1. Einleitung

Bei Simulation geht es darum, ein System, das sich für die Durchführung von realen Tests nicht eignet, durch ein Modell zu beschreiben, das die wesentlichen Eigenschaften des Systems abbildet. Mit Hilfe dieses Modells können dann Experimente durchgeführt werden, deren Ergebnisse Rückschlüsse auf das Verhalten des simulierten Systems zulassen sollen. Diese Erklärung von Simulation deckt sich mit der Definition von Gehring [Geh98], nach der Simulation das Nachbilden von Prozessen realer Systeme in einem Modell und das anschließende Durchführen von Experimenten an diesem Modell ist.

Auf eine exakte Definition des Systembegriffs wird hier verzichtet. Man kann ein System als Gegenstand des Interesses, das definierte Ein- und Ausgabeschnittstellen zu seiner Umwelt hat, ansehen. Bei Page [Pag91] wird ein System als ein vom Menschen abgegrenzter Ausschnitt der Gesamtmenge der Objekte und der Beziehungen der Objekte in der Umwelt des Menschen angesehen, wobei es sich um natürliche, technische oder soziale Systeme handeln kann.

Neben Simulationen, die die Aufgabe haben, Rückschlüsse auf das Verhalten eines simulierten Systems ziehen zu können, gibt es auch Simulationen, die einen realen Vorgang in Echtzeit abbilden und/oder einen interaktiven Eingriff durch einen menschlichen Akteur (z.B. Flugsimulatoren zum Training von Piloten) erlauben sollen. Man spricht dann von Echtzeitsimulation bzw. Interaktiver Simulation. In dieser Arbeit werden diese Arten der Simulation nur im Ausblick behandelt.

In Abschnitt 1.1 werden mögliche Gründe aufgezählt, warum man Experimente nicht direkt auf dem zu untersuchenden System, sondern mit Hilfe von Simulation auf einem Rechner durchführt. Abschnitt 1.2 verdeutlicht den Zusammenhang zwischen dem simulierten System (Realsystem), den Vorgängen im Realsystem, dem Modell sowie der Simulation am Modell und formalisiert den Vorgang der Modellbildung und Simulation. In Abschnitt 1.3 wird eine Einteilung von Simulationssystemen in diskret und kontinuierlich vorgenommen. In Abschnitt 1.4 werden einige weit verbreitete Simulationssysteme vorgestellt. Abschnitt 1.5 beschäftigt sich speziell mit Agentenbasierter Simulation, wobei auch hier einige weit verbreitete agentenbasierte Simulationssysteme vorgestellt werden. In Abschnitt 1.6 werden die Ergebnisse dieser Arbeit in gebündelter Form vorgestellt.

1.1 Gründe für Simulation

Will man Ergebnisse über mögliches Verhalten von Systemen erzielen, so bieten sich in erster Linie Experimente auf diesem System, dem "Realsystem", an. In vielen Fällen ist eine solche Vorgehensweise aber nicht zweckmäßig oder gar nicht möglich. Dies ist z.B. in den folgenden Situationen[1] der Fall:

a) Es soll das Verhalten eines Realsystems untersucht werden, das (noch) nicht existiert.

b) Experimente mit dem Realsystem wären zu gefährlich oder zu teuer.

c) Das Realsystem würde durch Experimente zerstört.

d) Es soll eine Prognose für die zukünftige Entwicklung des Realsystems erstellt werden.

e) Die Vorgänge laufen in der Realität zu schnell oder zu langsam ab.

Im Folgenden werden Beispiele für solche Situationen genannt. An den Beispielen sieht man, dass oftmals gleich mehrere Gründe gegen Experimente mit einem Realsystem sprechen.

- Ermittlung der maximalen Belastbarkeit eines Fahrstuhls (b, c).
- Ermittlung der optimalen Flugbahn eines Satelliten (b, e).
- Ermittlung der zukünftigen Bevölkerungsentwicklung (d).
- Ermittlung der durchschnittlichen Auslastung einer zu bauenden Straße (a).

[1] angelehnt an Sauerbier [Sau99]

Als Alternative bietet sich Simulation an, da hier die genannten Nachteile nicht auftreten.

Eine weitere Alternative wäre die Verwendung analytischer Methoden zur Ermittlung von Ergebnissen auf einem Modell. Allerdings sind reale Systeme im Allgemeinen zu komplex, um für sie eine analytische Lösung zu entwickeln [Sau99].

1.2 Modellbildung

In der Simulation wird das Realsystem durch ein Abbild, das Modell, dargestellt. Man führt dann Experimente auf dem Modell durch und zieht aus den Ergebnissen Rückschlüsse auf das Verhalten des Realsystems.

Meist sind Modelle vereinfachte Abbildungen des Realsystems. Dadurch wird die Komplexität des Systems reduziert. Allerdings darf man nicht so stark vereinfachen, dass die Ergebnisse für das Realsystem keine Relevanz mehr besitzen. Man muss bei der Modellbildung also einen Kompromiss zwischen Einfachheit und Aussagekraft des Modells finden. Je nach Anforderungen und Ziel der Simulationsstudie eröffnet sich dem Modellierer dabei ein weites Spektrum von einer groben Annäherung des Systems (starke Vereinfachung) und der vollständigen Emulation des Systems (keine Vereinfachung).

Formal kann man den Simulationsvorgang[2] wie folgt beschreiben. Sei S ein System, das eine Eingabe x aus der Menge X in eine Ausgabe y aus der Menge Y transformiert. Sei ferner M (das Modell von S) ein System, das eine Eingabe x' aus der Menge X' in eine Ausgabe y' aus der Menge Y' transformiert. Man kann dabei der Einfachheit halber S und M als Funktionen $S: X \rightarrow Y$ und $M: X' \rightarrow Y'$ betrachten[3]. Seien $g: X \rightarrow X'$ und $h: Y' \rightarrow Y$ Funktionen, die der Umcodierung dienen. Oft sind g und h bijektive Funktionen. Dann kann man die Ausgabe des Systems S unter x, also S(x) durch Simulation ermitteln, indem man $S(x) = h(M(g(x)))$ setzt[4]. Das Prinzip besteht also darin, die Eingabe erst umzucodieren, dann das Modell anzuwenden und das Ergebnis zurückzucodieren. Die folgende informelle Grafik (Abbildung 1) verdeutlicht die Formalisierung des Simulationsvorgangs. Dabei ist der Weg der direkten Anwendung des Systems weiß und der Weg der Ergebnisermittlung mittels Simulation grau eingezeichnet.

