Auftragsvermittlung (siehe Abschnitt 6.2). Dieses Basismodell kann anhand vorliegender empirischer Daten validiert werden und dient als Bezugspunkt für den Vergleich mit alternativen Strategien. Zwei Logistikstrategien werden näher betrachtet und in Modellvarianten implementiert. Hierbei handelt es sich um Konzepte zur zweistufig gestalteten Beförderung von Kuriersendungen (Hubsystem), die teilweise den Organisationsformen von Paket- und Expressdiensten (UPS, FedEx, DHL) oder flächendeckend arbeitenden Speditionen bzw. Speditionskooperationen entlehnt sind (*Hub and Shuttle*, siehe Abschnitt 6.3, sowie *Innen-Außen*, siehe Abschnitt 6.4).

Für alle Modellvarianten werden mehrere Simulationsläufe mit unterschiedlichen Eingabedaten (Szenarien) durchgeführt. Jedes Szenario wird an Zielgrößen gemessen, die durch Auswertungsvorschriften für die Simulationsergebnisse operationalisiert werden. Anhand der Zielgrößen lässt sich feststellen, ob und in welcher Hinsicht ein alternatives Szenario gegenüber dem Status quo Vorteile hat oder ob es die Situation verschlechtert. Im Rahmen der Modellvalidität kann aus diesen Ergebnissen auf das Realsystem zurückgeschlossen werden. Die Zielgrößen, an denen die Szenarien gemessen werden, sind:

Ökologische Qualität Hier steht der Verkehrsaufwand im Vordergrund, der Umweltbelastungen durch Energieverbrauch, Emissionen oder dynamischen Flächenbedarf verursacht. Diese Größe wird durch die mittlere motorisiert zurückgelegte Strecke (motorisierte Fahrleistung) pro Auftrag operationalisiert.

[1] Das Forschungsprojekt „Nachhaltige Logistikkonzepte für Stadtkurierdienste“ konnte auf eine im Rahmen einer Fallstudie des MOBILE-Projektes durchgeführte Systemanalyse zurückgreifen (Hilty et al. 1998, S. 291ff) und diese durch zusätzliche Expertengespräche mit Unternehmensvertretern aktualisieren und erweitern (Deecke et al. 2000).

[2] Möglicherweise trägt das bestehende Verfahren auch zur Arbeitsmotivation der Kuriere bei. Die individuelle Tourenplanung in Verbindung mit der Transparenz der Auftragslage für alle Beteiligten (Verlesen der Aufträge über Funk) verschafft vermutlich mehr Befriedigung als das „Abfahren“ einer von der Zentrale geplanten und zugewiesenen Tour, insbesondere vor dem Hintergrund der Tatsache, dass sich eine geschickte Planung aufgrund der Selbständigkeit der Kuriere unmittelbar auf ihr Einkommen auswirkt.

Ökonomische Qualität (Servicequalität) Der Kunde bewertet die Leistung eines Kuriers danach, wie zuverlässig seine Aufträge ausgeführt werden. Wichtig ist hierbei insbesondere, dass zugesagte Termine eingehalten werden. Es sind also neben der Zeit von der Annahme des Auftrags bis zur Übergabe der Sendung an den Empfänger auch Anzahl und Summe der Verspätungen zu minimieren.

Soziale Qualität Dieses Kriterium bezieht sich auf die Organisation (das soziale System) der selbstständigen Kuriere. Wichtig ist hier das Einkommen der Kurierfahrer, aber auch die Einkommensverteilung. Wenn hohe Einkommensunterschiede auftreten, ist der Konsens über das Verteilungsverfahren gefährdet. Operationalisiert wird diese Größe daher durch Mittelwert und Standardabweichung des Gewinns eines Kuriers pro zurückgelegter Strecke.

6.1.3. Datenerhebung

Grundlage der Modellierung bilden Auftrags- und Flottenprofile verschiedener Hamburger Kurierdienstunternehmen. In der ersten Projektphase wurden Unternehmensgespräche mit insgesamt fünf Stadtkurierdiensten geführt, die aufgrund ihrer Größe bzw. ihres Auftragsvolumens für alternative Logistikstrategien geeignet sind. Ein konzeptuelles Modell der idealisierten Arbeitsabläufe in Kurierdiensten lag bereits aus einer Fallstudie des Projekts MOBILE (Hilty et al. 1998) vor und wurde im Rahmen der Gespräche ergänzt und aktualisiert.

Zwei der befragten fünf Stadtkurierunternehmen haben Daten in Form von Auszügen der von ihnen verwendeten Datenbanksysteme bereitgestellt. Sie werden im folgenden aus Gründen der Anonymisierung als „Kurierdienst A“ und „Kurierdienst B“ bezeichnet. Kurierdienst A stellte Auftragsdaten der Kalenderwoche 37 aus dem Jahr 2001 zur Verfügung, Kurierdienst B aus demselbem Zeitraum auftrags- und kurierbezogene Daten von zwei Tagen. Die unterschiedliche Größe der beiden Unternehmen spiegelt sich im Umfang des täglichen Auftragsaufkommens wider: Kurierdienst B bearbeitet etwa 300 Aufträge pro Tag, A dagegen knapp 2000 Aufträge.

Für die Modellierung der Aufträge werden neben Angaben zu den Zeitpunkten der Auftragsannahme und -vermittlung, der Art der Sendung sowie dem gewünschten Transportmittel hauptsächlich der jeweilige Abhol- und Bestimmungsort in Form von Geokoordinaten benötigt. Diese sind allerdings nicht vorhanden; die Ortsangaben liegen ausschließlich in Form von Adressen (Straße und ggf. Hausnummer) vor. Die Kundendaten sind in den Softwaresystemen hausnummerngenau hinterlegt. Der größte Teil der Lieferadressen wird aber während der telefonischen Auftragsannahme weiterhin manuell erfasst, so dass hier eine Aufbereitung der Straßennamen nötig war (u.a. Bereinigung von Tippfehlern, Ergänzung von abgekürzten Bezeichnungen).

Die durch Straßennamen und Hausnummern spezifizierten Auftragsorte wurden anschließend mit Rechnerunterstützung geocodiert, d.h. auf die nächstgelegenen Knoten des verwendeten Verkehrsnetzes abgebildet. Aufträge außerhalb des abgedeckten Bereichs (z.B. aus dem Hamburger Umland) blieben unberücksichtigt. Zuzüglich weiterer aufgrund fehlender Angaben uncodierbarer Aufträge war die Vorverarbeitung mit einem „Schwund“ von im Mittel 15 % des ursprünglichen Aufkommens verbunden.

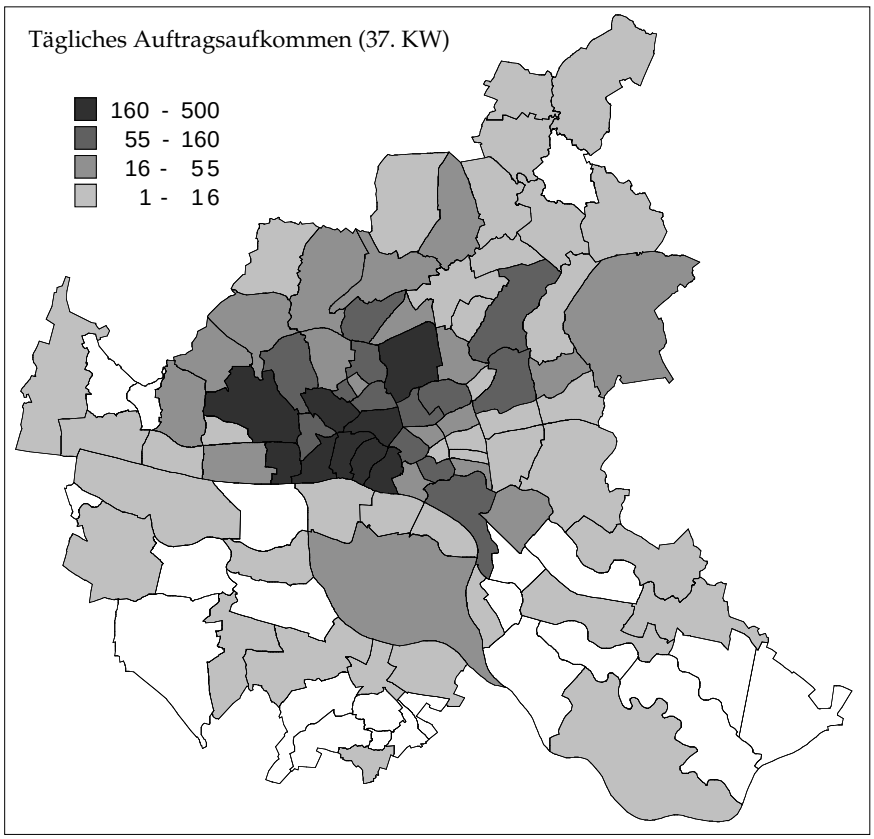

Abbildung 6.1.: Die Verteilung des Auftragsaufkommens auf die einzelnen Stadtteile.

Einen Eindruck von der räumlichen Verteilung der Aufträge gibt Abbildung 6.1. Die Auftragsdaten beider Kurierdienste wurden hier mit Hilfe eines GIS für einen beispielhaften Tag auf die Stadtteile Hamburgs bezogen. Jeder Auftrag ist zweimal vertreten, einmal durch seinen Abhol- und einmal durch seinen Lieferort. Es ist deutlich zu erkennen, dass der Schwerpunkt des Auftragsaufkommens in der Innenstadt liegt, während die Randbereiche Hamburgs wenig beitragen. Knapp 60 % aller Aufträge beginnen oder enden in den neun aufkommensstärksten Stadtteilen Neustadt, Hamburg-Altstadt, St. Pauli, Rotherbaum, Bahrenfeld, Ottensen, Winterhude und Altona-Altstadt.

Da sich für die anderen Tage die gleiche Situation ergibt, liegt der Schluss nahe, dass alternative Logistikkonzepte besonders erfolgversprechend sind, die diese räumliche Struktur ausnutzen.

Für die Modellierung der Kuriere werden Angaben zu Verkehrsmittel, Arbeitsbeginn und -ende sowie Standort zu Arbeitsbeginn (Heimatort) benötigt. Da entsprechende Datensätze nur von Kurierdienst B vorliegen, wurden die restlichen aus den Auftragsdaten ermittelt. Dabei wurde der Heimatort jedes Kuriers vereinfachend mit dem Abholort seines ersten Auftrags des Tages identifiziert. Der Arbeitsbeginn wurde fünf Minuten vor

Annahme des ersten, das (beabsichtigte) Arbeitsende 30 Minuten nach Annahme des letzten Auftrags gelegt. Da Kuriere auch Aufträge ausführen, deren Verkehrsmittelwunsch nicht mit ihrem Fahrzeug übereinstimmt, wurde dieses folgendermaßen ermittelt: Unter der vereinfachenden Annahme, dass Kuriere nur Aufträge für ein identisches oder „kleineres“ Fahrzeug ausführen (z.B. kann ein Autofahrer auch Fahrradaufträge bewältigen, jedoch nicht umgekehrt), wurde jedem Kurier der „größte“ Verkehrsmittelwunsch aller bearbeiteten Aufträge (z.B. ein Transporter) zugeordnet. Die Ermittlung der Kurierdaten ergab für den Kurierdienst A eine tägliche Flottengröße von ca. 160 Fahrern, darunter 20 % Fahrradkuriere, während beim Kurierdienst B lediglich ca. 50 Autokuriere beschäftigt sind.

Neben den auftrags- und kurierbezogenen Daten steht für die Simulationen folgendes empirische Datenmaterial zur Verfügung:

- Die Preisstruktur zur Berechnung der Auftragskosten, aus denen sich die Einnahmen der Kuriere ergeben. Pro Auftrag werden eine Anfahrtpauschale sowie eine Mindeststrecke von 3 km abgerechnet; längere Touren werden zusätzlich streckenbezogen vergütet.

- Das in einem geographischen Informationssystem (GIS) erstellte Verkehrsnetz der Stadt Hamburg in Form eines gerichteten Graphen. Kanten des Graphen sind Straßenabschnitte, während Knoten Kreuzungspunkte darstellen. Zusätzliche Kanteninformationen sind der Straßentyp (z.B. Autobahn) und die Durchschnittsgeschwindigkeit. Das Verkehrsnetz liegt in zwei Detaillierungsstufen vor, wobei das gröbere Modell nur Hauptstraßen beinhaltet und aufgrund des geringeren Rechenaufwands zur Modellverifikation dient. Das detaillierte Modell umfasst etwas mehr als 17.000 Knoten und 48.000 gerichtete Kanten, was bei der Simulation zu realistischeren Ergebnissen und deutlich höheren Rechenzeiten führt.

6.2. Das Basismodell: Ein Stadtkurierdienst mit Auftragsvermittlung

Das erste Projektziel bestand in der Entwicklung eines Modells, welches den Ist-Zustand typischer Hamburger Stadtkurierdienste widerspiegelt und als Grundlage bzw. Referenz für die Evaluation alternativer Logistikkonzepte dienen soll.

Die meisten Stadtkurierdienste sind als Vermittlungsagenturen ausgelegt. Die Fahrer arbeiten in der Regel als selbstständige Unternehmer. Nur bei wenigen, oftmals kleinen Kurierdienstunternehmen kommt es vor, dass Fahrer bei den Kurierdiensten angestellt sind. Die Vermittlungszentralen übernehmen gegen Entgelt das Marketing, die Auftragsannahme und -vermittlung sowie die kaufmännische Auftragsabwicklung, einschließlich des Inkasso für die Kurierfahrer. Der rechtliche Status der Fahrer als selbstständige Unternehmer hat zum Teil erhebliche Konsequenzen für die operativen Abläufe der Vermittlungszentralen. Um Scheinselbstständigkeit zu vermeiden, muss der Vermittlungsprozess z.B. die Entscheidungsfreiheit der Unternehmer prinzipiell gewährleisten, auch wenn dies die Servicequalität des Kurierdienstes als Ganzes negativ beeinflussen könnte. Weiterhin ergibt sich eine wettbewerbsrechtliche Problematik. Die Kurierfahrer bieten Leistungen am Markt zu abgesprochenen Preisen und Transportbedingungen an. Rechtlich bilden sie

damit ein Kartell. Mittlerweile sind Fahrervereinigungen als Mittelstandskartelle legalisiert, jedoch gibt es nicht bei allen Stadtkurierdiensten derartige Fahrervereinigungen.

Im folgenden wird von diesen juristischen und ökonomischen Randbedingungen abstrahiert und die Abläufe innerhalb eines typischen Stadtkurierdienstes mit Vermittlung von Aufträgen betrachtet.

6.2.1. Beschreibung des Verfahrens der Auftragsvermittlung

Der klassische Ablauf der Bearbeitung eines Auftrages beginnt damit, dass ein Auftraggeber in der Regel telefonisch einen Transportauftrag abgibt. Dieser wird so schnell wie möglich an einen Kurierfahrer vermittelt, der den Auftrag dann ausführt. Zu Stadtkuriertätigkeiten zählen aber auch regelmäßig anfallende Touren für einen oder mehrere Kunden oder Sendungen, die für einen bestimmten Zeitpunkt vorbestellt werden. Auch kommt es vor, dass einzelne Kurierfahrer für eine zunächst unbestimmte Anzahl von Aufträgen und Zeit von einem Kunden in Anspruch genommen werden.

Als Kurierfahrzeuge werden PKW, Kombis und LKW bis 3t, sowie Fahrräder und in seltenen Fällen Motorräder oder Motorroller eingesetzt. Die letzten drei Fahrzeugtypen werden von der Kurierdiensten meist zu der Kategorie *Biker* zusammengefasst. Die Fahrzeugtypen bieten diverse Möglichkeiten, eine unterschiedliche Anzahl nach Volumen und Masse verschieden großer Sendungsstücke zu befördern.

In den Kurierzentralen werden die eingehenden Aufträge meist an die Kuriere vermittelt. Eine Disposition, d.h. eine Zuordnung und Zuweisung von einzelnen Aufträgen an bestimmte Fahrer, findet dabei in der Regel nicht statt. Nachdem der Auftrag elektronisch erfasst ist, wird er zur Vermittlung freigegeben. Die Vermittler rufen die eingegangenen Aufträge über Sprechfunk aus. Im Anschluss daran können sich die Kurierfahrer um diese Aufträge bewerben. Der erste Fahrer, der sich für einen Auftrag meldet, bekommt die Tour in der Regel zugesprochen.

Bei großen Kurierdiensten in Hamburg werden normalerweise zwei Vermittler und Vermittlungsbereiche (Bereich Ost und Bereich West) für die motorisierten Kuriere eingerichtet, da ansonsten der Vermittlungsprozess zu schwierig und der Sprechfunkverkehr zu umfangreich wird. Außerdem gibt es separate Fahrradvermittlungen, falls bei einem Kurierdienst viele Biker fahren. Wenn Aufträge, für die von den Auftraggebern explizit Biker angefordert wurden, nach einer gewissen festgelegten Zeit nicht an einen Biker vermittelt wurden, werden sie automatisch auch den motorisierten Kurierfahrern angeboten.

Die Kurierdienste unterscheiden sich darin, ob die Kurierfahrer im Normalfall mehrere Aufträge gleichzeitig bearbeiten dürfen. Bei Kurierdiensten, die besonderen Wert auf eine kurze Reaktionszeit legen, werden Touren nur dann zusammengelegt, wenn Engpässe auftreten, d.h. falls aufgrund ungewöhnlich hoher Auftragseingänge nicht genügend Fahrer zur Verfügung stehen, oder falls die Bearbeitungszeiten aufgrund der schlechten Verkehrssituation stark ansteigen.

Für die einzelnen Kurierfahrer ist die gleichzeitige Bearbeitung mehrerer Aufträge ein probates Mittel, um die Ertragssituation zu verbessern. Als grober Richtwert gilt, dass ein motorisierter Kurierfahrer pro gefahrenem Kilometer einen Umsatz von 1,00 € bis 1,05 € erzielen muss, um ausreichende Erträge zu erwirtschaften. Die Tarifstrukturen zeigen, dass die Bündelung von Aufträgen unter diesen Umständen eine wichtige Rolle spielt,

zumal zu den abrechenbaren Fahrdistanzen noch Leerfahrten für die Anfahrt sowie eventuelle Fahrten in Freistellungsbereiche (siehe unten) hinzukommen, die nicht abrechenbar sind. Der Kuriertarif setzt sich aus einer Anfahrtpauschale und entfernungsbezogenen Tarifbestandteilen zusammen. In Hamburg hat sich in der Regel eine Anfahrtpauschale von 2,50 € sowie 0,85 € pro gefahrenem Kilometer durchgesetzt. Für einen Kurierauftrag werden immer die Anfahrtpauschale sowie mindestens drei Kilometer Strecke berechnet, d. h. mindestens 5,05 €.

Falls regelmäßig Touren von den Kurierfahrern gebündelt werden, achten die Vermittler darauf, dass die einzelnen Fahrer nicht so viele Aufträge übernehmen, dass die zeitgerechte Auslieferung der Sendungen gefährdet ist. Sie können ggf. die Zuweisung eines Auftrags an einen Kurierfahrer ablehnen.

Der Vermittlungsprozess verläuft dann etwas anders, wenn es Kurierfahrer gibt, die aktuell keinen Auftrag übernommen haben. Diese Kurierfahrer melden sich, nachdem sie die letzte Sendung ausgeliefert haben, bei der Kurierzentrale „frei“. Der Stadtraum ist dafür in sogenannte Freistellungsbereiche eingeteilt. Diese Freistellungsbereiche haben bei den verschiedenen Kurierdiensten unterschiedliche Größen und Zuschnitte. Postleitzahlbereiche sind die kleinsten ermittelten Freistellungsbereiche bei den befragten Kurierdiensten. Liegt der Quellpunkt eines Auftrages in einem Freistellungsbereich, in dem ein oder mehrere Kurierfahrer frei gemeldet sind, dann bekommt der erste frei gemeldete Kurierfahrer diesen Auftrag exklusiv angeboten, dann der zweite usw. Die Vermittler werden bei einigen Kurierdiensten dadurch unterstützt, dass die frei gemeldeten Fahrer durch die Software am Bildschirm angezeigt werden. Kurierfahrer können sich in einem beliebigen Freistellungsbereich frei melden. Sie fahren u. U. in Freistellungsbereiche, in denen sie sich eine höhere Wahrscheinlichkeit für die Vermittlung eines Auftrages ausrechnen.

6.2.2. Beschreibung des Modells

In das Modell des Status quo gehen folgende Annahmen ein, welche von beobachteten Organisationsformen verschiedener Kurierdienste abstrahieren und auch den weiteren Modellvarianten zu Grunde liegen:

1. Der Normalfall der Auftragsvergabe ist die Vermittlung, wobei ein Auftrag allen geeigneten Kurieren der Flotte angeboten wird. Die Kuriere einer Fahrzeugkategorie bewerten jeden Auftrag nach identischen Kriterien und geben umso schneller eine Bewerbung ab, je besser die Bewertung ausfällt. Der erste interessierte Fahrer bekommt den Auftrag zugesprochen.

2. Es wird nur zwischen motorisierten und unmotorisierten Kurieren unterschieden. Verschiedene KFZ-Typen bleiben unberücksichtigt.

3. Bei der Auftragsvergabe werden Verkehrsmittelwünsche und Freistellungen, jedoch keine unterschiedlichen Freistellungsbezirke oder -dauern beachtet.

4. Kuriere können mehrere Aufträge gleichzeitig übernehmen und in geschachtelter Form abarbeiten. Jeder Kurier plant dabei eigenständig seine Touren. Neben der Minimierung der Fahrstrecke berücksichtigt er auch die Einhaltung der vom Kurierdienst garantierten maximalen Lieferzeit von 2,5 Stunden.

5. Kann ein Auftrag über längere Zeit nicht vermittelt werden, so führt der Vermittler eine sogenannte „Druckvermittlung“ durch, die als Disposition abgebildet ist.

Die Umsetzung dieser Annahmen in ein Multiagentenmodell umfasst die Modellierung der Umwelt, der Agenten sowie ihrer Organisationsstrukturen und Kommunikationsprotokolle.

6.2.2.1. Modellierung der Umgebung

Die Umgebung der Kuriere bildet das Hamburger Straßennetz, welches auf ein graphbasiertes Raummodell in FAMOS abbildbar ist. Neben den Kurieren, denen zu jedem Simulationszeitpunkt die Position eines Verkehrsknotens zugeordnet ist, beinhaltet die Umwelt keine weiteren Objekte und auch keine eigene Dynamik. Letztere wäre z. B. zur Darstellung des tageszeitabhängigen Verkehrsaufkommens erforderlich.

Die Kuriere bewegen sich zwischen Auftragspunkten entlang vorab geplanter Routen. Da Kurierfahrer allgemein über sehr gute Ortskenntnisse verfügen, erscheint der Einsatz einer optimalen Suchstrategie für die Routensuche angemessen. Gewählt wird eine Variante des Dijkstra-Algorithmus zur Bestimmung kürzester Wege in Graphen, die von FAMOS bereits zur Verfügung gestellt wird. Über eine geeignete Bewertungsfunktion lässt sich die Routenplanung so beeinflussen, dass die schnellste Verbindung zwischen Start und Ziel in Abhängigkeit vom Verkehrsmittel des jeweiligen Kuriers bestimmt wird. Den Kanten des Graphen (Streckenabschnitten des Straßennetzes) sind dafür verschiedene durchschnittliche Geschwindigkeiten für motorisierte und unmotorisierte Kuriere zugeordnet. Auf diese Weise ist auch die unterschiedliche Nutzung von Straßen durch Autofahrer und Biker modellierbar. Bikern wird die Fahrt auf Autobahnen untersagt, indem die zugehörigen Geschwindigkeiten auf Null gesetzt und entsprechende Pfade daher niemals gewählt werden. Andererseits wird Bikern üblicher Praxis gehorchend das Befahren von Einbahnstraßen in beiden Richtungen gestattet. Zu diesem Zweck werden auch Einbahnstraßen auf zwei gerichtete Kanten abgebildet.

6.2.2.2. Modellierung der Organisationsstrukturen

Aus Sicht der Funker ist die Kurierflotte in mehrere funktionale Rollen unterteilt, die bei der Auftragsvermittlung unterschiedlich zu behandeln sind:

- Das *Verkehrsmittel* jedes Kuriers ist für die Einhaltung der Verkehrsmittelpräferenzen von Aufträgen zu beachten.
- Der *Beschäftigungsstatus* jedes Kuriers ist aufgrund der Bevorzugung freigestellter Kuriere bedeutsam.

Diese Organisationsform kann durch eine hierarchische Gruppenstruktur modelliert werden, wobei die übergeordnete Gruppe aller angemeldeten Kuriere vier Untergruppen für jede mögliche Kombination der Kriterien (freigestellte Biker, beschäftigte Biker, usw.) enthält. Verändert sich der Status eines Kuriers bei Arbeitsbeginn, Arbeitsende, Freistellung oder Neubeschäftigung, so wechselt er entsprechend die Gruppenzugehörigkeit.

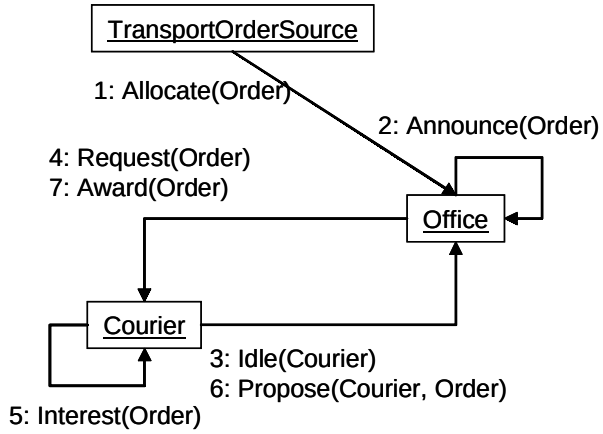

Abbildung 6.2.: Vermittlungsprotokoll des Kurierdienst-Modells

Abbildung 6.2 zeigt das Kommunikationsprotokoll der Auftragsvermittlung als UML-Kollaborationsdiagramm . Da es sich aus theoretischer Sicht prinzipiell um eine Variante des bekannten Kontraktnetz-Protokolls (siehe Abschnitt 2.3.3) handelt, wurden die englischen Bezeichnungen der standardisierten Nachrichtentypen beibehalten. An der Kommunikation beteiligte Rollen sind Kuriere, Vermittler und Kunden, die über typisierte Nachrichten kommunizieren. Von mehreren beschäftigten Vermittlern wird durch Zusammenfassung zu einem einzigen Objekt *Office* (Vermittlungszentrale) abstrahiert. Kunden werden zu einer Auftragsquelle zusammengefasst, welche anhand der vorliegenden Auftragsdaten zu bestimmten Zeitpunkten Transportaufträge ins System schleust. Die Auftragsvermittlung beginnt, indem die Auftragsquelle ein mit dem Auftrag parametrisiertes Signal *Allocate* an die Zentrale sendet. Die Zentrale merkt sich daraufhin selbst ein Signal *Announce* zum sofortigen Termin der ersten Auftragsverlesung vor.

Die Zentrale verliest die eingegangenen Aufträge in einem festen Zeitabstand nacheinander vor jeder Kuriergruppe, die zu deren Bearbeitung fähig ist. Abhängig vom Verkehrsmittelwunsch wird dabei folgende Verlesungsreihenfolge eingehalten:

- Auto-Aufträge werden zuerst allen freigestellten und anschließend allen beschäftigten Autofahrern angeboten. Biker können diese Auftragsart nicht ausführen.
- Fahrrad-Aufträge werden erst den freigestellten Bikern, dann den beschäftigten Bikern, als nächstes den freigestellten Autofahrern und zuletzt den beschäftigten Autofahrern angeboten.

Sobald eine Verlesung ansteht, sendet der Zentralen-Agent ein mit dem Auftrag parametrisiertes Signal *Request* an die nächste Kuriergruppe und merkt sich selbst die folgende Verlesung zum nächsten Verlesezeitpunkt vor. Alle angesprochenen Kuriere erhalten das *Request*-Signal und bewerten den Auftrag quantitativ anhand der im folgenden Abschnitt genannten Kriterien. Das auf diese Weise ermittelte Auftragsinteresse jedes Kuriers wird auf eine Reaktionszeit abgebildet, die umso kürzer ausfällt, je höher das Interesse ist. Jeder Kurier merkt sich im Anschluss an die Bewertung selbst ein Signal *Interest* nach Ablauf

seiner Reaktionszeit vor und sendet bei Eintritt dieses Signals als Abgabe seiner Bewerbung ein *Propose*-Signal an die Zentrale. Der erste Kurier, dessen Bewerbung bei der Zentrale eintrifft, bekommt den Auftrag durch Zusendung des Signals *Award* zugesprochen und ordnet diesen nach einer weiter unten beschriebenen Strategie in seine Tour ein. Alle später eingehenden *Propose*-Signale werden von der Zentrale verworfen.

Die Zentrale liest jeden Auftrag allen geeigneten Kuriergruppen vor, bis der erste Kurier Interesse bekundet. Zeigt kein Kurier Interesse, beginnt ein neuer Verlesungszyklus. Sonderfälle in diesem Ablauf ergeben sich durch die bevorzugte Behandlung freigemeldeter Kuriere und die sogenannte „Druckvermittlung“. Letztere tritt ein, wenn ein Auftrag nicht innerhalb der vorgegebenen Zeitspanne vermittelt werden konnte. Die Zentrale wählt dann einen geeigneten Kurier aus (modelliert als derjenige Kurier, dessen Standort dem Abholpunkt am nächsten liegt) und weist ihm per *Award*-Signal den kritischen Auftrag zu. Dieses Verfahren entspricht einer Disposition. Es kommt auch zum Einsatz, wenn sich ein Kurier durch Senden eines *Idle*-Signals bei der Zentrale freimeldet.

6.2.2.3. Modellierung der Kuriere

Das Verhalten eines Kuriers kann wie in Abbildung 6.3 durch ein hierarchisches Zustandsdiagramm modelliert werden und setzt sich im Wesentlichen aus den Aspekten „Kommunikation mit der Zentrale“ und „Auftragsausführung“ zusammen. Der Lebenszyklus jedes Kuriers beginnt zum Zeitpunkt seines aus den Fahrerdaten ermittelten Arbeitsbeginns. Zunächst nimmt er den Zustand *Im_Dienst* ein und merkt sich ein Zeitsignal für den Zeitpunkt des geplanten Arbeitsendes vor. Im nächsten Schritt nimmt der Kurier den Zustand *Freigestellt* ein, wobei er sich freimeldet, d.h. der Zentrale ein Signal *Idle* sendet und sich der seinem Verkehrsmittel zugeordneten Gruppe freigestellter Kuriere anschließt. Da über längere Zeit freigemeldete Kuriere meist nicht an ihrem Standort verharren, sondern in Richtung eines höheren Auftragsaufkommens fahren, wird in *Freigestellt* ein weiteres Zeitsignal vorgemerkt. Nach Ablauf dieser Wartezeit verliert der Kurier quasi die Geduld und beginnt, sich zu einem markanten Punkt in der Stadtmitte zu bewegen, wo generell ein hohes Auftragsaufkommen zu erwarten ist.[3]

Die Kommunikation zwischen Kurier und Zentrale erfolgt durch statische Reaktionen des Zustands *Im_Dienst*. Auf diese Weise ist die Kommunikation wie im Realsystem unabhängig vom aktuellen Zustand der Auftragsausführung. Erhält der Kurier ein *Request*-Signal von der Zentrale, so berechnet er sein Interesse[4] am beigefügten Auftrag. Falls dieses einen Grenzwert überschreitet, erscheint der Auftrag dem Kurier akzeptabel. Das Interesse wird dann auf die Reaktionszeit des Kuriers abgebildet: Je größer das Interesse, desto kürzer die Zeit, die vergeht, bis der Kurier sein Angebot zur Auftragsübernahme an die Zentrale sendet. Für den Ablauf der Reaktionszeit merkt sich der Kurier ein *Inte-*

[3] Alternativ zu dieser einfachen Heuristik könnte er das Auftragsaufkommen beobachten und den daraus errechneten Schwerpunkt aller relevanten Aufträge ansteuern, was allerdings ein komplexeres Verhaltensmodell zur Folge hätte.

[4] In die Berechnung des Auftragsinteresses gehen die Kriterien Auftragsnutzen, Auftragsbedarf und individuelle Auslastung ein. Als Ausschlusskriterium fungiert die vom Kurierdienst garantierte maximale Lieferdauer; wird diese bei der Einplanung des neuen Auftrags in die bestehende Tour des Kuriers überschritten, so lehnt der Kurier den Auftrag ab. Eine detaillierte Beschreibung der verwendeten Funktionen ist im Endbericht des Forschungsprojekts (Knaak et al. 2004, S. 42ff) zu finden.

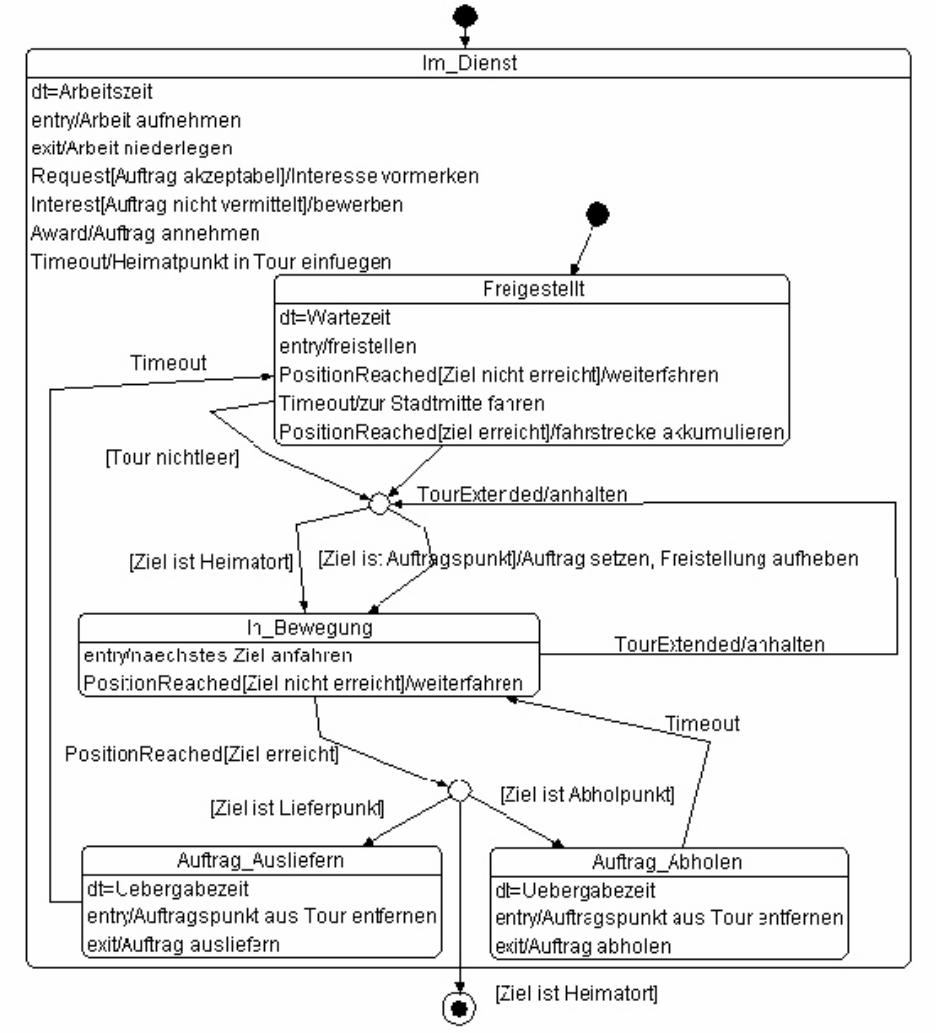

Abbildung 6.3.: Zustandsdiagramm eines Kuriers

rest-Signal vor und bekundet erst bei Eintritt dieses Signals der Zentrale sein Interesse. Bekommt der Kurier den Auftrag zugesprochen, so erhält er ein *Award*-Signal, auf das er mit der Einsortierung des Auftrags in seine Tour reagiert. Die Tour enthält die Abhol- und Lieferpunkte aller derzeit vom Kurier übernommenen Aufträge in der Reihenfolge der geplanten Abarbeitung. Zur Tourenplanung verwendet der Kurier eine suboptimale Strategie[5], bei der Abhol- und Lieferpunkt eines neu übernommenen Auftrags mit möglichst geringem Umweg in die bestehende Tour eingefügt werden.

Mit der letzten Reaktion des Zustands *Im_Dienst* behandelt der Kurier das zum geplanten Arbeitsende eintretende Zeitsignal. Auf dieses Signal kann nicht mit unmittelbarer Niederlegung der Arbeit reagiert werden, da die Tour möglicherweise noch Aufträge enthält, die zuvor abgearbeitet werden müssen. Statt dessen fügt der Kurier seinen speziell gekennzeichneten Heimatort als letzten anzufahrenden Punkt in die Tour ein. Auf diese Weise fährt er im Anschluss an alle noch ausstehenden Auftragspunkte zum Heimatort und beendet bei dessen Erreichung seinen Lebenszyklus. Nach Ablauf der regulären Arbeitszeit bewirbt sich der Kurier weiterhin um angebotene Aufträge, berücksichtigt bei der Berechnung des Auftragsinteresses jedoch, inwiefern deren Ausführung mit der Heimfahrt vereinbar ist.

Bei jedem Einfügen eines weiteren Punktes in die Tour wird ein Signal *TourExtended* vorgemerkt. Dieses signalisiert, dass der Kurier nun den ersten Punkt der Tour anfahren soll. Im Zustand *Freigestellt* wird das Signal durch einen Übergang nach *In_Bewegung* behandelt. Zuvor prüft der Kurier jedoch, ob der anzufahrende Punkt ein Auftragspunkt oder der Heimatort ist. Im ersten Fall wählt er den zugehörigen Auftrag zur Bearbeitung aus und hebt die Freistellung auf, indem er von der Gruppe der freigestellten in die der beschäftigten Kuriere wechselt. Bei Eintritt in den Zustand *In_Bewegung* berechnet er in der Eintrittsaktion die schnellste Verbindung zwischen seinem momentanen Standort und dem nächsten Punkt der Tour und beginnt anschließend, diesen Pfad im Verkehrsnetz abzufahren. Wenn das angestrebte Ziel erreicht ist, wechselt der Kurier in den Zustand der Warenübergabe. Werden während der Fahrt weitere Auftragspunkte in die Tour eingefügt, tauscht der Kurier gegebenenfalls sein aktuelles Ziel gegen einen neuen, günstiger gelegenen Auftragsort aus.

Die Abholung bzw. Auslieferung der Aufträge in den Zuständen *Auftrag_Abholen* und *Auftrag_Ausliefern* unterscheidet sich nur geringfügig. In beiden Fällen wird der nunmehr erreichte Auftragspunkt aus der Tour entfernt und anschließend eine im Modell mit fünf Minuten veranschlagte Übergabezeit abgewartet. Beim Austritt aus dem jeweiligen Zustand erfolgt die Übergabe der Sendung, die in der Implementation hauptsächlich im Setzen einiger Auftragsattribute und Aktualisierung von Statistiken besteht. Beim Verlassen des Zustands *Auftrag_Abholen* erfolgt ein direkter Übergang nach *In_Bewegung*, da auf jeden Fall noch der Lieferpunkt des entsprechenden Auftrags in der Tour enthalten ist. Beim Verlassen von *Auftrag_Ausliefern* kann die Tour hingegen auch leer sein. Zur Vereinfachung der Diagrammstruktur wird hier in jedem Fall der Zustand *Freigestellt* ein-

[5] Der Einsatz suboptimaler Tourenplanung erscheint angemessen, da Kuriere über Sprechfunk eine Vielzahl von Auftragsangeboten verfolgen und sich in kurzer Zeit entscheiden müssen. Zur Abschätzung der Entfernungen zwischen Auftragspunkten wird nicht das Straßennetz, sondern die mit einem fahrzeugspezifischen Umwegfaktor gewichtete Luftliniendistanz verwendet. Zusätzlich werden größere Hindernisse im Stadtgebiet berücksichtigt, die zu umfahren sind, wie beispielsweise die Außenalster.

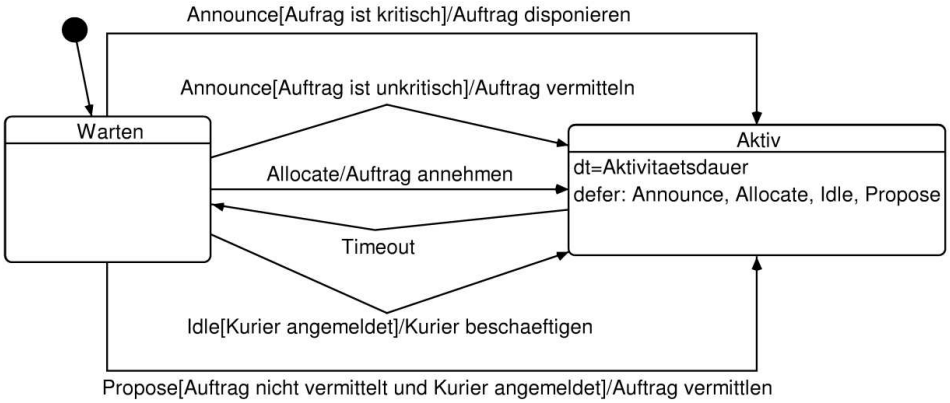

Abbildung 6.4.: Zustandsdiagramm der Kurierdienst-Zentrale

genommen und, falls die Tour nicht leer ist, über die implizite Transition sofort wieder verlassen.

6.2.2.4. Modellierung der Zentrale

Das Verhalten der Zentrale ist einfacher als das der Kuriere und stellt im Wesentlichen eine Variante des Vermittlers aus dem Kontraktnetz-Protokoll dar. Die Zentrale verfügt über eine Liste aller zur Vermittlung stehenden Aufträge und behandelt diese entsprechend dem in Abbildung 6.4 dargestellten Zustandsdiagramm. Zu Beginn ihres Lebenszyklus wartet sie im Zustand *Warten* auf eingehende Signale. Sobald das Signal *Allocate* den Eingang eines Auftrags meldet, trägt sie diesen in die Auftragsliste ein, bestimmt alle geeigneten Kuriergruppen und merkt ein *Announce*-Signal für die erste Verlesung vor. Anschließend geht sie in den Zustand *Aktiv* über, der im Anschluss an jede Handlung besucht wird. Der Aufenthalt in *Aktiv* bildet die Zeit ab, welche die Zentrale zur Ausführung ihrer Aktivitäten benötigt und in der keine weiteren Signale bearbeitet werden können.

Zu jedem Verlesungstermin eines Auftrags prüft die Zentrale, ob der kritische Zeitpunkt, ab dem mit der Druckvermittlung begonnen wird, bereits erreicht ist. In diesem Fall wird versucht, den Auftrag zu disponieren. Dazu bestimmt die Zentrale unter Berücksichtigung der Verkehrsmittelpräferenz und der Freistellungen denjenigen Kurier, dessen Standort dem Abholpunkt am nächsten liegt und sendet diesem ein *Award*-Signal. Wird kein Kurier gefunden, so merkt sie den nächsten Dispositionsversuch vor. Kann der Auftrag noch herkömmlich vermittelt werden, sendet sie allen Kurieren der nächsten anstehenden Kuriergruppe ein *Request*-Signal, woraufhin diese den Auftrag bewerten. Anschließend merkt sie die Verlesung vor der folgenden relevanten Kuriergruppe zum nächsten Verlesezeitpunkt vor.

Geht das Angebot *Propose* eines Kuriers ein, so prüft die Zentrale, ob der betreffende Auftrag nicht bereits vermittelt wurde, und der anbietende Kurier noch angemeldet ist. Wenn beides der Fall ist, gewährt sie dem Kurier als erstem Bieter durch ein *Award*-Signal den Zuschlag und löscht anschließend den Auftrag aus der Auftragsliste; damit gilt der

Auftrag als vermittelt. Auch beim Eingang einer Freimeldung *Idle* prüft die Zentrale zunächst, ob der freimeldende Kurier noch angemeldet ist, da er möglicherweise in der Zeit zwischen der Abgabe der Freimeldung und deren Bearbeitung durch die Zentrale die Arbeit niedergelegt hat. Ist der Kurier noch angemeldet, versucht die Zentrale, ihn mit dem am besten geeigneten Auftrag der Auftragsliste zu beschäftigen.

6.2.3. Implementation in FAMOS

Das bereits weitgehend operationalisierte Modell kann unter Verwendung der Klassen des Frameworks FAMOS mit relativ geringem Aufwand implementiert werden. Wie aus Abbildung 6.5 zu entnehmen ist, stellt die Implementation der Agenten die wesentliche Aufgabe eines Modellierers dar. Für die Realisierung der Umwelt und der Organisations- und Kommunikationsstrukturen kann dagegen auf fertige Bausteine des Frameworks zurückgegriffen werden.

6.2.3.1. Implementation der Zentrale

Da die Zentrale als rein kommunizierender Agent modelliert ist, wird die Klasse `Office` direkt von der Klasse `Agent` abgeleitet. Ihr Verhalten wird im von FAMOS bereitgestellten Editor für Zustandsdiagramme entsprechend dem Modell aus Abbildung 6.4 implementiert. Dabei sind lediglich die verbalen Diagrammbeschriftungen in Java-Anweisungen umzusetzen. Für jede elementare Handlung (Effektor) oder Bedingungsabfrage (Sensor) der Zentrale erhält die Klasse `Office` eine Methode, die vom Zustands-Automaten aus aufgerufen werden kann. So entspricht beispielsweise die Methode `takeOrder()` der informellen Anweisung „Auftrag annehmen“ und die Methode `occupy()` der Anweisung „Kurier beschäftigen“. Mit Hilfe des in Abschnitt 5.2.3 ab Seite 150 beschriebenen Codegenerators wird aus der Basisklasse `Office` und dem im Editor spezifizierten Zustandsdiagramm eine abgeleitete Klasse `FSMOffice` mit einem Verhaltens-Objekt der Klasse `StateMachine` erzeugt. Diese Klasse realisiert das Verhalten der Kurierdienstzentrale als ablauffähiger Zustandsautomat.

6.2.3.2. Implementation der Kuriere

Auf ähnliche Weise wird die Klasse `Courier` mit Zustandsattributen sowie Sensor- und Effektor-Methoden der Kuriere implementiert und daraus mit Hilfe des Codegenerators die Klasse `FSMCourier` zur Kapselung des Kurierverhaltens erzeugt. Im Gegensatz zur Zentrale sind Kuriere als situierte und im Verkehrsnetz mobile Agenten von `MobileAgent` abgeleitet. Ihre Fähigkeit zur Bewegung wird als ein Objekt der Klasse `OdoRoute` repräsentiert. Diese Klasse ermöglicht die zielgerichtete Bewegung im Graphen des Verkehrsnetzes entlang einer vorab ermittelten Route zwischen einem Anfangs- (Position des Kuriers) und einem Zielpunkt (Auftrags- bzw. Heimatpunkt) und protokolliert dabei die zurückgelegten Strecken. Dem `OdoRoute`-Objekt obliegt auch die Ermittlung der schnellsten Verbindung zwischen diesen Punkten, welche – wie von FAMOS als Standard vorgegeben – mit Hilfe der Suchstrategie `GraphPathDijkstra` erfolgt. Um den jeweils schnellsten Weg in Abhängigkeit des Kurierfahrzeugs zu bestimmen, erhält die

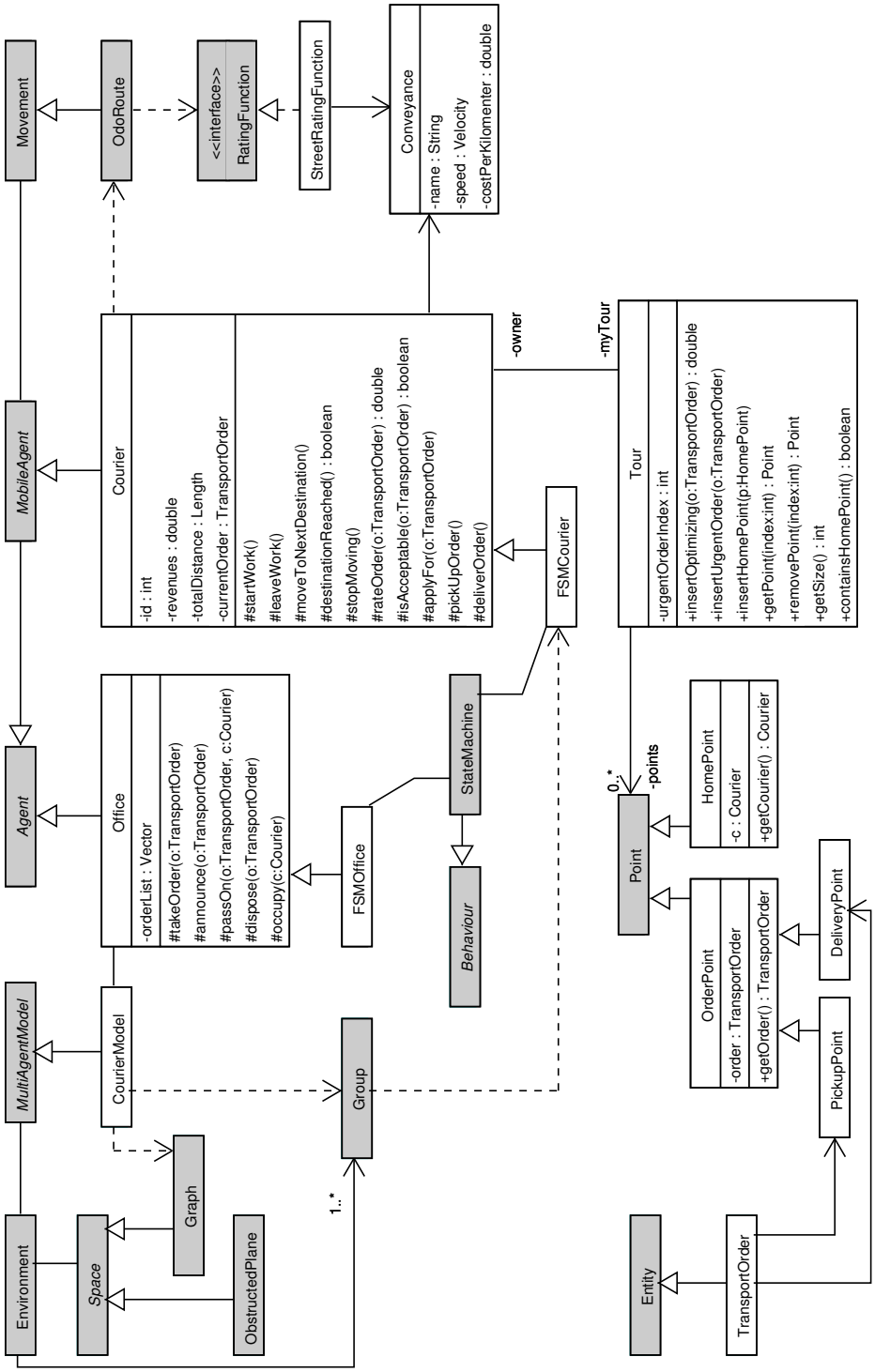

Abbildung 6.5.: Implementation des Kurierdienstmodells. Gezeigt sind nur die wesentlichen Klassen des Basismodells; die verwendeten Klassen des Frameworks FAMOS sind grau unterlegt.

OdoRoute zusätzlich eine benutzerdefinierte Bewertungsfunktion der Klasse StreetRatingFunction. Diese berücksichtigt neben der Kantenlänge auch die vom Verkehrsmittel abhängige Durchschnitts- bzw. zulässige Höchstgeschwindigkeit entlang einer Kante.

Die Implementation der Kurierbewegung ist bereits weitgehend vom Framework FAMOS vorgegeben (vgl. Abschnitt 5.2.1.2). Im Konstruktor werden der Oberklasse MobileAgent die erforderlichen Bewegungs- und Pfadsuche-Strategien übergeben. Wenn der Kurier den ersten Punkt seiner Tour anfährt, setzt er diesen als neuen Zielpunkt der OdoRoute, welche dabei die schnellste Route zum Ziel berechnet. Durch Aufruf der Methode move() von MobileAgent beginnt der Kurier, sich zum ersten Knotenpunkt der Route zu bewegen. Die Bewegung wird entsprechend dem Ansatz der ereignisgesteuerten Simulation durch Vormerkung eines Ankunftsereignisses realisiert. Nach Verstreichen der Fahrzeit zum nächsten Knotenpunkt wird dem Kurier die Ankunft durch ein Signal der Klasse PositionReached signalisiert, während seine Position bereits automatisch aktualisiert worden ist. Der erneute Aufruf von move() setzt die Fahrt zum nächsten Knotenpunkt fort, bis das Ziel erreicht ist. Erhält der Kurier auf dem Weg zwischen zwei Knotenpunkten einen neuen Auftrag, so unterbricht er die Fahrt mittels stopMoving() und tauscht gegebenenfalls sein Ziel gegen einen günstiger gelegenen Auftragspunkt aus. Die OdoRoute akkumuliert automatisch die bisher auf der geplanten Route zurückgelegte Strecke und ermittelt anschließend eine neue Route von der aktuellen Position zum neuen Ziel.

Die Tourenplanung und Auftragsbewertung der Kuriere wird nicht durch Framework-Klassen unterstützt und ist deshalb von Hand zu implementieren. Die Tour eines Kuriers wird durch ein Objekt der Klasse Tour repräsentiert, welches die Auftragspunkte aller bearbeiteten Aufträge in der Reihenfolge ihrer geplanten Abarbeitung enthält. Bei den von Point abgeleiteten modellspezifischen Punkten wird zwischen Abhol- und Lieferpunkten (PickupPoint bzw. DeliveryPoint) der Aufträge sowie Heimatpunkten (HomePoint) der Kuriere unterschieden. Anhand der Klasse eines Zielpunkts bestimmt der Kurier, welche Handlungen bei dessen Erreichen auszuführen sind. Neue Punkte können auf verschiedene Weise in die Tour eingefügt werden. Die Methode insertOptimizing() ordnet Abhol- und Lieferpunkt eines Auftrags entsprechend einer suboptimalen Strategie mit geringstem Umweg in die Tour ein und gibt für die Auftragsbewertung den ermittelten Umweg zurück. Mit insertUrgentOrder() ordnet der Kurier einen disponierten und als zeitkritisch deklarierten Auftrag am Anfang der Tour bzw. hinter dem letzten bereits enthaltenen kritischen Auftrag ein.

6.2.3.3. Implementation der Umwelt und Organisationsstrukturen

Zur Zusammenfassung der Agenten und der weiteren Modellkomponenten dient die von MultiAgentModel abgeleitete Klasse CourierModel. Jedem Multiagentenmodell ist in FAMOS ein Objekt der Klasse Environment zugeordnet, welches die Umwelt der Agenten realisiert und ein Raummodell (Klasse Space) beinhalten kann. Für das Kurierdienstmodell wird zur Abbildung des Straßennetzes ein Raummodell der Klasse Graph gewählt. Bei der Initialisierung des Modells wird das im Geographischen Informationssystem erstellte Hamburger Straßennetz mit seinen Knoten, Kanten und Kantenattributen

über die entsprechenden Import-Schnittstellen von FAMOS (vgl. Abschnitt 5.2.4.3) aus einer Datei eingelesen und in ein `Graph`-Objekt umgewandelt. Des weiteren dient ein kontinuierliches Raummodell der Klasse `ObstructedPlane` zur Entfernungsabschätzung bei der Auftragsbewertung durch die Kuriere.

Neben dem Raummodell verwaltet das `Environment`-Objekt auch die Organisationsstrukturen; in diesem Fall die zur Multicast-Kommunikation bei der Vermittlung benötigte hierarchische Gruppenstruktur. Diese besteht aus einer übergeordneten Gruppe der Klasse `Group` für alle angemeldeten Kuriere mit vier Untergruppen für jede Kombination der berücksichtigten Kriterien „Verkehrsmittel“ und „Beschäftigungs-Status“. Auf diese Weise kann die Zentrale gezielt eine bestimmte Kuriergruppe (z.B. die frei gemeldeten Biker) ansprechen, indem sie an die entsprechende Gruppe ein Signal sendet. Das `Group`-Objekt verteilt dieses an alle Mitglieder.

Die Kommunikation zwischen Kurieren und Zentrale sowie die Vormerkung von Handlungsabsichten erfolgt mit den im Kollaborationsdiagramm von Abbildung 6.2 aufgeführten parametrisierten Signalen. Jedes Signal ist in FAMOS als eigene Klasse zu repräsentieren. Dabei sind die agenten-intern als Handlungsabsicht vorgemerkten Signale von `InternalSignal` abzuleiten und die für die Kommunikation verwendeten Signale von `ExternalSignal`. Die im Kurierdienstmodell benötigten Signale wurden im Signal-Dialog des Diagrammeditors definiert und vom Codegenerator automatisch in modellspezifische Klassen mit passenden Konstruktoren und Zugriffsmethoden übersetzt.

Transportaufträge sind als passive Entitäten realisiert. Die Klasse `TransportOrder` kapselt lediglich eine Reihe hier nicht näher betrachteter Auftragsattribute wie Abhol- und Lieferpunkt, Vermittlungsdauer usw., die im Simulationslauf gesetzt und bei der Datensammlung ausgelesen werden. Zur Datensammlung dienen die von `Individual-Observer` abgeleiteten Klassen `CourierObserver` und `OrderObserver`, welche während der Simulation gesetzte Attribute der Kuriere bzw. Aufträge über benutzerdefinierte Sonden auslesen und in tabellarischer Form im Ergebnisreport ausgeben. Das Modell beinhaltet zwei `CourierObserver` zur getrennten Beobachtung von Bikern und Autokurieren sowie einen `OrderObserver` zur Beobachtung aller erledigten Transportaufträge. Des weiteren wurden zwei von `ExternalObjectArrival` abgeleitete Ankunftsprozesse implementiert, welche aus den über die Import-Schnittstelle `FileDataSource` eingelesenen empirischen Eingabedaten des Kuriermodells Objekte der Klassen `Transport-Order` bzw. `FSMCourier` erzeugen und zu den gegebenen Simulationszeitpunkten (Ankunft eines Auftrags bzw. Arbeitsbeginn eines Kuriers) ins System schleusen. Diese Klassen für den Im- und Export von Daten sind in Abbildung 6.5 nicht aufgeführt.

6.3. Modellvariante 1: Die Logistikstrategie Hub and Shuttle

Auf der Grundlage des Basismodells konnten mit relativ geringen Modifikationen Modelle der alternativen Logistikkonzepte *Hub and Shuttle* und *Innen/Außen* (siehe Abschnitt 6.4) entwickelt werden. Dabei wurde versucht, das Verhalten der beteiligten Agenten (Kuriere und Zentrale) so geringfügig wie möglich zu verändern. Dies erlaubt es einerseits, Auswirkungen der Logistikstrategien isoliert betrachten zu können, und erfüllt andererseits die Forderung nach möglichst weitgehender Beibehaltung bestehender Kurierdienst-

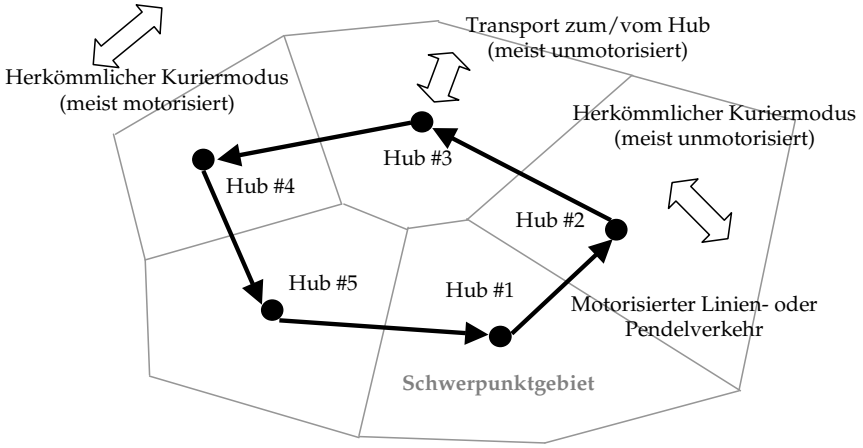

Abbildung 6.6.: Prinzip des alternativen Logistikkonzepts *Hub and Shuttle*.

Strukturen.

6.3.1. Beschreibung der Strategie

Bei dieser an der Organisation von Expressdiensten orientierten Strategie werden im Stadtbereich Schwerpunktgebiete festgelegt, die sich durch ein hohes Auftragsaufkommen auszeichnen. In diesen Gebieten wird jeweils ein Sammelzentrum (Umschlagpunkt, *Hub*) eingerichtet, über das mit wenigen Ausnahmen alle Sendungen aus dem oder in das Gebiet abgewickelt werden. Zwischen den Hubs werden die Sendungen durch größere Fahrzeuge gebündelt transportiert, die z.B. im Linienverkehr nacheinander alle Sammelzentren anfahren oder zwischen je zwei Hubs pendeln (*Shuttle*), vgl. Abbildung 6.6. Innerhalb eines Schwerpunktgebiets holen Kuriere die Sendungen wie üblich beim Absender ab und bringen sie zum Sammelzentrum bzw. liefern Sendungen vom Sammelzentrum zum Empfänger aus. Sie können mehrere Aufträge parallel bearbeiten und bleiben auch in der Tourenplanung autonom. In der Regel ordnen sie sich jedoch einem Schwerpunktgebiet zu.

Ausgenommen von diesem Prinzip sind Aufträge, bei denen Quellpunkt (der Sitz des Absenders) und Zielpunkt (der Sitz des Empfängers) innerhalb desselben Schwerpunktgebiets liegen, bei denen mindestens einer der beiden Punkte außerhalb der definierten Schwerpunktgebiete liegt, die Sendungen eilbedürftig sind oder eine Mindestdistanz zwischen Quell- und Zielpunkt unterschritten wird. Diese Sendungen werden nicht über das *Hub*-System transportiert, sondern weiterhin von einem Kurier direkt zwischen Absender und Empfänger befördert.

Der Erfolg dieser Strategie ist offensichtlich abhängig von der Wahl geeigneter Schwerpunktgebiete. Anzahl, Lage und Größe der Gebiete sollten auf die räumliche Verteilung der Aufträge abgestimmt sein. Der gebrochene Transport über die *Hubs* ermöglicht einen verstärkten Einsatz von Radkurieren innerhalb der Schwerpunktgebiete, da die zurückzulegenden Strecken innerhalb eines Gebietes kurz genug sind. Die Anlieferung größerer

Abbildung 6.7.: Aufteilung des Hamburger Stadtgebiets in drei (links) bzw. fünf (rechts) Schwerpunktgebiete für das *Hub and Shuttle*-Modell.

Mengen von Aufträgen zum selben Zeitpunkt (Eintreffen des *Shuttle*-Fahrzeugs am *Hub*) bedeutet ein erhöhtes Bündelungspotential für die Radkuriere, falls Größe und Gewicht der zu befördernden Güter dies zulassen. Im Idealfall lässt sich der motorisierte Verkehr im zentralen Stadtgebiet auf die zwischen den *Hubs* pendelnden Transporter beschränken.

Dagegen ist auch bei geschickter Organisation des Pendelverkehrs eine Erhöhung der Lieferzeiten zu erwarten. Für die meisten Kuriersendungen reicht es erfahrungsgemäß jedoch aus, wenn sie innerhalb eines Tages den Empfänger erreichen. Ein weiteres Problem ist neben der kostenintensiven Einrichtung der Infrastruktur das kompliziertere Abrechnungs- und Vermittlungsverfahren. So ist ein neuer Verrechnungsmodus für Aufträge zu finden, da bei Transport über das *Hub*-System mehrere Kuriere an der Auslieferung eines Auftrags beteiligt sind. Eine Lösung besteht darin, die Fahrer der *Shuttle*-Fahrzeuge beim Kurierdienst fest anzustellen, so dass nur der anliefernde und der abholende Kurier bei der Verteilung der Auftragsvergütung berücksichtigt werden müssen. Dann ließe sich jeder über Umschlagpunkte abgewickelte Auftrag in zwei getrennt vermittelte und abgerechnete Touren vom Versender zum *Hub* des Startgebiets sowie vom *Hub* des Zielgebiets zum Empfänger aufteilen.

6.3.2. Beschreibung des Modells

Für das Modell der *Hub and Shuttle*-Strategie wurden die vorhandenen Auftragsdaten einer manuellen räumlichen Analyse unterzogen, um geeignete Schwerpunktgebiete mit zugehörigen Umschlagpunkten zu ermitteln. Dabei war zu berücksichtigen, dass die einzelnen Hubs nicht mehr als fünf bis sechs Kilometer auseinander liegen sollten, damit jeder *Hub* zweimal pro Stunde angefahren werden kann. Zwei verschiedene Aufteilungen des inneren Stadtgebiets mit drei bzw. fünf Schwerpunktgebieten und *Hubs* an verkehrsgünstig gelegenen Standorten wurden gewählt (vgl. Abbildung 6.7), wobei sich die Aufteilung in drei Schwerpunktgebiete in Kalibrierungsläufen als günstiger herausgestellt hat.

Bei der Ausgestaltung des *Shuttle*-Verkehrs ist ein geeigneter Kompromiss aus der An-

zahl benötigter Transporter, der Einhaltung maximaler Lieferzeiten und hinreichender Sendungsbündelung zu finden. Hier wurde versucht, mit einer möglichst geringen Anzahl an Fahrzeugen auszukommen, um die Kosten gering zu halten; im Modell verkehren daher zwischen den *Hubs* zwei Lieferanten in festem Linienverkehr mit gegenläufiger Richtung.

Den Aufträgen werden als zusätzliche Attribute ein Start- und ein Ziel-*Hub* zugeordnet, welche nur dann gesetzt sind, wenn der Auftrag über das *Hub*-System abzuwickeln ist. Bei Eingang eines Auftrags überprüft die Zentrale anhand der Auftragsattribute zunächst, ob dieser für den *Hub*-Transport in Frage kommt. Dies ist der Fall, wenn Abhol- und Lieferpunkt in unterschiedlichen Schwerpunktgebieten liegen und die unter Berücksichtigung des verkehrsmittel-spezifischen Umwegfaktors geschätzte Auftragsdistanz drei Kilometer überschreitet. Dementsprechend vermittelt die Zentrale den Auftrag entweder vom Sender direkt zum Empfänger oder zum *Hub* des Startgebiets. Das übrige Vermittlungsverfahren unterscheidet sich nicht vom Basismodell. Die Anfahrtpauschale der über Umschlagpunkte abgewickelten Aufträge wird halbiert, da diese aufgrund der doppelten Vermittlung zwei Anfahrten, jedoch oft nur geringe Anfahrtswege erfordern.

Auch das Verhalten der Kuriere bleibt weitgehend unverändert. Zusätzlich ist lediglich das Abholen und Deponieren von Aufträgen am Umschlagpunkt zu berücksichtigen. Von Kurieren zu unterscheiden sind dagegen die Lieferanten. Jeder Lieferant fährt nacheinander alle Umschlagpunkte seiner festgelegten Tour an. Bei der Ankunft liefert er zunächst die für das jeweilige Gebiet bestimmten Aufträge ab, indem er diese am *Hub* deponiert. Anschließend teilt er der Zentrale mit, welche Aufträge abgeliefert wurden, damit diese mit der zweiten Vermittlung vom Ziel-*Hub* zum Empfänger beginnt. Die Meldung erfolgt durch ein *Allocate*-Signal und wird somit als modifizierter Auftragseingang behandelt. Im letzten Schritt entnimmt der Lieferant alle Aufträge aus dem *Hub*, die zur Lieferung in ein anderes Schwerpunktgebiet bestimmt sind, und setzt seine Tour fort.

Der Lieferverkehr beginnt mit dem Start der Simulation und endet zu einem Zeitpunkt, zu dem keine relevanten Auftragsmengen mehr für den Transport über das *Hub*-System eingehen. Im Modell wird dies für 19:30 Uhr des simulierten Arbeitstags festgelegt. Die von den *Shuttles* zurückgelegten Gesamtstrecken gehen zusätzlich in die Berechnung des motorisierten Verkehrsaufwands ein.

6.3.3. Implementation in FAMOS

Die Implementation des Basismodells kann mit einigen Modifikationen direkt für das *Hub and Shuttle*-Modell übernommen werden. Transportaufträge (Klasse `TransportOrder`) werden um zusätzliche Attribute für den Start- und Ziel-*Hub* erweitert. Diese werden ähnlich wie Auftragspunkte durch Unterklassen `StartHubPoint` und `DestHubPoint` von `Point` realisiert. Auf diese Weise können Kuriere Fahrten zu Umschlagpunkten wie Auftragsorte in ihre Tour einplanen und müssen dort lediglich andere Handlungen zum Deponieren bzw. Entnehmen der Aufträge ausführen. Dazu wurde die Kurier-Klasse um die Effektor-Methoden `depositOrderAtStartHub()` und `redrawOrderFromDestHub()` ergänzt und der das Verhalten beschreibende Zustandsautomat um entsprechende Transitionen erweitert. Zudem ist bei jeder Vermittlung eines Auftrags anzuzeigen, zwischen welchen Auftragspunkten die Tour vermittelt wird (z.B. vom Abholpunkt zum Start-*Hub*).

Dies erfolgt durch ein zusätzliches Attribut der Klasse `TransportOrder`, welches beim Eingang des Auftrags von der Zentrale gesetzt und bei der Bearbeitung von den Kurieren interpretiert und bei Ablieferung des Auftrags verändert wird.

Vollständig neu zu implementieren ist dagegen die *Hub and Shuttle*-Infrastruktur. Für Umschlagpunkte dient eine Klasse `Hub`, welche zwei Warteschlangen zum Lagern ein- und ausgehender Aufträge besitzt. Durch Verwendung der Warteschlangen des FAMOS zugrundeliegenden Simulationsframeworks DESMO-J erscheinen Statistiken über die an den *Hubs* anfallenden Wartezeiten und Auftragszahlen automatisch im Ergebnisreport. Die Schwerpunktgebiete sind als Attributgebiete realisiert, denen als einziges Attribut der jeweilige *Hub* zugeordnet ist. Die im Geographischen Informationssystem festgelegten Gebiete werden über die entsprechende Import-Schnittstelle von FAMOS eingelesen und in Objekte der Klasse `AttributeArea` gewandelt. Um den Zugriff auf die Gebiete und ihre Hubs zu erleichtern, werden die Attributgebiete von einem `HubAreaManager` verwaltet. Diese Klasse stellt Methoden zur Verfügung, mit denen die Zentrale die Zuordnung der Aufträge zu den Hubs durchführen kann.

Lieferanten sind durch eine von `MobileAgent` abgeleitete Klasse `Shuttle` realisiert, deren Verhalten auf prozessorientierte Weise beschrieben wird. Der Lebenszyklus eines Lieferanten beginnt am ersten Hub der festgelegten Tour. In einer Endlos-Schleife entlädt der Lieferant zunächst alle für den aktuellen Hub bestimmten Aufträge und überprüft dann, ob sein Arbeitsende erreicht ist. Ist dies der Fall, wird die Schleife verlassen. Anderenfalls entnimmt der Lieferant alle Aufträge, die zum Weitertransport entlang seiner Tour bestimmt sind, belädt sein Fahrzeug und fährt zum nächsten *Hub*. Die für das Deponieren bzw. Entnehmen von Sendungen benötigte Zeit wird jeweils über ein `hold()` abgebildet und variiert mit der Anzahl der Sendungen. Für die Bewegung auf dem Verkehrsnetz-Graphen benutzt ein *Shuttle* ebenso wie die Kuriere ein Objekt der Klasse `OdoRoute`. Dieses berechnet jeweils die schnellste Route von einem Hub zum nächsten und zeichnet die zurückgelegten Strecken auf. Zur Datensammlung über die Lieferanten dient eine wiederum von `IndividualObserver` abgeleitete spezielle Klasse `ShuttleObserver`. Die über *Hubs* abgewickelten Aufträge werden gesondert in einem `HubOrderObserver` erfasst.

6.4. Modellvariante 2: Die Logistikstrategie Innen/Außen

Eine wohl nicht nur für Hamburg typische Verteilung des Auftragsaufkommens besteht darin, dass in der Innenstadt eine deutlich größere Anzahl an Aufträgen anfällt als in den äußeren Bereichen oder im Umland. Der Kilometer-Aufwand ist daher für Sendungen in oder aus dem Umland wesentlich höher als für innerstädtische Sendungen. Mit dem Logistikkonzept *Innen-Außen* sollen auch für das Umland Optimierungspotentiale erschlossen werden. Gleichzeitig weist dieses Konzept auch einen Weg zur Gewinnung neuer Kunden im Umland, die derzeit mittels des traditionellen Kurierprinzips aus Kostengründen nicht bedient werden.

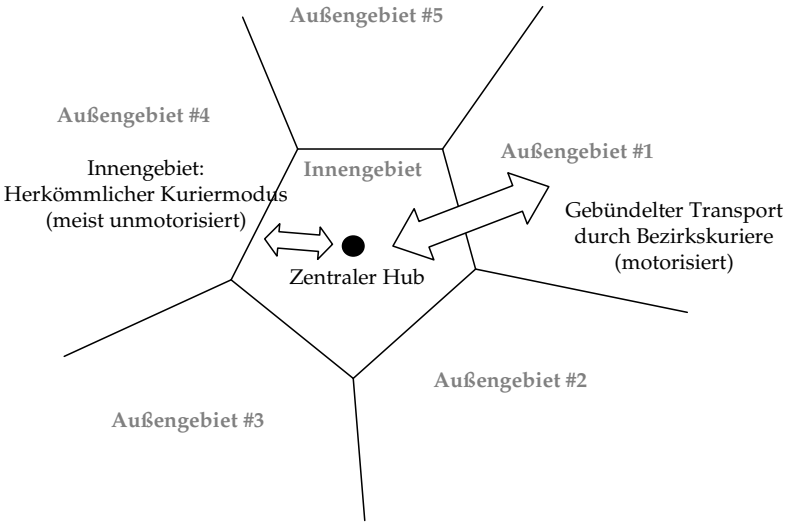

Abbildung 6.8.: Prinzip des alternativen Logistikkonzepts *Innen/Außen*.

6.4.1. Beschreibung der Strategie

Kern des Logistikkonzepts *Innen/Außen* ist eine Unterteilung des Stadtgebiets in einen Innen- und einen Außenbereich (Abbildung 6.8). Der Außenbereich wird weiter in wenige einzelne Bezirke aufgeteilt, für die ein Sammel- und Verteildienst eingeführt wird. Dazu wird ein zentraler innenstadtnaher Umschlagpunkt eingerichtet, an dem Sendungen umgeschlagen werden, die von außen nach innen bzw. von innen nach außen oder von einem Außenbezirk in einen anderen Außenbezirk zuzustellen sind.

Pro Außenbezirk gibt es einen oder mehrere Kuriere, die ausschließlich für diesen Bezirk zuständig sind. Sie sammeln alle Sendungen ein, die in ihrem Bezirk anfallen, und bringen sie zum Umschlagpunkt. Dort holen sie die aus den anderen Bereichen für ihren Bezirk bestimmten Sendungen ab und verteilen sie an die jeweiligen Empfänger. Aufträge, deren Quell- und Zielpunkte innerhalb eines Außenbezirks liegen, werden selbstverständlich ohne Umweg über den Umschlagpunkt direkt vom „Bezirkskurier“ ausgeführt.

Im Innenbereich, der sich durch ein hohes internes Auftragsaufkommen auszeichnet, werden die Sendungen wie bisher nach dem klassischen Kurierprinzip durch einzelne Kuriere per Fahrrad oder Auto transportiert. Diese Kuriere bringen außerdem Sendungen, die aus dem Innenbereich in einen der Außenbezirke gehen, zum Umschlagpunkt und übernehmen die Auslieferung der dort für den Innenbereich eingegangenen Sendungen.

Jeder Kurier des Kurierdienstes kann entscheiden, ob er als konventioneller Kurier im Innenbereich oder als Bezirkskurier in einem bestimmten Außenbezirk arbeiten möchte. Er kann auch während eines Tages – je nach Auftragslage – zwischen den Bereichen wechseln und zeitweilig als konventioneller oder als Bezirkskurier tätig sein. Fahrradkuriere werden allerdings als Bezirkskuriere nicht in Frage kommen, da die zurückzulegenden Distanzen in den Außenbereichen im Durchschnitt größer als im Innenbereich sind und

auf dem Fahrrad nur eine begrenzte Anzahl von Sendungen gleichzeitig befördert werden kann.

Im Zusammenhang mit diesem Logistikkonzept ist es sinnvoll, zumindest für Aufträge aus dem bzw. in den Außenbereich von der klassischen Kurierdienstleistung abzuweichen und eine Auslieferung innerhalb eines bestimmten Zeitraums (Zeitfenster) anzubieten, entweder in Form von individuellen Laufzeitgarantien (z. B. Zustellung innerhalb von vier Stunden) oder indem der Tag in feste Intervalle eingeteilt wird (z. B. 6–9, 9–12, 12–15 Uhr usw.). Auf diese Weise können die Bezirkskuriere ihre Touren besser planen und müssen den Umschlagpunkt nur zu bestimmten Zeitpunkten anfahren.

Dem Konzept *Innen/Außen* liegt die Annahme zugrunde, dass die Auftragsströme weitgehend radial gerichtet sind, so dass mit Einführung eines zentralen Umschlagpunktes erhebliche Kosten und Kilometer eingespart werden können. Zusammen mit den Bündelungspotentialen, die sich aus der zeitlichen und räumlichen Entzerrung bei der Bedienung der Außenbezirke ergeben, kann auch von diesem Logistikkonzept eine Reduzierung der motorisierten Fahrleistung erwartet werden.

Entscheidende Einflussfaktoren für dieses Logistikkonzept sind Größe und Zuschnitt der Außenbezirke, die Lage des Umschlagpunktes und die Ausgestaltung der Zeitfenster. Die Simulation kann auch hier bei der Optimierung der Faktoren helfen.

6.4.2. Beschreibung des Modells

Aufbauend auf dem Basismodell wird im *Innen/Außen*-Modell ein zusätzlicher Agententyp *Bezirkskurier* eingeführt. Jeder Bezirkskurier ist genau einem Außenbezirk zugeordnet und führt ausschließlich Aufträge aus, deren Abhol- oder Lieferpunkt in diesem Bezirk liegt. Dabei werden Aufträge, die vollständig innerhalb des Außenbezirks liegen, direkt ausgeliefert, während die Abwicklung von Aufträgen zwischen Außen- und Innenbereich bzw. zwischen zwei verschiedenen Außenbezirken über den zentralen *Hub* erfolgt.

Nach Festlegung einer geeigneten Aufteilung des Stadtgebiets in Innen- und Außenbezirke, die wiederum aufgrund einer manuellen räumlichen Analyse der vorhandenen Auftragsdaten durchgeführt wurde, besteht das erste Modellierungsproblem in der Gebietszuordnung der Kuriere. Da diese zugunsten größerer Flexibilität nicht statisch (d.h. fest in den Eingangsdaten codiert) erfolgen soll, wird sie anhand des folgenden einfachen Verfahrens bestimmt: Für jeden Außenbezirk wird auf Grundlage des lokalen Auftragsaufkommens eine maximale Kapazität dort beschäftigter Kuriere bestimmt, z. B. sieben Kuriere für Bezirke mit stärkerem Auftragsaufkommen, vier Kuriere für weniger frequentierte Bezirke (vgl. Abbildung 6.9). Bei Arbeitsbeginn wird ein motorisierter Kurier demjenigen Außenbezirk zugeordnet, in dem momentan am wenigsten Kuriere beschäftigt sind und dessen Kapazität nicht überschritten ist. Ist die Kapazität aller Außenbezirke überschritten, wird der Kurier dem Innenbezirk zugeordnet. Biker werden stets dem Innenbezirk zugewiesen, da ihr Verkehrsmittel für die in den Außenbezirken erwünschte hohe Auftragsbündelung ungeeignet ist.