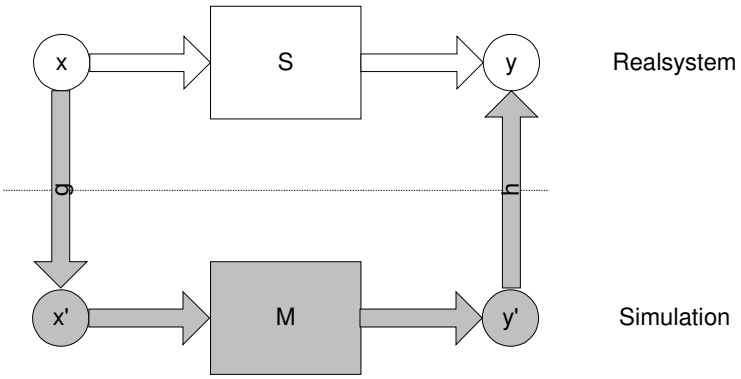

Abbildung 1: Formalisierung des Simulationsvorgangs

Die Anwendung von M findet fast immer rechner- bzw. softwareunterstützt statt, so dass man bei Verwendung des Begriffs Simulation von rechnerunterstützter Simulation ausgehen kann. Law und Kelton [LK82] stellen klar, dass eine Simulation im Prinzip per Hand durchgeführt werden könnte,

[2] angelehnt an Page [Pag91]

[3] Systeme als Funktionen zu charakterisieren, macht nur bei terminierenden Systemen Sinn. Ansonsten bietet sich eine Formalisierung als Eingabe- bzw. Ausgabestrom an. Das Prinzip der Formalisierung des Simulationsvorgangs bleibt aber davon unberührt.

[4] Nur bei der Emulation des Systems ist diese Gleichung auch immer erfüllt. Bei einer Vereinfachung des Systems ist es möglich, dass h(M(g(x))) vom tatsächlichen S(x) abweicht. Da man das tatsächliche S(x) im Allgemeinen aber nicht kennt und gerade deshalb eine Simulation durchführt, bei der man möglicherweise bewusst eine Vereinfachung in Kauf genommen hat, verwendet man h(M(g(x))) trotzdem als Ergebnis für das zu ermittelnde S(x).

aber die Vielzahl der Bestandteile, der Zusammenhänge und der Wiederholungen bestimmter Abläufe einen Einsatz von Rechenanlagen unerlässlich macht.

Bisher wurden damit nur deterministische Simulationen betrachtet. Bei Simulationen, die vom Zufall abhängen, kann man die Ausgabe von M als eine Funktion $M: X' \times Z \rightarrow Y'$ ansehen, wobei Z den Zufall darstellt, also die Menge aller möglichen Zufallsfolgen. Die Ausgabe einer Simulation hängt somit von der Eingabe und vom Zufall ab.

In [MD+02] wird als Anforderung an Plattformen für Agentenbasierte Simulation gestellt, dass sie kontrollierbare Simulationen mit wiederholbaren Ergebnissen ermöglichen sollen. Diese Forderung ist allgemein für Simulation sinnvoll. Bei deterministischen Simulationen bedeutet das, dass bei gleicher Eingabe das Simulationssystem die gleiche Ausgabe liefern soll. Bei Simulationen, die vom Zufall abhängen, erscheint diese Forderung auf den ersten Blick nicht sinnvoll. Jedoch lassen sich Zufallszahlengeneratoren mit einem Startwert initialisieren, so dass bei gleichem Startwert die gleiche Zufallszahlenfolge erzeugt wird.[5] Die Forderung lautet somit, dass bei gleicher Eingabe und bei gleichem Startwert des Zufallszahlengenerators die gleiche Ausgabe erzeugt wird. Das bedeutet mit anderen Worten, dass das Verhalten des Simulationssystems vollständig spezifiziert werden muss und nur dort vom Zufall abhängiges Verhalten erlaubt ist, wo explizit von einer Zufallszahl Gebrauch gemacht wird.

1.3 Diskrete und kontinuierliche Simulation

In der Simulationstheorie unterscheidet man zwischen diskreten und kontinuierlichen Simulationssystemen, wobei sich die Begriffe auf die Art der Änderung des Systemzustands beziehen. Bei Sauerbier [Sau99] werden Simulationssysteme getrennt nach *Größe* und *Prozess* in kontinuierliche bzw. diskrete Systeme eingeteilt. Dabei werden beim Begriff *Größe* die in einem System bzw. Modell betrachteten Größen in kontinuierlich und diskret eingeteilt, während beim Begriff *Prozess* zwischen stetigen und nichtstetigen Prozessen unterschieden wird. Bei nichtstetigen Prozessen spricht man von *diskreter Simulation*, während man bei stetigen Prozessen (diese setzen immer kontinuierliche Größen voraus) von *kontinuierlicher Simulation* spricht. Abbildung 2 (angelehnt an Sauerbier) verdeutlicht den Zusammenhang der Begriffe.

		Größen	
		kontinuierlich	**diskret**
Prozess	**stetig**	kontinuierliche Simulation	nicht möglich
	nichtstetig	diskrete Simulation	

Abbildung 2: Zusammenhang der Begriffe Größe und Prozess

Die kontinuierlichen Systeme können durch physikalische oder biologische Gesetze in Form von Differentialgleichungen dargestellt werden. Hier schreitet die Zeit in der Simulation zwar in diskreten Schritten weiter, die jedoch so klein sind, dass man annähernd kontinuierliches Verhalten wahrnehmen kann. Insofern gibt es keine starre Abgrenzung zwischen kontinuierlichen und diskreten Prozessen. Auch bei Größen kann man nicht immer klar zwischen diskret und kontinuierlich unterscheiden. Geldbeträge sind z.B. zwar eigentlich diskrete Größen, aber wenn es um die Prognose des Jahresumsatzes eines großen Unternehmens geht, spielt es keine Rolle, dass es keine weitere Unterteilung eines Cents geben kann. Insofern sind Geldbeträge oft eine quasi-kontinuierliche Größe.

Bei diskreten Systemen wird bei Page [Pag91] zwischen ereignisorientierten, prozessorientierten und transaktionsorientierten Ansätzen zur Simulation unterschieden.

Im *ereignisorientierten Modell* werden Zustandsänderungen durch das Auftreten von Ereignissen, die zu bestimmten Zeitpunkten stattfinden, vorgenommen, während zwischen zwei Ereignissen der Zustand unverändert bleibt. Ändert man in der Simulation den Zustand gemäß des Auftretens eines

[5] Normalerweise wählt man als Startwert die aktuelle Systemzeit, um gerade zu vermeiden, dass die gleiche Zufallszahlenfolge mehrmals erzeugt wird.

Ereignisses, so spricht man auch von der *Ausführung* des Ereignisses. Man betrachtet beim ereignisorientierten Modell eine Ereignismenge, in der sich alle noch nicht ausgeführten Ereignisse befinden. In einem Ausführungsschritt wird das Ereignis aus der Ereignismenge mit dem frühesten Zeitpunkt ausgewählt und ausgeführt, wodurch sich nicht nur der Zustand ändern kann, sondern außerdem neue Ereignisse in die Ereignismenge eingefügt werden. Bei solchen Ereignissen spricht man von *Folgeereignissen* des ausgeführten Ereignisses. Das ereignisorientierte Modell zeichnet sich durch seine klare und verständliche Struktur aus, die es auch ermöglicht, eine einfache mathematische Formalisierung zu finden, wie in Kapitel 2 aufgezeigt wird.

Im *prozessorientierten Modell* werden die auf ein Objekt bezogenen Aktivitäten zu einem Prozess zusammengefasst. Ein Prozess führt während seiner aktiven Phasen Zustandsänderungen durch, die konzeptuell zeitverzugslos ablaufen. Ein Prozess hat die Möglichkeit, in einen inaktiven Zustand einzutreten. Zu einem späteren Zeitpunkt kann der Prozess dann wieder reaktiviert werden. Prozesse können sich also deaktivieren und andere Prozesse aktivieren. Fasst man die Reaktivierung eines deaktivierten Prozesses als Ereignis auf, so ist das prozessorientierte Modell ähnlich dem ereignisorientierten Modell, wobei die Menge der zukünftigen Ereignisse der Menge der in der Zukunft zu reaktivierenden Prozesse entspricht. Allerdings kann die Struktur im prozessorientierten Modell natürlicher beschrieben werden, weil die Auflösung logisch zusammengehöriger Abläufe auf verschiedene Ereignisroutinen entfällt. Jedoch ist das prozessorientierte Modell schwerer mathematisch zu formalisieren und schwerer in einem Simulationssystem zu implementieren.

Liebl [Lie95] spricht dagegen vom *Process Interaction Approach*, der dem prozessorientierten Modell von Page entspricht. Dabei wird das System so aufgefasst, dass es von einzelnen Individuen bzw. Objekten durchwandert wird. Deren Lebenszyklen weisen dabei genug Ähnlichkeiten auf, um bestimmte Formen von durchlebten Aktivitäten und charakteristische Lebenszyklen zu generischen Lebenszyklen zusammenzufassen. Für jeden Systembestandteil, der das System durchwandert, läuft ein Prozess ab. Es laufen so viele Prozesse parallel ab, wie Systembestandteile zur gleichen Zeit durch das System geschleust werden. Prozesse bestehen aus einer Abfolge von aktiven Perioden und Ruhe- bzw. Wartezeiten, wobei die Aktivitäten mehrerer Prozesse sich manchmal überlappen und manchmal sequentiell ablaufen. Zu den Zeitpunkten, an denen Zustandsänderungen stattfinden, werden die nicht unmittelbar betroffenen Prozesse angehalten und nach dem Zustandssprung reaktiviert, so dass hier also nicht auf die Betrachtung von Ereignissen verzichtet werden kann.

Grundlage des *transaktionsorientierten Modells* bei Page ist die aus der Systemanalyse bekannte Blockdiagrammtechnik. Es gibt neben Blöcken mit fest vorgegebenen Funktionen Transaktionen, die auf ihrem Weg durch die einzelnen Blöcke verändert werden. Dabei handelt es sich bei den Blöcken um die statischen Systemkomponenten (z.B. Bedienstationen, Speicher, Leitstellen), während die Transaktionen die temporären Elemente, nämlich die dynamischen Systemkomponenten, darstellen. Eine Zustandsänderung wird durch eine Transaktion in einem Block ausgelöst. Das Aufeinandertreffen von Transaktion und Block kann als Ereignis betrachtet werden. Die Transaktionen sind in der Ereignisliste nach dem Zeitpunkt des Eintritts in den nächsten Block und nach Prioritäten geordnet.

Bei Liebl gibt es noch den *Activity Scanning Approach*. Dabei wird für jede Zustandsänderung ermittelt, unter welchen Voraussetzungen sie stattfinden kann. Tritt eine Zustandsänderung ein, so wird für jeden anderen Zustand bzw. für jede Aktivität untersucht, ob die Voraussetzungen für eine Änderung erfüllt sind. Die Ereignismenge wird also durch eine Abfrage von Bedingungen ersetzt. Anstatt wie im ereignisorientierten Modell bei Ausführung eines Ereignisses alle Folgeereignisse des Ereignisses zu ermitteln, wird hier für die Ereignisse erst bei ihrer geplanten Ausführung ermittelt, ob sie stattfinden oder nicht.

Zelluläre Automaten (siehe [GS+02]) lassen sich ebenfalls als diskrete Simulation klassifizieren. Dabei handelt es sich um ein regelmäßiges Muster von Zellen, die jeweils einen endlichen Automaten beinhalten. Die Zellen aktualisieren ihren Zustand abhängig von einer Zustandsübergangsfunktion, die ihren eigenen (alten) Zustand und den (alten) Zustand der Nachbarzellen berücksichtigt. Trotz der einfachen lokalen Regeln lässt sich dabei oft komplexes globales Verhalten beobachten. So lassen sich z.B. Ansiedlungen in einer Stadt oder Vorgänge in der Biologie, wie z.B. die Ausbreitung einer Bakterienkultur, gut durch zelluläre Automaten beschreiben.

Eine weitere, ebenfalls als diskret zu klassifizierende Methode der Simulation ist die Spieltheorie [McC97]. Die Spieltheorie ist ein interdisziplinärer Ansatz zur Untersuchung menschlichem

Verhaltens. Dabei sind *Spiele* eine menschliche Interaktion, bei denen das Ergebnis von den Entscheidungen der Mitspieler abhängt, wobei der Spielbegriff recht weit gefasst ist. Typisches Beispiel ist das sogenannte Gefangenendilemma [O'Ri00], bei dem zwei Gefangene, die getrennt voneinander verhört werden, die Wahl haben, zu schweigen oder jeweils den anderen Gefangenen zu belasten. Schweigen beide, erhalten sie nur eine geringe Strafe, weil ihnen nicht viel nachgewiesen werden kann. Belasten sie sich gegenseitig, erhalten beide eine höhere Strafe. Belastet einer den anderen, während der andere schweigt, so wird ersterer als Kronzeuge freigelassen, während der andere eine noch höhere Strafe bekommt. Die Spieltheorie hat in zweierlei Hinsicht etwas mit Simulation zu tun. Zum einen ist sie ein Ansatz zur Verhaltensmodellierung innerhalb von Simulationen, zum anderen ist sie aber auch ein Anwendungsfall der Simulation, etwa, wenn man Strategien für das Gefangenendilemma wie bei [O'Ri00] gegeneinander antreten lässt.

Am weitesten verbreitet ist diskrete Simulation unter Verwendung des ereignisorientierten Modells, die diskrete ereignisorientierte Simulation, auch kurz *Diskrete-Ereignis-Simulation* genannt. Sie wird in Kapitel 2 genauer beschrieben und formalisiert.

1.4 Simulationssysteme

In diesem Abschnitt wird eine repräsentative Auswahl von in der Praxis verwendeten Simulationssystemen vorgestellt. Auf agentenbasierte Simulationssysteme wird dabei zunächst verzichtet, sie werden in Abschnitt 1.5 berücksichtigt.