Die Gebietszuteilung und Vermittlung der Aufträge obliegt wiederum der Zentrale. Bei Eingang eines Auftrags bestimmt diese zunächst dessen Start- und Zielgebiet. Aufträge mit Abhol- und Lieferpunkt im Innengebiet werden unter Berücksichtigung der bekannten Prioritäten (Freistellung und Verkehrsmittelwunsch) vor den Kurieren des Innenge-

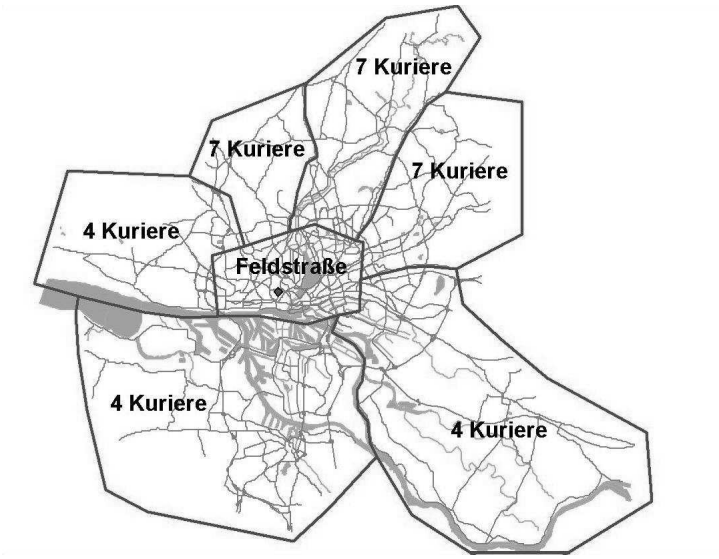

Abbildung 6.9.: Aufteilung des Hamburger Stadtgebiets in ein Innengebiet mit zentralem Umschlagpunkt und sechs Außenbezirke im Modell *Innen/Außen*. Für jeden Außenbezirk ist die maximale Anzahl der Bezirkskuriere angegeben.

biets verlesen, die sich anschließend wie im Basismodell bewerben. Ein Auftrag für einen Außenbezirk wird dagegen stets demjenigen Bezirkskurier zugeordnet, der den Auftrag mit minimalem Umweg bezüglich seiner bisherigen Tour ausliefern kann. Um eine gleichmäßige Auslastung der Bezirkskuriere zu gewährleisten, werden auch hier Freistellungen bevorzugt. Dieses Verfahren bildet eine sorgfältigere Absprache der (wenigen) Bezirkskuriere eines Außengebiets bei der Tourenplanung ab, da die Verwendung des herkömmlichen Vermittlungsverfahrens für die Außenbezirke in ersten Simulationsläufen nicht zu befriedigenden Ergebnissen führte.

Das Verhalten der Bezirkskuriere weicht im Gegensatz zu den im Innenbezirk eingesetzten Kurieren vom Basismodell ab. Aufträge, die über den zentralen *Hub* abzuwickeln sind, werden vom Bezirkskurier nicht unmittelbar bei Eingang ausgeführt. Statt dessen werden sie während dreißigminütiger Intervalle gesammelt. Erst nach Ablauf eines solchen Zeitfensters werden die angesammelten Aufträge in die momentane Tour des Kuriers eingeplant und auf diese Weise die Bündelungschancen für die Fahrt zum zentralen *Hub* erhöht. Bei Ankunft am *Hub* werden alle dort abgelieferten Aufträge wie im *Hub and Shuttle*-Modell der Zentrale durch ein *Allocate*-Signal zur zweiten Vermittlung gemeldet. Aufträge, die vollständig innerhalb eines Außenbezirks liegen, werden dagegen von den Bezirkskurieren nicht gesammelt, sondern unmittelbar in die Tour eingeplant.

6.4.3. Implementation in FAMOS

Die Bezirkskuriere des *Innen/Außen*-Modells unterscheiden sich implementationstechnisch von herkömmlichen Kurieren durch die Verwaltung zweier verschiedener Touren. Neben der aktuellen Tour der anzufahrenden Auftragspunkte verfügen sie über eine weitere

Auftragsliste, welche die Abhol- und Lieferpunkte der innerhalb eines dreißigminütigen Zeitfensters gesammelten Hub-Aufträge beinhaltet. Die Zusammenführung beider Touren erfolgt einerseits bei Einplanung der gesammelten Aufträge in die aktuelle Tour nach Ablauf eines Zeitfensters. Andererseits werden die Touren temporär vereinigt, wenn bei der Bewertung eines Hub-Auftrags der zu dessen Ausführung benötigte Umweg zu bestimmen ist.

6.5. Ausgewählte Ergebnisse

Um die Beschreibung der Simulationsstudie zu Stadtkurierdiensten abzurunden, sollen im folgenden die wichtigsten Ergebnisse des Kurierdienst-Projekts kurz vorgestellt werden. Dies betrifft vor allem die Resultate der mit den zuvor beschriebenen Modellvarianten durchgeführten Simulationsexperimente. Aber auch das Problem der Validierung agentenbasierter Modelle wird thematisiert. Für eine detailliertere Darstellung sei auf den Abschlussbericht des Projekts (Knaak et al. 2004) verwiesen.

6.5.1. Modellvalidierung und -kalibrierung

Bevor eine vergleichende Bewertung der alternativen Logistikstrategien vorgenommen werden kann, muss zunächst das Basismodell sorgfältig validiert werden. Dabei sind insbesondere die zahlreichen Modellparameter anzupassen. Während einige Größen wie z. B. die garantierte Lieferzeit oder die kritische Zeitspanne bis zum Beginn der Druckvermittlung auf Grundlage der Systemanalyse abschätzbar sind, stellen die Parameter der Auftragsbewertung Modellartefakte dar, deren Werte in Testläufen kalibriert werden müssen.

Da die dem Projekt zur Verfügung gestellten Auftragsprofile des Kurierdienstes A detaillierte Informationen enthalten, welcher Auftrag von welchem Kurier ausgeführt wurde, konnte bei der Validierung und Kalibrierung des Modells wie folgt verfahren werden:

1. Im ersten Schritt wurde zu jedem Datensatz ein Simulationslauf durchgeführt, in dem jeder Kurier genau die Aufträge erhielt, die er laut Auftragsdaten tatsächlich ausgeführt hatte (*empirische Auftragszuordnung*). Die Tourenplanung und Auftragsauslieferung wurde dagegen wie im herkömmlichen Basismodell simuliert.

2. Die mit dem Modell der empirischen Auftragszuordnung erzielten Resultate wurden einem Domänenexperten vorgelegt und nach Durchführung einiger Modifikationen und Fehlerbereinigungen am Modell (z.B. Einführung straßentyp-abhängiger Geschwindigkeiten) als akzeptabel eingestuft. Somit konnte die Modellierung der Tourenplanung und Auftragsausführung als valide hinsichtlich der untersuchten Größen (Auftragslieferzeiten, Fahrleistungen, Auslastungen, Umsätze) angesehen werden.

3. Die Resultate der Simulationen mit empirischer Auftragszuordnung dienten anschließend als Referenz für die Kalibrierung des Modells der Auftragsvermittlung und -bewertung. Hierbei erfolgte insbesondere eine Anpassung der verkehrsmittelabhängigen Faktoren für die Gewichtung der bisherigen Auslastung eines Kuriers.

Modellparameter	Wert	Beschreibung
t_{crit}	20.0 [min]	Dauer bis zum Beginn der Druckvermittlung
t_{anc}	60.0 [sec]	Zeit zwischen Auftragsverlesungen
t_{active}	0.0 [sec]	Dauer jeder Aktivität der Zentrale (vernachlässigt)
$t_{handover}$	5.0 [min]	Dauer der Sendungsübergabe beim Kunden
T_{max}	2.5 [h]	Garantierte maximale Lieferzeit
w_{car}	0.02	Gewichtung der Auslastung bei motorisierten Kurieren
w_{bike}	0.035	Gewichtung der Auslastung bei Bikern
δ	50.0 [sec]	Faktor zur Berechnung der Reaktionszeit von Kurieren
q	1.0	Faktor zur Berechnung des Auftragsbedarfs

Tabelle 6.1.: Parameterwerte der Simulationen

Diese dient als Schwellwert bei der Auftragsbewertung und hat somit einen großen Einfluss auf das Verhalten eines Kuriers und damit des gesamten Modells.

Die Parameterkalibrierung wurde ausgehend von einigen aus der Vorstudie (Hilty et al. 1998, S. 291ff) bekannten Initialwerten manuell durchgeführt und erwies sich als schwierig, da die Gewichtungsfaktoren starke und vorab schwer einschätzbare Auswirkungen auf die Simulationsergebnisse zeigten. Annehmbare Resultate konnten mit den in Tabelle 6.1 aufgelisteten Werten erzielt werden, wobei insbesondere die stärkere Gewichtung der Auslastung bei Fahrradkurieren aufgrund früherer physikalischer Erschöpfung plausibel erscheint.

Tabelle 6.2 zeigt, dass die über vier Simulationsläufe mit jeweils unterschiedlichen Datensätzen gemittelten Ergebnisse hinsichtlich der Mittelwerte und Standardabweichungen aller betrachteten Ausgangsgrößen recht gut mit den Resultaten der empirischen Auftragszuordnung übereinstimmen. Eine detaillierte Untersuchung der Verteilung von Auftragslieferzeiten, Fahrleistungen und Umsätzen innerhalb einzelner Simulationsläufe lässt zwar stärkere Abweichungen zu Tage treten, doch lässt sich dies zurückführen auf die mit Datenverlust einhergehende Geocodierung der Auftragsdaten. Da ca. 15 % des Auftragsaufkommens eines Tages aufgrund fehlender Angaben nicht als Eingabedaten für das Modell zur Verfügung stand, kann die empirische Auftragszuordnung nicht exakt die reale Situation widerspiegeln. Die Ergebnisse der Kalibrierungsläufe wurden daher trotz der Abweichungen als im Rahmen der Modellierungsgenauigkeit akzeptabel eingestuft.

Ergänzend zur manuellen Kalibrierung wurden automatische Kalibrierungsläufe mit dem simulationsbasierten Optimierungssystem DISMO (Gehlsen und Page 2001; Gehlsen 2004) durchgeführt. Die Grundidee der automatischen Kalibrierung besteht darin, den Vorgang als simulationsbasiertes Optimierungsproblem aufzufassen, dessen Ziel die Minimierung von Abweichungen an Modell und Originalsystem gemessener Ausgangsgrößen ist.

Bei der automatischen Kalibrierung des Kuriermodells wurden der Mittelwert und die Standardabweichung der Auftragslieferzeiten sowie die Summe und die Standardabweichung der individuellen Fahrleistungen (für Biker und motorisierte Kuriere getrennt) berücksichtigt. Die Abweichungen dieser Größen von den Referenzergebnissen wurden in

	Fahrstrecke [km]	Umsatz [€/km]	Auslastung [%]	Auftragslieferzeit [min]
Emp. Auftragszuordnung				46.7 (32.5)
Fahrradkuriere	47.7 (25.4)	1.80 (0.43)	68.0 (17.0)	
Motorisierte Kuriere	91.4 (44.6)	1.12 (0.45)	64.4 (18.3)	
Simulierte Vermittlung				47.9 (28.2)
Fahrradkuriere	46.1 (16.9)	1.65 (0.34)	65.4 (6.1)	
Motorisierte Kuriere	91.3 (32.1)	1.15 (0.28)	59.1 (14.7)	

Tabelle 6.2.: Ergebnisse der Kalibrierung des Basismodells. Dargestellt sind über die Kurierflotte bzw. das Auftragsaufkommen gemittelte Werte verschiedener relevanter Ausgangsgrößen (Standardabweichungen in Klammern).

einer quadratischen Fehlerfunktion verrechnet, welche als multikriterielle Zielfunktion der simulationsbasierten Optimierung diente. Optimiert wurden wiederum die Parameter q, w_{bike} und w_{car}, deren Werte das DISMO-System während der Kalibrierung automatisch mit Hilfe eines genetischen Algorithmus variierte.

In jeder Generation erzeugt der genetische Algorithmus verschiedene Parameterkonfigurationen, die in Simulationsläufen evaluiert werden müssen. Da diese Simulationsläufe unabhängig sind, ist es im DISMO-System möglich, sie parallel in einem Rechnernetz auszuführen. Die Simulationen der beschriebenen Kalibrierungsstudie wurden innerhalb einer Nacht auf ca. 20 SUN-Workstations des Informatik-Rechenzentrums ausgeführt. Die automatisierte Kalibrierung konnte die Ergebnisse der (wesentlich zeitaufwendigeren) manuellen Kalibrierung bestätigen, wobei sich die Betrachtung allerdings bislang auf einen Datensatz beschränkt hat. Der Ansatz und die Ergebnisse der automatischen Kalibrierung werden detaillierter in (Bachmann et al. 2004) beschrieben.

6.5.2. Vergleich alternativer Logistikkonzepte

Den Schwerpunkt der im Rahmen des Forschungsprojekts durchgeführten Simulationsstudie bildet ein Vergleich der bestehenden Organisationsform von Stadtkurierdiensten mit den oben beschriebenen Modellen alternativer Logistikkonzepte. Zu diesem Zweck wurden Simulationen der drei Modelle *Status quo*, *Hub and Shuttle* und *Innen/Außen* mit den fünf von Kurierdienst A bereitgestellten Datensätzen durchgeführt. Daten von Kurierdienst B wurden zunächst nicht verwendet, da dieses Unternehmen über ein zu geringes Auftragsvolumen verfügt, als dass auf erhöhter Auftragsbündelung beruhende Logistikkonzepte sinnvoll einsetzbar wären.

Während die Einstellungen der allen Modellen gemeinsamen Parameter unverändert aus den Kalibrierungsläufen übernommen werden können (vgl. Tabelle 6.1), verfügen die alternativen Logistikkonzepte über zusätzliche Freiheitsgrade. In beiden Modellen ist zunächst eine geeignete Gebietsaufteilung vorzunehmen. Im *Hub and Shuttle*-Modell muss zusätzlich die Organisation des Lieferverkehrs, im *Innen/Außen*-Modell die Zahl der in den Außengebieten eingesetzten Bezirkskuriere festgelegt werden. Die Dimensionierung dieser Parameter wurde manuell anhand der Auftragsdaten durchgeführt und in einigen Kalibrierungsläufen verfeinert. Für die Simulationen der *Hub and Shuttle*-Strategie wurde

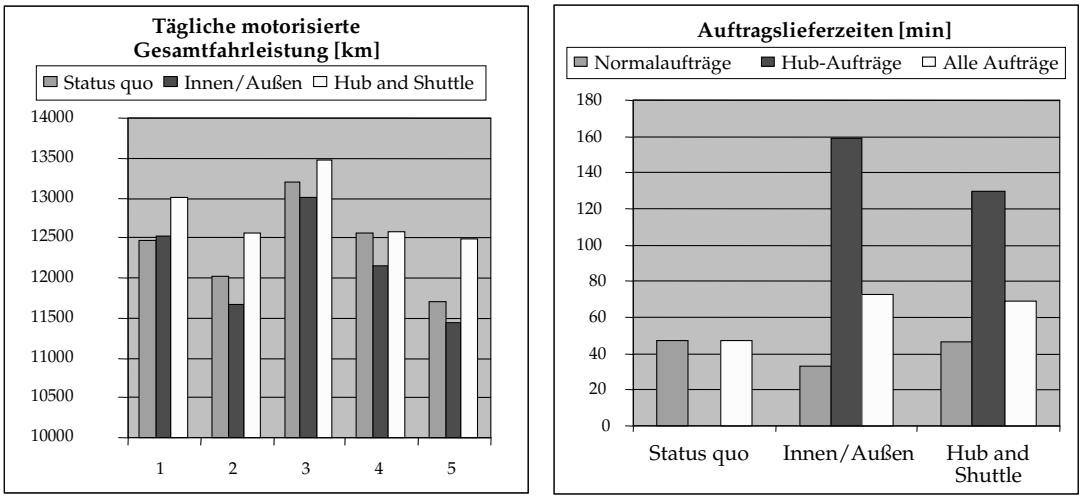

Abbildung 6.10.: Vergleich der täglichen motorisierten Gesamtfahrleistung (links) und der mittleren Auftragslieferzeit (rechts) in Simulationen des Status quo und der alternativen Logistikstrategien

eine Aufteilung des inneren Stadtgebiets in drei Schwerpunktgebiete mit bekannt hohem Auftragsaufkommen gewählt und deren *Hubs* an verkehrsgünstigen Standorten positioniert (vgl. Abbildung 6.7, rechts). Zwischen den Hubs verkehren zwei *Shuttle*-Fahrzeuge in gegenläufigem Linienverkehr, welche Aufträge jeweils nur zum Folgehub transportieren. Für das *Innen/Außen*-Modell wurde die in Abbildung 6.9 gezeigte Aufteilung in ein Innengebiet und sechs umliegende Außengebiete vorgenommen, welche die räumliche Verteilung des Auftragsaufkommens recht angemessen widerspiegelt. Den drei aufkommensstärkeren Außengebieten im Norden und Osten der Stadt wurden jeweils sieben, den anderen je vier Bezirkskuriere zugeordnet.

Abbildung 6.10 zeigt die auf die Zielgrößen *ökologische Qualität* und *ökonomische Qualität* (vgl. Abschnitt 6.1.2) bezogenen Resultate eines Vergleichs der drei Modellvarianten in Simulationen mit den fünf Datensätzen des Kurierdienstes A. Im linken Diagramm ist die täglich anfallende motorisierte Fahrleistung aus den fünf Simulationen jeder Modellvariante dargestellt. Dabei ist im *Innen/Außen*-Modell im Mittel eine Senkung dieser ökologisch relevanten Kenngröße gegenüber dem Basismodell (Status quo) zu beobachten. Das Ausmaß der Einsparungen erscheint allerdings bei einer mittleren Ersparnis von täglich 238,17 km (bzw. jährlich 59541,5 km bei angenommenen 250 Arbeitstagen) für die gesamte Kurierflotte recht gering und bleibt hinter den anfänglichen Erwartungen zurück. Im *Hub and Shuttle*-Modell tritt gegenüber dem Basismodell sogar eine Erhöhung der motorisierten Fahrleistung auf, welche sowohl durch die von den beiden *Shuttles* zusätzlich aufgewendete Fahrleistung als auch durch geringfügig verlängerte individuelle Fahrten der Kuriere verursacht wird.

Bezüglich der im rechten Diagramm von Abbildung 6.10 dargestellten, über alle Simulationsläufe gemittelten durchschnittlichen Auftragslieferzeit weisen beide alternative Logistikkonzepte eine ähnliche Charakteristik auf. Die mittlere Lieferzeit der über einen

Hub abgewickelten (und folglich zweimal vermittelten) Aufträge ist gegenüber der mittleren Lieferzeit im Basismodell deutlich erhöht. Dies betrifft allerdings stets nur einen mit 20 - 30 % relativ geringen Anteil des gesamten Auftragsaufkommens. Darüber hinaus könnten nach Aussage der beteiligten Kurierdienste längere Zeitfenster für bestimmte Aufträge durchaus akzeptiert werden, so dass das in den Simulationen erzielte Resultat nicht unbedingt eine Einschränkung für die Konkurrenzfähigkeit eines Kurierdienstes bedeuten muss. Für den höheren Anteil der nicht über einen *Hub* abgewickelten Aufträge ist im *Innen/Außen*-Modell sogar eine mittlere Senkung der Lieferzeit um mehr als zehn Minuten gegenüber dem Status quo zu beobachten. Diese wird wahrscheinlich durch deutliche Verkürzungen der Auftragsdistanzen verursacht, welche von den Kurieren des Innengebiets zurückzulegen sind. Die Verlagerung langer Auftragsdistanzen auf die motorisierten Bezirkskuriere der Außengebiete würde eine generelle Erhöhung des Bikeranteils im Innengebiet ermöglichen, falls Maße und Gewichte der anfallenden Sendungen den Fahrradtransport erlauben.[6]

Hinsichtlich der als soziales Bewertungskriterium betrachteten Umsätze der Kuriere zeigt sich im *Innen/Außen*-Modell gegenüber dem Status quo eine wesentliche Steigerung bei den motorisierten Kurieren bei gleichzeitigem Anstieg der Standardabweichung (vgl. Abbildung 6.11). Dies ist wahrscheinlich einerseits auf Verdienstunterschiede zwischen Bezirkskurieren und „herkömmlichen“ motorisierten Kurieren im Innengebiet und andererseits auf die stark variierende Auslastung der Kuriere in den verschiedenen Außengebieten zurückzuführen. Im *Hub and Shuttle*-Modell steigt sowohl der Umsatz der motorisierten Kuriere als auch der Umsatz der Biker. Dies ist neben besseren Bündelungsmöglichkeiten eventuell auf eine erhöhte Anzahl von Kurzaufträgen zurückzuführen, welche aufgrund der Anfahrtpauschale finanziell vorteilhaft erscheinen.

Angesichts des relativ geringen ökologischen Optimierungspotentials beider alternativer Logistikstrategien wurden weitere Experimente durchgeführt, in denen mit synthetischen Datensätzen ein erhöhtes Auftragsvolumen simuliert wurde. Dazu wurden zum einen die Datensätze zweier Tage zu einem zusammengefasst, um das Auftragsaufkommen näherungsweise zu verdoppeln, und zum anderen Datensätze der Kurierdienste A und B kombiniert. Die Ergebnisse der Simulationsläufe zeigen für das *Innen/Außen*-Modell eine wesentlich stärkere Senkung der motorisierten Gesamtfahrleistung gegenüber dem Basismodell, während das *Hub and Shuttle*-Modell wiederum keinen ökologischen Vorteil besitzt. Dies lässt den Schluss zu, dass das ökologische Optimierungspotential der *Innen/Außen*-Strategie erst bei deutlich höheren Auftragszahlen als denen der betrachteten Kurierdienste in vollem Maße nutzbar wird. Eine denkbare Umsetzungsmöglichkeit bestünde in der Bearbeitung von Aufträgen mehrerer Kurierdienste durch die Bezirkskuriere. Allerdings bedürfen die Resultate weiterer statistischer Absicherung, wobei insbesondere die nichtlineare Reaktion des Modells (mehr als verzehnfachte Senkung der täglichen motorisierten Gesamtfahrleistung bei Verdoppelung des Auftragsvolumens) genauer zu untersuchen ist. Auch könnte die Gültigkeit des Modells bei erhöhtem Auftragsvolumen eventuell nicht ausreichen, da bestimmte Einschränkungen realer Kurierfahrer (z.B. Arbeitspausen) nur ungenügend berücksichtigt werden. Generell verdeutlichen je-

[6] Die bereitgestellten Datensätze beinhalten keinerlei Informationen über Maße und Gewichte der Sendungen, so dass zu diesem Punkt keine Aussage möglich ist.

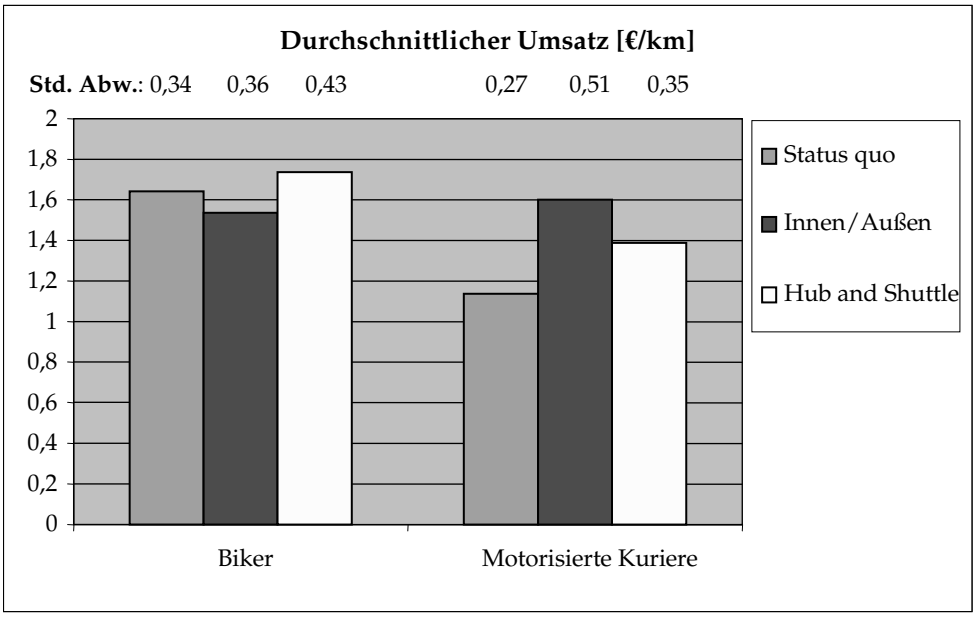

Abbildung 6.11.: Durchschnittlicher Kurier-Umsatz pro Kilometer in Simulationen mit den Datensätzen 1 - 5. Die Standardabweichung ist oberhalb der Balken angegeben.

doch alle Simulationsergebnisse eine deutliche Überlegenheit der *Innen/Außen*-Strategie gegenüber dem *Hub and Shuttle*-Modell. Diese ist wahrscheinlich in erster Linie auf bessere Berücksichtigung der charakteristischen Auftragsprofile mit stark ausgelastetem Innengebiet und geringer ausgelasteten Außenbereichen zurückzuführen und kann als wesentliches Ergebnis der Untersuchung angesehen werden.

6.5.3. Umsetzungspotential

Die Simulationen dieser explorativen Studie haben gezeigt, dass bei Stadtkurierdiensten erhebliche Effizienzverbesserungen hinsichtlich ökologischer Parameter bei der Auftragsabwicklung zu erzielen sind, ohne dass die betriebswirtschaftliche Situation der Kuriere oder der Vermittlungszentralen beeinträchtigt wird. Ergebnis der Untersuchung ist allerdings auch, dass nicht alle Modelle in jeder Stadt und in jeder Konstellation zu einem derartigen Erfolg führen. Die durchgeführten Simulationsstudien und die Konstruktion der alternativen Organisationsmodelle zeigen, wie außerordentlich voraussetzungsvoll die Realisierung innovativer Modelle ist.

Eine wichtige Erfolgsbedingung ist sicherlich ein hohes Sendungsaufkommen. Zusammen mit der geographischen Struktur der Aufträge (Verteilung von Quell- und Zielpunkten) verweisen die Simulationsergebnisse darauf, dass alternative Organisationsstrukturen nur für größere Kurierunternehmen oder den operativen Zusammenschluss mehrerer kleinerer Unternehmen erfolgreich sind. Damit hängt eng die Einsicht zusammen, dass derartige Bedingungen vor allem in größeren Städten und Metropolregionen zu finden sein dürften.

Ein in den bisherigen Ausführungen vernachlässigter, jedoch entscheidender Aspekt der praktischen Umsetzung alternativer Logistikkonzepte sind die Unterhaltskosten der erforderlichen Infrastruktur. Die Kosten eines bemannten *Hubs* müssen aufgrund von Erfahrungswerten mit etwa 3500 € monatlich veranschlagt werden. Zusätzlich ist beim *Hub and Shuttle*-Konzept der Transport zwischen den *Hubs* zu finanzieren, wobei pro Fahrzeug monatliche Kosten von ca. 1000 € (inkl. Kraftstoff) sowie pro Fahrer ca. 2000 € anfallen. Bereits für das einfachste denkbare System mit zwei *Hubs* und einem *Shuttle* ergeben sich somit monatliche Gesamtkosten von rund 10.000 €. Bei durchschnittlich 21 Tagen Betriebszeit zu jeweils zehn Stunden sind dies etwa 48 € Kosten pro Stunde.

Bei einem täglichen Aufkommen von 1000 Sendungen, von denen die Hälfte über den *Hub* abgewickelt wird, bedeutet dies, dass jede Sendung mit etwa 1,05 € zusätzlicher Kosten belastet wird, den die Kurierfahrer von ihrem Umsatz abgeben müssen. Der Mindestauftragswert für eine Sendung, die über die *Hubs* abgewickelt wird, müsste damit bei etwa 13 € liegen. Dann kann sowohl der anliefernde aus auch der ausliefernde Kurier den Mindestumsatz von 6 € pro Auftrag (Anfahrtpauschale plus drei km minimale Strecke) erzielen. Der Mindestauftragswert kann dann geringer sein, wenn die Kuriere bei der An- und Ablieferung bündeln können. Für die *Innen/Außen*-Strategie fallen entsprechend niedrigere Kosten an, da lediglich ein einzelner bemannter *Hub* zu finanzieren ist.

6.6. Diskussion

Modellierung und Simulation ist eine angemessene – und oft die einzige – Methode, um die Auswirkungen möglicher neuer Organisationsformen für Unternehmen im voraus abzuschätzen. Für die Modellierung von Stadtkurierdiensten erscheint die Simulation auf Basis eines agentenbasierten Modells aus folgenden Gründen besonders geeignet:

1. Kurierdienste verfügen zumindest bei Anwendung einer Vermittlungsstrategie über ein geringes Maß an zentraler Kontrolle. Kuriere handeln weitgehend autonom, indem sie eigenständig über die Annahme von Aufträgen entscheiden und ihre Touren planen. Es erscheint daher besonders naheliegend, die Vorgänge in einem Kurierdienst aus Sicht der beteiligten Akteure (Kuriere und Funker) zu modellieren.

2. Die Kuriere bewegen sich in einer räumlichen Umwelt, dem Straßennetz. Dabei fahren sie eigenständig geplante Routen, die während der Fahrt gegebenenfalls aufgrund neuer Informationen (z.B. Übernahme eines Auftrags) angepasst werden müssen. Diese Balance reaktiven und pro-aktiven Verhaltens ist ein typisches Merkmal von Agenten.

3. Die Bewegung der Kuriere auf dem städtischen Verkehrsnetz – und damit das Verkehrsnetz selbst – ist explizit im Modell abzubilden, da dies das Verhalten der Kuriere maßgeblich beeinflusst. Ein Kurier entscheidet beispielsweise abhängig von seiner aktuellen Position, geplanten Route und Auftragslage über die Annahme oder Ablehnung eines neuen Auftrags.

4. Das in Kurierdiensten eingesetzte Vermittlungsprotokoll stellt aus theoretischer Sicht

eine Variante des aus der verteilten künstlichen Intelligenz bekannten Kontraktnetz-Protokolls (vgl. Abschnitt 2.3.3) zur verteilten Allokation von Aufgaben dar.

6.6.1. Eignung der agentenorientierten Weltsicht

Das System „Kurierdienst“ ist inhärent ereignisgesteuert: Die Ankunft eines Auftrags in der Zentrale löst den Vermittlungsvorgang aus, dessen erfolgreiches Ende – Zuteilung des Auftrags an einen Kurier – wiederum den Liefervorgang anstösst, welcher schließlich mit der Ablieferung der Sendung beim Empfänger beendet wird. Weitere Ereignisse sind Arbeitsbeginn und Arbeitsende eines Kuriers, welche das Betreten bzw. Verlassen des Systems durch einen Agenten markieren. Die im Rahmen dieser Arbeit eingeführte agentenorientierte Weltsicht, d. h. die agentenbasierte Simulation mit ereignisgesteuertem Zeitfortschritt (vgl. Abschnitt 3.2.5), ermöglicht eine natürliche Abbildung dieses Systems.

Für die in diesem Kapitel beschriebene Simulationsstudie von Stadtkurierdiensten lagen empirischen Daten vor mit exakten Zeitpunkten für externe Ereignisse, d. h. diejenigen Ereignisse, welche die Systemgrenzen betreffen: Ankunft von Aufträgen sowie Arbeitsbeginn und Arbeitsende von Kurieren. Die Zeitpunkte aller anderen Ereignisse ergeben sich davon abhängig aus der Simulation des Systems. Die Verwendung von Ereignissteuerung erlaubt hier die Beibehaltung der von den empirischen Daten vorgegebenen Genauigkeit. Wenn die Dauer von Aktivitäten ebenfalls exakt auf Simulationszeit abgebildet wird, lässt sich beispielsweise die Verweilzeit eines Auftrags im System präzise bestimmen als das Intervall zwischen den Ereignissen Auftragseingang und Ablieferung der Sendung beim Empfänger.

Voraussetzung hierfür ist, dass die Dauer der relevanten Aktivitäten genau genug bestimmt werden kann. Im vorliegenden Kurierdienstmodell wird von der Annahme ausgegangen, dass gleichförmige, zustandsunabhängige Aktivitäten wie die auftragsbezogenen Handlungen der Zentrale (Annehmen eines neuen Auftrags, Verlesen eines Auftrags über Funk, Zuteilen eines Auftrags an einen Kurier) im Durchschnitt immer die gleiche Zeit in Anspruch nehmen, so dass sie mit einer konstanten Dauer versehen wurden. Gleiches gilt für das Abholen bzw. Abliefern einer Sendung durch einen Kurier.

Im Gegensatz dazu sind die Fahrzeiten der Kuriere sowie deren Reaktionszeit auf Auftragsangebote vom aktuellen Zustand des Systems abhängig und müssen daher während der Simulation bestimmt werden. Die Berechnung der Fahrzeiten kann an das verwendete Framework FAMOS delegiert werden, da sie nicht modellspezifisch ist und FAMOS explizit die Bewegung im Raum unterstützt. Wie schnell ein Kurier auf ein Auftragsangebot reagiert, wird im Modell durch ein Auftragsinteresse approximiert, welches sich aus Auftragsnutzen, Auftragsbedarf und individueller Auslastung des Kuriers zusammensetzt. Das Auftragsinteresse wird umgekehrt proportional zur Reaktionszeit des Kuriers angenommen, d. h. je höher das Interesse, desto geringer die Reaktionszeit.

Wie in Kapitel 2.1 bereits dargelegt, bewirkt die Ereignissteuerung, dass jedes Ereignis zum korrekten Ereigniszeitpunkt eintritt und nur die mit ihm assoziierten Agenten betrifft. Auf diese Weise besteht das Problem der Abarbeitungsreihenfolge nur bei zeitgleichen Ereignissen. Diese sind in der Regel rar – die Ausnahme im Kurierdienstmodell bildet das vom Vermittlungsprotokoll diktierte Broadcasting eines Auftrages über Funk an eine Gruppe von Kurieren. Diese erhalten alle gleichzeitig die Nachricht und begin-

nen mit der Bewertung des Auftrags. Für den Fall, dass mehrere Kuriere einen Auftrag gleich bewerten und damit identische Reaktionszeiten aufweisen, muss eine Konfliktlösung gefunden werden. Das Kurierdienstmodell greift hier auf die zufällige Abarbeitung zeitgleicher Ereignisse zurück, welche mit der Erweiterung von DESMO-J zur Verfügung steht (vgl. Abschnitt 5.1.2.1).

Während die Ereignissteuerung wie gezeigt auf den system-immanenten Ereignissen aufbaut, um die Simulation voranzutreiben, ist eine zeitgesteuerte Umsetzung weniger direkt und mit größeren Kosten bezüglich Rechenaufwand verbunden. Letzteres ist u. a. darin begründet, dass in der Ereignissteuerung die Abstände zwischen zwei Ereignissen flexibel und damit genau an die Situation angepasst sind, wohingegen sich die Zeitsteuerung durch ein festes Intervall Δt auszeichnet. Um die gleiche Genauigkeit zu erreichen, müsste dieser Zeitschritt so klein gewählt werden wie der minimale Abstand zwischen zwei nicht-gleichzeitigen, unabhängigen Ereignissen. Da dieser nicht im voraus bekannt ist, kann der Wert für Δt nur abgeschätzt werden, indem beispielsweise aus den vorhandenen empirischen Auftragsdaten das kleinste Intervall zwischen nicht-gleichzeitigen Auftragseingängen ermittelt wird. Die Dauer einer Aktivität wird in jedem Fall zu einem Vielfachen von Δt vergröbert.

Desweiteren ist in jedem Zeitschritt darauf zu achten, die korrekte Reihenfolge bei der Aktualisierung der Agenten und Umgebung einzuhalten, um Simulationsartefakte zu vermeiden. Dies betrifft zum einen die Abfolge der Aktualisierung der unterschiedlichen Komponenten Umgebung (Auftrags- und Kurierquellen), Zentrale und Kurierflotte, welche im Modell explizit festgelegt werden muss, während sie sich bei der Ereignissteuerung automatisch ergibt. Zum anderen ist die Reihenfolge bei der Aktualisierung der einzelnen Kuriere innerhalb der Kurierflotte von Bedeutung, falls identische Reaktionszeiten auftreten. Hier kann jedoch dieselbe Strategie – zufällige Abarbeitung der Kurierliste – zum Einsatz kommen wie bei der Ereignissteuerung.

6.6.2. Einsatz von FAMOS

Die Kurierdienststudie hat sich als hervorragend geeignet erwiesen, das Framework FAMOS unter realen Anforderungen zu evaluieren. Seine Schwerpunkte wie Ereignissteuerung (siehe Abschnitt 6.6.1), graphbasiertes Raummodell, besondere Unterstützung der Bewegung im Raum und Verhaltensmodellierung mit Zustandsdiagrammen, welche FAMOS gegenüber anderen Softwarepaketen für agentenbasierte Simulation auszeichnen, finden im Kurierdienstmodell eine idealtypische Anwendung.

6.6.2.1. Raummodell und Unterstützung der Bewegung im Raum

Wie bereits in Abschnitt 4.3.1.1 diskutiert, stellen (gerichtete) Graphen eine natürliche Repräsentationsform für Straßennetze dar. Da FAMOS nicht nur die in der agentenbasierten Simulation verbreiteten Gitter, sondern explizit gerichtete Graphen als Raummodell zur Verfügung stellt, lässt sich auch das im Kurierdienstmodell verwendete detaillierte Hamburger Straßennetz, welches aus über 17.000 Kreuzungspunkten (Knoten) und 48.000 Straßenabschnitten (Kanten) besteht, problemlos abbilden. Fahrrad- und Autokuriere bewegen sich auf diesem Netz je nach Straßentyp mit unterschiedlichen Geschwindigkeiten.

Dies wird durch Kantenattribute ausgedrückt, die jedem Verkehrsmittel eine bestimmte Geschwindigkeit zuordnen. Ist ein Straßenabschnitt für ein Verkehrsmittel nicht zugelassen, wie beispielsweise Autobahnen für Fahrräder oder Wege durch Parks für Autos, so wurde die entsprechende Geschwindigkeit auf 0 gesetzt. Auf diese Weise konnten auch Einbahnstraßen in umgekehrter Richtung ausschließlich für Fahrradkuriere befahrbar gemacht werden.

Die Bewegung der Kuriere auf dem Straßennetz wird von FAMOS übernommen und ist fast ausschließlich mit Blackbox-Klassen des Frameworks realisiert. Kuriere benutzen Instanzen der Klasse `OdoRoute`, um ihre Bewegung durchzuführen. Diese Klasse ermöglicht die Bewegung anhand einer vorab geplanten Route und zeichnet dabei die zurückgelegte Distanz auf (vgl. Abschnitt 5.2.1.2). Ein Kurier-Agent steuert seine Bewegung auf dem Straßennetz einfach durch Aufruf der Methoden `setDestination()`, `move()` und `stop()`. Mit `setDestination()` gibt er ein Ziel (Abhol- oder Lieferadresse) vor, zu dem dann automatisch die schnellste Route ermittelt wird. Wiederholter Aufruf von `move()` bewegt den Agenten entlang dieser Route, während `stop()` einen Bewegungsvorgang unterbricht. Letzteres erlaubt den Kurieren, sofort auf Nachrichten von der Zentrale zu reagieren. Die Bestimmung der Bewegungsdauer erfolgt abhängig von Verkehrsmittel und Kantenattributen über die spezielle Bewertungsfunktion `StreetRatingFunction`. Dies ist die einzige anwendungsspezifische Anpassung des Frameworks, die für das Kurierdienstmodell benötigt wurde.