Arena [Arena00, TP97] ist ein von der *Rockwell Software Inc.* entwickeltes Werkzeug, das es Systemanalytikern erlaubt, animierte Simulationsmodelle zu erschaffen. Arena verwendet ein objektorientiertes Design für eine komplett grafische Modellentwicklung. Dabei platzieren Systemanalytiker grafische Objekte – Module genannt – zum Zwecke der Definition von Systemkomponenten wie Maschinen, Bediener oder Geräte. Arena basiert auf der Simulationssprache SIMAN, wobei aus dem grafischen Arena-Modell automatisch ein SIMAN-Modell generiert und ausgeführt wird. Arena beinhaltet u.a. einen Flussdiagrammeditor, um Prozesse zu beschrieben und eine automatische Animation von Flussdiagrammen, um eine dynamische Visualisierung von Prozessflüssen zu ermöglichen.

Bei *eM-Plant* [Tec04] (früher SiMPLE++) handelt es sich um eine von der Firma *Tecnomatix* entwickelte, vollständig objektorientierte Software zur grafischen Modellierung, Visualisierung, Simulation und Optimierung von Produktion, Logistik und Geschäftsprozessen. Es können sowohl diskrete als auch kontinuierliche Prozesse modelliert, simuliert und animiert werden. Bei eM-Plant erfolgt die Modellierung nach dem Bausteinkonzept. Es gibt eine Menge von Grundbausteinen, die ein Benutzer verwenden und aus denen er neue Bausteine (Anwenderbausteine) grafisch und interaktiv zusammensetzen kann. Die Steuerung der Anwenderbausteine kann mit Hilfe der Steuerungssprache SimTALK durch den Benutzer spezifiziert werden.

Silk [Kil00] ist eine Java-Programmbibliothek, die objektorientierte Simulation unterstützt. Silk-Objekte drücken das exakte Verhalten von individuellen Entitäts-Threads aus. Sie sind wiederverwendbar, weil sie leicht verändert, editiert und zusammengebaut werden können. Man kann Silk als Menge von Simulations-Klassenbibliotheken auffassen, aber auch als eine prozessorientierte Modellierungssprache oder als eine visuelle Modellierungsumgebung. Silk ist ein Tool für die Erstellung von objektorientierten Simulationskomponenten und domänenspezifischen Simulatoren. Es handelt sich nach [Kil00] nicht um eine Simulationssprache, sondern eine Simulationserweiterung von Java. Diese Ansicht wird hier nicht geteilt, da es eine Java-Programmbibliothek ist und nicht die Sprachkonstrukte der Sprache Java erweitert. Silk ist am mächtigsten, wenn man als Benutzer einem konsistenten Entwurfsmuster für objektorientierte Modellierung folgt, bei dem jede intelligente Komponente als unabhängige Silk-Entitätsklasse modelliert wird. Da man nur einen Java-Compiler braucht, kann man jede beliebige Java-Entwicklungsumgebung verwenden.

1.5 Agentenbasierte Simulation

Da die Modellierung und Simulation komplexer Systeme, die aus mehreren, ggf. hierarchisch strukturierten Bestandteilen bzw. Subsystemen bestehen, in der Diskreten-Ereignis-Simulation schnell an ihre Grenzen stößt, weil es schwierig ist, einzelne Systembestandteile unabhängig voneinander zu modellieren, hat sich in der Praxis ein neues Paradigma, die Agentenbasierte Simulation entwickelt.

In Abschnitt 1.5.1 wird zunächst der allgemeine Agentenbegriff erläutert. In Abschnitt 1.5.2 werden die Grundlagen Agentenbasierter Simulation vorgestellt. In den folgenden Abschnitten werden eine repräsentative Auswahl von Plattformen für Agentenbasierte Simulation (*Swarm* [Swarm00, Swarm96], *RePast* [RePast04], *Spades* [RR03], *Sesam* [Klü01, SeSAm04], *MadKit* [MadKit00] und *CORMAS* [Cormas01, Cormas00]) in Abschnitt 1.5.3, sowie die als internationale Technologie-Wettbewerbe konzipierten Simulationssysteme *RoboCup* [Robo02, Robo98] in Abschnitt 1.5.4 und *Trading Agent Competition* [TAC04] in Abschnitt 1.5.5 diskutiert.

1.5.1 Agenten

Bevor auf Agentenbasierte Simulation eingegangen wird, wird zunächst der Agentenbegriff erläutert. Er subsummiert sowohl natürliche als auch künstliche Systeme.

In den Wirtschaftswissenschaften wird ein Agent als jemand bezeichnet, der im Auftrag eines anderen handelt (z.B. Verträge abschließt). Bei Software-Agenten, die im Auftrage eines menschlichen Benutzers handeln, z.B. das Internet nach Informationen durchzusuchen, hat das Wort *Agent* ebenfalls diese Bedeutung (siehe [Klü01]).

Bei Wooldridge und Jennings [WJ95] sind Agenten Entitäten, die miteinander und mit ihrer Umwelt interagieren. Die an einem System beteiligten Agenten – Tiere, Menschen, soziale Institutionen (wie Vereine, Firmen oder Behörden), Softwaresysteme oder Maschinen – können selbständig Handlungen ausführen, ihre Umgebung wahrnehmen und verändern und auf Veränderungen der Umgebung reagieren; sie verfügen über einen mentalen Zustand, der aus Komponenten wie Wissen, Zielen, Erinnerungen und Verpflichtungen besteht.

Nach Franklin und Graesser [FG97] ist ein Agent ein System, das Teil seiner Umgebung ist, diese wahrnimmt und in ihr Aktionen ausführt, und damit im Zeitablauf beeinflusst, was es in der Zukunft wahrnehmen wird.

Wagner [Wag97b] unterscheidet zwischen drei Ansätzen, um das Agentenkonzept zu definieren: dem softwaretechnischen Ansatz, dem Wissensrepräsentationsansatz und dem erkenntnistheoretischen Ansatz.

Nach dem softwaretechnischen Ansatz werden künstliche Informationsverarbeitungssysteme in den Vordergrund gestellt. So definieren Genesereth und Ketchpel [GK94], dass eine Enitität ein Softwareagent ist, genau dann, wenn sie korrekt in einer Agentenkommunikationssprache (*agent communication language, ACL*), wie z.B. KQML, kommuniziert. Solch eine Sprache basiert auf getypten Nachrichten.

Der Wissensrepräsentationsansatz entwickelt formale Konzepte für menschliche Wahrnehmungsfunktionen, um sie als Softwaresysteme zu rekonstruieren. Shoham [Sho93] definiert das mentalistische Agentenmodell, nach der ein Agent eine Entität ist, deren Zustand als aus mentalen Komponenten (Wissen, Fähigkeiten, Wahlmöglichkeiten und Verpflichtungen) bestehend angesehen wird. Die dynamischen Konzepte eines Agenten wie Wahrnehmungen und Aktionen berücksichtigt die Definition von Hayes-Roth [Hay95], nach der intelligente Agenten sich dadurch auszeichnen, dass sie permanent drei Funktionen ausführen: die Wahrnehmung dynamischer Zustände in der Umwelt, die Ausführung von Aktionen, um diese Zustände zu ändern und das Nachdenken, um Wahrnehmungen zu interpretieren, Probleme zu lösen, Schlüsse zu ziehen und um Aktionen festzulegen.