FAMOS unterstützt die Bewegung im Raum somit auf einem hohen Niveau. Ein Vergleich mit anderen Simulationswerkzeugen für agentenbasierte Simulation zeigt, dass dies immer noch eine Ausnahme darstellt. In der Regel werden nur Basis-Operationen wie Hinzufügen bzw. Löschen eines Agenten an einer bestimmten Position im Raum und agenten-internes Update der Position zur Verfügung gestellt, aus denen der Modellierer selbst die Agenten-Bewegung zusammensetzen muss (Railsback et al. 2006). Dies trifft u. a. auf die verbreiteten Simulationspakete Swarm (Minar et al. 1996), Repast (North et al. 2006, 2013) und Mason (Luke et al. 2005) zu. Andere Werkzeuge wie NetLogo (Wilensky 1999) oder SeSAm (Klügl 2001) bieten zwar grundlegende Methoden für schrittweise Bewegung in eine Richtung oder Sprünge zu einer spezifizierten Position an, jedoch sind diese auf zweidimensionale Gitter beschränkt. Desweiteren findet keinerlei Berechnung der Dauer des Bewegungsvorgangs statt; da alle genannten Softwarepakete eine zeitgesteuerte Fortschreibung der Simulationszeit (zumindest als Default) vorgeben, dauert jeder Bewegungsvorgang genau einen Zeitschritt.

6.6.2.2. Kommunikation und Organisationsstrukturen

Die von FAMOS zur Verfügung gestellte Funktionalität in Bezug auf Agenten-Kommunikation und Organisationsstrukturen deckt den Bedarf des Kurierdienstmodells vollständig ab. Während die einzelnen Kuriere ausschließlich mit der Zentrale Nachrichten austauschen, verwendet diese sowohl $1 : 1$ als auch $1 : n$ Nachrichtenaustausch in ihrer Kommunikation mit den Kurieren: Zu vermittelnde Aufträge werden einer Teilmenge der Kurierflotte gleichzeitig über Funk angeboten, die weitere Verhandlung findet nur noch mit einem einzelnen Kurier statt. Aus der Sicht des Modellierers ist beides gleichermaßen einfach zu benutzen – durch Aufruf der Methode `send()`, welche in der Klasse `Agent`

definiert ist. Im Fall der Multicast-Kommunikation ist als Empfänger statt eines einzelnen Agenten eine definierte Gruppe von Agenten anzugeben, das ist der einzige Unterschied. Den Rest, d. h. die Weiterleitung der Nachricht an den bzw. die Empfänger, übernimmt das Framework FAMOS.

Das Konzept hierarchischer Gruppen ist hilfreich bei der Umsetzung der Aufteilung der Kurierflotte in freigemeldete bzw. Aufträge abarbeitende Fahrrad- und Autokuriere. Auf diese Weise lässt sich jeweils die gewünschte Untermenge von Kurieren ansprechen, so dass die Zentrale problemlos Aufträge ausschließlich an Fahrradkuriere vermitteln oder freigemeldete Kuriere bei der Auftragsvergabe bevorzugen kann.

Im Vergleich mit anderen Werkzeugen für agentenbasierte Simulation bietet nur MadKit, ein Multiagenten-Toolkit mit Simulationserweiterung (Gutknecht et al. 2000), vergleichbare Funktionalität. Dies ist nicht weiter überraschend, setzt MadKit doch ebenfalls das von Ferber und Gutknecht entwickelte Konzept von Gruppen und Rollen (Ferber und Gutknecht 1998; Ferber et al. 2004) um, welches als Vorlage für die Organisationsstrukturen in FAMOS gedient hat (vgl. Abschnitt 4.4.3).

Während reine MAS-Werkzeuge selbstverständlich die Kommunikation von Agenten auf Basis von getypten Nachrichten beinhalten, fehlt diese Funktionalität in gängigen Softwarepaketen für die agentenbasierte Simulation bisher vollständig. Weder Swarm, Repast oder Mason noch NetLogo bieten Unterstützung der Agenten-Kommunikation an. Dies ist darauf zurückzuführen, dass die genannten Softwarepakete die Modellierung von Agenten auf einem relativ niedrigen Level anbieten.

6.6.2.3. Verhaltensmodellierung

Wie aus der Modellbeschreibung in Abschnitt 6.2.2 zu ersehen, ist das Verhalten der Zentrale und insbesondere der Kuriere so komplex, dass eine Modellierung auf höherer Ebene angeraten ist. Da sowohl das individuelle Verhalten als auch die Interaktion zwischen Kurieren und Zentrale im Wesentlichen von Ereignissen wie Eingang oder Erledigung eines Auftrags gesteuert wird, bietet sich die Verwendung von Zustandsautomaten an. Die Implementation ausführbarer Zustandsdiagramme in FAMOS ist konform zur UML-Semantik und unterstützt hierarchische Zustände, nebenläufige Regionen, Inter-Level-Transitionen und statische Reaktionen, d. h. Reaktionen auf Ereignisse, die keine Zustandsänderung nach sich ziehen. Im hierarchischen Zustandsdiagramm der Kuriere erlauben letztere die Entkopplung der Reaktion auf Nachrichten vom aktuellen Zustand der Auftragsbearbeitung.

Der graphische Editor ermöglicht eine visuelle Modellierung des Agentenverhaltens als UML-Zustandsdiagramm, welches per Codegenerator automatisch in entsprechenden Programmcode überführt wird. Die Umsetzung des konzeptuellen Modells in ein ausführbares Computermodell wird dadurch wesentlich erleichtert. Darüber hinaus hat sich die visuelle Modellierung als hilfreich beim Debugging erwiesen.

Der deliberative Anteil des Kurierverhaltens kann dagegen mit Zustandsdiagrammen nicht adäquat dargestellt werden. Auftragsbewertung und Tourenplanung mussten direkt in Java implementiert werden. Sie werden als Aktionen der entsprechenden Transitionen aufgerufen, ihr Art und Umfang ist somit im Zustandsdiagramm nicht sichtbar. FAMOS bietet derzeit keinen designierten Verhaltensbaustein für deliberatives Verhalten. Die

modulare Architektur lässt eine Erweiterung jedoch problemlos zu; Czogalla und Matzen (2003) belegen dies mit ihrer Arbeit zu zielgerichteten Bewegungen von Passagieren in Flugzeugkabinen.

In einer Machbarkeitsstudie wurde im Rahmen einer studentischen Projektarbeit (Sadikni 2003) das Verhaltensmodell der Zentrale von einem Zustandsautomaten in ein regelbasiertes Modell überführt. Die bereits vorhandenen Sensor- und Effektormethoden der Klasse `Office` sowie die Signalklassen konnten direkt wiederverwendet werden. Bei Verzicht auf die Abbildung der Dauer einzelner Aktionen, die im Zustandsdiagramm durch die Verzögerung im Zustand *Aktiv* modelliert wird, reicht es aus, für jede Transition eine gleichwertige Regel zu spezifizieren. Das resultierende deklarative Verhaltensmodell ist übersichtlich und – Kenntnis der JESS-Skriptsprache vorausgesetzt – leicht nachvollziehbar. Die mit diesem Modell erzielten Simulationsergebnisse sind mit denen des Basismodells vergleichbar: Die Mittelwerte der einzelnen Zielgrößen stimmen überein. Die Abweichungen lassen sich dadurch erklären, dass die Aktionen der Zentrale zeitverzugslos durchgeführt werden. Es sind prinzipiell sogar mehrere Aktionen zum gleichen Zeitpunkt möglich, da der Regelinterpreter alle ausführbaren Regeln schaltet. Für das Kurierdienstmodell hat sich die zustandsbasierte Modellierung jedoch bewährt.

6.6.2.4. Aufzeichnung von Daten

Mit der abstrakten Klasse `IndividualObserver` bietet FAMOS die Möglichkeit, Daten über einzelne Agenten oder Entitäten aufzuzeichnen und im Ergebnisreport auszugeben. Für numerische Attribute können zusätzlich statistische Größen wie Minimum, Maximum, Mittelwert und Standardabweichung automatisch berechnet werden. Zur Beobachtung modellspezifischer Entitäten ist ein konkreter `IndividualObserver` zu implementieren, der in der Methode `init()` festlegt, welche Merkmale aufzuzeichnen sind. Der gemeinsame Typ der Entitäten wird bei der Instantiierung eines Beobachter-Objekts spezifiziert.

FAMOS ermöglicht für das Kurierdienstmodell auf diese Weise die individuelle Auftragsverfolgung inklusive Berechnung von Makro-Kenngrößen wie kürzeste, durchschnittliche und längste Lieferzeit, Anzahl von verspäteten Aufträgen und Summe aller Verspätungen. Bezüglich der Kuriere werden Daten wie gefahrene Strecke, Anzahl erledigter Aufträge, Verdienst und Auslastung aufgezeichnet. Um die Daten für Auto- und Fahrradkuriere getrennt zu halten, beinhaltet das Kurierdienstmodell zwei `CourierObserver`. Dies vereinfacht die Bestimmung der Zielgröße „Ökologische Qualität“, welche als mittlere motorisiert zurückgelegte Strecke pro Auftrag operationalisiert wurde (vgl. Abschnitt 6.1.2).

6.6.2.5. Import empirischer Daten

Das Kurierdienstmodell benötigt als Eingabedaten detaillierte empirischen Daten über Aufträge und Kuriere. Aus diesen sind während eines Simulationslaufs entsprechende Auftragsobjekte bzw. Kurier-Agenten zu erzeugen und zu ihrem spezifizierten Ankunftszeitpunkt ins System einzuschleusen. FAMOS unterstützt dies – so weit wie modellunabhängig möglich – über die Import-Schnittstelle für empirische Daten in Verbindung

mit dem verallgemeinerten Ankunftsprozess für beliebige Objekte (Klasse `External-ObjectArrival`). Die im ASCII-Format vorliegenden Daten können direkt über die Klasse `FileDataSource` eingelesen und an die beiden modellspezifischen, von `External-ObjectArrival` abgeleiteten Ankunftsprozesse zur Verarbeitung übergeben werden. In den abgeleiteten Klassen sind nur die drei abstrakten Methoden `createObject()`, `getArrivalTime()` und `scheduleObject()` geeignet zu überschreiben.

6.6.2.6. GIS-Schnittstelle

Als Raummodell des Kurierdienstmodells dient das Hamburger Straßennetz, welches als Objekt der Klasse `Graph` repräsentiert wird. Das detaillierte Netz besteht aus etwas mehr als 17.000 Knoten und 48.000 Kanten. Die XML-Datei mit der Spezifikation dieses Netzes hat eine Größe von dreizehn MB. Der Import über die von FAMOS zur Verfügung gestellte GIS-Schnittstelle funktioniert, allerdings unter immensem Speicher- und Zeitbedarf – auf einem durchschnittlichen PC dauerte der Import des gesamten Netzes mehr als 2 Stunden.[7]

Da zu Beginn eines jeden Experiments diese Datei einzulesen und in ein `Graph`-Objekt zu wandeln ist, musste eine praktikablere Lösung gefunden werden. Die aus diesem Grund zusätzlich implementierte maßgeschneiderte Schnittstelle für Daten im ASCII-Format (Klasse `famos.util.Import`) hat sich bewährt. Die Textdatei mit der Netz-Spezifikation ist nur noch 5.5 MB groß und der Import dauert wenige Minuten.

6.6.2.7. Graphische Benutzungsschnittstelle

Abbildung 6.12 zeigt exemplarisch die simulationsbegleitende Visualisierung für das Kurierdienstmodell. Der in DESMO-J integrierte graphische Experiment-Starter (vgl. Abschnitt 5.1.2.1) ermöglicht die Verfolgung modellspezifischer Zustandsgrößen durch Spezifikation sogenannter graphischer Beobachter (Klasse `desmoj.gui.GraphicalObserver`). Zur Verfügung stehen in der derzeitigen Implementation Zeitreihen für numerische Größen (Klasse `desmoj.gui.TimeSeriesPlotter`) und eine Darstellung der räumlichen Umgebung eines Multiagentenmodells einschließlich Animation der Agentenbewegung (Klasse `famos.gui.EnvironmentView`).

Beide wurden für das Kurierdienstmodell verwendet: Zeitreihen für auftrags- und kurierbezogene Größen wie die Anzahl eingetroffener bzw. bereits erledigter Aufträge sowie im System befindlicher Kuriere (in Abbildung 6.12 rechts) und ein `EnvironmentView` für die Darstellung der Kuriere auf dem Hamburger Straßennetz. Aus der Farbkodierung der Agenten kann ihr Verkehrsmittel (rot für Auto, grün für Fahrrad) und ihr aktueller Beschäftigungsstatus (dunkel für freigemeldet, hell für Aufträge ausführend) abgelesen werden. Zu diesem Zweck ist in der Klasse `Courier` nur die von `SituatedAgent` geerbte Methode `getColor()` zu überschreiben. Die Darstellung des Netzes und der Agenten wird von FAMOS geliefert.

[7] Die GIS-Schnittstelle in FAMOS arbeitet mit einer frühen Version des Data-Binding-Frameworks Castor. Es ist zu erwarten, dass sich mit der Weiterentwicklung von Castor zumindest der Zeitbedarf inzwischen verbessert hat.

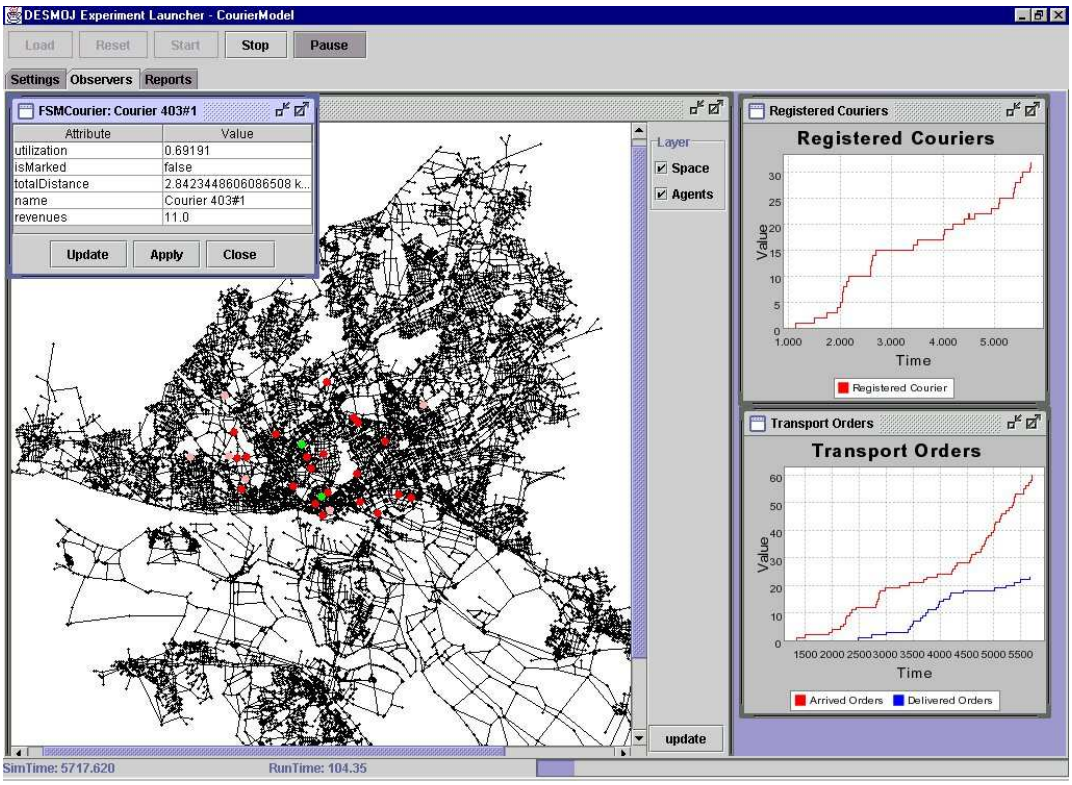

Abbildung 6.12.: Screenshot des Kurierdienstmodells

FAMOS erlaubt darüber hinaus, den aktuellen Zustand einzelner Kuriere genauer zu inspizieren. Ein Mausklick auf einen der als Kreise dargestellten Agenten öffnet ein sogenanntes Inspektions-Fenster (siehe Abbildung 6.12 oben links) mit einer Liste von Attributen. Welche Attribute angezeigt werden, liegt in der Hand des Modellierers und ist in der Methode `getProbeMap()` zu definieren. Für die Kuriere ersetzen z. B. die modellspezifischen Größen Auslastung, zurückgelegte Distanz und Einnahmen die als Default definierten Attribute Sensor-Reichweite, aktuelle Position und agenten-interne Ereignisliste.

Die von FAMOS bereitgestellte Visualisierung des Modellgeschehens hat sich als nützlich bei Kalibrierung und Verifikation des Modells erwiesen. So kann beispielsweise relativ schnell erkannt werden, ob einzelne Kuriere keinerlei Aufträge erhalten. Die graphische Darstellung war außerdem hilfreich in der Kommunikation mit den beteiligten Kurierdiensten und anderen Interessenvertretern.

Auf der anderen Seite verlangsamte die graphische Darstellung die Modellausführung wesentlich, so dass Simulationsläufe in der Experimentierphase der Kurierdienststudie ohne GUI durchgeführt wurden. Dies ist hauptsächlich auf hohen Speicherbedarf für das detaillierte Straßennetz in Verbindung mit Performanzproblemen in der Implementation von Javas Graphik API zurückzuführen (Sun 2001).

6.6.2.8. Abschließende Bewertung

Das Framework FAMOS hat sich im Einsatz für die Simulationsstudie zu Stadtkurierdiensten als sehr gut geeignet erwiesen. Das erste Projektziel bestand in der Entwicklung eines Modells, welches den Ist-Zustand typischer Hamburger Stadtkurierdienste widerspiegelt und als Grundlage bzw. Referenz für die Evaluation alternativer Logistikkonzepte dienen kann. Ein ähnliches Modell wurde bereits 1997 als Teil einer Fallstudie im Rahmen des MOBILE-Projekts erstellt, allerdings in einem anderen technischen Rahmen implementiert (vgl. Hilty et al. 1998, S. 291ff). Durch Einsatz von FAMOS und der zugehörigen Modellierhilfen konnten verschiedene Aspekte anschaulicher und näher am realen System modelliert werden.

Die Umsetzung der in Abschnitt 3.2.5 definierten agentenorientierten Weltsicht mit Ereignissteuerung und expliziter Raumrepräsentation in FAMOS erlaubte eine transparente Abbildung aller relevanten Prozesse des Realsystems in das Kurierdienstmodell. Die inhärente Ereignissteuerung eines Kurierdienstes blieb ebenso erhalten wie Routenplanung und Bewegung der Kuriere auf dem detailliert modellierten Hamburger Straßennetz sowie das Kommunikationsprotokoll zwischen Kurieren und Zentrale. Die Verwendung von Zustandsdiagrammen für die Modellierung des Agentenverhaltens ist ideal auf die Ereignissteuerung abgestimmt. Während sich diese Form der Modellierung für das System „Kurierdienst“ als adäquat in Bezug auf Ausdruckskraft und Anwendungsfreundlichkeit erwiesen hat, wird sie für komplexeres Agentenverhalten unter Umständen nicht mehr zweckmäßig sein, da deliberative Aspekte nicht angemessen abgebildet werden können und das Zustandsdiagramm zu groß und unübersichtlich werden kann (Knaak 2004). Letzteres ließe sich jedoch durch die erweiterte Modularität von Zustandsdiagrammen in UML 2.0, welche eine Verfeinerung von Zuständen zu Unterzustandsdiagrammen erlaubt (vgl. z.B. Rupp et al. 2007, S. 367f), verbessern und sollte für die zukünftige Ent-

wicklung von FAMOS berücksichtigt werden.

7. Zusammenfassung und Ausblick

The goal of science is to make the wonderful and complex understandable and simple – but not less wonderful.
– Herbert Simon

Agentenbasierte Modellierung und Simulation, d. h. die Verwendung der Multiagentensystem-Metapher zur Modellierung komplexer Systeme, ist ein relativ neues Forschungsgebiet, das dessen ungeachtet bereits als Modellierungsparadigma in einem breiten Spektrum von Anwendungsgebieten etabliert ist. Hierzu gehören die Sozialwissenschaften (Epstein 2007; Edmonds und Meyer 2013), Wirtschaftswissenschaften (Tesfatsion 2002), die Geographie (Heppenstall et al. 2012) und Ökologie (Grimm und Railsback 2005). Die vorliegende Dissertation zeigt auf, wie die agentenbasierte Simulation als eine Weltsicht der ereignisdiskreten Simulation definiert werden kann. Damit ist die Arbeit im Schnittpunkt der Bereiche diskrete Simulation und Multiagentensysteme angesiedelt.

Im folgenden wird zunächst der in dieser Arbeit erbrachte wissenschaftliche Beitrag zusammengefasst und bewertet (Abschnitt 7.1), bevor abschließend Erweiterungsmöglichkeiten (Abschnitt 7.2) aufgezeigt werden.

7.1. Zusammenfassende Betrachtung der Arbeit

Verschiedene Disziplinen haben die Entstehung der agentenbasierten Simulation beeinflusst (Abschnitt 3.2.1), darunter die Komplexitätsforschung und die Verteilte Künstliche Intelligenz, während die traditionelle Simulation den entstehenden neuen Modellierungsstil weitgehend ignoriert hat. Dies führte u. a. dazu, dass eine theoretische Grundlage für die agentenbasierte Simulation bisher fehlt. Erste Ansätze streben interessanterweise mehrheitlich an, die agentenbasierte Simulation in der ereignisgesteuerten Simulation zu verankern (Abschnitt 3.2.4), obwohl Zeitsteuerung als *de facto* Standard für ABMS gelten kann (Abschnitt 3.2.2).

Mehrere Gründe sprechen für die Verwendung des ereignisgesteuerten Zeitfortschritts in agentenbasierten Modellen. Zum einen ist Ereignissteuerung effizienter, da nur die Zeitpunkte von Zustandsänderungen betrachtet und somit inaktive Phasen des simulierten Systems übersprungen werden. Sie ist flexibler, da sich die Auflösung der Zeit-Diskretisierung dynamisch an die Abläufe des betrachteten Systems anpasst, während bei der Zeitsteuerung eine künstliche Taktrate die Diskretisierung der Zeit starr vorgibt. Darüber hinaus lässt sich Zeitsteuerung problemlos auf Ereignissteuerung abbilden, während umgekehrt dies nicht ohne weiteres möglich ist (Abschnitt 2.1.2). Damit verbunden ist die Tatsache, dass manche Systeme eine korrekte Abbildung der die Systemdynamik vorantreibenden Ereignisse benötigen; ein Beispiel ist der in Kapitel 6 beschriebene Stadtkurierdienst. Eine Zeitsteuerung würde entweder die Ergebnisse verfälschen, da aufeinanderfolgende Ereignisse als zeitgleich behandelt werden, wenn die Taktrate Δt zu groß

gewählt wird, oder sehr ineffizient sein, weil die Taktrate so klein gewählt werden muss wie der kleinste Abstand zwischen zwei Ereignissen. Bei letzterem besteht darüber hinaus das Problem, dass dieses Intervall nicht vorher bekannt ist (siehe Abschnitt 2.1.3). Ereignissteuerung ermöglicht somit eine umfangreichere Klasse von Modellen abzubilden als eine reine Zeitsteuerung.

Mit der Definition und nachfolgenden Umsetzung der agentenorientierten Weltsicht in einem dedizierten Simulationswerkzeug trägt diese Arbeit dazu bei, komplexe Systeme, welche inhärent von Ereignissen gesteuert werden, adäquat modellieren zu können. Bei einem Vergleich mit existierenden ABMS-Werkzeugen stellt sich die Frage, wie notwendig die Konzeption und Realisierung eines neuen Werkzeugs ist. Mehrere Simulationsframeworks führen in ihrer Funktionsbeschreibung die Bereitstellung eines ereignisgesteuerten Scheduler auf; die verbreitetsten sind Swarm, Mason und Repast bzw. Repast Simphony. Damit ist die Verwendung von Ereignissteuerung in agentenbasierten Modellen zumindest möglich. Ich möchte jedoch argumentieren, dass es nicht ausreicht, einen ereignisgesteuerten Scheduler zur Verfügung zu stellen. Auch die Benutzung der Ereignissteuerung muss entsprechend unterstützt werden und hier weisen die existierenden Simulationswerkzeuge Defizite auf.

Zum einen konzentriert sich deren Dokumentation auf Beschreibungen, wie zeitgesteuerte Modelle mit dem jeweiligen Werkzeug erstellt werden können (vgl. z. B. Minar et al. 1996, S. 3; Luke 2013, S. 16f; North et al. 2013, S. 18ff). Dies ist die in der agentenbasierten Simulation vorherrschende Sichtweise, die auch von einführenden Lehrbüchern wie z. B. (Klügl 2001; Gilbert und Troitzsch 2005) vertreten wird (siehe Abschnitt 3.2.2). Dies ist insofern problematisch, als es die Wahrnehmung eines Modellierers beeinflussen kann, so dass er die Welt nur noch mit den „Augen“ seines bevorzugten Frameworks sieht (Edmonds 2001, S. 18).

Zum anderen fehlt eine Abstimmung der Konstrukte für die Modellierung des Agenten-Verhaltens, der Umgebung einschließlich Raummodell sowie der Interaktionen zwischen Agenten und Umgebung auf die Anforderungen, die von der Ereignissteuerung gestellt werden. Dies betrifft, wie bereits in Abschnitt 1.1 dargelegt,

1. die Bestimmung von Aktionsdauern, so dass das zugehörige Ende-Ereignis rechtzeitig auf der Ereignisliste eingetragen werden kann;

2. die Gewährleistung, dass jeder Agent sofort auf Veränderungen in der Umgebung reagieren kann, obwohl er während der Ausführung einer zeitverbrauchenden Handlung programmtechnisch passiv ist;

3. die Gewährleistung, dass die Aktualisierung des Umgebungszustands, der die aktuellen Positionen sich im Raum bewegender Agenten beinhaltet, trotz asynchroner Agenten-Aktualisierung effizient gehalten wird, d. h. nicht vor jedem Zugriff durch einen Agenten aktualisiert werden muss.

Mit der Konzeption und Realisierung des Frameworks für agentenorientierte Modellierung und Simulation (FAMOS) wurde in dieser Arbeit ein Simulationswerkzeug entwickelt, das diese Anforderungen umsetzt und die agentenorientierte Weltsicht angemessen unterstützt. Das bedeutet, dass mit der Bereitstellung einer entsprechenden Simulationsinfrastruktur – in diesem Fall durch Integration eines vorhandenen diskreten

Simulators – nicht nur die technischen Voraussetzungen für eine Verwendung der Ereignissteuerung geschaffen, sondern einem Modellierer dedizierte Konstrukte auf der Ebene der Agenten- und Raum-Modellierung angeboten werden. Besonderer Fokus lag auf der Entwicklung eines flexiblen Raummodells und der Unterstützung der Bewegung im Raum, weil dies von vielen Anwendungen benötigt wird und letztere auf die Ereignissteuerung abgestimmt werden muss. Übergeordnetes Entwurfsziel war es, FAMOS anwendungsunabhängig zu gestalten, um das in der Multiagentensimulation existierende breite Spektrum von Anwendungen abdecken zu können.

Der Aufbau auf einem objektorientierten Simulationspaket für diskrete Simulation, welches die ereignisorientierte und prozessorientierte Weltsicht unterstützt, hat neben der direkten Wiederverwendung der Simulationsfunktionalität weitere Vorteile. So kann auf Entitäten und Ereignisroutinen bzw. Prozesse zurückgegriffen werden, um die Eigendynamik der Agenten-Umgebung auf einem angemessenen Abstraktionsniveau zu modellieren. Darüber hinaus sind Zufallszahlenströme verschiedener Verteilungen vorhanden, die für die Bestimmung von Aktionsdauern eingesetzt werden können. Voraussetzung ist, eine geeignete Integration des Agentenkonzepts mit dem bestehenden System zu finden, d. h. Agenten auf der technischen Grundlage von Ereignissen und Entitäten oder Prozessen zu realisieren. Damit folgt das Framework FAMOS dem immer noch bestehenden Trend, Agenten nur auf der konzeptuellen Ebene zu unterstützen, statt sie ebenfalls für die Implementation des Modells einzusetzen, wie von (Drogoul et al. 2003) gefordert.

Für die Anforderungen, die sich aus der Wahl eines ereignisgesteuerten Zeitfortschritts ergeben, sind in FAMOS folgende Lösungsansätze realisiert:

1. Die Dauer modell-unabhängiger Agenten-Aktionen wie Wahrnehmung der Umgebung, Manipulation von Objekten, Senden von Nachrichten oder Bewegen im Raum wird vom Framework automatisch bestimmt. In der vorliegenden Version werden alle solche Aktionen als zeitverzugslos angenommen, mit Ausnahme der Bewegung. Deren Dauer wird in Abhängigkeit von der zurückgelegten Distanz und der aktuellen Geschwindigkeit des Agenten berechnet. Für modell-spezifische Aktionen, die ein Modellierer selbst implementiert, stellt FAMOS über das integrierte Simulationspaket DESMO-J stochastische Verteilungen zur Verfügung.

2. Um zu gewährleisten, dass jeder Agent sofort auf Veränderungen in der Umgebung reagieren kann, lässt FAMOS die Umgebung denjenigen Agenten ein spezielles Signal schicken, die von der Veränderung betroffen sind, d. h. in deren Wahrnehmungsbereich die Veränderung eingetreten ist. Ein Agent wird über dieses Signal automatisch zur Reaktivierung vorgemerkt, kann also seine aktuelle, zeitverbrauchende Handlung unterbrechen und selbst entscheiden, ob und wie er auf die Veränderung reagieren will.

3. In der diskreten Simulation sind Zustandsänderungen mit zeitverzugslosen Ereignissen verknüpft, so dass der Zustand des Systems zwischen Ereignissen konstant bleibt. Damit ist gewährleistet, dass auch bei einer asynchronen Aktualisierung von Entitäten der Zustand des Systems zu jedem Zeitpunkt aktuell ist. Zeitverbrauchende Handlungen müssen auf eine Folge von Ereignissen abgebildet werden und die

von ihnen bewirkte Zustandsänderung wird üblicherweise erst zu ihrem Ende wirksam. Für die von FAMOS zur Verfügung gestellten Agenten-Aktionen stellt dies nur für die Bewegung im Raum ein Problem dar, da die aktuellen Positionen sich bewegender Agenten bei länger andauernder Bewegung zu ungenau werden, wenn sie erst am Ende des Bewegungsvorgangs aktualisiert werden. Um Positionen ausreichend genau aktuell zu halten und so eine Aktualisierung des Umgebungszustands vor jedem möglichen Zugriff eines Agenten zu vermeiden, werden längere Bewegungsvorgänge in FAMOS in atomare Schritte zerlegt. Als atomar wird dabei die Bewegung zwischen benachbarten diskreten Raumelementen (Zellen eines Gitters, Knoten eines Graphen) definiert. Die eigentliche Zustandsänderung (Wechsel der Position) erfolgt dabei bereits nach der Hälfte der Zeitdauer, während der Agent selbst erst zum Ende der Aktion über diese informiert wird (vgl. Abschnitte 4.3.2 und 4.6.3).

Diese Abstimmung der räumlichen Bewegung auf die Ereignissteuerung ist für Benutzer des Frameworks transparent in einer speziellen Komponente `Movement` gekapselt. Sie bietet für die Modellierung des Agenten-Verhaltens eine Methode `move()` an, welche automatisch die Berechnung der Dauer und Vormerkung des entsprechenden Reaktivierungsereignis für den Agenten übernimmt. Über die Auswahl bzw. Implementation einer bestimmten Bewegungsstrategie lässt sich die Bewegungskomponente an modellspezifische Bedürfnisse anpassen; vordefiniert sind die Strategien Zufallsbewegung (*Random Walk*), Gradientensuche, Bewegung in eine Richtung und Bewegung entlang einer geplanten Route. Die Unterstützung der Bewegung in FAMOS geht damit weit über die von vergleichbaren Simulationsframeworks geleistete Funktionalität hinaus, in der ein Modellierer die Bewegung aus primitiven Befehlen selbst zusammensetzen muss.

Eine weitere Stärke von FAMOS ist das flexible, diskrete Raummodell, dem es gelingt, die in der agentenbasierten Simulation vorherrschenden Gitter mit gerichteten Graphen zu kombinieren (vgl. Abschnitt 4.3). Dazu wird die Tatsache ausgenutzt, dass zu jeder Tesselation ein dualer Graph existiert; ein im Zusammenhang mit GIS bekanntes Beispiel sind Voronoi-Diagramm und Delaunay-Triangulation (siehe Abschnitt 2.2.2). Das Modell betrachtet den diskreten Raum als aus Raumelementen aufgebaut, die über Nachbarschaftsbeziehungen verbunden sind, welche Wahrnehmung und Bewegung von Agenten beeinflussen. Zur einheitlichen Beschreibung dieser räumlichen Struktur werden gerichtete Graphen verwendet, indem die Raumelemente auf Knoten und die Nachbarschaftsbeziehungen auf gerichtete Kanten abgebildet werden. Diese Abbildung kann in der Regel automatisch vom Framework generiert werden, so dass kein zusätzlicher Aufwand für den Benutzer entsteht. In FAMOS ist dies derzeit realisiert für regelmäßige Gitter mit rechteckigen oder hexagonalen Zellen und als Punktmenge spezifizierte unregelmäßige Gitter, für welche sowohl das Voronoi-Diagramm als auch der zugehörige duale Graph berechnet werden.

Die Verwendung eines gerichteten Graphen als grundlegendes Raummodell ermöglicht die einheitliche Anwendung aller Bewegungsstrategien auf verschiedene Ausprägungen des Raummodells. Darüber hinaus wird mit der expliziten Repräsentation von Positionen als speziellen Positionsobjekten eine Abstraktionsschicht zwischen den Agenten und dem Raummodell eingeschoben, welche die Austauschbarkeit des Raummodells erleich-

tert und die Anforderung umsetzt, den Zugriff auf die räumliche Umgebung für Agenten unabhängig vom verwendeten Raummodell zu machen. Der Zugriff auf das Raummodell findet ausschließlich über die Umgebung statt. Diese bildet gemäß dem Entwurfsmuster Fassade die Schnittstelle zwischen Agenten und Raummodell, Kommunikationsinfrastruktur und als Gruppen realisierte Organisationsstrukturen.

Um von einer bestimmten Agenten-Architektur (siehe Abschnitt 2.3.4) zu abstrahieren, sind Agenten in FAMOS modular aufgebaut. Sie besitzen eine Menge von Fähigkeiten (Kommunikation, Zugriff auf die Umgebung, Bewegung), die modellspezifisch erweitert werden kann, und ein austauschbares Verhalten, das in einem Baustein gekapselt ist. Im Rahmen einer Diplomarbeit (Knaak 2002) wurden vier Verhaltensbausteine in FAMOS realisiert, welche verschiedene Methoden zur Modellierung des Agentenverhaltens anbieten (Abschnitt 5.2.3). Jede dieser Methoden besitzt ihre Berechtigung innerhalb eines klar umgrenzten Einsatzbereichs. Die modifizierte ereignisorientierte Modellierung eignet sich aufgrund ihrer Effizienz besonders zur Beschreibung des Verhaltens großer Populationen einfacher reaktiver Agenten. Sie war daher ideal, um das Verhalten der Agenten im Segregationsmodell zu implementieren (Abschnitt 5.3). Die prozessorientierte Variante ist dagegen besser für die Modellierung relativ starren proaktiven Verhaltens geeignet, welches nur gelegentlich durch Ereignisse unterbrochen wird. Sie fand Einsatz in der Re-Implementation des Sugarscape-Modells (Abschnitt 5.4). Der deklarative Verhaltensbaustein eignet sich aufgrund der Möglichkeiten zur Symbolverarbeitung besonders dann, wenn die Verhaltensbeschreibung der Agenten in Form von qualitativen Kausalbeziehungen vorliegt oder komplexe interne Weltmodelle berücksichtigt werden sollen; FAMOS bietet derzeit keine weitere Unterstützung deliberativer Agenten an. Als besonders flexibel hat sich die Modellierung mit Zustandsautomaten erwiesen, welche für die Agenten im Kurierdienstmodell verwendet wurde (siehe Kapitel 6). Die graphische Spezifikation und automatische Übersetzung in ablauffähigen Code ermöglichen eine anschauliche und leicht verifizierbare Implementation des Verhaltens subkognitiver Agenten.

Das Framework FAMOS konnte im Rahmen einer Studie zu Stadtkurierdiensten unter realen Anforderungen evaluiert werden. Seine Schwerpunkte wie Ereignissteuerung, graphbasiertes Raummodell, besondere Unterstützung der Bewegung im Raum und Verhaltensmodellierung mit Zustandsdiagrammen, welche FAMOS gegenüber anderen Softwarepaketen für agentenbasierte Simulation auszeichnen, konnten im Kurierdienstmodell mit Gewinn eingesetzt werden. Einzig die im Verhaltensmodell der Kuriere notwendigen Bewertungs- und Planungsmechanismen mussten manuell implementiert werden, da das Framework deliberative Aspekte bisher nicht unterstützt.

7.2. Ausblick auf zukünftige Arbeiten

Viele Aspekte des Frameworks für agentenorientierte Modellierung und Simulation konnten im Rahmen dieser Arbeit bisher nur prototypisch realisiert werden. Dies betrifft vor allem die Unterstützung der Ergebnisanalyse und Validierung, für die nur die Minimalanforderungen bezüglich Datenaufzeichnung und simulationsbegleitender Visualisierung umgesetzt wurden. Um die Laufzeiteffizienz komplexer Multiagentenmodelle wie des Kurierdienstmodells zu erhöhen, sollte zumindest die Möglichkeit ergänzt werden, ani-

mationsrelevante Daten während eines Simulationslaufs in einer Datei zu speichern und diese Datei für eine nachträgliche Animation zu nutzen. Eine solche von der Durchführung des Simulationslaufs entkoppelte Animation könnte zusätzliche Interaktionsmöglichkeiten wie die Unterbrechung der Animation, Zugriff auf Detailinformationen und Veränderung des Zoom-Faktors der Anzeige integrieren. Hier bietet es sich an, eine neu entwickelte 2D-Animationskomponente entsprechend zu erweitern, welche seit Version 2.3 Bestandteil von DESMO-J ist (Müller 2011).

Für die Verhaltensmodellierung von Agenten sind weitere Bausteine wünschenswert, die beispielsweise kognitives oder deliberatives Verhalten abbilden. Die Modellierung mit Zustandsautomaten hat sich zwar in der Studie zu Kurierdiensten in Bezug auf Ausdruckskraft und Anwendungsfreundlichkeit bewährt, wird jedoch für komplexeres Agentenverhalten unter Umständen nicht mehr zweckmäßig sein, da die Zustandsdiagramme zu groß und unübersichtlich werden und deliberative Aspekte nicht entsprechend abgebildet werden können. Im Zusammenhang mit der ausführlichen Validierung des Kurierdienstmodells mittels *Process Mining* im Rahmen der Dissertation von Nicolas Denz zeigte sich, dass die Modellierung des Kurierverhaltens durch ein einzelnes, hierarchisches Zustandsdiagramm die Identifikation eines klaren Prozessflusses behinderte (Denz 2013, S. 382ff). Das Zustandsdiagramm kombiniert verschiedene Aspekte des Kurierverhaltens wie Kommunikation mit der Zentrale, Bewegung auf dem Verkehrsnetz, Abholung bzw. Zustellung von Aufträgen und die Rückmeldung über den Beschäftigungszustand miteinander, welche zum Teil konzeptuell nebenläufig sind. Dies legt den Schluss nahe, die Verhaltensmodellierung analog zu den Agentenfähigkeiten zu modularisieren und statt eines monolithischen Verhaltensbausteins die Verwendung mehrerer, aufgabenspezifischer Komponenten zuzulassen. Diese könnten darüber hinaus dynamisch instantiiert werden, wodurch eine Änderung des Verhaltens zur Laufzeit und somit adaptive Agenten ermöglicht würden. Dieser Ansatz wird derzeit in einem weiteren Dissertationsprojekt verfolgt (Kruse 2007).