Der erkenntnistheoretische Ansatz befasst sich mit der Analyse natürlicher Informationsverarbeitungssysteme (z.B. von Tieren oder Menschen). Von Rao und Georgeff [RG91] stammt ein Modell, dass Wissen, Wünsche und Absichten in den Vordergrund stellt (*Belief-Desire-Intention, BDI*) und auf Arbeiten in der Philosophie, Erkenntnistheorie und künstlichen Intelligenz basiert. Aspekte wie Wahrnehmung, Reaktion und Kommunikation werden dabei aber vernachlässigt.

Wagner [Wag97b] verdeutlicht außerdem, dass es keine allgemeingültige formale Definition eines Agenten geben kann. Stattdessen definiert er wissens- und wahrnehmungsbasierte Agenten (*Knowledge- and Perception-Based Agents, KP-Agents*), deren Zustand eine Wissensbasis und Wahrnehmungen in Form einer Ereignisschlange beinhaltet. Ferner definiert er KPTI-Agenten (*Knowledge-Perception-Task-Intention)*, die zusätzlich zu den Funktionen eines KP-Agenten die Fähigkeit haben, Aktionen auszuführen sowie Pläne zu generieren und auszuführen. Dabei wird

zwischen Aktionen, die ausgeführt werden, um eine Aufgabe zu erfüllen oder ein Ziel zu erreichen und Reaktionen, die durch Kommunikations- oder Umweltereignisse ausgelöst werden, unterschieden.

Bei Klügl [Klü01] werden sechs charakteristische Eigenschaften von Agenten aufgeführt: situiert, reaktiv, autonom, sozial, rational und anthropomorph. Ein Agent ist situiert, wenn er sich in einer Umgebung befindet, sie über Sensoren wahrnimmt und über Effektoren verändert. Er ist reaktiv, wenn er Veränderungen in seiner Umwelt erkennen und sein Verhalten diesen Veränderungen anpassen kann. Er ist autonom, wenn er seine Aktionen selbst bestimmt. Sozial ist ein Agent, wenn er mit anderen Agenten interagiert. Er ist rational, wenn er Ziele besitzt und diese verfolgt. Als anthropomorph wird ein Agent bezeichnet, wenn er sich ähnlich wie ein Mensch verhält, d.h. mentale Konzepte wie Wissen, Wünsche und Absichten besitzt.

1.5.2 Grundlagen

In der Agentenbasierten Simulation geht es darum, komplexe Realsysteme, die aus miteinander und mit ihrer Umwelt interagierenden Entitäten bestehen, als *Multiagentensysteme* [WJ95] zu interpretieren und gemäß dieser Interpretation zu simulieren. Im Vergleich zu vielen traditionellen Methoden der Simulation – wie kontinuierliche Simulation (Differentialgleichungen), Diskrete-Ereignis-Simulation, zelluläre Automaten und Spieltheorie – ist die Agentenbasierte Simulation weniger abstrakt und realitätsnäher.

Agentenbasierte Simulation wird heute in vielen Bereichen der Forschung angewendet, insbesondere in

- der **Biologie**, z.B. zur Untersuchung von Ökosystemen oder in der Populationsethologie (speziell Ameisen, Hummeln, Honigbienen), siehe z.B. [Klü01];
- der **Bioinformatik**, z.B. bei der Genexpressionsanalyse, bei der ein Stoffwechsel-/ Signaltransduktions-Netzwerk mittels eines multiagentenbasierten Ansatzes simuliert werden soll [Schob02];
- den **Wirtschaftswissenschaften**: z.B. in der Simulation von Auktionen und Märkten, siehe *Trading Agent Competition* [TAC04];
- den **Sozialwissenschaften**: z.B. wird in [EW02] ein spezielles Modell der Rassentrennung als Simulationsbeispiel verwendet und in [Hal02] wird die Kooperation in Teams untersucht;
- den **Ingenieurwissenschaften**: zur Analyse und zum Design komplexer (sozio-) technischer Systeme, wie z.B. Fahrerloser Transportsysteme (siehe Kapitel 5 und Kapitel 6).

In [MD+02] werden Anforderungen an Plattformen für Agentenbasierte Simulation zusammengestellt. Insbesondere sollen sie

- kontrollierbare Simulationen mit wiederholbaren Ergebnissen ermöglichen;
- asynchronen Nachrichtenaustausch zwischen Agenten zulassen;
- außer externen Ereignissen (wie z.B. Nachrichtenaustausch, Handlungen von Agenten) auch interne/kognitive Ereignisse beobachtbar machen;
- verschiedene (vom Simulator kontrollierte und nicht vom Simulator kontrollierte) ‚Umgebungen' in die Simulation einbinden können;
- Mechanismen bereitstellen, die eine Intervention des Experimentators in das Simulationsgeschehen ermöglichen.

Wie die Autoren bemerken, werden speziell die letzten zwei dieser Anforderungen von den heute verfügbaren Plattformen nicht erfüllt.

1.5.3 Plattformen für Agentenbasierte Simulation

Swarm [Swarm00, Swarm96] ist ein weit verbreitetes System zur Programmierung von Multiagentensimulationen, das ursprünglich am *Santa Fe Institut* (USA) entwickelt wurde und mittlerweile durch die *Swarm Development Group* weiterentwickelt wird. Es hat seinen Ursprung im

Bereich der Artificial-Life-Forschung, dient aber allgemein zur Simulation komplexer adaptiver Systeme. Ein ‚Swarm' (engl. für *Schwarm*) ist eine Menge von Agenten, die mit Hilfe diskreter Ereignisse miteinander interagieren. Softwaretechnisch ist ein Swarm ein Objekt, das Speicherallokation und Ereignisscheduling implementiert. Eine einfache Swarm-Simulation besteht aus einem ‚Modell-Swarm' und einem ‚Observer-Swarm', dessen Agenten zur Beobachtung der Simulation dienen.

RePast (Recursive Porous Agent Simulation Toolkit) [RePast04] ist ein Software-Framework, das von der *Social Science Research Computing* der *Universität Chicago* (USA) entwickelt wurde. Mit RePast können Agentenbasierte Simulationen mit Hilfe der Sprache Java durchgeführt werden. Es bietet eine Klassenbibliothek an, um eine Agentenbasierte Simulation zu erzeugen, laufen zu lassen, anzuzeigen und Daten zu sammeln. Außerdem kann man Schnappschüsse und Filme von laufenden Simulationen erstellen. RePast benutzt Teile von Swarm und hat zusätzliche Features wie Manipulation des Laufzeitmodells über eine grafische Benutzerschnittstelle. In RePast wird eine Simulation als Zustandsmaschine betrachtet, wobei sich der Gesamtzustand in die Zustände der einzelnen Komponenten aufteilt. Jede Komponente besitzt eine Infrastruktur und eine Repräsentation. Die Infrastruktur beinhaltet Mechanismen, um z.B. die Simulation laufen zu lassen und Daten zu sammeln. Die Repräsentation ist das Simulationsmodell selbst.

Spades (System for Parallel Agent Discrete Event Simulation) [RR03] ist ein Middleware-System zur Erstellung von agentenbasierten KI-Simulationen. Dabei handelt es sich bei den Agenten ausschließlich um Software-Agenten, die einen kontinuierlichen Wahrnehmungs-, Nachdenk- und Handlungszyklus durchlaufen. Spades erlaubt beliebige Softwaresysteme zur Implementierung der Agenten. Voraussetzung ist lediglich, dass die Agenten die vorgegebene Schnittstelle implementieren. Die einzelnen Agenten kommunizieren über TCP/IP mit dem Simulator und können somit auf verschiedene Rechner verteilt werden. Die Zeit, die ein Agent zum Nachdenken benötigt, wird explizit erfasst, so dass das Ergebnis einer verteilten Simulation unabhängig von der aktuellen Netzlast und von der Auslastung der beteiligten Rechner ist und somit Ergebnisse reproduzierbar sind. Das Metamodell, das dem Simulator von Spades zugrunde liegt, basiert auf der Diskreten-Ereignis-Simulation, wobei es möglich ist, dass die Reihenfolge der ausgeführten Ereignisse von der tatsächlichen abweicht, solange die Kausalordnung eingehalten und damit das Ergebnis der Simulation nicht beeinflusst ist.

SeSAm (Shell for Simulated Multi-Agent Systems) [Klü01, SeSAm04] ist eine an der *Universität Würzburg* entwickelte Simulationsplattform. Sie findet Anwendung in der Verkehrssimulation und in biologischer Simulation (Ameisen, Hummeln, Honigbienen), ist aber auf jede spezielle Domäne adaptierbar. SeSAm besteht aus mehreren Komponenten: einem grafischen Modellierungstool, einer expliziten Modellrepräsentation, einem Simulator, einem Visualisierungstool, Protokolldaten und Analyseroutinen. Die direkte Manipulation einzelner Agenten wird durch einen Interventions-mechanismus ermöglicht. Der Zustand eines Agenten wird durch eine Menge von Variablen repräsentiert. Sein Verhalten wird durch UML-ähnliche Aktivitätsdiagramme beschrieben. Ein Agent befindet sich zu jedem Zeitpunkt in genau einer Aktivität, die aus einer Folge von Aktionen besteht. Von einer Aktivität in eine andere wechselt der Agent unter Benutzung von Regeln.

MadKit (Multi-Agent Development Kit) [MadKit00] wurde an der *Universität Montpellier* (Frankreich) entwickelt und basiert auf einem organisationsorientierten Paradigma, dem AGR-Ansatz [FG98] mit den drei Grundbegriffen *Agent*, *Gruppe* und *Rolle*. MadKit erlaubt die Entwicklung von effizienten verteilten Simulationen, wobei die technischen Grundlagen der Verteilung (Sockets, Ports) für den Programmierer transparent sind.

CORMAS (Common-pool Resources and Multi-Agent Systems) [Cormas01, Cormas00] ist ein Simulationstool, das vom *Center de Cooperation International en Recherche Agronomique pour le Dévelopment (CIRAD)* in Frankreich speziell zur Untersuchung von Problemen beim Management erneuerbarer natürlicher Ressourcen entwickelt wurde. Es unterstützt Interaktionen zwischen Individuen und Gruppen, die sich solche Ressourcen teilen. CORMAS unterteilt die Welt in räumliche, passive und soziale Entitäten. Die räumlichen Entitäten besitzen natürliche Ressourcen. Passive Entitäten sind Objekte und Nachrichten, während soziale Entitäten Agenten sind. CORMAS basiert auf der Smalltalk-Programmierumgebung *VisualWorks*. Der Modellierungsprozess mit Hilfe von CORMAS besteht aus drei Teilen. Der erste Teil ist die Definition von Modellentitäten. Beim zweiten Teil handelt es sich um die Kontrolle der globalen Entwicklung des Modells, das ist die Definition der Modellklasse selbst. Der dritte Teil ist die Definition der Modellbeobachtung.

1.5.4 RoboCup Simulationsliga

Die Simulationsliga der Roboterfußballinitiative *RoboCup* ist ein Wettbewerb, in dem zwei Mannschaften gegeneinander antreten, deren Spieler durch Softwareagenten simuliert werden. Ein Spiel wird als verteilte Simulation auf der Basis einer Client-/Serverarchitektur durchgeführt. Ein spezielles Programm, der *Soccer Server* [Robo02, Robo98], verwaltet den Zustand eines virtuellen Spielfelds einschließlich aller Bewegungs- und Kommunikationshandlungen von Spielern. Der Soccer Server erfährt von allen Handlungsentscheidungen der Spieler-Clients, berechnet ihre Auswirkungen auf den Zustand des Spielfelds (z.B. Positionsveränderungen, Kollisionen, Ball im Tor) und ermittelt für jeden Spieler dessen Wahrnehmungen (in Abhängigkeit seiner Position und Gesichtsfeldausrichtung). Wahrnehmungen beziehen sich auf andere Objekte (andere Spieler, Ball, Eckfahne, Torpfosten), deren Entfernung und Ausrichtung sowie auf Zurufe von anderen Spielern.

Der Server arbeitet in Zyklen, die jeweils ein diskretes Zeitintervall abbilden. Jeder Zyklus hat eine feste Zeitlänge, wodurch die Clients gezwungen sind, sicherzustellen, dass der Server ihre Aktionen auch innerhalb des Zyklus-Zeitintervalls erfährt. Die Spiele können durch ein auf die RoboCup-Domäne zugeschnittenes Visualisierungsprogramm, den *Soccer Monitor*, mitverfolgt werden.

Das RoboCup-Simulationssystem ist zwar keine allgemeine Simulationsplattform, kann aber für die Entwicklung von funktional-verteilten Simulationssystemen als Vorbild genommen werden. Da das System die Einbindung von (unabhängig erstellten) Fremdsimulatoren – den Spieler-Clients – erlaubt, kann es als Interaktive Agentenbasierte Echtzeitsimulation klassifiziert werden.

1.5.5 Trading Agent Competition

Trading Agent Competition (TAC) [TAC04] ist ein Wettbewerb, bei dem Reisebüro-Softwareagenten in einem fiktiven Touristikszenario Waren kaufen und verkaufen, um die Wünsche ihrer Kunden zu befriedigen. Jeder Agent hat dabei die Aufgabe, Reisepakete (Flug, Hotel, Veranstaltungskarten) für Reisen eines 5-Tage-Zeitraums zusammenzustellen und dabei die Summe der Befriedigungen der individuellen Wünsche seiner acht Kunden zu maximieren. Flüge, Hotelübernachtungen und Veranstaltungskarten werden in separaten Märkten nach unterschiedlichen Regeln gehandelt. Flüge werden von der Fluggesellschaft zum simulations-aktuellen Preis verkauft, Hotelübernachtungen in einer Englischen Auktion versteigert und Veranstaltungskarten auf einer Börse gehandelt, bei der die Agenten untereinander kaufen und verkaufen können. Die einzelnen Aktionen der Agenten, z.B. Abgabe von Geboten, werden als XML-basierte Nachrichten an ein Auktionssystem gesendet, das die Auktionen durchführt und den teilnehmenden Reisebüro-Agenten deren Ergebnisse mitteilt.