Ein Aspekt, der in FAMOS bisher keine Berücksichtigung fand, ist die Modellierung sozialer Netze (vgl. Abschnitt 2.2.1). Diese spielen insbesondere in sozialwissenschaftlichen und ökonomischen Multiagentenmodellen eine große Rolle in ihrem Einfluss auf die Interaktionen der Agenten und ersetzen oder ergänzen in manchen Modellen das physikalische Raummodell. Soziale Netze können dynamisch während der Simulation entstehen, indem Agenten neue Beziehungen zu anderen Agenten knüpfen und alte Beziehungen fallen lassen, oder sie können zu Beginn der Simulation aus empirischen Daten oder bewährten Topologien initialisiert werden (Amblard und Quattrocicocchi 2013). In beiden Fällen wird eine Erweiterung der Agenten-Fähigkeiten um die Verwaltung sozialer Kontakte benötigt.

Für eine breitere Benutzung des Frameworks werden darüber hinaus Effizienz-Verbesserungen (geringerer Speicherbedarf, kürzere Modell-Laufzeiten) und vor allem eine umfangreichere Dokumentation mit einführendem Tutorial benötigt. Zwar stehen eine vollständige API-Dokumentation der Quelltexte und mit den Testbeispielen zwei Beispielmodelle zur Verfügung (Meyer 2013), doch sind Schritt-für-Schritt-Anleitungen und Schablonen für die Modellerstellung gerade für Anwender mit weniger Programmiererfahrung unerlässlich (Railsback et al. 2006, S. 622). Diese sollten ebenfalls helfen, einem interessierten Modellierer die Konzepte der agentenorientierten Weltsicht zu vermitteln.

Um die agentenorientierte Weltsicht stärker ins Bewusstsein der ABMS-Forschungsgemeinde zu rücken, ist zusätzlich deren Integration in die Ansätze zur Definition der theoretischen Grundlagen (Abschnitt 3.2.4) angeraten. Hier scheint das Referenzmodell von (Siegfried et al. 2009) den vielversprechendsten Ansatzpunkt zu bieten, da es bereits grundsätzlich eine Dauer von Aktionen und eine Zeitbasis $T_s \subset \mathbb{R}_0^+$, d. h. die Verwendung reeller Zahlen für Simulationszeitpunkte, zulässt.

Um abschließend auf das Zitat von Siebers et al. (2010) aus der Einleitung zurückzukommen, ereignis-diskrete Simulation ist nicht tot, sondern bleibt – zumindest in ihrer agentenorientierten Ausprägung – eine der am besten geeigneten Methoden, komplexe Systeme zu untersuchen. Sie hat und wird weiterhin, wie Zeigler et al. (2000, S. 66) voraussagten, eine immer größere Rolle in allen Arten der Modellierung spielen. Die vorliegende Arbeit liefert mit der Unterstützung von Agenten in Raum und Zeit einen Beitrag, diese Vision umzusetzen.

Literaturverzeichnis

GML2 2001 Cox, Simon (Hrsg.) ; Cuthbert, Adrian (Hrsg.) ; Lake, Ron (Hrsg.) ; Martell, Richard (Hrsg.): Geography Markup Language (GML) 2.0 / Open Geospatial Consortium (OGC). URL `http://portal.opengeospatial.org/files/?artifact_id=1034`, 2001 – OpenGIS Recommendation Paper 01-029.

XML0 2004 Fallside, David C. (Hrsg.) ; Walmsley, Priscilla (Hrsg.): XML Schema Part 0: Primer Second Edition / World Wide Web Consortium. URL `http://www.w3.org/TR/xmlschema-0/`, 2004 – W3C Recommendation 28 October 2004.

XML1 2004 Thompson, Henry S. (Hrsg.) ; Beech, David (Hrsg.) ; Maloney, Murray (Hrsg.) ; Mendelsohn, Noah (Hrsg.): XML Schema Part 1: Structures Second Edition / World Wide Web Consortium. URL `http://www.w3.org/TR/xmlschema-1/`, 2004 – W3C Recommendation 28 October 2004.

XML2 2004 Biron, Paul V. (Hrsg.) ; Malhotra, Ashok (Hrsg.): XML Schema Part 2: Datatypes Second Edition / World Wide Web Consortium. URL `http://www.w3.org/TR/xmlschema-2/`, 2004 – W3C Recommendation 28 October 2004.

ISO 2007 ISO/TC-211 (Hrsg.): ISO 19136:2007 Geographic Information – Geography Markup Language (GML) / International Organization for Standardization. 2007 – Internationaler Standard.

Aigner 2006 Aigner, Martin: *Diskrete Mathematik : mit 600 Übungsaufgaben*. 6., korrig. Aufl. Wiesbaden : Vieweg, 2006

Amblard und Quattrocicocchi 2013 Amblard, Frédéric ; Quattrocicocchi, Walter: Social Networks and Spatial Distribution. In: (Edmonds und Meyer 2013), S. 401–430

Anderson 1997 Anderson, Scott D.: Simulation of multiple time-pressured agents. In: Andradóttir, S. (Hrsg.) ; Healy, K.J. (Hrsg.) ; Withers, D.H. (Hrsg.) ; Nelson, B.L. (Hrsg.): *Proceedings of the 1997 Winter Simulation Conference*, 1997, S. 397–404

Armstrong 1988 Armstrong, Mark P.: Temporality in Spatial Databases. In: *Proceedings of GIS/LIS'88: Accessing the World, Volume 2*. Falls Church, VA : American Society for Photogrammetry and Remote Sensing, 1988, S. 880–889

Aurenhammer 1991 Aurenhammer, Franz: Voronoi diagrams – a survey of a fundamental geometric data structure. In: *ACM Computing Surveys (CSUR)* 23 (1991), Nr. 3, S. 345–405

Austin 1962 Austin, John L.: *How To Do Things With Words*. Oxford : Oxford University Press, 1962

Axelrod 1997 Axelrod, Robert: *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton, NJ : Princeton University Press, 1997 (Princeton Studies in Complexity 2)

Axtell 2001 Axtell, Robert: Effects of Interaction Topology and Activation Regime in Several Multi-Agent Systems. In: (Moss und Davidsson 2001), S. 33–48

Axtell 2013 Axtell, Robert: Full-scale models: What can be learned from studying entire economies computationally? In: *9th Conference of the European Simulation Association (ESSA 2013), 16-20 September 2013, Warsaw, Poland*, URL `http://www.youtube.com/watch?v=tlcbasKtZyg`, 2013. – Ein Video dieses eingeladenen Vortrags ist unter der angegeben URL verfügbar.

Bachmann 2003 Bachmann, Ralf: *Ein flexibler, CORBA-basierter Ansatz für die verteilte, komponentenorientierte Simulation*, Universität Hamburg, Fachbereich Informatik, Dissertation, 2003

Bachmann et al. 2004 Bachmann, Ralf ; Gehlsen, Björn ; Knaak, Nicolas: Werkzeuggestützte Kalibrierung agentenbasierte Simulationsmodelle. In: *Proceedings der Tagung Simulation und Visualisierung*. Magdeburg, 2004, S. 115–126

Bagnoli 1998 Bagnoli, Franco: Cellular Automata. In: Bagnoli, F. (Hrsg.) ; Lió, P. (Hrsg.) ; Ruffo, S. (Hrsg.): *Dynamical Modelling in Biotechnologies*. Singapore : World Scientific, 1998, S. 3–46

Bala et al. 2012 Bala, Saroj ; Ahson, S. I. ; Agarwal, R. P.: A Pheromone Based Model for Ant Based Clustering. In: *International Journal of Advanced Computer Science and Applications* 3 (2012), Nr. 11, S. 180–183

Banks 1998 Banks, Jerry: Principles of Simulation. In: Banks, Jerry (Hrsg.): *Handbook of Simulation. Principles, Methodology, Advances, Applications, and Practice*. New York; Chichester : Wiley, 1998, S. 3–30

Banks et al. 1999 Banks, Jerry ; Carson, John S. ; Nelson, Barry L.: *Discrete-Event System Simulation*. 2. ed. Upper Saddle River, NJ : Prentice Hall, 1999 (Prentice-Hall International Series in Industrial and Systems Engineering)

Barabási und Bonabeau 2003 Barabási, Albert-László ; Bonabeau, Eric: Scale-free networks. In: *Scientific American* (2003), Nr. May, S. 50–59

Barnes und Chu 2010 Barnes, David J. ; Chu, Dominique: *Introduction to Modeling for Biosciences*. London : Springer, 2010

Barros 1997 Barros, Fernando J.: Modeling Formalisms for Dynamic Structure Systems. In: *ACM Transactions on Modeling and Computer Simulation* 7 (1997), Nr. 4, S. 501–515

Bartelme 2005 Bartelme, Norbert: *Geoinformatik: Modelle, Strukturen, Funktionen*. 4. vollständig überarbeitete Aufl. Berlin : Springer, 2005

Batty und Jiang 2000 Batty, Michael ; Jiang, Bin: Multi-agent simulation: computational dynamics within GIS. In: Atkinson, Peter (Hrsg.) ; Martin, David (Hrsg.): *GIS and Geocomputation*. London : Taylor and Francis, 2000 (Innovations in GIS 7), S. 55–71

Batty und Xie 1994 Batty, Michael ; Xie, Yichun: From cells to cities. In: *Environment and Planning B: Planning and Design* 21 (1994), Nr. 7, S. s31–s48

Bauer et al. 2001 Bauer, Bernhard ; Müller, Jörg P. ; Odell, James: Agent UML: A Formalism for Specifying Multiagent Interaction. In: *Agent-Oriented Software Engineering: First International Workshop AOSE 2000, Limerick, Ireland, 10 June 2000; Revised Papers*. Berlin : Springer, 2001, S. 91–103

Bauer und Odell 2005 Bauer, Bernhard ; Odell, James: UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard. In: *Engineering Applications of Artificial Intelligence* 18 (2005), Nr. 2, S. 141–157

Baveco und Lingeman 1992 Baveco, J. M. ; Lingeman, R.: An object-oriented tool for individual-oriented simulation: host-parasitoid system application. In: *Ecol. Model.* 61 (1992), S. 267–286

Bedau 2003 Bedau, Mark A.: Artificial life: organization, adaptation and complexity from the bottom up. In: *Trends in Cognitive Sciences* 7 (2003), Nr. 11, S. 505–512

Benenson und Torrens 2003 Benenson, Itzhak ; Torrens, Paul M.: Geographic Automata Systems: A New Paradigm for Integrating GIS and Geographic Simulation. In: *Proceedings of the 7th International Conference on GeoComputation, University of Southampton, UK, September 2003*, URL `http://www.geocomputation.org/2003/Papers/Benenson_Paper.pdf`, 2003

Benenson und Torrens 2004 Benenson, Itzhak ; Torrens, Paul M.: *Geosimulation: Automata-Based Modeling of Urban Phenomena*. Chichester : Wiley, 2004

Berg et al. 1997 Berg, Mark d. ; Kreveld, Marc v. ; Overmars, Mark ; Schwarzkopf, Otfried: *Computational Geometry: Algorithms and Applications*. Berlin; Heidelberg : Springer, 1997

Bergenti et al. 2004 Bergenti, Federico (Hrsg.) ; Gleizes, Marie-Pierre (Hrsg.) ; Zambonelli, Franco (Hrsg.): *Methodologies and Software Engineering for Agent Systems : the Agent-Oriented Software Engineering Handbook*. Boston : Kluwer Academic, 2004 (Multiagent systems, artificial societies, and simulated organizations 11)

Berlekamp et al. 1982 Berlekamp, Elwyn R. ; Conway, John H. ; Guy, Richard K.: *Winning Ways for Your Mathematical Plays Volume 2: Games in Particular*. London : Academic Press, 1982

Bernard et al. 2002 Bernard, Lars ; Simonis, Ingo ; Wytzisk, Andreas: Monitoring und Simulation raum-zeitlicher Prozesse in Geodateninfrastrukturen. In: Schubert, Sigrid E. (Hrsg.) ; Reusch, Bernd (Hrsg.) ; Jesse, Norbert (Hrsg.): *Informatik bewegt: Informatik 2002 - 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 30. September - 3.Oktober 2002 in Dortmund*, GI, 2002, S. 756–761

Bernard und Streit 2000 Bernard, Lars ; Streit, Ulrich: Three-dimensional Boundary Layer Modelling and GIS. In: Fotheringham, A. S. (Hrsg.) ; Wegener, Michael (Hrsg.): *Spatial Models and GIS. New Potentials and New Models*. Taylor and Francis, 2000 (GISDATA 7), S. 109–118

Bernon et al. 2005 Bernon, Carole ; Camps, Valerie ; Gleizes, Marie-Pierre ; Picard, Gauthier: Engineering Adaptive Multi-Agent Systems: the ADELFE Methodology. In: (Henderson-Sellers und Giorgini 2005), S. 172–202

Bernon et al. Bernon, Carole ; Gleizes, Marie-Pierre ; Picard, Gauthier ; Glize, Pierre: The ADELFE Methodology for an Intranet System Design. In: *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002 at CAiSE'02)*, CEUR-WS.org (CEUR Workshop Proceedings 57)

Bersini und Detours 1994 Bersini, H. ; Detours, V.: Asynchrony induces stability in cellular automata based models. In: Brooks, Rodney A. (Hrsg.) ; Maes, Patty (Hrsg.): *Artificial Life IV, Proceedings of the Fourth International Conference on Artificial Life*. Cambridge, Massachusetts : MIT Press, 1994, S. 382–387

Bigbee et al. 2007 Bigbee, Anthony ; Cioffi-Revilla, Claudio ; Luke, Sean: Replication of Sugarscape Using MASON. In: Terano, T. (Hrsg.) ; Kita, H. (Hrsg.) ; Deguchi, H. (Hrsg.) ; Kijima, K. (Hrsg.): *Agent-Based Approaches in Economic and Social Complex Systems IV: Post-Proceedings of the AESCS International Workshop 2005*, 2007, S. 183–190

Biggs 2002 Biggs, Norman L.: *Discrete Mathematics*. 2nd ed. Oxford : Oxford University Press, 2002

Bill 2010 Bill, Ralf: *Grundlagen der Geo-Informationssysteme, Band 1: Hardware, Software und Daten*. 5. Aufl. Heidelberg : Wichmann Verlag, 2010

Bill und Zehner 2001 Bill, Ralf ; Zehner, Marco L.: *Lexikon der Geoinformatik*. Heidelberg : Wichmann, 2001. – Auch online verfügbar unter http://www.geoinformatik.uni-rostock.de/lexikon.asp

Bischak und Roberts 1991 Bischak, Diane P. ; Roberts, Stephen D.: Object-Oriented Simulation. In: Nelson, B.L. (Hrsg.) ; Kelton, W.D. (Hrsg.) ; Clark, G.M. (Hrsg.): *1991 Winter Simulation Conference*, 1991, S. 194–203

Bisgambiglia und Franceschini 2013 Bisgambiglia, Paul-Antoine ; Franceschini, Romain: Agent-oriented Approach Based on Discrete Event Systems. In: Wainer, Gabriel A. (Hrsg.) ; Mostermann, Pieter (Hrsg.) ; Barros, Fernando (Hrsg.) ; Zacharewicz, Gregory (Hrsg.): *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (DEVS 2013), Part of Spring Simulation Multi-Conference (SpringSim'13), San Diego, CA*. San Diego, CA : Society for Computer Simulation International, 2013, S. 190–195

Bithell und Macmillan 2007 Bithell, M. ; Macmillan, W.D.: Escape from the cell: Spatially explicit modelling with and without grids. In: *Ecological Modelling* 200 (2007), Nr. 1–2, S. 59–78

Bloch 2008 Bloch, Joshua: *Effective Java: Programming Language Guide*. 2nd ed. Upper Saddle River, NJ : Addison-Wesley, 2008 (The Java Series)

Boer et al. 2007 Boer, Katalin ; Kaymak, Uzay ; Spiering, Jaap: From discrete-time models to continuous-time, asynchronous models of financial markets. In: *Computational Intelligence* 23 (2007), Nr. 2, S. 142–161

Bond und Gasser 1988a Bond, Alan H. ; Gasser, Les: An Analysis of Problems and Research in DAI. In: Bond, Alan H. (Hrsg.) ; Gasser, Les (Hrsg.): *Readings in Distributed Artificial Intelligence*. San Mateo, CA : Morgan Kaufmann, 1988, S. 3–35

Bond und Gasser 1988b Bond, Alan H. ; Gasser, Les: *Readings in Distributed Artificial Intelligence*. San Mateo, CA : Morgan Kaufmann, 1988

Bordini et al. 2005 Bordini, Rafael H. (Hrsg.) ; Dastani, Mehdi (Hrsg.) ; Dix, Jürgen (Hrsg.) ; El Fallah Seghrouchni, Alam (Hrsg.): *Multi-Agent Programming: Languages, Platforms and Applications*. New York : Springer, 2005 (Multiagent Systems, Artificial Societies, and Simulated Organizations 15)

Bordini und Dix 2013 Bordini, Rafael H. ; Dix, Jürgen: Programming Multiagent Systems. In: (Weiss 2013), S. 587–640

Boulaire et al. 2013 Boulaire, Fanny ; Utting, Mark ; Drogemuller, Robin: Parallel ABM for electricity distribution grids: A case study. In: Scarano, V. (Hrsg.) ; Cordasco, G. (Hrsg.) ; Chiara, R. D. (Hrsg.) ; Erra, U. (Hrsg.): *Proceedings of the 1st Workshop on Parallel and Distributed Agent-Based Simulations (PADABS), Satellite Workshop of Euro-Par 2013, 26-30 August 2013, Aachen, Germany*, 2013

Bousquet 2001 Bousquet, François: *SugarScape*. Centre de coopération internationale en recherche agronomique pour le développement (CIRAD). 2001. – URL `http://cormas.cirad.fr/en/applica/sugarScape.htm`

Bradshaw 1997 Bradshaw, Jeffrey: An Introduction to Software Agents. In: Bradshaw, Jeffrey (Hrsg.): *Software Agents*. AAAI Press; The MIT Press, 1997, S. 3–46

Braitenberg 1986 Braitenberg, Valentino: *Vehicles: Experiments in Synthetic Psychology*. Cambridge, MA : MIT Press, 1986

Bresciani et al. 2004 Bresciani, Paolo ; Giorgini, Paolo ; Giunchiglia, Fausto ; Mylopoulos, John ; Perini, Anna: Tropos: An Agent-Oriented Software Development Methodology. In: *Journal of Autonomous Agents and Multi-Agent Systems* 8 (2004), Nr. 3, S. 203–236

Brooks 1986 Brooks, Rodney: A Robust Layered Control System for a Mobile Robot. In: *IEEE Journal of Robotics and Automation* 2 (1986), Nr. 1, S. 14–23

Brooks 1991a Brooks, Rodney: Intelligence without Reason. In: Myopoulos, John (Hrsg.) ; Reiter, Ray (Hrsg.): *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI '91), August 1991, Sydney, Australia*. San Mateo, CA : Morgan Kaufman, 1991, S. 569–595

Brooks 1991b Brooks, Rodney: Intelligence without Representation. In: *AI* 47 (1991), Nr. 1-3, S. 139–159

Brown et al. 2005 Brown, Daniel G. ; Riolo, Rick ; Robinson, Derek T. ; North, Michael ; Rand, William: Spatial process and data models: Toward integration of agent-based models and GIS. In: *Journal of Geographical Systems* 7 (2005), Nr. 1, S. 25–47. – URL http://www.cscs.umich.edu/research/projects/sluce/publications/jgs_abmgis.pdf

Burkhard 1998 Burkhard, Hans-Dieter: Einführung in die Agenten-Technologie. In: *it+ti* 40 (1998), Nr. 4, S. 6–11

Burkhart 2006 Burkhart, Helmar: Parallele Programmierung. In: (Rechenberg und Pomberger 2006), S. 633–662

Burrough et al. 2013 Burrough, Peter A. ; McDonnell, Rachael A. ; Lloyd, Christopher D.: *Principles of Geographical Information Systems*. 3. ed. Oxford : Oxford University Press, 2013

Buss und Sánchez 2005 Buss, Arnold H. ; Sánchez, Paul J.: Simple Movement and Detection in Discrete Event Simulation. In: Kuhl, M. E. (Hrsg.) ; Steiger, N. M. (Hrsg.) ; Armstrong, F. B. (Hrsg.) ; Joines, J. A. (Hrsg.): *Proceedings of the 2005 Winter Simulation Conference*, 2005, S. 992–1000

Caire et al. 2001 Caire, Giovanni ; Coulier, Wim ; Garijo, Francisco ; Gomez, Jorge ; Pavon, Juan ; Leal, Francisco ; Chainho, Paulo ; Kearney, Paul ; Stark, Jamie ; Evans, Richard ; Massonet, Philippe: Agent Oriented Analysis Using MESSAGE/UML. In: *Agent-Oriented Software Engineering II: Second International Workshop AOSE 2001, Montreal, Canada, 29 May 2001*. Berlin : Springer, 2001 (LNCS 2222), S. 119–135

Cervenka und Trencansky 2007 Cervenka, Radovan ; Trencansky, Ivan: *The Agent Modeling Language – AML: A Comprehensive Approach to Modeling Multi-Agent Systems*. Basel : Birkhäuser, 2007

Chan et al. 2010 Chan, Wai Kin V. ; Son, Young-Jun ; Macal, Charles M.: Agent-based simulation tutorial – Simulation of emergent behaviour and differences between agent-based simulation and discrete-event simulation. In: Johansson, B. (Hrsg.) ; Jain, S. (Hrsg.) ; Montoya-Torres, J. (Hrsg.) ; Hugan, J. (Hrsg.) ; Yücesan, E. (Hrsg.): *Proceedings of the 2010 Winter Simulation Conference*, 2010, S. 135–150

Chow 1996 Chow, Alex C.-H.: Parallel DEVS: a parallel, hierarchical, modular modeling formalism and its distributed simulator. In: *Transactions of the Society for Computer Simulation International* 13 (1996), Nr. 2, S. 55–67

Christen und Franklin 2002 Christen, Markus ; Franklin, Laura R.: The Concept of Emergence in Complexity Science: Finding Coherence between Theory and Practice. In: *Proceedings of the SFI Complex Systems Summer School, June 9 – July 6, 2002, Santa Fe, NM*, URL http://www.ini.uzh.ch/admin/extras/doc_get.php?id=41950, 2002

Claassen 1999 Claassen, Sönke: *Erweiterung des Simulationsframework DESMO-J um höhere Simulationskonstrukte und Statistikfunktionen*, Universität Hamburg, Fachbereich Informatik, Studienarbeit, 1999

Claassen 2001 Claassen, Sönke: *Entwicklung anwendungsspezifischer Simulationskomponenten im Rahmen eines Simulationsframeworks*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 2001

Claramunt et al. 1997 Claramunt, Christophe ; Parent, Christine ; Thériault, Marius: Design Patterns for Spatio-temporal Processes. In: Spaccapietra, S. (Hrsg.) ; Maryanski, F. (Hrsg.): *Searching for Semantics: Data Mining, Reverse Engineering*. Chapman & Hall, 1997, S. 415–428

Clark und Rose 1997 Clark, Mark E. ; Rose, Kenneth A.: Individual-based model of stream-resident rainbow trout and brook char: model description, corroboration, and effects of sympatry and spawning season duration. In: *Ecological Modelling* 94 (1997), Nr. 2, S. 157–175

Cohen et al. 1989 Cohen, Paul R. ; Greenberg, Michael L. ; Hart, David M. ; Howe, Adele E.: Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. In: *AI Magazine* 10 (1989), Nr. 3, S. 32–48

Cook 2004 Cook, Matthew: Universality in Elementary Cellular Automata. In: *Complex Systems* 15 (2004), Nr. 1, S. 1–40

Corkill 2003 Corkill, Daniel D.: Collaborating Software: Blackboard and Multi-Agent Systems & the Future. In: *Proceedings of the International Lisp Conference, 12-15 October 2003, New York*, 2003

Cornforth et al. 2002 Cornforth, David ; Green, David G. ; Newth, David ; Kirley, Michael: Do Artificial Ants March in Step? Ordered Asynchronous Processes and Modularity in Biological Systems. In: Standish, Russell (Hrsg.) ; Bedau, Mark A. (Hrsg.) ; Abbass, Hussein (Hrsg.): *Artificial Life VIII, Proceedings of the Eighth International Conference on Artificial Life*. Cambridge, Massachusetts : MIT Press, 2002, S. 28–32

Cossentino und Potts 2002 Cossentino, Massimo ; Potts, Colin: A CASE tool supported methodology for the design of multi-agent systems. In: *Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP'02), Las Vegas, June 24–27*, CSREA Press, 2002, S. 315–321

Cota und Sargent 1992 Cota, Bruce A. ; Sargent, Robert G.: A Modification of the Process Interaction World View. In: *ACM Transactions on Modeling and Computer Simulation* 2 (1992), Nr. 2, S. 109–129

Crosbie 1982 Crosbie, R. E.: Interactive and Real-Time Simulation. In: Cellier, Francois E. (Hrsg.): *Progress in Modelling and Simulation*. London : Academic Press, 1982, S. 393–406

Czogalla et al. 2006 Czogalla, Rainer ; Knaak, Nicolas ; Page, Bernd: Simulating the Eclipse Way: A Generic Experimentation Environment Based on the Eclipse Platform. In: Borutzky, W. (Hrsg.) ; Orsoni, A. (Hrsg.) ; Zobel, R. (Hrsg.): *Proceedings of the 20th European Conference on Modelling and Simulation (ECMS 2006), Bonn, September 2006*, 2006, S. 260–265

Czogalla und Matzen 2003 Czogalla, Rainer ; Matzen, Bente: *Entwicklung eines Werkzeugs zur agentenbasierten Simulation von Personenbewegungen innerhalb einer Flugzeugkabine*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 2003

Dahl und Nygaard 1966 Dahl, Ole-Johan ; Nygaard, Kristen: SIMULA: An ALGOL-based Simulation Language. In: *Communications of the ACM* 9 (1966), Nr. 9, S. 671–678

Daniel 2006 Daniel, Gilles: *Asynchronous Simulations of a Limit Order Book*, University of Manchester, Faculty of Science and Engineering, Dissertation, 2006

Davidsson 2001 Davidsson, Paul: MABS: Beyond Social Simulation. In: (Moss und Davidsson 2001), S. 97–107

Davidsson et al. 2007 Davidsson, Paul ; Holmgren, Johan ; Kyhlbäck, Hans ; Mengistu, Dawit ; Persson, Marie: Applications of Agent Based Simulation. In: Antunes, Luis (Hrsg.) ; Takadama, Keiki (Hrsg.): *Multi-Agent-Based Simulation VII; International Workshop, MABS 2006, Hakodate, Japan, May 8, 2006; Revised and Invited Papers*. Berlin, Heidelberg : Springer, 2007 (Lecture Notes in Artificial Intelligence; 4442), S. 15–27

Davis und Smith 1983 Davis, Randall ; Smith, Reid G.: Negotiation as a Metaphor for Distributed Problem Soving. In: *AI* 20 (1983), Nr. 1, S. 63–109

DeAngelis und Gross 1992 DeAngelis, Donald L. ; Gross, Louis J.: *Individual-based models and approaches in ecology – populations, communities, and ecosystems*. New York : Chapman & Hall, 1992

Deecke et al. 2000 Deecke, Helmut ; Meyer, Ruth ; Page, Bernd ; Reick, Christian: Erster Zwischenbericht des Forschungsprojekts Nachhaltige Logistikkonzepte für Stadtkurierdienste: Zusammenfassung der Unternehmensgespräche / Universität Hamburg. 2000 – Interner Bericht.

DeLoach 2004 DeLoach, Scott: The MaSE Methodology. In: (Bergenti et al. 2004), S. 107–126

DeLoach und Garcia-Ojeda 2010 DeLoach, Scott A. ; Garcia-Ojeda, Juan C.: O-MaSe: a customizable approach to designing and building complex, adaptive multi-agent systems. In: *International Journal of Agent-Oriented Software Engineering* 4 (2010), Nr. 3, S. 244–280

Denz 2013 Denz, Nicolas: *Process-Oriented Analysis and Validation of Multi-Agent-Based Simulations*, Universität Hamburg, Fakultät für Mathematik, Informatik und Naturwissenschaften, Dissertation, 2013

Deursen 1995 Deursen, W. P. A. v.: *Geographical Information Systems and Dynamic Models – Development and application of a prototype spatial modelling language*, Faculty of Spatial Sciences, University of Utrecht, Ph.D. Thesis, 1995. – URL `http://www.carthago.nl/__miracle/doc/thesis1.pdf`. – NGS Publication 190

Dijkstra 1959 Dijkstra, Edsger W.: A Note on Two Problems in Connexion with Graphs. In: *Numerische Mathematik* 1 (1959), Nr. 1, S. 269–271

d'Inverno et al. 1997 d'Inverno, Mark ; Kinny, David ; Luck, Michael ; Wooldridge, Michael: A Formal Specification of dMARS. In: *ATAL'97: Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*. Berlin : Springer, 1997 (LNAI 1365), S. 155–176

Dix und Fisher 2013 Dix, Jürgen ; Fisher, Michael: Specification and Verification of Multiagent Systems. In: (Weiss 2013), S. 641–693

Dolk et al. 2001 Dolk, Daniel R. ; Scholl, Hans J. ; Chaturvedi, Alok ; Dickieson, Jan: Agent-based Simulation and System Dynamics. In: *Proceedings of the 34th Annual Hawaii International Conference on Systems Sciences (HICSS-34), Volume 3*, IEEE Computer Society, 2001, S. 3002. – URL `http://www.computer.org/csdl/proceedings/hicss/2001/0981/03/09813002.pdf`

Douglass 1999 Douglass, Bruce P.: *Doing Hard Time*. Reading, MA : Addison-Wesley, 1999

Doyle und Reed 2001 Doyle, Allan ; Reed, Carl: Introduction to OGC Web Services / Open GIS Consortium. URL `http://portal.opengeospatial.org/files/?artifact_id=14973`, May 2001 – An OGC White Paper.

Drogoul und Ferber 1994 Drogoul, Alexis ; Ferber, Jacques: Multi-Agent Simulation as a Tool for Modeling Societies: Application to Social Differentiation in Ant Colonies. In: Castelfranchi, Cristiano (Hrsg.) ; Werner, Eric (Hrsg.): *Artificial Social Systems: 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'92, S. Martino al Cimino, Italy, July 1992; Selected Papers*. Berlin; Heidelberg; New York : Springer, 1994 (Lecture Notes in Artificial Intelligence 830), S. 3–23

Drogoul et al. 2003 Drogoul, Alexis ; Vanbergue, Diane ; Meurisse, Thomas: Multi-Agent Based Simulation: Where are the Agents? In: Sichman, Jaime S. (Hrsg.) ; Bousquet, François (Hrsg.) ; Davidsson, Paul (Hrsg.): *Multi-Agent-Based Simulation II : Third International Workshop, MABS 2002, Bologna, Italy, July 15-16, 2002. Revised Papers*. Berlin; Heidelberg; New York : Springer, 2003 (Lecture Notes in Artificial Intelligence 2581), S. 1–15

Duboz et al. 2006 Duboz, Raphael ; Versmisse, David ; Quesnel, Gauthier ; Muzzy, Alexandre ; Ramat, Eric: Specification of Dynamic Structure Discrete Event Multiagent Systems. In: Ören, Tuncer (Hrsg.) ; Madey, Gregory (Hrsg.) ; Yilmaz, Levent (Hrsg.): *Proceedings of Agent-Directed Simulation (ADS 2006), Part of Spring Simulation Multiconference, Huntsville, AL*, 2006

Durrett 1999 Durrett, Rick: Stochastic Spatial Models. In: *SIAM Review* 41 (1999), Nr. 4, S. 677–718

Edmonds 2001 Edmonds, Bruce: The Use of Models – Making MABS More Informative. In: Moss, Scott (Hrsg.) ; Davidsson, Paul (Hrsg.): *Multi-Agent-Based Simulation, Second International Workshop, MABS 2000, Boston, MA, Revised and Additional Papers* Bd. 1979. Berlin; Heidelberg : Springer, 2001, S. 15–32

Edmonds und Meyer 2013 Edmonds, Bruce (Hrsg.) ; Meyer, Ruth (Hrsg.): *Simulating Social Complexity: A Handbook*. Berlin : Springer, 2013 (Understanding Complex Systems)

Ellis und Gibbs 1989 Ellis, Clarence A. ; Gibbs, Simon J.: Active Objects: Realities and Possibilities. In: Kim, Won (Hrsg.) ; Lochovsky, Frederick H. (Hrsg.): *Object-Oriented Concepts, Databases, and Applications*. New York : ACM Press, 1989, S. 561–572

Epstein 2007 Epstein, Joshua M.: *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton, N.J. : Princeton University Press, 2007

Epstein und Axtell 1996 Epstein, Joshua M. ; Axtell, Robert: *Growing Artificial Societies : Social Science from the Bottom Up*. Washington, D.C. / Cambridge, MA : Brookings Institution Press / The MIT Press, 1996

Etienne et al. 2003 Etienne, Michel ; LePage, Christophe ; Cohen, Mathilde: A Step-by-step Approach to Building Land Management Scenarios Based on Multiple Viewpoints on Multi-agent System Simulations. In: *Journal of Artificial Societies and Social Simulations* 6 (2003), Nr. 2. – URL `http://jasss.soc.surrey.ac.uk/6/2/2.html`

Fedra 1996 Fedra, Kurt: Distributed Models and Embedded GIS: Strategies and Case Studies of Integration. In: Goodchild, Michael F. (Hrsg.) ; Steyaert, Louis T. (Hrsg.) ; Parks, Bradley O. (Hrsg.) ; Johnston, Carol (Hrsg.) ; Maidment, David R. (Hrsg.) ; Crane, Michael P. (Hrsg.) ; Glendinning, Sandi (Hrsg.): *GIS and Environmental Modeling: Progress and Research Issues*. Chichester : Wiley, 1996, S. 413–417

Feng und Feng 1996 Feng, Huijun ; Feng, Yuncheng: Towards an Object-oriented Simulation World View. In: *Systems Science and Systems Engineering* 5 (1996), Nr. 3, S. 319–326

Ferber 1999 Ferber, Jacques: *Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence*. Harlow, UK : Addison-Wesley, 1999

Ferber und Gutknecht 1998 Ferber, Jacques ; Gutknecht, Olivier: A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. In: *Proceedings of the Third International Conference on Multi Agent Systems (ICMAS-98)*. Paris, July 4 - 7 1998, S. 128–135

Ferber et al. 2004 Ferber, Jacques ; Gutknecht, Olivier ; Michel, Fabien: From Agents to Organizations: An Organizational View of Multi-Agent Systems. In: Giorgini, P. (Hrsg.) ; Müller, J. (Hrsg.) ; Odell, J. (Hrsg.): *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, 15 July 2003; Revised Papers*. Berlin : Springer, 2004 (LNCS 2935), S. 214–230

Ferber und Müller 1996 Ferber, Jacques ; Müller, Jean-Pierre: Influences and Reaction: A Model of Situated Multiagent Systems. In: Tokoro, M. (Hrsg.): *Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)*, AAAI Press, 1996, S. 72–79

Ferguson 1992 Ferguson, Innes A.: *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. Cambridge, UK, University of Cambridge, Dissertation, 1992. – Also: Technical Report 273, Computer Laboratory, University of Cambridge, UK, November 1992

Fishman 1973 Fishman, George S.: *Concepts and Methods in Discrete Event Digital Simulation*. New York : Wiley, 1973

Flache und Hegselmann 2001 Flache, Andreas ; Hegselmann, Rainer: Do Irregular Grids make a Difference? Relaxing the Spatial Regularity Assumption in Cellular Models of Social Dynamic. In: *Journal of Artificial Societies and Social Simulation* 4 (2001), Nr. 4. – URL `http://jasss.soc.surrey.ac.uk/4/4/6.html`

Forgy 1982 Forgy, Charles L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: *Artificial Intelligence* 19 (1982), Nr. 1, S. 17–37

Fowler 2004 Fowler, Martin: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed. Reading, MA : Addison-Wesley, 2004 (Object Technology Series)

Franklin und Graesser 1996 Franklin, Stan ; Graesser, Art: Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. In: *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer, 1996 (Intelligent Agents III), S. 21–35

Frisch et al. 1986 Frisch, Uriel ; Hasslacher, Brosl ; Pomeau, Yves: Lattice-Gas Automata for the Navier-Stokes Equation. In: *Physical Review Letters* 56 (1986), Nr. 14, S. 1505–1508

Fujimoto 1990 Fujimoto, Richard: Parallel Discrete Event Simulation. In: *Communications of the ACM* (1990), S. 30–53

Fujimoto 1998 Fujimoto, Richard M.: Time Management in the High Level Architecture. In: *Simulation* 71 (1998), Nr. 6, S. 388–400. – URL `http://www.cc.gatech.edu/computing/pads/PAPERS/Time_mgmt_High_Level_Arch.pdf`

Gamma et al. 1994 Gamma, Erich ; Helm, Richard ; Johnson, Ralph E. ; Vlissides, John M.: *Design Patterns - Elements of Reusable Object-Oriented Software*. Reading, MA : Addison-Wesley, 1994

Gardner 1970 Gardner, Martin: The fantastic combinations of John Conway's new solitaire game of "Life". In: *Scientific American* 223 (1970), S. 120–123

Gasser et al. 1987 Gasser, Les ; Braganza, Carl ; Herman, Nava: MACE: A Flexible Testbed for Distributed AI Research. In: Huhns, Michael J. (Hrsg.): *Distributed Artificial Intelligence*. San Mateo, CA : Morgan Kaufmann, 1987, S. 119–152

Gehlsen 2004 Gehlsen, Björn: *Automatisierte Experimentplanung im Rahmen von Simulationsstudien: Konzeption und Realisierung eines verteilten simulationsbasierten Optimierungssystems*, Universität Hamburg, Fachbereich Informatik, Dissertation, 2004

Gehlsen und Page 2001 Gehlsen, Björn ; Page, Bernd: A Framework for Distributed Simulation Optimization. In: *Proceedings of the 2001 Winter Simulation Conference*. Arlington, VA, 2001, S. 508–514

Genesereth und Nilsson 1987 Genesereth, Michael R. ; Nilsson, Nils J.: *Logical Foundations of Artificial Intelligence*. Los Altos, CA : Morgan Kaufman, 1987

Georgeff und Ingrand 1989 Georgeff, Michael P. ; Ingrand, François Félix: Decision-Making in an Embedded Reasoning System. In: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*. Detroit, Mich., August 20-25 1989, S. 972–978

Georgeff und Lansky 1987 Georgeff, Michael P. ; Lansky, Amy L.: Reactive Reasoning and Planning: An Experiment with a Mobile Robot. In: *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*. Seattle, Wa., July 13-17 1987, S. 677–682

Gerken und Meyer 1995 Gerken, Barbara ; Meyer, Ruth: *Geographische Informationssysteme – Eine Einführung für Informatiker*, Universität Hamburg, Fachbereich Informatik, Studienarbeit, 1995. – Auch veröffentlicht als Mitteilung Nr. 249 des Fachbereichs Informatik der Universität Hamburg, FBI-HH-M-249/95.