Wie bei RoboCup handelt es sich auch bei TAC um eine Interaktive Agentenbasierte Echtzeit-Simulation. Im Unterschied zu RoboCup sind bei TAC jedoch nicht alle Teilnehmer Simulatoren. Die Reisebüroagenten und das Auktionssystem repräsentieren nämlich reale E-Commerce-Software zur Automatisierung von Touristikdienstleistungen. Insofern ist TAC eine Interaktive Agentenbasierte Simulation, bei der die eingebundenen Simulationsteilnehmer Softwaresysteme sind.

1.6 Ergebnisse dieser Arbeit

Für die Agentenbasierte Simulation sind insbesondere die folgenden Anforderungen von Belang:

- die Simulationsspezifikationssprache und der Simulator basieren auf einem agentenorientierten Metamodell
- die Simulationsspezifikationssprache ist deklarativ
- es gibt eine visuelle, UML-basierte Simulationsspezifikationssprache

Die Forderung nach einem Metamodell entspringt aus dem Wunsch, einem Simulator eine theoretische Grundlage zu geben, was in der Praxis oft nicht der Fall ist. Durch die Deklarativität der Simulationsspezifikationssprache sollen Spezifikation und Ausführung voneinander entkoppelt werden, so dass es auch möglich wird, ein einmal spezifiziertes System in verschiedenen Simulationsplattformen auszuführen. Außerdem ist eine deklarative Spezifikation natürlicher, und gegenüber einer imperativen Spezifikation ist die Lesbarkeit erhöht. Der Vorteil bei einer visuellen Spezifikationssprache ist, dass sie leichter erfassbar ist und damit die Simulationsmodelle leichter kommunizierbar sind. UML-Basierung ist sinnvoll, da sich UML als Standard in Forschung und

Industrie etabliert und die Konzepte der Softwaremodellierung der letzten Jahrzehnte in sich aufgenommen hat.

In Abschnitt 1.5 wird deutlich, dass die genannten Anforderungen von den verfügbaren Systemen gar nicht oder nur teilweise erfüllt werden (siehe Abbildung 3).

Anforderung	*Swarm*	*RePast*	*Spades*	*SeSAm*	*MadKit*	*CORMAS*	*RoboCup*	*TAC*
Metamodell	-	√	√	-	√	√	-	-
Deklarativ	-	-	-	√	-	-	-	-
Visuell	√	√	-	√	-	√	-	-
UML-basiert	-	-	-	-	-	-	-	-

Abbildung 3: Vergleich einiger agentenbasierter Simulationssysteme

Das in dieser Arbeit vorgestellte Simulationssystem erfüllt alle genannten Anforderungen. Es basiert auf einem Agentenbasierten Metamodell, das in Abschnitt 4.3 vorgestellt wird. Die in Kapitel 7 vorgestellte grafische High-Level-Spezifikationssprache für die Agentenbasierte Simulation ist deklarativ, da sie auf Reaktionsregeln basiert, und sie hat die Form eines UML-Profils, so dass man jedes beliebige UML-Tool zur Spezifikation verwenden kann.

Die Simulationsparadigmen der Diskreten-Ereignis-Simulation, die sich über Jahrzehnte hinweg etabliert hat und der Agentenbasierten Simulation, die strukturierte und realitätsnahe Simulationen erlaubt, werden als getrennte und voneinander unabhängige Ansätze betrachtet.

In dieser Arbeit wird hingegen gezeigt, wie man durch Differenzierung und Verfeinerung der Grundbegriffe der Diskreten-Ereignis-Simulation und ihrer Ausführungssemantik zu einer Form der Agentenbasierten Simulation gelangt, die man als Agentenbasierte Diskrete-Ereignis-Simulation auffassen kann. Zu diesem Zweck wird zunächst das in der Praxis heterogene Konzept der Diskreten-Ereignis-Simulation mathematisch formalisiert. Auf dieser Formalisierung aufbauend wird dann eine Formalisierung für Agentenbasierte Simulation vorgestellt. Man kann agentenbasierte Simulationsmodelle, die dieser Formalisierung gehorchen, mit einem im Rahmen dieser Arbeit implementierten Simulator ausführen.

Etwas detaillierter: Die Diskrete-Ereignis-Simulation wird erst in ihrer einfachsten Form formalisiert, woraus ein Basismodell der Diskreten-Ereignis-Simulation entsteht. Es werden dann Erweiterungen des Basismodells der Diskreten-Ereignis-Simulation vorgenommen. Dabei werden als drei mögliche Erweiterungen der objektorientierte Systemzustand, die Unterscheidung zwischen exogenen Ereignissen und Folgeereignissen sowie die Ersetzung der Angabe von altem und neuem Zustand durch Angabe von Zustandsbedingung und Zustandseffekt vorgestellt. Danach wird dann das Prinzip der auf der Diskreten-Ereignis-Simulation aufbauenden Objektbasierten Simulation vorgestellt, das diese Erweiterungen in sich vereinigt. Anschließend wird die auf der Objektbasierten Simulation aufbauende Agentenbasierte Simulation vorgestellt. In der Agentenbasierten Simulation wird das Modell der Objektbasierten Simulation um die Modellierung von Nachrichten und um die Trennung zwischen externem und internem Agentenzustand, die die Einführung eines Umgebungssimulators nach sich zieht, erweitert.

Die hier formalisierte Agentenbasierte Simulation beinhaltet im Wesentlichen die Aufteilung des zu simulierenden Systems in aktive Entitäten (Agenten) und passive Entitäten (Objekte). In der Simulation gibt es einen Umgebungssimulator, der die Umgebung sowie die (passiven) Objekte verwaltet, sowie für jeden Agenten einen Agentensimulator. Die Simulation läuft in Zyklen ab, wobei der Ablauf eines Zyklus in der Simulation wie in Abbildung 4 dargestellt aussieht.