Gignoux und Visco 2005 Gignoux, Sebastien ; Visco, Keith: *Castor XML Mapping*. 2005. – URL `http://castor.codehaus.org/1.2/xml-mapping.html`

Gilbert und Troitzsch 2005 Gilbert, Nigel ; Troitzsch, Klaus G.: *Simulation for the Social Scientist*. 2. ed. Maidenhead : Open University Press, 2005

Gimblett 2002 Gimblett, H. R. (Hrsg.): *Integrating Geographic Information Systems and Agent-Based Modeling Techniques for Simulating Social and Ecological Processes*. Oxford : University Press, 2002 (Santa Fe Institute Studies in the Sciences of Complexity)

Giorgini und Henderson-Sellers 2005 Giorgini, Paolo ; Henderson-Sellers, Brian: Agent-Oriented Methodologies: An Introduction. In: (Henderson-Sellers und Giorgini 2005), S. 1–19

Gold et al. 1997 Gold, Christopher M. ; Remmele, Peter R. ; Roos, Thomas: Voronoi methods in GIS. In: Nievergeld, J. (Hrsg.) ; Roos, T. (Hrsg.) ; Kreveld, M. van (Hrsg.) ; Widmeyer, P. (Hrsg.): *Algorithmic Foundations of GIS*. Berlin : Springer, 1997 (Lecture Notes in Computer Science 1340), S. 21–35

Grimm 1999 Grimm, Volker: Ten years of individual-based modelling in ecology: What have we learned and what could we learn in the future? In: *Ecol. Model.* 115 (1999), Nr. 2, S. 129–148

Grimm et al. 2006 Grimm, Volker ; Berger, Uta ; Bastiansen, Finn ; Eliassen, Sigrunn ; Ginot, Vincent ; Giske, Jarl ; Goss-Custard, John ; Grand, Tamara ; Heinz, Simone K. ; Huse, Geir ; Huth, Andreas ; Jepsen, Jane U. ; Jorgensen, Christian ; Mooi, Wolf M. ; Müller, Birgit ; Pe'er, Guy ; Piou, Cyril ; Railsback, Steven F. ; Robbins, Andrew M. ; Robbins, Martha M. ; Rossmanith, Eva ; Rüger, Nadja ; Strand, Espen ; Souissi, Sami ; Stillman, Richard A. ; Vabø, Rune ; Visser, Ute ; DeAngelis, Donald L.: A standard protocol for describing individual-based and agent-based models. In: *Ecological Modelling* 198 (2006), S. 115–126

Grimm et al. 2010 Grimm, Volker ; Berger, Uta ; DeAngelis, Donald L. ; Polhill, J. G. ; Giske, Jarl ; Railsback, Steven F.: The ODD protocol: A review and first update. In: *Ecological Modelling* 221 (2010), S. 2760–2768

Grimm et al. 2013 Grimm, Volker ; Polhill, Gary ; Touza, Julia: Documenting Social Simulation Models: The ODD Protocol as a Standard. In: (Edmonds und Meyer 2013), S. 117–133

Grimm und Railsback 2005 Grimm, Volker ; Railsback, Steven F.: *Individual-based modeling and ecology*. Princeton, NJ : Princeton University Press, 2005 (Princeton series in theoretical and computational biology)

Griss et al. 2003 Griss, Martin L. ; Fonseca, Steven ; Cowan, Dick ; Kessler, Robert: Using UML State Machine Models for More Precise and Flexible JADE Agent Behaviours. In: Giunchiglia, F. (Hrsg.) ; Odell, J. (Hrsg.) ; Weiss, G. (Hrsg.): *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, 15 July 2002; Revised Papers and Invited Contributions*. Berlin : Springer, 2003 (LNCS 2585), S. 113–125

Gronewold und Sonnenschein 1998 Gronewold, Anja ; Sonnenschein, Michael: Event-based modelling of ecological systems with asynchronous cellular automata. In: *Ecological Modelling* 108 (1998), S. 37–52

Gross 1996 Gross, Louis J.: Individual-based ecological models for spatially-explicit investigation and computational ecology. In: *Third Autumn Workshop on Mathematical Ecology*. International Centre for Theoretical Physics, Trieste, Italy, October 1996

Guessoum 2000 Guessoum, Zahia: A Multi-Agent Simulation Framework. In: *Transactions of the SCS* 17 (2000), Nr. 1, S. 2–11

Guessoum und Briot 1999 Guessoum, Zahia ; Briot, Jean-Pierre: From Active Objects to Autonomous Agents. In: *IEEE Concurrency* 7 (1999), Nr. 3, S. 68–76

Gulyás et al. 1999 Gulyás, László ; Kozsik, Tamás ; Corliss, John B.: The Multi-Agent Modelling Language and the Model Design Interface. In: *JASSS* 2 (1999), Nr. 3. – URL `http://jasss.soc.surrey.ac.uk/2/3/8.html`

Gutknecht et al. 2000 Gutknecht, Olivier ; Ferber, Jacques ; Michel, Fabien: The MadKit Agent Platform Architecture / LIRM, Université Montpellier. URL `http://www.madkit.net/documents/others/MadkitTechnicalReport.pdf`, 2000 – Rapport de Recherche, LIRMM 00061.

Gutowitz 1991 Gutowitz, Howard (Hrsg.): *Cellular Automata: Theory and Experiment*. Cambridge, MA : MIT Press, 1991 (Special issues of Physica D)

Hammond und Axelrod 2006 Hammond, Ross A. ; Axelrod, Robert: The Evolution of Ethnocentrism. In: *Journal of Conflict Resolution* 50 (2006), Nr. 6, S. 926–936

Hanks et al. 1993 Hanks, Steve ; Pollack, Martha E. ; Cohen, Paul R.: Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures. In: *AI Magazine* 14 (1993), Nr. 4, S. 17–42

Hanski und Gilpin 1991 Hanski, I. ; Gilpin, M.: Metapopulation dynamics: brief history and conceptual domain. In: *Biological Journal of the Linnean Society* 42 (1991), Nr. 1–2, S. 3—-16

Harel 1987 Harel, David: Statecharts: A Visual Formalism for Complex Systems. In: *Science of Computer Programming* 8 (1987), Nr. 3, S. 231–274

Heath und Hill 2010 Heath, Brian L. ; Hill, Raymond R.: Some insights into the emergence of agent-based modelling. In: *Journal of Simulation* 4 (2010), September, Nr. 3, S. 163–169

Hegselmann 1996 Hegselmann, Rainer: Cellular automata in the social sciences: Perspectives, restrictions and artefacts. In: Hegselmann, R. (Hrsg.) ; Mueller, U. (Hrsg.) ; Troitzsch, K. G. (Hrsg.): *Modelling and simulation in the social sciences from a philosophy of science point of view*. Dordrecht : Kluwer, 1996, S. 209–234

Hegselmann und Flache 1998 Hegselmann, Rainer ; Flache, Andreas: Understanding Complex Social Dynamics – A Plea For Cellular Automata Based Modelling. In: *Journal of Artificial Societies and Social Simulation* 1 (1998), Nr. 3. – URL `http://jasss.soc.surrey.ac.uk/1/3/1.html`

Helleboogh et al. 2007 Helleboogh, Alexander ; Vizzari, Giuseppe ; Uhrmacher, Adelinde ; Michel, Fabien: Modeling Dynamic Environments in Multi-Agent Simulation. In: *Autonomous Agents and Multi-Agent Systems* 14 (2007), Nr. 1, S. 87–116

Henderson-Sellers und Giorgini 2005 Henderson-Sellers, Brian (Hrsg.) ; Giorgini, Paolo (Hrsg.): *Agent-oriented Methodologies*. Hershey, PA : Idea Group Publ., 2005

Heppenstall et al. 2012 Heppenstall, Alison J. (Hrsg.) ; Crooks, Andrew T. (Hrsg.) ; See, Linda M. (Hrsg.) ; Batty, Michael (Hrsg.): *Agent-Based Models of Geographical Systems*. Dordrecht : Springer, 2012

Hewitt 1977 Hewitt, Carl: Viewing Control Structures as Patterns of Message Passing. In: *AI* 8 (1977), Nr. 3, S. 323–374

Hewitt 1986 Hewitt, Carl: Offices are Open Systems. In: *TOIS* 4 (1986), Nr. 3, S. 271–287

Heywood et al. 2011 Heywood, Ian ; Cornelius, Sarah ; Carver, Steve: *An Introduction to Geographic Information Systems*. 4. ed. Harlow : Pearson Education / Prentice Hall, 2011

Hill 1996 Hill, David R. C.: *Object-oriented analysis and simulation*. Harlow : Addison-Wesley, 1996

Hilty et al. 1998 Hilty, Lorenz M. ; Page, Bernd ; Meyer, Ruth ; Mügge, Holger ; Deecke, Helmut ; Reick, Christian H. ; Gehlsen, Björn ; Hupf, Martin ; Becken, Olaf ; Bosselmann, Michael ; Neumann, Marc ; Poll, Martina ; Lechler, Tim ; Böttger, Thorsten: Instrumente für die ökologische Bewertung und Gestaltung von Verkehrs- und Logistiksystemen: Abschlussbericht des Forschungsprojekts MOBILE / Fachbereich Informatik, Universität Hamburg; FAW Ulm. URL `http://mobile-www.informatik.uni-hamburg.de/MOBILE/Abschlussbericht/Aufbau.html`, 1998 – Forschungsbericht.

Himmelspach und Uhrmacher 2007 Himmelspach, Jan ; Uhrmacher, Adelinde M.: Plug'n Simulate. In: *Proceedings of the 40th Annual Simulation Symposium (ANSS-40 2007), 26-28 March 2007, Norfolk, VA*, IEEE Computer Society, 2007, S. 137–143

Hogeweg 1988 Hogeweg, P.: Cellular Automata as a Paradigm for Ecological Modeling. In: *Applied mathematics and computation* 27 (1988), S. 81–100

Hogeweg und Hesper 1990 Hogeweg, P. ; Hesper, B.: Individual-Oriented Modelling in Ecology. In: *Mathl Comput. Modelling* 13 (1990), Nr. 6, S. 83–90

Hogeweg 2010 Hogeweg, Paulien: Multilevel Cellular Automata as a Tool for Studying Bioinformatic Processes. In: Hoekstra, A.G. (Hrsg.) ; Kroc, J. (Hrsg.) ; Sloot, P.M.A. (Hrsg.): *Simulating Complex Systems by Cellular Automata*. Berlin : Springer, 2010 (Understanding Complex Systems), S. 19–28

Holland 1995 Holland, John H.: *Hidden Order: How Adaptation Builds Complexity*. Cambridge, MA : Helix Books, 1995

Holland 2006 Holland, John H.: Studying Complex Adaptive Systems. In: *Journal of Systems Science and Complexity* 19 (2006), Nr. 1, S. 1–8

Holler und Illing 2009 Holler, Manfred J. ; Illing, Gerhard: *Einführung in die Spieltheorie*. 7. Aufl. Berlin : Springer, 2009 (Springer-Lehrbuch)

Horling und Lesser 2005 Horling, Bryan ; Lesser, Victor: A Survey of Multi-Agent Organizational Paradigms. In: *Knowledge Engineering Review* 19 (2005), Nr. 4, S. 281–316

Huang und Iyer 1998 Huang, Yiqing ; Iyer, Ravishankar K.: An Object-Oriented Environment for Fast Simulation Using Compiler Techniques. In: Medeiros, D.J. (Hrsg.) ; Watson, E.F. (Hrsg.) ; Carson, J.S. (Hrsg.) ; Manivannan, M.S. (Hrsg.): *Proceedings of the 1998 Winter Simulation Conference*, 1998, S. 531–538

Huber 1999 Huber, Marcus J.: JAM: a BDI-theoretic mobile agent architecture. In: *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents, Seattle, WA*. New York, NY : ACM Press, 1999, S. 236–243

Huberman und Glance 1993 Huberman, Bernardo A. ; Glance, Natalie S.: Evolutionary Games and Computer Simulations. In: *Proceedings of the National Academy of Sciences of the United States of America (PNAS)* 90 (1993), S. 7715—-18

Huhns und Singh 1998 Huhns, Michael N. ; Singh, Munindar P.: *Readings in agents*. San Francisco : Kaufmann, 1998

Huhns und Stephens 1999 Huhns, Michael N. ; Stephens, Larry M.: Multiagent Systems and Societies of Agents. In: (Weiss 1999), S. 79–120

Huston et al. 1988 Huston, Michael ; DeAngelis, Donald L. ; Post, Wilfred: New Computer Models Unify Ecological Theory. In: *BioScience* 38 (1988), Nr. 10, S. 682–691

Iglesias et al. 1999 Iglesias, Carlos A. ; Garijo, Mercedes ; González, José C.: A Survey of Agent-oriented Methodologies. In: Müller, Jörg P. (Hrsg.) ; Singh, Munindar P. (Hrsg.) ; Rao, Anand S. (Hrsg.): *Intelligent Agents V: Agent Theories, Architectures, and Languages; 5th International Workshop, ATAL'98, Paris, France, July 1998, Proceedings*. Berlin; Heidelberg; New York : Springer, 1999 (Lecture Notes in Artificial Intelligence 1555), S. 317–330

Iglesias et al. 1998 Iglesias, Carlos A. ; Garijo, Mercedes ; Gonzalez, Jose C. ; Velasco, Juan R.: Analysis and Design of Multiagent Systems using MAS-CommonKADS. In: *Intelligent Agents IV: Agent Theories, Architectures, and Languages* Bd. 1365. Berlin, Heidelberg : Springer, 1998, S. 313–327

INFORMS Simulation Society 2013 *Winter Simulation Conference Archive*. 2013. – URL `http://informs-sim.org`. – Zuletzt aufgerufen am 21.12.2013.

Ingrand et al. 1996 Ingrand, F.F. ; Chatila, R. ; Alami, R. ; Robert, F.: PRS: A high level supervision and control language for autonomous mobile robots. In: *IEEE International Conference on Robotics and Automation, 22–28 April 1996, Minneapolis, MN* Bd. 1, 1996, S. 43–49

Ingrand et al. 1992 Ingrand, François Félix ; Georgeff, Michael P. ; Rao, Anand S.: An Architecture for Real-Time Reasoning and System Control. In: *IEEE Expert* 7 (1992), Nr. 6, S. 33–44

Itami 2002 Itami, Robert M.: Mobile Agents with Spatial Intelligence. In: (Gimblett 2002), S. 191–210

Ito et al. 2010 Ito, Takayuki (Hrsg.) ; Zhang, Minjie (Hrsg.) ; Robu, Valentin (Hrsg.) ; Fatima, Shaheen (Hrsg.) ; Matsuo, Tokuro (Hrsg.) ; Yamaki, Hirofumi (Hrsg.): *Innovations in Agent-Based Complex Automated Negotiations*. Berlin : Springer, 2010

Jacobs et al. 2004 Jacobs, Bruce I. ; Levy, Kenneth N. ; Markovitz, Harry M.: Financial Market Simulation in the 21st Century. In: *The Journal of Portfolio Management* (2004), Nr. 30th Anniversary Issue, S. 142–151

Jennings 2000 Jennings, Nicholas R.: On Agent-Based Software Engineering. In: *AI* 117 (2000), Nr. 2, S. 277–296

Jennings 2001 Jennings, Nicholas R.: An Agent-Based Approach for Building Complex Software Systems. In: *Communications of the ACM* 44 (2001), Nr. 4, S. 35–41

Jennings et al. 2001 Jennings, Nicholas R. ; Faratin, P. ; Lomuscio, A. R. ; Parsons, S. ; Wooldridge, Michael J.: Automated Negotiation: Prospects, Methods and Challenges. In: *International Journal of Group Decision and Negotiation* 10 (2001), Nr. 2, S. 199–215

Jennings et al. 1998 Jennings, Nicholas R. ; Sycara, Katia ; Wooldridge, Michael J.: A Roadmap of Agent Research and Development. In: *Autonomous Agents and Multi-Agent Systems* 1 (1998), Nr. 1, S. 7–38

Jessen und Valk 1987 Jessen, Eike ; Valk, Rüdiger: *Rechensysteme: Grundlagen der Modellbildung*. Berlin : Springer, 1987 (Studienreihe Informatik)

Juan et al. 2002 Juan, Thomas ; Pearce, Adrian ; Sterling, Leon: ROADMAP: Extending the Gaia Methodology for Complex Open Systems. In: *Proceedings of the 1st ACM Joint Conference on Autonomous Agents and Multiagent Systems*. New York : ACM Press, July 2002, S. 3–10

Judson 1994 Judson, Olivia P.: The rise of the individual-based model in ecology. In: *TREE* 9 (1994), Nr. 1, S. 9–14

Kelton et al. 2010 Kelton, W. D. ; Sadowski, Randall P. ; Swets, Nancy B.: *Simulation with Arena*. 5. ed. Boston : McGraw-Hill Higher Education, 2010

Kiviat 1967 Kiviat, Philip J.: Digital Computer Simulation: Modeling Concepts / RAND Corp. 1967 – RAND Memo, RM-5376-PR.

Kiviat 1969 Kiviat, Philip J.: Digital Computer Simulation: Computer Programming Languages / RAND Corp. 1969 – RAND Memo, RM-5883-PR.

Klügl 2001 Klügl, Franziska: *Multiagentensimulation: Konzepte, Werkzeuge, Anwendungen*. München : Addison-Wesley, 2001 (Agententechnologie)

Klügl 2007 Klügl, Franziska: Towards a formal framework for multi-agent simulation models / Universität Würzburg, Institut für Informatik. April 2007 – Technical report, 412.

Klügl et al. 2006 Klügl, Franziska ; Herrler, Rainer ; Fehler, Manuel: SeSAm: Implementation of agent-based simulation using visual programming. In: Nakashima, H. (Hrsg.) ; Wellman, M. P. (Hrsg.) ; Weiss, G. (Hrsg.) ; Stone, P. (Hrsg.): *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), 8-12 May 2006, Hakodate, Japan*, ACM, 2006, S. 1439–1440

Klügl et al. 2004 Klügl, Franziska ; Oechslein, Christoph ; Puppe, Frank ; Dornhaus, Anna: Multi-Agent Modelling in Comparison to Standard Modelling. In: *Simulation News Europe* 40 (2004), S. 3–9

Knaak 2002 Knaak, Nicolas: *Konzepte der agentenbasierten Simulation und ihre Umsetzung im Rahmen des Simulationsframeworks DESMO-J*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 2002

Knaak 2004 Knaak, Nicolas: Modifications of the Fujaba Statechart Interpreter for Multiagent-Based Discrete Event Simulation. In: *Proceedings of the 2004 Fujaba Days*. Technische Universität Darmstadt, Institut für Datentechnik, Darmstadt, Sept 2004, S. 23–26

Knaak und Meyer 2005 Knaak, Nicolas ; Meyer, Ruth: Simulation Model Descriptions with UML 2. In: *The Java Simulation Handbook: Simulating Discrete Event Systems with UML and Java*. (Page und Kreutzer 2005), S. 59–95

Knaak et al. 2002a Knaak, Nicolas ; Meyer, Ruth ; Page, Bernd: Agentenbasierte Simulation mit einem objektorientierten Framework in Java. In: *Proceedings der ASIM 2002, 16. Symposium Simulationstechnik*. Rostock, September 2002, S. 247–252

Knaak et al. 2002b Knaak, Nicolas ; Meyer, Ruth ; Page, Bernd: Zweiter Zwischenbericht des Forschungsprojekts Nachhaltige Logistikkonzepte für Stadtkurierdienste: Entwurf und Implementation von agentenbasierten Simulationsmodellen / Universität Hamburg. 2002 – Interner Bericht.

Knaak et al. 2004 Knaak, Nicolas ; Meyer, Ruth ; Page, Bernd ; Deecke, Helmut: Agentenbasierte Simulation nachhaltiger Logistikkonzepte: Methoden, Werkzeuge, Anwendung. Bericht des Forschungsprojekts Nachhaltige Logistikkonzepte für Stadtkurierdienste / Universität Hamburg. 2004 – Fachbereichsbericht, FBI-B 260/2004.

Knaak und Page 2006 Knaak, Nicolas ; Page, Bernd: Applications and Extensions of the Unified Modelling Language UML 2 for Discrete Event Simulation. In: *International Journal of Simulation* 7 (2006), Nr. 6, S. 33–43

Köhler 1999 Köhler, H. J.: *Codegenerierung für UML Kollaborations-, Sequenz- und Statechart-Diagramme*, Universität Paderborn, Diplomarbeit, 1999

Kohler und Gumerman 2000 Kohler, Timothy ; Gumerman, George J.: *Dynamics of Human and Primate Societies: Agent-Based Modeling of Social and Spatial Processes*. New York : Oxford University Press, 2000 (Santa Fe Institute Studies in the Sciences of Complexity)

Krahl 2002 Krahl, David: The Extend Simulation Environment. In: Yücesan, E. (Hrsg.) ; Chen, C.-H. (Hrsg.) ; Snowdon, J. L. (Hrsg.) ; Charnes, J. M. (Hrsg.): *Proceedings of the 2002 Winter Simulation Conference*, 2002, S. 205–213

Krahl 2008 Krahl, David: ExtendSim 7. In: Manson, S. J. (Hrsg.) ; Hill, R. R. (Hrsg.) ; Mönch, L. (Hrsg.) ; Rose, O. (Hrsg.) ; Jefferson, T. (Hrsg.) ; Fowler, J. W. (Hrsg.): *Proceedings of the 2008 Winter Simulation Conference*, 2008, S. 215–221

Kranakis et al. 1999 Kranakis, Evangelos ; Singh, Harvinder ; Urrutia, Jorge: Compass Routing on Geometric Networks. In: *11th Canadian Conference on Computational Geometry (CCCG'99)*. Vancouver, Canada, August 15-18 1999, S. 51–54

Kreutzer 1986 Kreutzer, Wolfgang: *System Simulation: Programming Styles and Languages*. Reading, MA : Addison-Wesley, 1986 (International Computer Science Series)

Kruse 2007 Kruse, Sven: Komponentenbasierte Simulation lernfähiger Agenten. In: *Simulation und Visualisierung – Beiträge zum Doktorandenforum Diskrete Simulation, Magdeburg, September 2007*, SCS Publishing House, 2007

Labrou et al. 1999 Labrou, Yannis ; Finin, Tim ; Peng, Yun: Agent Communication Languages: The Current Landscape. In: *IEEE Intelligent Systems* 14 (1999), Nr. 2, S. 45–52

Lackner 1962 Lackner, Michael R.: Toward a General Simulation Capability. In: *Proceedings of the SJCC*. San Francisco, Ca., May 1-3 1962, S. 1–14

Lackner 1964 Lackner, Michael R.: Digital Simulation and System Theory / System Development Corp. 1964 – SDC Document, SP-1612.

Lake et al. 2004 Lake, Ron ; Burggraf, David ; Trninic, Milan ; Rae, Laurie: *Geography Mark-Up Language: Foundation for the Geo-Web*. Chichester : Wiley, 2004

Lamberson 2002 Lamberson, Roland H.: What Does It Take to Make Individual-Based Models Realize Their Potential? Introduction to NRM Special Issue on Individual-Based Models. In: *Natural Resource Modeling* 15 (2002), Nr. 1, S. 1–4

Lamport 1978 Lamport, Leslie: Time, Clocks, and the Ordering of Events in a Distributed System. In: *Communications of the ACM* 21 (1978), Nr. 7, S. 558–565

Langran 1992 Langran, G. E.: *Time in Geographic Information Systems*. London : Taylor & Francis, 1992

Langran und Chrisman 1988 Langran, Gail ; Chrisman, N. R.: A Framework for Temporal Geographic Information. In: *Cartographica* 25 (1988), Nr. 1, S. 1–14

Langton 1988 Langton, Christopher: Artificial Life. In: *Artificial Life*. Reading, MA : Addison-Wesley, 1988 (Santa Fe Institute Studies in the Sciences of Complexity VI), S. 1–47

Langton 1990 Langton, Christopher G.: Computation at the edge of chaos: Phase transitions and emergent computation. In: Forest, Stephanie (Hrsg.): *Emergent Computation*. Cambridge, MA : MIT Press, 1990, S. 12–37

Langton 2000 Langton, Christopher G.: *Artificial Life: An Overview*. 5th printing (original: 1995). Cambridge, MA : MIT Press, 2000 (Complex Adaptive Systems)

Lavender und Schmidt 1996 Lavender, R. G. ; Schmidt, Douglas C.: Active Object: An Object Behavioral Pattern for Concurrent Programming. In: Vlissides, John M. (Hrsg.) ; Coplien, James O. (Hrsg.) ; Kerth, Norman L. (Hrsg.): *Pattern Languages of Program Design 2: Papers presented at PLoP'95, the 1995 Pattern Languages on Programming Conference*. Reading, MA : Addison-Wesley, 1996, S. 483–499

Law und Kelton 2000 Law, Averill M. ; Kelton, W. D.: *Simulation Modeling and Analysis*. 3rd. Boston : McGraw-Hill, 2000

Lawson und Park 2000 Lawson, Barry G. ; Park, Steve: Asynchronous Time Evolution in an Artificial Society Model. In: *Journal of Artifiical Societies and Social Simulation* 3 (2000), Nr. 1. – URL http://jasss.soc.surrey.ac.uk/3/1/2.html

Lechler 1999 Lechler, Tim: *Entwurf und Implementierung eines Frameworks für diskrete Simulatoren in Java*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 1999

Lee et al. 1994 Lee, Jaeho ; Huber, Marcus J. ; Durfee, Edmund H. ; Kenny, Patrick G.: UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications. In: Erickson, J.D. (Hrsg.): *AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS'94), 23–24 March 1994, Houston, TX*. Houston, Texas, 1994 (NASA Conference Publications 3251), S. 842–849

LePage et al. 2012 LePage, Christophe ; Becu, Nicolas ; Bommel, Pierre ; Bousquet, François: Participatory Agent-Based Simulation for Renewable Resource Management: The Role of the Cormas Simulation Platform to Nurture a Community of Practice. In: *Journal of Artificial Societies and Social Simulation* 15 (2012), Nr. 1, S. 10. – URL http://jasss.soc.surrey.ac.uk/15/1/10.html

Lesser und Corkill 1983 Lesser, Victor R. ; Corkill, Daniel D.: The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. In: *AI Magazine* 4 (1983), Nr. 3, S. 15–33

Li und Wilensky 2009 Li, J. ; Wilensky, U.: *NetLogo Sugarscape 1 Immediate Growback model*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. 2009. – URL http://ccl.northwestern.edu/netlogo/models/Sugarscape1ImmediateGrowback

Liebl 1995 Liebl, Franz: *Simulation - Problemorientierte Einführung*. 2. Aufl. München; Wien : Oldenbourg, 1995

Lorek 1998 Lorek, Helmut: *Computerwerkzeuge zur Unterstützung bei der Entwicklung und beim Einsatz individuen-orientierter ökologischer Modelle*. Berlin : Logos-Verl., 1998. – Zugl.: Dissertation, Universität Oldenburg

Luck et al. 2004 Luck, Michael ; Ashri, Ronald ; d'Inverno, Mark: *Agent-Based Software Development*. London : Artech House, 2004

Luke 2013 Luke, Sean: Multiagent Simulation and the MASON Library / Department of Computer Science, George Mason University, Fairfax, VA. URL http://cs.gmu.edu/~eclab/projects/mason/manual.pdf, May 2013 – Manual Version 17.

Luke et al. 2005 Luke, Sean ; Cioffi-Revilla, Claudio ; Panait, Liviu ; Sullivan, Keith ; Balan, Gabriel: MASON: A Multi-Agent Simulation Environment. In: *Simulation* 82 (2005), Nr. 7, S. 517–527

Macal 2013 Macal, Charles M.: *Re: Execution algorithm for agent-based simulation*. Beitrag zur Mailingliste SIMSOC (Computer Simulation in the Social Sciences). 25. Februar 2013. – URL https://www.jiscmail.ac.uk/cgi-bin/webadmin?A2=ind1302&L=simsoc&F=&S=&P=31929

Macal und North 2006 Macal, Charles M. ; North, Michael J.: Tutorial on Agent-based Modeling and Simulation Part 2: How to Model with Agents. In: Perrone, L. F. (Hrsg.) ; Wieland, F. P. (Hrsg.) ; Liu, J. (Hrsg.) ; Lawson, B. G. (Hrsg.) ; Nicol, D. M. (Hrsg.) ; Fujimoto, R. M. (Hrsg.): *Proceedings of the 2006 Winter Simulation Conference*, 2006, S. 73–83

Macal und North 2008 Macal, Charles M. ; North, Michael J.: Agent-based modeling and simulation: ABMS examples. In: Mason, S. J. (Hrsg.) ; Hill, R. R. (Hrsg.) ; Mönch, L. (Hrsg.) ; Rose, O. (Hrsg.) ; Jefferson, T. (Hrsg.) ; Fowler, J. W. (Hrsg.): *Proceedings of the 2008 Winter Simulation Conference*, 2008, S. 101–112

Macal und North 2010a Macal, Charles M. ; North, Michael J.: Toward teaching agent-based simulation. In: Johansson, B. (Hrsg.) ; Jain, S. (Hrsg.) ; Montoya-Torres, J. (Hrsg.) ; Hugan, J. (Hrsg.) ; Yücesan, E. (Hrsg.): *Proceedings of the 2010 Winter Simulation Conference*, 2010, S. 268–277

Macal und North 2010b Macal, Charles M. ; North, Michael J.: Tutorial on agent-based modelling and simulation. In: *Journal of Simulation* 4 (2010), Nr. 1, S. 151–162

Macklin und Edgerton 2010 Macklin, Paul ; Edgerton, Mary E.: Agent-based cell modeling: application to breast cancer. In: Cristini, Vittorio (Hrsg.) ; Lowengrub, John (Hrsg.): *Multiscale Modeling of Cancer*. Cambridge : Cambridge University Press, 2010, S. 206–234

Macy und Willer 2002 Macy, Michael W. ; Willer, Robert: From Factors to Actors: Computational Sociology and Agent-Based Modeling. In: *Annual Review of Sociology* 28 (2002), Nr. 1, S. 143–166

Maes 1995 Maes, Pattie: Artificial life meets entertainment: lifelike autonomous agents. In: *Communications of the ACM* 38 (1995), Nr. 11, S. 108–114

Malsch und Schulz-Schaeffer 2007 Malsch, Thomas ; Schulz-Schaeffer, Ingo: Socionics: Sociological Concepts for Social Systems of Artificial (and Human) Agents. In: *Journal of Artificial Societies and Social Simulation* 10 (2007), Nr. 1. – URL `http://jasss.soc.surrey.ac.uk/10/1/11.html`

Mangina 2002 Mangina, Eleni: Review of Software Products for Multi-Agent Systems / AgentLink. URL `http://www.agentlink.org/admin/docs/2002/2002-47.pdf`, June 2002 – Report.

Mascardi et al. 2005 Mascardi, V. ; Demergasso, D. ; Ancona, D.: Languages for Programming BDI-style Agents: an Overview. In: Corradini, F. (Hrsg.) ; Paoli, F. D. (Hrsg.) ; Merelli, E. (Hrsg.) ; Omicini, A. (Hrsg.): *Proceedings of WOA 2005: Dagli Oggetti agli Agenti. 6th AI*IA/TABOO Joint Workshop "From Objects to Agents"*, Pitagora Editrice Bologna, 2005, S. 9–15

Maxwell und Costanza 1997 Maxwell, Thomas ; Costanza, Robert: An Open Geographic Modeling Environment. In: *Simulation* 68 (1997), Nr. 3, S. 175–185

McLaughlin und Edelson 2007 McLaughlin, Brett D. ; Edelson, Justin: *Java & XML*. 3rd ed. Sebastopol, CA : O'Reilly, 2007

Menczer und Belew 1993 Menczer, Filippo ; Belew, Richard K.: LEE: A Tool for Artificial Life Simulations / University of California, San Diego. URL `http://informatics.indiana.edu/fil/Papers/TR2.ps`, 1993 – Technical Report, CS93-301.

Meyer 1998 Meyer, Ruth: Eine Klassenbibliothek zur individuenbasierten Verkehrs- und Logistikmodellierung in Java. Siehe (Hilty et al. 1998), S. 87–100.

Meyer 2001 Meyer, Ruth: Bewegungsgraphen – Ein Konzept für die räumliche Modellierung der Umgebung in der individuen- und agenten-basierten Simulation. In: Wittmann, Jochen (Hrsg.) ; Bernard, Lars (Hrsg.): *Simulation in Umwelt- und Geowissenschaften, Workshop Münster 2001*. Aachen : Shaker, 2001 (Berichte aus der Umweltinformatik), S. 47–60

Meyer 2013 Meyer, Ruth: *FAMOS-Homepage*. 2013. – URL `http://famos.sourceforge.net`

Meyer et al. 2005a Meyer, Ruth ; Möller, Andreas ; Page, Bernd: Modellierungswerkzeuge zur Simulation umweltökonomischer Systeme – Interaktives Modul im Rahmen der universitären Präsenzlehre der Angewandten Informatik und der betrieblichen Fortbildung / Universität Hamburg, Fachbereich Informatik. 2005 – Endbericht des ELCH-Projekts A006.

Meyer et al. 2005b Meyer, Ruth ; Page, Bernd ; Kreutzer, Wolfgang ; Knaak, Nicolas ; Lechler, Tim: DESMO-J – A Framework for Discrete Event Modelling & Simulation. In: *The Java Simulation Handbook: Simulating Discrete Event Systems with UML and Java*. (Page und Kreutzer 2005), S. 263–335

Meyer et al. 1995 Meyer, Ruth ; Poll, Martina ; Mügge, Holger ; Gerken, Barbara ; Hilty, Lorenz M.: Anforderungen für den Einsatz eines Geographischen Informationssystems (GIS) in der umweltbezogenen Verkehrssimulation. In: Kremers, H. (Hrsg.) ; Pillmann, W. (Hrsg.): *Raum und Zeit in Umweltinformationssystemen — Space and Time in Environmental Information Systems. 9th International Symposium on Computer Science for Environmental Protection CSEP '95*. Marburg : Metropolis, 1995 (Umweltinformatik aktuell; 7), S. 376–384

Michel et al. 2009 Michel, Fabien ; Ferber, Jacques ; Drogoul, Alexis: Multi-Agent Systems and Simulation: A Survey from the Agents Community's Perspective. In: (Uhrmacher und Weyns 2009), S. 3–52

Minar et al. 1996 Minar, Nelson ; Burkhart, Roger ; Langton, Christopher G. ; Askenazi, Manor: The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations / Santa Fe Institute. URL `http://www.santafe.edu/media/workingpapers/96-06-042.pdf`, 1996 – Working Paper, 96-06-042.

Moss und Davidsson 2001 Moss, S. (Hrsg.) ; Davidsson, P. (Hrsg.): *Multi-Agent-Based Simulation; Second International Workshop, MABS 2000; Revised and Additional Papers*. Berlin : Springer, 2001 (LNAI 1979)

Moss et al. 1998 Moss, Scott ; Gaylard, Helen ; Wallis, Steve ; Edmonds, Bruce: SDML: A Multi-Agent Language for Organizational Modelling. In: *Computational and Mathematical Organization Theory* 4 (1998), Nr. 1, S. 43–69

Mügge und Meyer 1996 Mügge, Holger ; Meyer, Ruth: MSL – Eine Modell- und Experimentbeschreibungssprache. In: Ranze, K.C. (Hrsg.) ; Tuma, A. (Hrsg.) ; Hilty, L.M. (Hrsg.) ; Haasis, H.-D. (Hrsg.) ; Herzog, O. (Hrsg.): *Intelligente Methoden zur Verarbeitung von Umweltinformationen*. Marburg : Metropolis, 1996, S. 165–180

Mühlhäuser 2006 Mühlhäuser, Max: Verteilte Systeme. In: (Rechenberg und Pomberger 2006), S. 705–737

Müller 2011 Müller, Christian: Ein Ansatz zur Visualisierung von Desmo-J Simulationen / TH Wildau. URL `http://www.tfh-wildau.de/cmueller/Desmo-J/Visualization2d/Visualisierung_Desmoj_Simulationen.pdf`, Februar 2011 – Forschungsbericht.

Müller et al. 1998 Müller, H. J. ; Malsch, Thomas ; Schulz-Schaeffer, I.: SOCIONICS: Introduction and Potential. In: *JASSS* 1 (1998), Nr. 3. – URL `http://jasss.soc.surrey.ac.uk/1/3/5.html`

Müller 2009 Müller, Jean-Pierre: Towards a formal semantics of event-based multi-agent simulations. In: David, Nuno (Hrsg.) ; Sichman, Jaime S. (Hrsg.): *Multi-Agent-Based Simulation IX; International Workshop, MABS 2008, Estoril, Portugal, May 12-13, 2008, Revised Selected Papers*. Heidelberg : Springer, 2009 (LNAI 5269), S. 110–126

Müller 1996 Müller, Jörg P.: *The design of intelligent agents – a layered approach*. Berlin; u.a. : Springer, 1996 (Lecture Notes in Artificial Intelligence 1177)

Murphy 2003 Murphy, John T.: *A RePast Tutorial*, 2003. – URL `http://www.perfectknowledgedb.com/H2R/main.htm`

Nagel et al. 2000 Nagel, Kai ; Esser, Jörg ; Rickert, Marcus: Large-scale traffic simulations for transportation planning. In: Stauffer, Dietrich (Hrsg.): *Annual Review of Computational Physics VII*. Singapur : World Scientific Publishing, 2000, S. 151–202

Nagel und Schreckenberg 1992 Nagel, Kai ; Schreckenberg, Michael: A cellular automaton model for freeway traffic. In: *J. Physique I* 2 (1992), S. 2221–2229

Nance 1981 Nance, Richard E.: The Time and State Relationships in Simulation Modeling. In: *Communications of the ACM* 24 (1981), Nr. 4, S. 173–179

Neidhardt 2000 Neidhardt, Olaf: *Erstellung eines WWW-basierten Tutorials und Evaluation eines Frameworks zur zeitdiskreten Simulation in Java*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 2000

von Neumann 1966 Neumann, John von: *Theory of Self-Reproducing Automata. Edited and completed by Arthur W. Burks*. Urbana : University of Illinois Press, 1966

Niazi und Hussain 2011 Niazi, Muaz ; Hussain, Amir: Agent-based computing from multi-agent systems to agent-based models: a visual survey. In: *Scientometrics* 89 (2011), Nr. 2, S. 479–499

Nonaka und Holme 2007 Nonaka, Etsuko ; Holme, Petter: Agent-based model approach to optimal foraging in heterogeneous landscapes: Effects of patch clumpiness. In: *Ecography* 30 (2007), S. 777–788

North et al. 2013 North, Michael J. ; Collier, Nicholson T. ; Ozik, Jonathan ; Tatara, Eric R. ; Macal, Charles M. ; Bragen, Mark ; Sydelko, Pam: Complex adaptive systems modeling with Repast Simphony. In: *Complex Adaptive Systems Modeling* 1 (2013), S. 3. – URL `http://www.casmodeling.com/content/1/1/3`

North et al. 2006 North, Michael J. ; Collier, Nicholson T. ; Vos, Jerry R.: Experiences creating three implementations of the Repast agent modeling toolkit. In: *ACM Trans. Model. Comput. Simul.* 16 (2006), Nr. 1, S. 1–25

Nwana und Ndumu 1997 Nwana, Hyacinth S. ; Ndumu, Divine T.: An Introduction to Agent Technology. In: Nwana, Hyacinth S. (Hrsg.) ; Azarmi, Nader (Hrsg.): *Software Agents and Soft Computing – Towards Enhancing Machine Intelligence; Concepts and Applications*. Berlin; Heidelberg; New York : Springer, 1997 (Lecture Notes in Artificial Intelligence 1198), S. 3–26

Nwana und Ndumu 1999 Nwana, Hyacinth S. ; Ndumu, Divine T.: A Perspective on Software Agents Research. In: *Knowledge Engineering Review* 14 (1999), Nr. 2, S. 1–18

Nygaard und Dahl 1978 Nygaard, Kristen ; Dahl, Ole-Johan: The development of the SIMULA languages. In: *The First ACM SIGPLAN Conference on History of Programming Languages*, 1978, S. 245–272

Odell et al. 2001 Odell, James ; Parunak, H. Van D. ; Bauer, Bernhard: Representing Agent Interaction Protocols in UML. In: Ciancarini, Paolo (Hrsg.) ; Wooldridge, Michael J. (Hrsg.): *Agent-Oriented Software Engineering: First International Workshop, Limerick, Ireland, June 10, 2000; Revised Papers*. Berlin : Springer, 2001 (Lecture Notes in Computer Science 1957), S. 121–140

Okabe et al. 2000 Okabe, Atsuyuki ; Boots, Barry ; Sugihara, Kokichi ; Chiu, Sung N.: *Spatial Tesselations – Concepts and Applications of Voronoi Diagrams*. 2. ed. Chichester : Wiley, 2000

OMG 2005 OMG: *Unified Modeling Language: Superstructure Specification, Version 2.0, formal/05-07-04*. Object Management Group. 2005. – URL `http://www.omg.org/spec/UML/2.0/Superstructure/PDF`

Onggo 2010 Onggo, Bhakti S.: Running Agent-Based Models on a Discrete-Event Simulator. In: *Proceedings of the 24th European Simulation and Modelling Conference (ESM 2010), 25–27 October 2010, Hasselt, Belgium*, Eurosis-ETI, 2010, S. 51–55

Ortúzar und Willumsen 1994 Ortúzar, Juan de D. ; Willumsen, Luis G.: *Modelling Transport*. 2. ed. Chichester : Wiley, 1994

O'Sullivan 2000 O'Sullivan, David: *Graph-based Cellular Automaton Models of Urban Spatial Processes*, Centre of Advanced Spatial Analysis, University of London, Dissertation, 2000

O'Sullivan und Torrens 2000 O'Sullivan, David ; Torrens, Paul M.: Cellular Models of Urban Systems. In: Bandini, S. (Hrsg.) ; Worsch, T. (Hrsg.): *Theoretical and Practical Issues on Cellular Automata, Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry (ACRI 2000)*. London : Springer, 2000, S. 108–116

Ott und Swiaczny 2001 Ott, Thomas ; Swiaczny, Frank: *Time-Integrative Geographic Information Systems. Management and Analysis of Spatio-Temporal Data*. Berlin : Springer, 2001

Overstreet 1987 Overstreet, C. M.: Using Graphs to Translate Between World Views. In: Thesen, A. (Hrsg.) ; Grant, H. (Hrsg.) ; Kelton, W. D. (Hrsg.): *Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA*. New York : ACM Press, 1987, S. 582–589

Ozik et al. 2013 Ozik, Jonathan ; Collier, Nicholson T. ; Murphy, John T. ; North, Michael J.: The ReLogo Agent-Based Modeling Language. In: Pasupathy, R. (Hrsg.) ; Kim, S.-H. (Hrsg.) ; Tolk, A. (Hrsg.) ; Hill, R. (Hrsg.) ; Kuhl, M.E. (Hrsg.): *Proceedings of the 2013 Winter Simulation Conference*, 2013, S. 1560–1568

Padgham und Winikoff 2002 Padgham, Lin ; Winikoff, Michael: Prometheus: A pragmatic methodology for engineering intelligent agents. In: *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, Seattle, WA*. Sydney : Centre for Object Technology Applications and Research, 2002, S. 97–108

Padgham und Winikoff 2004 Padgham, Lin ; Winikoff, Michael: *Developing Intelligent Agent Systems: A Practical Guide*. Chichester, UK : Wiley, 2004

Page 1991 Page, Bernd: *Diskrete Simulation – Eine Einführung mit Modula-2*. Berlin : Springer, 1991

Page et al. 2005a Page, Bernd ; Knaak, Nicolas ; Meyer, Ruth: First Steps in Simulation Programming. In: *The Java Simulation Handbook: Simulating Discrete Event Systems with UML and Java*. (Page und Kreutzer 2005), S. 143–158

Page und Kreutzer 2005 Page, Bernd ; Kreutzer, Wolfgang: *The Java Simulation Handbook: Simulating Discrete Event Systems with UML and Java*. Aachen : Shaker, 2005

Page et al. 2005b Page, Bernd ; Kreutzer, Wolfgang ; Wohlgemuth, Volker: Simulation Software. In: *The Java Simulation Handbook: Simulating Discrete Event Systems with UML and Java*. (Page und Kreutzer 2005), S. 239–262

Page et al. 2000 Page, Bernd ; Lechler, Tim ; Claassen, Sönke: *Objektorientierte Simulation in Java mit dem Framework DESMO-J*. Libri Books on Demand, 2000

Page 1997 Page, Scott E.: On Incentives and Updating in Agent-Based Models / Division of the Humanities and the Social Sciences, California Institute of Technology. URL `http://www.hss.caltech.edu/SSPapers/wp1001.pdf`, Februar 1997 – Social Science Working Papers, 1001.

Panait und Luke 2004 Panait, Liviu ; Luke, Sean: Ant Foraging Revisited. In: Pollack, J. (Hrsg.) ; Bedau, M. (Hrsg.) ; Husbands, P. (Hrsg.) ; Ikegami, T. (Hrsg.) ; Watson, R.E. (Hrsg.): *Artificial Life IX: Proceedings of the 9th International Conference on the Simulation and Synthesis of Living Systems*. Cambridge, MA : MIT Press, 2004, S. 569–574

Parker 2001 Parker, Miles T.: What is Ascape and Why Should You Care? In: *Journal of Artificial Societies and Social Simulation* 4 (2001), Nr. 1, S. 5. – URL `http://jasss.soc.surrey.ac.uk/4/1/5.html`

Parunak 2000 Parunak, H. Van D.: The Process-Interface-Topology Model: Overlooked Issues in Modeling Social Systems. In: *Modelling Artificial Societies and Hybrid Organizations; Workshop at ECAI 2000 (European Conference on Artificial Intelligence)*, 20–25 Aug. 2000

Parunak et al. 2000 Parunak, H. Van D. ; Brückner, Sven ; Sauter, John ; Matthews, Robert S.: Distinguishing Environmental and Agent Dynamics: A Case Study in Abstraction and Alternate Modeling Technologies. In: *Engineering Societies in the Agents' World: First International Workshop, ESAW 2000, Berlin, Germany, August 21, 2000, Revised Papers*. 2000, S. 19–33

Parunak et al. 1998 Parunak, H. Van D. ; Savit, Robert ; Riolo, Rick L.: Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and User's Guide. In: Sichman, Jaime S. (Hrsg.) ; Conte, Rosaria (Hrsg.) ; Gilbert, Nigel (Hrsg.): *Multi-agent systems and agent based simulation – Proceedings of the first international workshop MABS '98, Paris, July 4-6, 1998*. Berlin : Springer, 1998 (Lecture notes in artificial intelligence 1534), S. 10–25

Pegden 2010 Pegden, C. D.: Advanced Tutorial: Overview of Simulation World Views. In: Johansson, B. (Hrsg.) ; Jain, S. (Hrsg.) ; Montoya-Torres, J. (Hrsg.) ; Hugan, J. (Hrsg.) ; Yücesan, E. (Hrsg.): *Proceedings of the 2010 Winter Simulation Conference*, 2010, S. 210–215

Pelekis et al. 2004 Pelekis, Nikos ; Theodoulidis, Babis ; Kopanakis, Ioannis ; Theodoridis, Yannis: Literature review of spatio-temporal database models. In: *The Knowledge Engineering Review* 19 (2004), Nr. 3, S. 235–274

Peuquet und Duan 1995 Peuquet, Donna ; Duan, N.: An Event-Based Spatio-temporal Data Model (ESTDM) for Temporal Analysis of Geographical Data. In: *International Journal of Geographical Information Systems* 9 (1995), Nr. 1, S. 7–24

Phipps 1992 Phipps, Michel J.: From Local to Global: The Lesson of Cellular Automata. In: DeAngelis, Donald L. (Hrsg.) ; Gross, Louis J. (Hrsg.): *Individual-based models and approaches in ecology – populations, communities, and ecosystems*. New York : Chapman & Hall, 1992, S. 165–187

Pidd 1995 Pidd, Michael: Object orientation, discrete simulation and the three-phase approach. In: *Journal of the Operational Research Society* 46 (1995), S. 362–374

Pokahr und Braubach 2009 Pokahr, Alexander ; Braubach, Lars: A Survey of Agent-oriented Development Tools. In: Bordini, R.H. (Hrsg.) ; Dastani, M. (Hrsg.) ; Dix, J. (Hrsg.) ; Seghrouchni, A. El F. (Hrsg.): *Multi-Agent Programming: Languages, Tools and Applications*. Berlin : Springer, 2009, S. 289–329

Polani und Uthmann 2000 Polani, Daniel ; Uthmann, Thomas: XRaptor – Eine flexible Umgebung für die Simulation virtueller zeit- und raumkontinuierlicher 2D- und 3D-Agentenszenarien. In: Szczerbicka, Helena (Hrsg.) ; Uthmann, Thomas (Hrsg.): *Modellierung, Simulation und Künstliche Intelligenz*. Erlangen : SCS Publishing House, 2000 (Frontiers in simulation 4), S. 111–137

Pomberger und Pree 2004 Pomberger, Gustav ; Pree, Wolfgang: *Software-Engineering: Architektur-Design und Prozessorientierung*. 3., vollständig überarb. Aufl. München; Wien : Hanser, 2004

Pree 1997 Pree, Wolfgang: *Komponentenbasierte Softwareentwicklung mit Frameworks*. Heidelberg : dpunkt-Verlag, 1997

Railsback und Grimm 2011 Railsback, Steven F. ; Grimm, Volker: *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton, N.J. : Princeton University Press, 2011

Railsback et al. 2006 Railsback, Steven F. ; Lytinen, Steven L. ; Jackson, Stephen K.: Agent-based Simulation Platforms: Review and Development Recommendations. In: *Simulation* 82 (2006), Nr. 9, S. 609–623

Rechenberg und Pomberger 2006 Rechenberg, Peter (Hrsg.) ; Pomberger, Gustav (Hrsg.): *Informatik-Handbuch*. 4., aktualisierte und erweiterte Aufl. München; Wien : Hanser, 2006

Repast 2008 Repast: *Repast Agent Simulation Toolkit: Examples*. Complex Adaptive Systems Group, Decision and Information Sciences Division, Argonne National Laboratory, Argonne, IL. 2008. – URL `http://repast.sourceforge.net/repast_3/examples/index.html`

Resnick 2000 Resnick, Mitchel: *Turtles, Termites, and Traffic Jams*. 6th printing (first paperback ed. 1997). Cambridge, MA : MIT Press, 2000 (Complex Adaptive Systems)

Reynolds 1987 Reynolds, Craig: Flocks, Herds, and Schools: A Distributed Behavioral Model. In: *Computer Graphics* 21 (1987), Nr. 4, S. 25–34

Roberts und Dessouky 1998 Roberts, Chell A. ; Dessouky, Yasser M.: An Overview of Object-Oriented Simulation. In: *Simulation* 70 (1998), Nr. 6, S. 359–368

Ropella et al. 2002 Ropella, Glen E. ; Railsback, Steven F. ; Jackson, Stephen K.: Software Engineering Considerations for Individual-Based Models. In: *Natural Resource Modeling* 15 (2002), Nr. 1, S. 5–22

Rosenschein und Genesereth 1985 Rosenschein, Jeffrey S. ; Genesereth, Michael R.: Deals Among Rational Agents. In: *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI)*. Los Angeles, Ca., August 1985, S. 91–99

Rupp et al. 2007 Rupp, Chris ; Queins, Stefan ; Zengler, Barbara: *UML 2 glasklar – Praxiswissen für die UML-Modellierung*. 3., aktualisierte Aufl. München : Hanser, 2007

Russell und Norvig 1995 Russell, Stuart J. ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ : Prentice-Hall, 1995

Russell und Norvig 2009 Russell, Stuart J. ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River, NJ : Prentice-Hall, 2009

Sadikni 2003 Sadikni, Remon: *Einsatz regelbasierter Expertensysteme in der agentenbasierten Simulation eines Stadtkurierdienstes mit Hilfe von JESS*, Universität Hamburg, Fachbereich Informatik, Baccalaureatsarbeit, 2003

Samuelson und Macal 2006 Samuelson, Douglas A. ; Macal, Charles M.: Agent-Based Simulation Comes of Age. In: *Operations Research / Management Science Today* 33 (2006), Nr. 4, S. 34ff

Sandhu und Treleaven 1996 Sandhu, Raghbir ; Treleaven, Philip: Client-Server Approaches to Model Integration within GIS. In: *Proceedings, Third International Conference/Workshop on Integrating GIS and Environmental Modeling, Santa Fe, NM, January 21-26, 1996*. Santa Barbara, CA : National Center for Geographic Information and Analysis, 1996. – URL `http://www.ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers/sandhu_raghbir/clientserver.html`

Sarkar 2000 Sarkar, Palash: A Brief History of Cellular Automata. In: *ACM Computing Surveys* 32 (2000), Nr. 1, S. 80–107

Schelling 1978 Schelling, Thomas C.: *Micromotives and Macrobehavior*. New York; London : Norton, 1978

Scheutz und Schermerhorn 2006 Scheutz, Matthias ; Schermerhorn, Paul: Adaptive Algorithms for the Dynamic Distribution and Parallel Execution of Agent-Based Models. In: *Journal of Parallel and Distributed Computing* 66 (2006), Nr. 8, S. 1037–1051

Schieritz und Milling 2003 Schieritz, Nadine ; Milling, Peter M.: Modeling the Forest or Modeling the Trees: A Comparison of System Dynamics and Agent-Based Simulation. In: al., Robert L. E. et (Hrsg.): *Proceedings of the 21st International Conference of the System Dynamics Society*. Albany, NY : System Dynamics Society, 2003

Schmidt 2000 Schmidt, Bernd: *Einführung in die Simulation mit SIMPLEX3*. Ghent : SCS Publishing House, 2000

Schniewind 1998 Schniewind, Thomas: *Entwurf und Implementierung eines Simulators für zeitdiskrete Simulation in C++*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 1998

Schreckenberg et al. 2003 Schreckenberg, M. ; Chrobok, R. ; Hafstein, S.F. ; Pottmeier, A.: OLSIM - Traffic Forecast and Planning using Simulations. In: Hohmann, R. (Hrsg.): *ASIM 2003 – 17. Symposium Simulationstechnik, 16.09.–19.09.2003 in Magdeburg*. Ghent : SCS European Publishing House, 2003, S. 11–18

Schriber und Brunner 1998 Schriber, Thomas J. ; Brunner, Daniel T.: How Discrete-Event Simulation Software Works. In: Banks, Jerry (Hrsg.): *Handbook of Simulation. Principles, Methodology, Advances, Applications, and Practice*. New York; Chichester : Wiley, 1998, S. 765–811

Schruben 1983 Schruben, Lee: Simulation Modeling with Event Graphs. In: *Communications of the ACM* 26 (1983), Nr. 11, S. 957–963

Schruben und Yucesan 1994 Schruben, Lee ; Yucesan, Enver: Transforming Petri Nets into Event Graph Models. In: *Proceedings of the 1994 Winter Simulation Conference*. Orlando, FL, 11-14 Dec 1994, S. 560–565

Schüle et al. 2004 Schüle, Michael ; Herrler, Rainer ; Klügl, Franziska: Coupling GIS and Multi-agent Simulation – Towards Infrastructure for Realistic Simulation. In: *Multiagent System Technologies. Second German Conference, MATES 2004, Erfurt, Germany, September 29–30, 2004. Proceedings*. Berlin, Heidelberg : Springer, 2004 (LNCS 3187), S. 228–242

Searle 1969 Searle, John R.: *Speech Acts: an Essay in the Philosophy of Language*. Cambridge : Cambridge University Press, 1969

Shoham 1993 Shoham, Yoav: Agent-oriented Programming. In: *AI* 60 (1993), Nr. 1, S. 51–92. – in Readings in Agents, S. 329-349

Sichman und Conte 2002 Sichman, Jaime S. ; Conte, Rosaria: Multi-Agent Dependence by Dependence Graphs. In: *First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Bologna, Italy, July 15-19 2002, S. 483–490

Sichman et al. 1994 Sichman, Jaime S. ; Conte, Rosaria ; Demazeau, Yves ; Castelfranchi, Cristiano: A Social Reasoning Mechanism Based on Dependence Networks. In: Cohn, A. G. (Hrsg.): *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94), Amsterdam*. Chichester : Wiley, 1994, S. 188–192

Siebers und Aickelin 2008 Siebers, Peer-Olaf ; Aickelin, Uwe: Introduction to Multi-Agent Simulation. In: Adam, Frederic (Hrsg.) ; Humphreys, Patrick (Hrsg.): *Encyclopedia of Decision Making and Decision Support Technologies*. Hershey, PA; London : Information Science Reference (IGI Global), 2008, S. 554–564

Siebers et al. 2010 Siebers, Peer-Olaf ; Macal, Charles M. ; Garnett, Jeremy ; Buxton, David ; Pidd, Mike: Discrete-event simulation is dead, long live agent-based simulation! In: *Journal of Simulation* 4 (2010), Nr. 3, S. 204–210

Siegfried et al. 2009 Siegfried, Robert ; Lehmann, Axel ; El Abdouni Khayari, Rachid ; Kiesling, Tobias: A reference model for agent-based modeling and simulation. In: Wainer, Gabriel (Hrsg.) ; Shaffer, Cliff (Hrsg.) ; McGraw, Robert (Hrsg.) ; Chinni, Michael J.

(Hrsg.): *SpringSim '09: Proceedings of the 2009 Spring Simulation Multiconference*. San Diego, CA : Society for Computer Simulation International, 2009

Simon 1996 Simon, Herbert: *The Sciences of the Artificial*. 3rd ed. Cambridge, MA : MIT Press, 1996

Sipper 1994 Sipper, Moshe: Non-Uniform Cellular Automata: Evolution in Rule Space and Formation of Complex Structures. In: Brooks, Rodney A. (Hrsg.) ; Maes, Patty (Hrsg.): *Artificial Life IV, Proceedings of the Fourth International Conference on Artificial Life*. Cambridge, Massachusetts : MIT Press, 1994, S. 394–399

Sloman und Poli 1996 Sloman, Aaron ; Poli, Riccardo: SIM_AGENT: A Toolkit for Exploring Agent Designs. In: Wooldridge, M. (Hrsg.) ; Müller, J. P. (Hrsg.) ; Tambe, M. (Hrsg.): *Intelligent Agents II, Proceedings of Agent Theories, Architectures, and Languages (ATAL), 19-20 August 1995, Montreal, Canada*. Heidelberg : Springer, 1996 (LNCS 1037), S. 392–407

Smith 1976 Smith, Alvy R.: Introduction to and survey of cellular automata or polyautomata theory. In: Lindenmayer, A. (Hrsg.) ; Rozenberg, G. (Hrsg.): *Automata, Languages and Development*. North Holland, 1976, S. 405–422

Smith 1980 Smith, Reid G.: The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. In: *IEEE Transactions on Computers* C-29 (1980), Nr. 12, S. 1104–1113

Sonnenschein und Lorek 1995 Sonnenschein, Michael ; Lorek, Helmut: Zeitdiskrete und ereignisdiskrete verteilte Simulation individuen-orientierter Modelle. In: Keller, H. B. (Hrsg.) ; Grützner, Rolf (Hrsg.) ; Paul, W. (Hrsg.): *5. Treffen des Arbeitskreises 5 der GI-Fachgruppe 4.6.1 "Werkzeuge für die Modellierung und Simulation im Umweltbereich"; Bericht FZKA 5622*. Forschungszentrum Karlsruhe, 1995, S. 9–16

Sonnenschein und Vogel 2001 Sonnenschein, Michael ; Vogel, Ute: Asymmetric Cellular Automata for the Modelling of Ecological Systems. In: Hilty, L. M. (Hrsg.) ; Gilgen, P. W. (Hrsg.): *Sustainability in the Information Society; 15th International Symposium Informatics for Environmental Protection*. Marburg : Metropolis, 2001, S. 631–636

Spaniol und Hoff 1995 Spaniol, Otto ; Hoff, Simon: *Ereignisorientierte Simulation: Konzepte und Systemrealisierung*. Bonn : Internat. Thomson Publishing, 1995 (Thomson's Aktuelle Tutorien 7)

Star und Estes 1990 Star, Jeffrey ; Estes, John: *Geographic Information Systems – An Introduction*. Englewood Cliffs : Prentice Hall, 1990

StarLogo 2001 StarLogo: *StarLogo Sample Projects - Sugarscape*. MIT Media Laboratory. 2001. – URL `http://education.mit.edu/starlogo/samples/sugarscape.htm`

Steyaert und Goodchild 1994 Steyaert, Louis T. ; Goodchild, Michael F.: Integrating Geographic Information Systems and Environmental Simulation Models – A Status Review. In: Michener, William K. (Hrsg.) ; Brunt, James W. (Hrsg.) ; Stafford, Susan G. (Hrsg.): *Environmental Information System Management and Analysis: Ecosystem to Global Scales*. London, Bristol : Taylor & Francis, 1994

Straßburger 2006 Straßburger, Steffen: Overview about the High Level Architecture for Modelling and Simulation and Recent Developments. In: *Simulation News Europe* 16 (2006), Nr. 2 (Special Issue: Parallel and Distributed Simulation Methods and Environments), S. 5–14

Sun 2001 Sun: High Performance Graphics. Graphics Performance Improvements in the Java 2 SDK, version 1.4 / Sun Microsystems / Oracle. URL `http://www.oracle.com/technetwork/java/perf-graphics-135933.html`, 2001 – White Paper.

Takeyama und Couclelis 1997 Takeyama, Masano ; Couclelis, Helen: Map dynamics: integrating cellular automata and GIS through Geo-Algebra. In: *International Journal of Geographical Information Science* 11 (1997), Nr. 1, S. 73–91

Tesfatsion 2002 Tesfatsion, Leigh: Agent-Based Computational Economics: Growing Economies from the Bottom Up. In: *Artificial Life* 8 (2002), Nr. 1, S. 55–82

Theodoropoulos et al. 2009 Theodoropoulos, Georgios ; Minson, Rob ; Ewald, Roland ; Lees, Michael: Simulation Engines for Multi-Agent Systems. In: (Uhrmacher und Weyns 2009), S. 77–108

Tobias und Hofmann 2004 Tobias, Robert ; Hofmann, Carole: Evaluation of free Java-libraries for social-scientific agent based simulation. In: *Journal of Artificial Societies and Social Simulation* 7 (2004), Nr. 1, S. 6. – URL `http://jasss.soc.surrey.ac.uk/7/1/6.html`

Toffoli und Margolus 1987 Toffoli, Tommaso ; Margolus, Norman: *Cellular Automata Machines: A New Environment for Modeling*. Cambridge, MA : MIT Press, 1987

Travers 1996 Travers, Michael D.: *Programming with Agents: New Metaphors for Thinking about Computation*. Cambridge, MIT, Dissertation, 1996

Troitzsch 2004 Troitzsch, Klaus: A Multi-Agent Model of Bilingualism in a Small Population. In: Coelho, Helder (Hrsg.) ; Espinasse, Bernard (Hrsg.): *5th Workshop on Agent-Based Simulation*. Erlangen; San Diego : SCS Publishing House, 2004, S. 38–43

Tyszer 1999 Tyszer, Jerzy: *Object-Oriented Computer Simulation of Discrete-Event Systems*. Dordrecht : Kluwer, 1999 (The Kluwer International Series on Discrete Event Dynamic Systems 10)

Uchmanski und Grimm 1996 Uchmanski, Janusz ; Grimm, Volker: Individual-based modelling in ecology: what makes the difference? In: *TREE* 11 (1996), Nr. 10, S. 437–441

Uhrmacher 1996 Uhrmacher, Adelinde M.: Variable structure modeling – Discrete events in simulation. In: *Proceedings of the Sixth Annual Conference on AI, Simulation and Planning in High Autonomy Systems*, IEEE Press, 1996, S. 133–140

Uhrmacher 1997 Uhrmacher, Adelinde M.: Concepts of Object- and Agent-Oriented Simulation. In: *Transactions of the SCS International* 14 (1997), Nr. 2, S. 59–67

Uhrmacher 2000 Uhrmacher, Adelinde M.: Agentenorientierte Simulation. In: Szczerbicka, Helena (Hrsg.) ; Uthmann, Thomas (Hrsg.): *Modellierung, Simulation und Künstliche Intelligenz*. Erlangen : SCS Publishing House, 2000 (Frontiers in simulation 4), S. 15–43

Uhrmacher 2001 Uhrmacher, Adelinde M.: Dynamic structures in modeling and simulation: a reflective approach. In: *ACM Transactions on Modeling and Computer Simulation* 11 (2001), Nr. 2, S. 206–232

Uhrmacher et al. 2006 Uhrmacher, Adelinde M. ; Himmelspach, Jan ; Röhl, Mathias ; Ewald, Roland: Introducing variable ports and multi-couplings for cell biological modeling in DEVS. In: Perronne, L. F. (Hrsg.) ; Wieland, F. P. (Hrsg.) ; Liu, J. (Hrsg.) ; Lawson, G. (Hrsg.) ; Nicol, D. M. (Hrsg.) ; Fujimoto, R. M. (Hrsg.): *Proceedings of the 2006 Winter Simulation Conference*, 2006, S. 832–840

Uhrmacher und Schattenberg 1998 Uhrmacher, Adelinde M. ; Schattenberg, Bernd: Agents in Discrete Event Simulation. In: *Proceedings of the ESS'98, Oct. 26-28, 1998, Nottingham, UK*. Ghent : SCS Publications, 1998, S. 129–138

Uhrmacher und Weyns 2009 Uhrmacher, Adelinde M. (Hrsg.) ; Weyns, Danny (Hrsg.): *Multi-Agent Systems: Simulation and Applications*. Boca Raton, FL : CRC Press/Taylor and Francis, 2009

Wagner 1997 Wagner, Daniel F.: Cellular automata and geographic information systems. In: *Environment and Planning B: Planning and Design* 24 (1997), Nr. 2, S. 219–234

Wagner 2004 Wagner, Gerd: AOR Modelling and Simulation — Towards a General Architecture for Agent-Based Discrete Event Simulation. In: Giorgini, Paolo (Hrsg.) ; Henderson-Sellers, Brian (Hrsg.) ; Winikoff, Michael (Hrsg.): *Agent-Oriented Information Systems, 5th International Bi-Conference Workshop, AOIS 2003, Melbourne, Australia, July 14, 2003 and Chicago, IL, USA, October 13, 2003, Revised Selected Papers*. Berlin : Springer, 2004 (LNAI 3030), S. 174–188

Wagner et al. 2008 Wagner, Gerd ; Giurca, Adrian ; Pehla, Marco ; Werner, Jens: Modellierung und Simulation von Multiagenten-Systemen. In: *Forum der Forschung* 21 (2008), S. 47–52

Watts 1999 Watts, Duncan J.: *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton, NJ : Princeton University Press, 1999

Weimar 1998 Weimar, Jörg R.: *Simulation with Cellular Automata*. Berlin : Logos-Verlag, 1998

Weiss 1998 Weiss, Gerhard: Agenten, Multiagentensysteme und Verteilte Künstliche Intelligenz – Eine einführende Literaturübersicht. In: *it+ti* 40 (1998), Nr. 4, S. 40–42

Weiss 1999 Weiss, Gerhard (Hrsg.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA : MIT Press, 1999

Weiss 2013 Weiss, Gerhard (Hrsg.): *Multiagent Systems*. 2nd ed. Cambridge, MA : MIT Press, 2013

Weiss und Jakob 2005 Weiss, Gerhard ; Jakob, Ralf: *Agentenorientierte Softwareentwicklung – Methoden und Tools*. Berlin : Springer, 2005 (Xpert.press)

Westervelt 2002 Westervelt, James D.: Geographic Information Systems and Agent-Based Modelling. In: (Gimblett 2002), S. 83 – 103

Weyns und Holvoet 2003 Weyns, Danny ; Holvoet, Tom: Model for Situated Multi-Agent-Systems with Regional Synchronization. In: Jardim-Goncalves, R. (Hrsg.) ; Cha, J. (Hrsg.) ; Steiger-Garcao, A. (Hrsg.): *Enhanced Interoperable Systems: Proceedings of the 10th International Conference on Concurrent Engineering (ISPE CE 2003), 26-30 July, Madeira, Portugal*, 2003, S. 177–188

Weyns et al. 2004 Weyns, Danny ; Parunak, H. Van D. ; Michel, Fabien ; Holvoet, Tom ; Ferber, Jacques: Environments for Multiagent Systems – State-of-the-Art and Research Challenges. In: *Proceedings of E4MAS'04* Bd. 3374. Berlin, Heidelberg : Springer, 2004, S. 1–47

White 1998 White, Roger: Cities and cellular automata. In: *Discrete Dynamics in Nature and Society* 2 (1998), Nr. 2, S. 111–125

White und Engelen 1997 White, Roger ; Engelen, Guy: Cellular Automata as the Basis of Integrated Dynamic Regional Modelling. In: *Environment and Planning B: Planning and Design* 24 (1997), Nr. 2, S. 235–246

Wilensky 1998 Wilensky, Uri: *NetLogo CA 1D Elementary model*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. 1998. – URL `http://ccl.northwestern.edu/netlogo/models/CA1DElementary`

Wilensky 1999 Wilensky, Uri: *Netlogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. 1999. – URL `http://ccl.northwestern.edu/netlogo/`

Winikoff und Padgham 2013 Winikoff, Michael ; Padgham, Lin: Agent-Oriented Software Engineering. In: (Weiss 2013), S. 695–757

Wittmann 1993 Wittmann, Jochen: *Eine Benutzerschnittstelle für die Durchführung von Simulationsexperimenten: Entwurf und Implementierung der Experimentierumgebung für das Simulationssystem Simplex II*, Universität Erlangen-Nürnberg, Technische Fakultät, Dissertation, 1993

Wittmann 2000 Wittmann, Jochen: Zellulare Automaten in der Umweltmodellierung. In: Möller, Dietmar P. F. (Hrsg.): *Simulationstechnik 14. Symposium in Hamburg, September 2000*. Delft; Erlangen; Ghent : SCS European Publishing House, 2000, S. 45–50

Wittmann 2011 Wittmann, Jochen: Immer Ärger mit der Zeit – Schnittstellenprobleme für Modellarchitekturen. In: *Simulation in Umwelt- und Geowissenschaften, Workshop Berlin 2011*. Aachen : Shaker, 2011, S. 193–202

Wolfram 1983 Wolfram, Stephen: Cellular automata. In: *Los Alamos Science* 9 (1983), S. 2–21

Wolfram 1984a Wolfram, Stephen: Cellular automata as models of complexity. In: *Nature* 311 (1984), S. 419–424

Wolfram 1984b Wolfram, Stephen: Universality and complexity in cellular automata. In: *Physica D* 10 (1984), S. 1–35

Wood und DeLoach 2001 Wood, Mark F. ; DeLoach, Scott A.: An Overview of the Multiagent Systems Engineering Methodology. In: *Agent-Oriented Software Engineering: First International Workshop AOSE 2000, Limerick, Ireland, 10 June 2000; Revised Papers*. Berlin : Springer, 2001, S. 207–221

Wooldridge et al. 2000 Wooldridge, Michael ; Jennings, Nicholas R. ; Kinny, David: The Gaia Methodology for Agent-Oriented Analysis and Design. In: *Journal of Autonomous Agents and Multi-Agent Systems* 3 (2000), Nr. 3, S. 285–312

Wooldridge 1999 Wooldridge, Michael J.: Intelligent Agents. In: Weiss, Gerhard (Hrsg.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, Mass.; London : MIT Press, 1999, S. 27–77

Wooldridge 2002 Wooldridge, Michael J.: *An Introduction to MultiAgent Systems*. Chichester : Wiley, 2002

Wooldridge 2009 Wooldridge, Michael J.: *An Introduction to MultiAgent Systems*. 2nd ed. Chichester : Wiley, 2009

Wooldridge 2013 Wooldridge, Michael J.: Intelligent Agents. In: (Weiss 2013), S. 3–50

Wooldridge und Jennings 1995 Wooldridge, Michael J. ; Jennings, Nicholas R.: Intelligent Agents: Theory and Practice. In: *Knowledge Engineering Review* 10 (1995), Nr. 2, S. 115–152

Worboys 1994 Worboys, Michael F.: A Unified Model for Spatial and Temporal Information. In: *The Computer Journal* 37 (1994), Nr. 1, S. 26–34

Worboys und Duckham 2004 Worboys, Michael F. ; Duckham, Matt: *GIS: A Computing Perspective*. 2. ed. CRC Press, 2004

Yuan 1999 Yuan, May: Use of a Three-Domain Representation to Enhance GIS Support for Complex Spatiotemporal Queries. In: *Transactions in GIS* 3 (1999), Nr. 2, S. 137–159

Zaft und Zeigler 2002 Zaft, Gordon C. ; Zeigler, Bernard P.: Discrete Event Simulation and Social Science: The XeriScape Artificial Society. In: *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002), 14–18 July 2002, Orlando, FL*, International Institute of Informatics and Systemics, 2002

Zambonelli et al. 2003 Zambonelli, Franco ; Jennings, Nicholas R. ; Wooldridge, Michael: Developing Multiagent Systems: The Gaia Methodology. In: *ACM Transactions on Software Engineering and Methodology* 12 (2003), Nr. 3, S. 317–370

Zeigler 1976 Zeigler, Bernard P.: *Theory of Modelling and Simulation*. New York; London : Wiley, 1976

Zeigler 1984 Zeigler, Bernard P.: *Multifacetted Modelling and Discrete Event Simulation*. London; Orlando : Academic Press, 1984

Zeigler et al. 2000 Zeigler, Bernard P. ; Praehofer, Herbert ; Kim, Tag G.: *Theory of Modeling and Simulation: Integrating Discrete Event and Continous Complex Dynamic Systems*. 2nd. San Diego; San Francisco; New York : Academic Press, 2000

Züllighoven und Raasch 2006 Züllighoven, Heinz ; Raasch, Jörg: Softwaretechnik. In: (Rechenberg und Pomberger 2006), S. 795–838

A. Anhang

A.1. Erweiterungen der Jess-Skriptsprache für regelbasierte Agenten in FAMOS

Die folgenden Funktionen stehen in FAMOS für die regelbasierte Modellierung des Agentenverhaltens mit *Jess* zur Verfügung.

(me): Referenz auf den regelbasierten Agenten.

(engine): Referenz auf das Verhaltensobjekt der Klasse `RuleEngine` des regelbasierten Agenten.

(model): Referenz auf das Modell, zu dem der regelbasierte Agent gehört.

(environment): Referenz auf die Umgebung des Agenten.

(schedule-at <signal> <t>): Merkt ein Signal für den angegebenen Zeitpunkt vor.

(schedule-in <signal> <dt>): Merkt ein Signal nach Ablauf der angegebenen Zeitdauer vor.

(assert-in (<fact>) <dt>): Merkt das Hinzufügen eines geordneten Fakts zur Wissensbasis nach Ablauf der angegeben Zeitdauer vor. Der regelbasierte Agent wird mit dem Hinzufügen des Fakts automatisch aktiviert.

(send <signal>): Sendet ein Signal.

(dump <text>): Gibt den als `String` angegebenen Text mit einem Zeitstempel versehen auf der Konsole aus.

(create-group <name> <hasRoles>): Erzeugt eine Gruppe des angegebenen Namens. Abhängig vom booleschen Wert `hasRoles` wird ein Objekt der Klasse `Group` oder ein Objekt der Klasse `RoleGroup` erzeugt.

(get-group <name>): Liefert die von der Umgebung des Agenten verwaltete Gruppe des angegeben Namens als `Group`-Objekt zurück.

(join-group <name>): Der Agent tritt der durch `name` spezifizierten Gruppe bei.

(leave-group <name>): Der Agent verläßt die durch `name` spezifizierte Gruppe.

(enter-environment): Der regelbasierte Agent betritt die Umgebung.

(exit-environment): Der regelbasierte Agent verläßt die Umgebung.

(observe): Gibt den Zustand der vom Agenten wahrnehmbaren Umwelt als Objekt der Klasse `EnvironmentState` zurück. Diese Funktion ist nur verfügbar, wenn der Agent von `SituatedAgent` abgeleitet ist.

(movement): Referenz auf die Bewegungsstrategie des Agenten. Diese Funktion ist nur verfügbar, wenn der Agent von `MobileAgent` abgeleitet ist.

(move): Der regelbasierte Agent führt einen Schritt seiner Bewegungsstrategie aus. Diese Funktion ist nur verfügbar, wenn der Agent von `MobileAgent` abgeleitet ist.

A.2. Skriptsprachen zur Verhaltensmodellierung in FAMOS

A.2.1. Formale Definition

Die folgenden DTDs definieren die Grammatiken der XML-Skriptsprachen, die FAMOS zur Modellierung des Verhaltens von Agenten zur Verfügung stellt. Dies sind eine Sprache zur Deklaration modellspezifischer Signalklassen und eine Sprache zur Spezifikation zustandsdiagramm- bzw. regelbasierter Verhaltensmodelle. Die entsprechenden Dateien *signalset.dtd* und *agent.dtd* befinden sich im Package `famos.xml`.

DTD der Skriptsprache für Signaldeklarationen

```
<!ELEMENT signalset (import*, signal+)>
<!ATTLIST signalset
    class ID #REQUIRED
    package CDATA #IMPLIED
>
<!ELEMENT import EMPTY>
<!ATTLIST import
    path CDATA #REQUIRED
>
<!ELEMENT signal (param*)>
<!ATTLIST signal
    name ID #REQUIRED
    activating (TRUE | FALSE) #IMPLIED
    internal (TRUE | FALSE) #IMPLIED
>
<!ELEMENT param EMPTY>
<!ATTLIST param
    name CDATA #REQUIRED
    type CDATA #REQUIRED
>
```

DTD der Skriptsprache zur Spezifikation von Verhaltensmodellen

```
<!ELEMENT agent (import*, declarations?, signals, (fsm | rulebase))>
```

```
<!ATTLIST agent
    class ID #REQUIRED
    extends CDATA #IMPLIED
    package CDATA #IMPLIED
>
<!ELEMENT declarations (#PCDATA)>
<!ELEMENT import EMPTY>
<!ATTLIST import
    path CDATA #REQUIRED
>
<!ELEMENT signals EMPTY>
<!ATTLIST signals
    file CDATA #REQUIRED
>
<!ELEMENT rulebase (#PCDATA)>
<!ELEMENT fsm (top, (in | transition)*)>
<!ELEMENT state EMPTY>
<!ATTLIST state
    name ID #REQUIRED
    id CDATA #IMPLIED
>
<!ELEMENT final EMPTY>
<!ATTLIST final
    name ID #REQUIRED
    id CDATA #IMPLIED
>
<!ELEMENT composite (composite | concurrent | state | final)*>
<!ATTLIST composite
        name ID #REQUIRED
    initial IDREF #IMPLIED
    history (NO_HISTORY | SHALLOW_HISTORY) #IMPLIED
    id CDATA #IMPLIED
>
<!ELEMENT concurrent (region*)>
<!ATTLIST concurrent
    name ID #REQUIRED
    id CDATA #IMPLIED
>
<!ELEMENT region (composite | state | final)*>
<!ATTLIST region
    initial IDREF #IMPLIED
    prio (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9) #IMPLIED
>
<!ELEMENT top (composite | concurrent | state | final)*>
<!ATTLIST top
```

```
        initial IDREF #REQUIRED
>
<!ELEMENT in (entry?, exit?, dt?, (on | defer)*)>
<!ATTLIST in
    state IDREF #REQUIRED
>
<!ELEMENT dt (#PCDATA)>
<!ATTLIST dt
    desc CDATA #IMPLIED
>
<!ELEMENT effect (#PCDATA)>
<!ATTLIST effect
    desc CDATA #IMPLIED
>
<!ELEMENT entry (#PCDATA)>
<!ATTLIST entry
    desc CDATA #IMPLIED
>
<!ELEMENT exit (#PCDATA)>
<!ATTLIST exit
    desc CDATA #IMPLIED
>
<!ELEMENT guard (#PCDATA)>
<!ATTLIST guard
    desc CDATA #IMPLIED
>
<!ELEMENT on (guard?, effect)>
<!ATTLIST on
    sig CDATA #REQUIRED
>
<!ELEMENT defer EMPTY>
<!ATTLIST defer
    sig CDATA #REQUIRED
>
<!ELEMENT transition (guard?, effect?)>
<!ATTLIST transition
    src IDREF #REQUIRED
    sig CDATA #IMPLIED
    dst IDREF #REQUIRED
    id CDATA #IMPLIED
>
```

A.2.2. Beschreibung der verfügbaren Elemente

Im folgenden werden die in den beiden Skriptsprachen zur Verfügung stehenden Elemente näher erläutert. Jedes Element ist durch seinen XML-Bezeichner (*Tag*) referenziert. Beispiele für Skripte, die diese Elemente verwenden, finden sich in Abschnitt 5.2.3.3 ab Seite 153.