Schritt 1	Der Umgebungssimulator ermittelt alle in diesem Zyklus stattfindenden Ereignisse (die *aktuellen Ereignisse*). Dabei handelt es sich um: a) die Aktionen, die die Agenten am Ende des vergangenen Zyklus ausgeführt haben b) Ereignisse aus der Ereignismenge des Umgebungssimulators, die in diesen Zyklus fallen c) exogene Ereignisse, die in diesen Zyklus fallen.
Schritt 2	Der Umgebungssimulator berechnet aus dem aktuellen Umgebungszustand und den aktuellen Ereignissen einen neuen Umgebungszustand, eine Menge von Folgeereignissen sowie für jeden Agenten seine Wahrnehmungen für den folgenden Zyklus. Zu den Wahrnehmungen gehört auch der Empfang von Nachrichten, die andere Agenten versendet haben.
Schritt 3	Der Umgebungssimulator sendet jedem Agentensimulator die (ggf. leere) Menge von Wahrnehmungen des durch ihn simulierten Agenten in diesem Zyklus (die er im vergangenen Zyklus ermittelt hat).
Schritt 4	Jeder Agentensimulator (auch die, die eine leere Menge von Wahrnehmungen empfangen haben) bestimmt zunächst die Menge der in diesem Zyklus stattfindenden internen Ereignisse (die *aktuellen internen Ereignisse*). Dabei handelt es sich um: a) die empfangenen Wahrnehmungen b) interne Zeitereignisse aus der Zeitereignismenge des Agentensimulators, die in diesen Zyklus fallen c) periodische interne Zeitereignisse, die in diesen Zyklus fallen. Anschließend berechnet der Agentensimulator aus dem aktuellem internen Agentenzustand und aus den aktuellen internen Ereignissen einen neuen internen Agentenzustand, eine Menge von internen Folge-Zeitereignissen sowie eine Menge von Aktionen, die der durch ihn simulierte Agent durchführt. Zu den Aktionen gehören auch an andere Agenten versendete Nachrichten. Die (ggf. leere) Menge von Aktionen sendet er an den Umgebungssimulator.
Schritt 5	Der Umgebungssimulator wartet ab, bis er von allen Agenten jeweils die Menge ausgeführter Aktionen empfangen hat, und setzt die Zeit weiter. Die Menge der ausgeführten Aktionen wird im darauf folgenden Zyklus berücksichtigt.

Abbildung 4: Ablauf eines Zyklus in der Agentenbasierten Simulation

Konkrete Agentenbasierte Simulationssysteme können durch Angabe der verwendeten Agenten- und Objekttypen, der Ereignis-, Wahrnehmungs- und Aktionstypen, der konkreten Agenten und Objekte, der auftretenden exogenen Ereignisse sowie der Reaktionsregeln für den Umgebungssimulator (U-Regeln) und für die Agentensimulatoren (A-Regeln) vollständig spezifiziert werden. Diese Spezifikation lässt sich in natürlicher Art und Weise in der Sprache UML durchführen. Dabei werden die Agenten-, Objekt-, Ereignis-, Wahrnehmungs- und Aktionstypen mit Hilfe von Stereotypes unterschieden. Reaktionsregeln werden als n-äre Assoziationen zwischen den beteiligten Typen ausgedrückt, wobei Bedingungen, unter denen die Regeln gelten, die Form von den Assoziationsenden zugeordneten OCL-Ausdrücken haben.

Um zu zeigen, dass es tatsächlich möglich ist, mit Hilfe eines UML-Modells ein ausführbares agentenbasiertes Simulationssystem zu spezifizieren, wird ein im Rahmen dieser Arbeit an der Freien Universität Berlin entwickeltes Simulationssystem vorgestellt.

Der Kern des Simulators ist eine Java-Programmbibliothek, die eine Klasse enthält, die die Ausführung eines agentenbasierten Simulationsmodells organisiert. Ein konkretes Simulationssystem wird ausgeführt, indem die verwendeten Typen (Agenten- und Objekttypen, Ereignis-, Wahrnehmungs- und Aktionstypen) als Unterklassen der generischen Typklassen realisiert werden. Um zu erreichen, dass man als Modellierer vollständig in UML spezifizieren kann und dieses Modell dann direkt ausführen kann, wird das Format XMI verwendet, eine standardisierte XML-basierte Sprache, die eine textuelle Repräsentation eines UML-Modells darstellt. Viele der heutigen UML-

Tools verwenden XMI direkt für die Speicherung der UML-Modelle oder bieten zumindest die Möglichkeit, ein UML-Modell in das Format XMI zu exportieren. Mit Hilfe von XSL-Transformationen wird dann das im XMI-Format vorliegende UML-Modell zunächst in ein XML-basiertes Zwischenformat und dann direkt in vom Simulator verwendbaren Java-Code transformiert. XSL-Transformationen sind eine XML-Anwendung, die es ermöglicht, Dokumente aus einem XML-basierten Format in ein anderes Format gemäß gewisser Transformationsregeln zu überführen.

Ein in dieser Arbeit ausführlich behandeltes Beispiel sind Fahrerlose Transportsysteme (FTS). Ein FTS besteht dabei aus Fahrerlosen Transportfahrzeugen (FTF), die automatisch gesteuert Transportstücke innerhalb einer Verkehrszone von einem Ort zum anderen transportieren. Ein Steuerungssystem ist für die Vergabe von Transportaufträgen, für die Wegplanung und für die Verkehrsregelung zuständig. FTS werden überwiegend für die innerbetriebliche Materialflussabwicklung, z.B. für die Verkettung von Fertigungseinrichtungen und Montageplätzen, eingesetzt. Aufgrund ihrer dezentralen Struktur, die FTS aufgrund ihrer natürlichen Aufteilung in FTF, Fahrkurs und Steuerungssystem haben, eignen sie sich sehr gut zur agentenorientierten Modellierung und Simulation. Wie auch in dieser Arbeit durch Simulation gezeigt, besitzen dezentral gesteuerte FTS eine höhere Robustheit und Flexibilität gegenüber zentral gesteuerten.

Der weitere Verlauf der Arbeit gliedert sich wie folgt. In Kapitel 2 wird ein Basismodell für die Diskrete-Ereignis-Simulation entwickelt und mathematisch formalisiert. Dieses Basismodell wird in Kapitel 3 um einige nützliche, nicht agentenorientierte Erweiterungen angereichert, die dann zur sogenannten Objektbasierten Simulation subsummiert werden. In Kapitel 4 hingegen wird das Modell der Objektbasierten Simulation agentenorientiert erweitert, so dass am Ende ein formales Modell für die Agentenbasierte Simulation steht. Kapitel 5 bietet eine Einführung in Fahrerlose Transportsysteme, die sowohl objekt- als auch agentenorientiert modelliert werden, wobei insbesondere auf den Aspekt der Auftragsvergabe eingegangen wird. In Kapitel 6 werden Fahrerlose Transportsysteme benutzt, um ein Beispiel für Agentenbasierte Simulation darzustellen. Kapitel 7 stellt eine grafische Spezifikationssprache für Agentenbasierte Simulationssysteme in Form eines UML-Profils und in Form eines AORML-Profils vor. In Kapitel 8 wird dann das Simulationssystem erläutert, welches es ermöglicht, in UML spezifizierte agentenbasierte Simulationsmodelle automatisch ausführen zu lassen. Ferner wird eine mit dem Simulationssystem durchgeführte komplexe Simulationsstudie Fahrerloser Transportsysteme vorgestellt. Kapitel 9 reißt verwandte und weiterführende Themen an, die in dieser Arbeit nicht ausführlich behandelt werden konnten. Dazu gehören eine generische Visualisierung für Simulationssysteme, Verteilte Simulation, Echtzeitsimulation und Interaktive Simulation. In Kapitel 10 werden die Ergebnisse dieser Arbeit zusammengefasst.