Skriptsprache für Signaldeklarationen

<signalset>: Leitet die Deklaration der Signaltypen eines Modells ein. Als Attribute werden der zur Identifikation dienende Name des zugehörigen Modells (`class`) und optional das Java-Package angegeben (`package`), in dem die Signalklassen erzeugt werden sollen. Eingebettete Tags sind `<signal>` und `<import>`.

<import>: Dient zur Spezifikation der für die Quelltexte der erzeugten Signalklassen benötigten importierten Klassen oder Packages. Deren Pfad wird mit dem Attribut `path` angegeben. Eine Signaldeklaration kann mehrere `<import>`-Tags beinhalten.

<signal>: Deklaration eines modellspezifischen Signaltyps. Das Attribut `name` dient zur Angabe des entsprechenden Klassennamens. Die Attribute `internal` und `activating` können die Werte `TRUE` oder `FALSE` annehmen und legen fest, ob es sich um ein internes oder externes Signal handelt bzw. ob der Agent bei Erhalt des Signals sofort zu aktivieren ist. Interne Signale werden von der Klasse `InternalSignal` abgeleitet, externe Signale von `ExternalSignal`. Zur Angabe der Signalparameter kann das Tag `<signal>` mehrere eingebettete `<param>`-Tags enthalten.

<param>: Deklaration eines Signalparameters mit den Attributen `name` (Parametername) und `type` (Parametertyp).

Skriptsprache zur Spezifikation von Verhaltensmodellen

<agent>: Deklariert die zu erzeugende Agentenklasse. Attribute sind der Klassenname (`class`), die Oberklasse (`extends`) und das Java-Package (`package`), in dem die Klasse erzeugt werden soll. Bei der Oberklasse muss es sich um `Agent` oder eine davon abgeleitete Klasse handeln. Untergeordnete Tags von `<agent>` sind `<signals>`, `<declarations>`, `<import>` und `<fsm>` bzw. `<rulebase>`.

<signals>: Referenziert mit dem Attribut `name` eine Skriptdatei, in der alle vom Verhaltensmodell behandelten Signalklassen im oben vorgestellten Format deklariert sind. Prinzipiell könnten Signalklassen auch direkt im Agenten-Skript deklariert werden. Da Signale jedoch meist für mehrere Agenten eines Modells relevant sind, ist eine Deklaration auf Modellebene vorzuziehen.

<declarations>: Dient zur Deklaration zusätzlicher Attribute oder Methoden. In dieses Tag eingebettete Java-Anweisungen werden vom Codegenerator unverändert in den erzeugten Quelltext der Agentenklasse übernommen. Syntaxfehler innerhalb der Deklarationen werden somit erst bei der Kompilierung des Quellcodes erkannt

und müssen dann zurückverfolgt werden. Daher sollte der Anteil im Skript eingebetteter Java-Anweisungen so gering wie möglich ausfallen und dieses Tag nur in Sonderfällen verwendet werden. In der praktischen Arbeit mit FAMOS hat es sich als sinnvoll erwiesen, Sensor- und Effektor-Methoden von Agenten mit Zustandsdiagrammen explizit in eine Oberklasse auszulagern und davon die eigentliche Agentenklasse abzuleiten.

<import>: Dient wie bei der Signaldeklaration zur Spezifikation der für den Quelltext der erzeugten Agentenklasse benötigten importierten Klassen oder Packages. Die Packages `famos.agent` und `desmoj` müssen nicht explizit angegeben werden, sie werden automatisch importiert.

<rulebase>: Definition des regelbasierten Verhaltens der Agentenklasse. Die einzelnen Regeln sind in der *Jess*-Skriptsprache ggf. unter Verwendung der unter A.1 genannten Erweiterungen aufzulisten. Diese Regelbasis wird unverändert nach *Jess* übernommen, eine Syntaxprüfung der *Jess*-Anweisungen erfolgt dabei nicht.

<fsm>: Leitet die Definition des Zustandsdiagramms ein. Diese unterteilt sich in drei Bereiche: Zunächst wird unter dem Tag `<top>` die Zustandshierarchie aufgebaut. Anschließend folgen mehrere Tags `<in state="S">` zur Angabe der Aktionen und Reaktionen von Zuständen. Zuletzt werden mit `<transition>`-Tags Transitionen deklariert.

<top>: Deklaration der Zustandshierarchie. Das Attribut `initial` kennzeichnet den Anfangszustand des Zustandsdiagramms auf oberster Ebene. Die Angabe der einzelnen Zustände erfolgt abhängig vom Zustandstyp mit den eingebetteten Tags `<state>`, `<final>`, `<composite>` und `<concurrent>`.

<state>, <final>: Kennzeichnet einen einfachen Zustand bzw. Endzustand. Als einziges Attribut wird der Zustandsname angegeben.

<composite>: Deklaration eines zusammengesetzten Zustands. Attribute sind der Zustandsname `name`, der Anfangszustand `initial` und die Art des Wiedereintrittszustands `history`. Der Wert von history kann SHALLOW_HISTORY (flacher Wiedereintrittszustand) oder NO_HISTORY (normaler Anfangszustand) sein. Das Tag `<composite>` enthält weitere Zustandsdeklarationen zur Angabe von Unterzuständen.

<concurrent>: Deklaration eines zusammengesetzten Zustands mit mehreren orthogonalen Regionen. Enthält eine Liste eingebetteter Tags `<region>` für jede orthogonale Region.

<region>: Deklaration einer orthogonalen Region. Attribute sind `initial`, `history` (gleiche Bedeutung wie bei `<composite>`) und `prio`. Dieses Attribut gibt die Ausführungspriorität der Region durch eine ganze Zahl an, wobei die Region mit der kleinsten Zahl die höchste Priorität erhält. Das Tag `<region>` enthält weitere Zustandsdeklarationen zur Angabe von Unterzuständen.

<in>: Dient zur Angabe der Aktionen, Reaktionen und aufgeschobenen Signale eines Zustands. Das Attribut `state` verweist auf den Namen des Zustands, dessen Aktionen spezifiziert werden sollen. Eingebettete Tags sind `<entry>`, `<exit>`, `<dt>`, `<on>` und `<defer>`. Sie können entfallen, falls ein Zustand nicht über das entsprechende Element (z. B. eine Austrittsaktion) verfügt.

<entry>, <exit>: Eintritts- bzw. Austrittsaktion eines Zustands. Beide Aktionen werden durch zwischen den Tags eingebettete Java-Anweisungen angegeben, wobei das abschließende Semikolon der letzten Anweisung optional ist.

<dt>: Spezifiziert die Verweildauer im Zustand bis zum Senden des `Timeout`-Signals durch einen beliebigen Ausdruck vom Typ `SimTime`.

<defer>: Deklaration eines aufgeschobenen Signals, dessen Name mit dem Attribut `sig` angegeben wird.

<on>: Tag zur Deklaration einer statischen Reaktion. Das Attribut `sig` dient zur Angabe der auslösenden Signalklasse. Die Tags `<guard>` und `<effect>` werden zur Spezifikation der Reaktionsbedingung und des Effekts eingebettet. Bei einer unbedingten Reaktion entfällt das Tag `<guard>`.

<effect>: Angabe des Effekts einer Reaktion oder Transition durch beliebige eingebettete Java-Anweisungen. Innerhalb der Anweisungen kann auf Parameter des auslösenden Signals zugegriffen werden. Diese werden über ihren mit einem vorangestellten $-Zeichen gekennzeichneten Namen angesprochen und können in den Anweisungen als Variablen eingesetzt werden.

<guard>: Angabe der Bedingung einer Reaktion oder Transition. Bedingungen sind beliebige Java-Ausdrücke vom Typ `boolean`, in denen Signalparameter auf die oben beschriebene Weise verwendbar sind.

<transition>: Deklaration einer Transition. Entspricht der Deklaration einer Reaktion mit dem Tag `<on>` wobei als zusätzliche Attribute der Ausgangszustand (`src`) und der Zielzustand (`dst`) anzugeben sind. Bei einer impliziten Transition entfällt das Attribut `sig`.

A.3. XML-Dialekt zum Import räumlicher Daten in FAMOS

A.3.1. Formale Definition

Das folgende XML-Schema definiert die Grammatik des XML-Dialekts, den FAMOS zur Beschreibung räumlicher Daten zur Verfügung stellt. Es besteht wie jedes XML-Schema aus einer Deklaration der gültigen Elemente und einer Definition der verwendeten Datentypen. Die entsprechende Datei *famos.xsd* befindet sich im Package `famos.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns=http://www.w3.org/2001/XMLSchema
           elementFormDefault="qualified"
```

```
           attributeFormDefault="unqualified">

<!-- element declarations -->

<element name="Space" type="SpaceType">
    <annotation>
        <documentation>The space model</documentation>
    </annotation>
</element>
<element name="Graph" type="GraphType">
    <annotation>
        <documentation>A directed graph</documentation>
    </annotation>
</element>
<element name="Node" type="NodeType">
    <annotation>
        <documentation>A node in a graph</documentation>
    </annotation>
</element>
<element name="Edge" type="EdgeType">
    <annotation>
        <documentation>A directed edge in a graph
        </documentation>
    </annotation>
</element>
<element name="Point" type="PointType">
    <annotation>
        <documentation>A 2D point </documentation>
    </annotation>
</element>
<element name="Attribute" type="AttributeType">
    <annotation>
        <documentation>An attribute of a space element
        </documentation>
    </annotation>
</element>
<element name="AttributeArea" type="AttributeAreaType">
    <annotation>
        <documentation>An area with the same attribute(s)
        </documentation>
    </annotation>
</element>
<element name="Group" type="GroupType">
    <annotation>
        <documentation>A (spatial) group</documentation>
```

```
    </annotation>
</element>
<element name="SituatedObject" type="SituatedObjectType">
    <annotation>
        <documentation>An object in space
        </documentation>
    </annotation>
</element>
<element name="SimpleValue" type="SimpleValueType"
            substitutionGroup="Object">
    <annotation>
        <documentation>A wrapper for primitive types
        </documentation>
    </annotation>
</element>
<element name="Length" type="LengthType"
            substitutionGroup="Object">
    <annotation>
        <documentation>A length with unit</documentation>
    </annotation>
</element>
<element name="Velocity" type="VelocityType"
            substitutionGroup="Object">
    <annotation>
        <documentation>A velocity with unit</documentation>
    </annotation>
</element>
<element name="Duration" type="DurationType"
            substitutionGroup="Object">
    <annotation>
        <documentation>A duration with unit</documentation>
    </annotation>
</element>
<element name="Weight" type="WeightType"
            substitutionGroup="Object">
    <annotation>
        <documentation>A weight with unit</documentation>
    </annotation>
</element>
<element name="Capacity" type="CapacityType"
            substitutionGroup="Object">
    <annotation>
        <documentation>A capacity with unit</documentation>
    </annotation>
</element>
```

```
<element name="Object" type="anyType" abstract="true">
    <annotation>
        <documentation>Models java.lang.Object
        </documentation>
    </annotation>
</element>

<!-- type definitions -->

<complexType name="GroupType">
    <attribute name="name" type="string" use="required"/>
</complexType>
<complexType name="PointType">
    <attribute name="x" type="double" use="required"/>
    <attribute name="y" type="double" use="required"/>
</complexType>
<complexType name="AttributeType">
    <all>
        <element name="Value">
            <complexType>
                <sequence>
                    <element ref="Object"/>
                </sequence>
            </complexType>
        </element>
    </all>
    <attribute name="name" type="string" use="required"/>
    <attribute name="type" type="string" use="optional"/>
</complexType>
<complexType name="AttributeAreaType">
    <sequence>
        <element ref="Point" maxOccurs="unbounded"/>
        <element ref="Group" minOccurs="0"/>
        <element ref="Attribute" minOccurs="0"
                                 maxOccurs="unbounded"/>
    </sequence>
</complexType>
<complexType name="EdgeType">
    <sequence>
        <element ref="Length"/>
        <element ref="Attribute" minOccurs="0"
                                 maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="integer" use="required"/>
    <attribute name="startNode" type="integer"
```

```
                  use="required"/>
    <attribute name="endNode" type="integer"
                  use="required"/>
</complexType>
<complexType name="NodeType">
    <sequence>
        <element ref="Point"/>
        <element ref="Attribute" minOccurs="0"
                                 maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="integer" use="required"/>
    <attribute name="x" type="double" use="optional"/>
    <attribute name="y" type="double" use="optional"/>
</complexType>
<complexType name="GraphType">
    <sequence>
        <element ref="Node" maxOccurs="unbounded"/>
        <element ref="Edge" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="origin-x" type="integer" use="optional"
                  default="1"/>
    <attribute name="origin-y" type="integer" use="optional"
                  default="-1"/>
</complexType>
<complexType name="SpaceType">
    <sequence>
        <element ref="Graph"/>
        <element ref="AttributeArea" minOccurs="0"
                                     maxOccurs="unbounded"/>
        <element ref="SituatedObject" minOccurs="0"
                                      maxOccurs="unbounded"/>
    </sequence>
</complexType>
<complexType name="SituatedObjectType">
    <sequence>
        <element name="Object">
            <complexType>
                <sequence>
                    <element ref="Object"/>
                </sequence>
                <attribute name="type" type="string"
                           use="optional"/>
            </complexType>
        </element>
        <element ref="Point"/>
```

```
    </sequence>
</complexType>
<complexType name="SimpleValueType">
    <attribute name="string" type="string" use="optional"/>
    <attribute name="boolean" type="boolean" use="optional"/>
    <attribute name="byte" type="byte" use="optional"/>
    <attribute name="short" type="short" use="optional"/>
    <attribute name="integer" type="integer" use="optional"/>
    <attribute name="long" type="long" use="optional"/>
    <attribute name="float" type="float" use="optional"/>
    <attribute name="double" type="double" use="optional"/>
</complexType>
<complexType name="DurationType">
    <attribute name="value" type="double" default="0.0"/>
    <attribute name="unit" default="1">
        <simpleType>
            <restriction base="integer">
                <enumeration value="0"/>
                <enumeration value="1"/>
                <enumeration value="2"/>
                <enumeration value="3"/>
            </restriction>
        </simpleType>
    </attribute>
</complexType>
<complexType name="LengthType">
    <attribute name="value" type="double" default="0.0"/>
    <attribute name="unit" default="6">
        <simpleType>
            <restriction base="integer">
                <enumeration value="4"/>
                <enumeration value="5"/>
                <enumeration value="6"/>
                <enumeration value="7"/>
            </restriction>
        </simpleType>
    </attribute>
</complexType>
<complexType name="VelocityType">
    <attribute name="value" type="double" default="0.0"/>
    <attribute name="unit" default="9">
        <simpleType>
            <restriction base="integer">
                <enumeration value="8"/>
                <enumeration value="9"/>
```

```
            </restriction>
        </simpleType>
    </attribute>
</complexType>
<complexType name="WeightType">
    <attribute name="value" type="double" default="0.0"/>
    <attribute name="unit" default="12">
        <simpleType>
            <restriction base="integer">
                <enumeration value="10"/>
                <enumeration value="11"/>
                <enumeration value="12"/>
                <enumeration value="13"/>
            </restriction>
        </simpleType>
    </attribute>
</complexType>
<complexType name="CapacityType">
    <attribute name="value" type="double" default="0.0"/>
    <attribute name="unit" default="15">
        <simpleType>
            <restriction base="integer">
                <enumeration value="14"/>
                <enumeration value="15"/>
            </restriction>
        </simpleType>
    </attribute>
</complexType>

</schema>
```

A.3.2. Beschreibung der verfügbaren Elemente

Im folgenden werden die im XML-Dialekt für die Beschreibung räumlicher Daten zur Verfügung stehenden Elemente näher erläutert. Jedes Element ist durch seinen XML-Bezeichner (*Tag*) referenziert. Für eine ausführliche Beschreibung der in XML-Schemata vorkommenden Elemente sei auf die entsprechenden Dokumente des World Wide Web Consortiums (W3C) verwiesen (XML0 (2004) für eine Einführung und XML1 (2004) und XML2 (2004) für die formale Spezifikation).

<Space> Beschreibt das zu erzeugende Raummodell. Dieses besteht aus einem Graphen (untergeordneter Tag <Graph>) und je einer optionalen Liste von Attributgebieten und im Raum positionierter Objekte (Tags <AttributeArea> und <Situated-Object>). Der Import von Gittern wird momentan in FAMOS nicht unterstützt.

<Graph> Der zu importierende Graph wird durch je eine nicht-leere Liste von Knoten und Kanten (untergeordnete Tags `<Node>` und `<Edge>`) definiert. Optional kann der Ursprung des verwendeten Koordinatensystems über die Attribute `origin-x` und `origin-y` angegeben werden.

<Node> Ein Knoten wird durch einen eindeutigen Bezeichner vom Typ Integer (Attribut <id>), seine Koordinaten in Form eines Punktes (untergeordneter Tag `<Point>`) und eine optionale Liste von Attributwerten (untergeordneter Tag `<Attribute>`) spezifiziert. Die Koordinaten können alternativ auch über die Element-Attribute `x` und `y` angegeben werden.

<Edge> Eine Kante benötigt die Angabe des Start- und Endknotens in Form der jeweiligen Knoten-ID (Attribute `startNode` und `endNode`), ihrer Länge (untergeordneter Tag `<Length>`) und eines eindeutigen Bezeichners (Attribut `id`). Optional kann wie bei Knoten eine Liste von Attributwerten hinzugefügt werden (untergeordneter Tag `<Attribute>`).

<Point> Definiert einen Punkt im zweidimensionalen Raum durch Angabe seiner Koordinaten (Attribute `x` und `y`).

<Attribute> Ein Attribut in FAMOS ist ein Paar aus Name (Element-Attribut `name`) und Wert (untergeordneter Tag `<Value>`, definiert als anonymer Typ). Als Werte sind beliebige Folgen von Objekten zugelassen. Optional kann der Datentyp des Attributwerts (Element-Attribut `type`) spezifiziert werden.

<AttributeArea> Beschreibt ein Attributgebiet als eine Folge von Punkten (untergeordneter Tag `<Point>`) verbunden mit einer Liste von Attributen (untergeordneter Tag `<Attribute>`) und optional einer Liste von räumlichen Gruppen (untergeordneter Tag `<Group>`).

<Group> Eine Gruppe ist durch Angabe ihres eindeutigen Bezeichners vom Typ String (Attribut `name`) definiert.

<SituatedObject> Beschreibt ein im Raum positioniertes Objekt, bestehend aus einem beliebigen Objekt (untergeordneter Tag `<Object>`, definiert als anonymer Typ) und dessen Position (untergeordneter Tag `<Point>`).

<SimpleValue> Eine Hülle für Java's primitive Datentypen `boolean`, `byte`, `short`, `int`, `long`, `float` und `double` sowie Zeichenketten (`String`). Durch die Deklaration einer sogenannten Ersetzungsgruppe (Element-Attribut `substitutionGroup`) mit dem anonymen Typ `Object` als Ursprungs-Element können SimpleValue-Elemente überall dort verwendet werden, wo ein Element vom Typ `Object` erwartet wird. Dies sind Werte von FAMOS-Attributen (Tag `<Value>` in `<Attribute>`) und im Raum positionierte Objekte (Tag `<Object>` in `<SituatedObject>`).

<Length> Dieses und die folgenden vier Elemente beschreiben physikalische Größen. Sie bestehen jeweils aus einem Zahlenwert (Attribut `value`) und einer Maßeinheit (Attribut `unit`). Die Einheiten sind als Konstanten zu spezifizieren, wie sie in der

Klasse `desmoj.Units` definiert sind. Für jede Größe ist eine Default-Einheit vorgegeben. Das Element `<Length>` dient zur Angabe einer Länge; Default-Einheit ist Meter.

`<Velocity>` Die physikalische Größe Geschwindigkeit; Default-Einheit ist Kilometer pro Stunde.

`<Duration>` Die physikalische Größe Zeit(dauer); Default-Einheit ist Sekunde.

`<Weight>` Die physikalische Größe Masse (Gewicht); Default-Einheit ist Kilogramm.

`<Capacity>` Die physikalische Größe Volumen (Rauminhalt); Default-Einheit ist Liter.

A.3.3. Abbildung von Java-Klassen auf XML-Strukturen im Data Binding

Das in FAMOS verwendete Data-Binding-Framework Castor besitzt mit dem sogenannten *XML Mapping* einen flexiblen Mechanismus zur Abbildung von Java-Klassen auf XML-Strukturen (Gignoux und Visco 2005). Die für eine solche Abbildung benötigten Informationen sind in einem speziellen XML-Dokument, dem *mapping file*, anzugeben. Castor stellt dazu einen als DTD definierten XML-Dialekt zur Verfügung. Innerhalb des Wurzelelements `<mapping>` kann für jede Klasse (untergeordneter Tag <class>) genau spezifiziert werden, wie deren Attribute auf XML-Elemente abzubilden sind. Auf diese Weise lassen sich nahezu beliebige Java-Klassen in XML-Dokumente überführen bzw. aus XML-Dokumenten einlesen.

Die einzige Einschränkung für die Klassendefinition ergibt sich aus der Notwendigkeit, bei der Objekterzeugung (*Unmarshalling*) auf die Java Reflection API zurückzugreifen. Daher muss jede im Data Binding zu verwendende Klasse zumindest einen parameterlosen Konstruktor sowie öffentlich (*public*) deklarierte Zugriffsmethoden für ihre Attribute besitzen. Für FAMOS bedeutet dies, dass die räumlichen Klassen des Package `famos.space` nicht direkt verwendet werden können, da deren Attribute vor Zugriff von außen geschützt sind. Das Raummodell ist nur über die Klasse `Environment` zugreifbar, um dessen Einsatz in Modellen einfacher und robuster zu gestalten (Entwurfsmuster Fassade). Diese bewusst getroffene Design-Entscheidung sollte durch Hinzufügen einer GIS-Schnittstelle nicht wieder rückgängig gemacht werden. Es wurde daher der Umweg über Java-Bean konforme Stellvertreter-Klassen (`SpaceDescription` anstelle von `Space`, `GraphDescription` für `Graph` usw.) gewählt, welche die zugehörigen Raumobjekte generieren können. Diese Klassen finden sich im Package `famos.xml`.

Im folgenden ist das *mapping file* für FAMOS zur Verwendung mit Castor aufgelistet.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapping PUBLIC
        "-//EXOLAB/Castor Mapping DTD Version 1.0//EN"
        "http://castor.org/mapping.dtd">

<mapping>
```

```
<!-- root-element: space -->

<class name="famos.xml.SpaceDescription">
    <map-to xml="Space"/>
    <field name="Graph"
        type="famos.xml.GraphDescription">
        <bind-xml name="Graph" node="element"/>
    </field>
    <field name="AttributeAreas"
        type="famos.xml.AttributeAreaDescription"
        collection="array">
        <bind-xml name="AttributeArea" node="element"/>
    </field>
    <field name="Objects"
        type="famos.xml.SituatedObjectDescription"
        collection="array">
        <bind-xml name="SituatedObject" node="element"/>
    </field>
</class>

<!-- classes describing graphs, nodes, and edges -->

<class name="famos.xml.GraphDescription">
    <map-to xml="Graph"/>
    <field name="OriginX"
        type="integer">
        <bind-xml name="origin-x" node="attribute"/>
    </field>
    <field name="OriginY"
        type="integer">
        <bind-xml name="origin-y" node="attribute"/>
    </field>
    <field name="NodeDescription"
           type="famos.xml.NodeDescription"
            collection="array"
            get-method="getNodes"
            set-method="setNodes">
            <bind-xml name="Node" node="element"/>
    </field>
    <field name="EdgeDescription"
           type="famos.xml.EdgeDescription"
            collection="array"
            get-method="getEdges"
            set-method="setEdges">
            <bind-xml name="Edge" node="element"/>
```

```
    </field>
</class>

<class name="famos.xml.NodeDescription">
    <field name="ID"
        type="integer">
        <bind-xml name="id" node="attribute"/>
    </field>
    <field name="Point"
        type="famos.xml.PointDescription">
        <bind-xml name="Point" node="element"/>
    </field>
    <field name="Attributes"
        type="famos.xml.AttributeDescription"
        collection="array">
        <bind-xml name="Attribute" node="element"/>
    </field>
</class>

<class name="famos.xml.EdgeDescription">
    <field name="ID"
        type="integer">
        <bind-xml name="id" node="attribute"/>
    </field>
    <field name="StartNodeID"
        type="integer">
        <bind-xml name="startNode" node="attribute"/>
    </field>
    <field name="EndNodeID"
        type="integer">
        <bind-xml name="endNode" node="attribute"/>
    </field>
    <field name="Length"
        type="desmoj.app.Length">
        <bind-xml name="Length" node="element"/>
    </field>
    <field name="Attributes"
        type="famos.xml.AttributeDescription"
        collection="array">
        <bind-xml name="Attribute" node="element"/>
    </field>
</class>

<!-- classes describing attribute areas -->
```

```
<class name="famos.xml.AttributeAreaDescription">
    <map-to xml="AttributeArea"/>
    <field name="Points"
        type="famos.xml.PointDescription"
        collection="array">
        <bind-xml name="Point" node="element"/>
    </field>
    <field name="Group"
        type="famos.xml.GroupDescription">
        <bind-xml name="Group" node="element"/>
    </field>
    <field name="Attributes"
        type="famos.xml.AttributeDescription"
        collection="array">
        <bind-xml name="Attribute" node="element"/>
    </field>
</class>

<class name="famos.xml.GroupDescription">
    <field name="Name"
        type="string">
        <bind-xml name="name" node="attribute"/>
    </field>
</class>

<class name="famos.xml.PointDescription">
    <field name="X"
        type="double">
        <bind-xml name="x" node="attribute"/>
    </field>
    <field name="Y"
        type="double">
        <bind-xml name="y" node="attribute"/>
    </field>
</class>

<!-- class describing attributes -->

<class name="famos.xml.AttributeDescription">
    <map-to xml="Attribute"/>
    <field name="Name"
        type="java.lang.String"
        required="true">
        <bind-xml name="name" node="attribute"/>
    </field>
```

```
    <field name="Type"
        type="java.lang.String">
        <bind-xml name="type" node="attribute"/>
    </field>
    <field name="Value"
        type="other"
        required="true">
        <bind-xml name="Value" node="element"/>
    </field>
</class>


<!-- pre-defined classes for attribute values -->

<class name="famos.xml.PrimitiveWrapper">
    <map-to xml="SimpleValue"/>
    <field name="Boolean"
        type="boolean">
        <bind-xml name="boolean" node="attribute"/>
    </field>
    <field name="Long"
        type="long">
        <bind-xml name="long" node="attribute"/>
    </field>
    <field name="Integer"
        type="integer">
        <bind-xml name="integer" node="attribute"/>
    </field>
    <field name="Short"
        type="short">
        <bind-xml name="short" node="attribute"/>
    </field>
    <field name="Byte"
        type="byte">
        <bind-xml name="byte" node="attribute"/>
    </field>
    <field name="Double"
        type="double">
        <bind-xml name="double" node="attribute"/>
    </field>
    <field name="Float"
        type="float">
        <bind-xml name="float" node="attribute"/>
    </field>
    <field name="Character"
```

```
            type="char">
            <bind-xml name="char" node="attribute"/>
        </field>
        <field name="String"
            type="string">
            <bind-xml name="string" node="attribute"/>
        </field>
    </class>

    <class name="desmoj.app.Length">
        <map-to xml="Length"/>
        <field name="Value"
            type="double">
            <bind-xml name="value" node="attribute"/>
        </field>
        <field name="Unit"
            type="integer">
            <bind-xml name="unit" node="attribute"/>
        </field>
    </class>

    <class name="desmoj.app.Velocity">
        <map-to xml="Velocity"/>
        <field name="Value"
            type="double">
            <bind-xml name="value" node="attribute"/>
        </field>
        <field name="Unit"
            type="integer">
            <bind-xml name="unit" node="attribute"/>
        </field>
    </class>

    <class name="desmoj.app.Duration">
        <map-to xml="Duration"/>
        <field name="Value"
            type="double">
            <bind-xml name="value" node="attribute"/>
        </field>
        <field name="Unit"
            type="integer">
            <bind-xml name="unit" node="attribute"/>
        </field>
    </class>
```

```
<class name="desmoj.app.Weight">
    <map-to xml="Weight"/>
    <field name="Value"
        type="double">
        <bind-xml name="value" node="attribute"/>
    </field>
    <field name="Unit"
        type="integer">
        <bind-xml name="unit" node="attribute"/>
    </field>
</class>

<class name="desmoj.app.Capacity">
    <map-to xml="Capacity"/>
    <field name="Value"
        type="double">
        <bind-xml name="value" node="attribute"/>
    </field>
    <field name="Unit"
        type="integer">
        <bind-xml name="unit" node="attribute"/>
    </field>
</class>

<!-- classes describing situated objects -->

<class name="famos.xml.DescriptionList">
    <map-to xml="List"/>
    <field name="Elements"
        type="other"
        collection="vector">
        <bind-xml name="Item" node="element"/>
    </field>
</class>

<class name="famos.xml.SituatedObjectDescription">
    <map-to xml="SituatedObject"/>
    <field name="Object"
        type="other">
        <bind-xml name="Object" node="element"/>
    </field>
    <field name="Point"
        type="famos.xml.PointDescription">
        <bind-xml name="Point" node="element"/>
    </field>
```

```
    </class>

</mapping>
```

A.3.4. Anbindung an GML

Die *Geography Markup Language* (GML) modelliert räumliche Phänomene als sogenannte Geo-Objekte (*Features*), die eine Menge von geometrischen und nicht-geometrischen Eigenschaften (*Properties*) besitzen. Hierzu stellt GML grundlegende Elemente und Datentypen zur Verfügung, die in einer Reihe von XML-Schemata (Basisschemata) definiert sind.

Um FAMOS mit einer GML-Schnittstelle zu versehen, ist ein sogenanntes GML-Anwendungsschema zu erstellen. Ein Anwendungsschema baut auf den GML-Basisschemata auf und legt Feature- und Property-Typen fest, die im Anwendungsbereich benötigt werden. Diese Typen können eine Teilmenge von GML-Typen und eigene Spezialisierungen von GML-Typen umfassen. Um die Gültigkeit eines Anwendungschemas zu gewährleisten, sind bei dessen Spezifikation bestimmte Regeln einzuhalten (GML2 2001, ch. 5).[1] So muss beispielsweise ein eigener Namensraums deklariert werden (Attribut `targetNamespace` des Elements `<schema>`), um Benennungskonflikte zu vermeiden, und sichergestellt sein, dass eigene Feature-Typen direkt oder indirekt von `gml:AbstractFeatureType` bzw. `gml:AbstractFeatureCollectionType` abgeleitet werden.

Im folgenden ist die prototypische Realisierung eines GML-Anwendungsschemas für FAMOS angegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
   targetNamespace="http://sourceforge.net/projects/famos"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:gml="http://www.opengis.net/gml"
   xmlns:famos="http://sourceforge.net/projects/famos"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="0.1">

<import namespace="http://www.opengis.net/gml"
   schemaLocation="feature.xsd"/>

<!-- global element declarations -->

<element name="Space" type="famos:SpaceType"
   substitutionGroup="gml:_FeatureCollection">
```

[1] Da zum Implementationszeitpunkt der Import-Schnittstelle GML in der Version 2.0 vorlag, werden hier die Vorschriften für GML 2 zitiert. Die aktuelle Version GML 3.2.1 ist gegenüber der Version 2.0 wesentlich erweitert, u. a. um zeitliche und topologische Aspekte.

```
   <annotation>
      <documentation>the abstract famos space</documentation>
   </annotation>
</element>
<element name="spaceMember" type="famos:SpaceMemberType"
   substitutionGroup="gml:featureMember">
   <annotation>
      <documentation>denotes the association type: only elements
       of SpaceMemberType are allowed as members of the Space
       feature collection</documentation>
   </annotation>
</element>
<element name="Graph" type="famos:GraphType"
   substitutionGroup="famos:_SpaceFeature">
   <annotation>
      <documentation>a graph is a valid space member
      </documentation>
   </annotation>
</element>
<element name="AttributeArea" type="famos:AttributeAreaType"
   substitutionGroup="famos:_SpaceFeature">
   <annotation>
      <documentation>an attribute area is a valid space member
      </documentation>
   </annotation>
</element>
<element name="attribute" type="famos:AttributeType">
   <annotation>
      <documentation>denotes an attribute property
      </documentation>
   </annotation>
</element>
<element name="_SpaceFeature" type="gml:AbstractFeatureType"
   abstract="true" substitutionGroup="gml:_Feature">
   <annotation>
      <documentation>a label for restricting membership in the
       Space collection</documentation>
   </annotation>
</element>
<element name="Length" type="famos:LengthType"
   substitutionGroup="famos:_Object">
   <annotation>
      <documentation>a length object</documentation>
   </annotation>
</element>
```

```
<element name="Velocity" type="famos:VelocityType"
   substitutionGroup="famos:_Object">
   <annotation>
      <documentation>a velocity object</documentation>
   </annotation>
</element>
<element name="_Object" type="anyType" abstract="true">
   <annotation>
      <documentation>a label for the java.lang.Object class
       (hierarchy)</documentation>
   </annotation>
</element>

<!-- type definitions -->

<complexType name="SpaceType">
   <annotation>
      <documentation>a space consists of a graph (or a grid)
       and 0-n attribute areas</documentation>
   </annotation>
   <complexContent>
      <extension base="gml:AbstractFeatureCollectionType"/>
   </complexContent>
</complexType>
<complexType name="SpaceMemberType">
   <annotation>
      <documentation>a space member is restricted to those
       features that are declared equivalent to
       famos:_SpaceFeature</documentation>
   </annotation>
   <complexContent>
      <restriction base="gml:FeatureAssociationType">
         <sequence minOccurs="0">
            <element ref="famos:_SpaceFeature"/>
         </sequence>
      </restriction>
   </complexContent>
</complexType>
<complexType name="GraphType">
   <annotation>
      <documentation>a graph consists of nodes and edges
      </documentation>
   </annotation>
   <complexContent>
      <extension base="gml:AbstractFeatureType">
```

```
         <sequence>
            <element name="node" type="famos:NodeType"
               minOccurs="0" maxOccurs="unbounded"/>
            <element name="edge" type="famos:EdgeType"
               minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
<complexType name="NodeType">
   <annotation>
      <documentation>A node of a graph has a Point position,
       an id, and may have a number of attributes</documentation>
   </annotation>
   <complexContent>
      <extension base="gml:AbstractFeatureType">
         <sequence>
            <element name="nodeID" type="integer"
               nillable="false"/>
            <element ref="gml:position"/>
            <element ref="famos:attribute" minOccurs="0"
               maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
<complexType name="EdgeType">
   <annotation>
      <documentation>An edge of a graph has a LineString center
       line, a length attribute and may have any additional
       attributes</documentation>
   </annotation>
   <complexContent>
      <extension base="gml:AbstractFeatureType">
         <sequence>
            <choice>
               <element ref="gml:centerLineOf"/>
               <sequence>
                  <element name="startNodeID" type="integer"/>
                  <element name="endNodeID" type="integer"/>
               </sequence>
            </choice>
            <element name="length" type="double">
               <annotation>
                  <documentation>the length of the edge
```

```
                  [currently: in meters]</documentation>
               </annotation>
            </element>
            <element ref="famos:attribute" minOccurs="0"
               maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
<complexType name="AttributeAreaType">
   <annotation>
      <documentation>An attribute area defines attributes for a
       part of the space</documentation>
   </annotation>
   <complexContent>
      <extension base="gml:AbstractFeatureType">
         <sequence>
            <choice>
               <element ref="gml:Polygon"/>
               <element name="nodeList" type="famos:NodeType"
                  maxOccurs="unbounded"/>
            </choice>
            <element ref="famos:attribute" minOccurs="0"
               maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
<complexType name="AttributeType">
   <annotation>
      <documentation>An attribute is basically a named value,
       i.e. a pair of name and value</documentation>
   </annotation>
   <sequence>
      <element name="name" type="string"/>
      <element name="valueType" type="string" use="optional">
         <annotation>
            <documentation>the java class of the value object
            </documentation>
         </annotation>
      </element>
      <element name="value">
         <complexType>
            <choice>
               <element ref="famos:_Object"/>
```

```
                <element name="simpleValue" type="anySimpleType"/>
            </choice>
         </complexType>
      </element>
   </sequence>
</complexType>
<complexType name="LengthType">
   <annotation>
      <documentation>models desmoj.app.Length</documentation>
   </annotation>
   <sequence>
      <element name="value" type="double"/>
      <element name="unitString">
         <simpleType>
            <restriction base="string">
               <enumeration value="mm"/>
               <enumeration value="cm"/>
               <enumeration value="m"/>
               <enumeration value="km"/>
            </restriction>
         </simpleType>
      </element>
   </sequence>
</complexType>
<complexType name="VelocityType">
   <annotation>
      <documentation>models desmoj.app.Velocity</documentation>
   </annotation>
   <sequence>
      <element name="value" type="double"/>
      <element name="unitString">
         <simpleType>
            <restriction base="string">
               <enumeration value="m/s"/>
               <enumeration value="km/h"/>
            </restriction>
         </simpleType>
      </element>
   </sequence>
</complexType>
<complexType name="DurationType">
   <annotation>
      <documentation>models desmoj.app.Duration</documentation>
   </annotation>
   <sequence>
```

```
      <element name="value" type="double"/>
      <element name="unitString">
         <simpleType>
            <restriction base="string">
               <enumeration value="ms"/>
               <enumeration value="s"/>
               <enumeration value="min"/>
               <enumeration value="h"/>
            </restriction>
         </simpleType>
      </element>
   </sequence>
</complexType>
<complexType name="WeightType">
   <annotation>
      <documentation>models desmoj.app.Weight</documentation>
   </annotation>
   <sequence>
      <element name="value" type="double"/>
      <element name="unitString">
         <simpleType>
            <restriction base="string">
            <restriction base="string">
               <enumeration value="mg"/>
               <enumeration value="g"/>
               <enumeration value="kg"/>
               <enumeration value="t"/>
            </restriction>
         </simpleType>
      </element>
   </sequence>
</complexType>
<complexType name="CapacityType">
   <annotation>
      <documentation>models desmoj.app.Capacity</documentation>
   </annotation>
   <sequence>
      <element name="value" type="double"/>
      <element name="unitString">
         <simpleType>
            <restriction base="string">
               <enumeration value="ml"/>
               <enumeration value="l"/>
            </restriction>
         </simpleType>
```

```
      </element>
   </sequence>
</complexType>

</schema>
```