Technical Report 53

Zustandsbasierte Modellierung zur vereinfachten Implementierung von Simulationsmodellen

Jörg Höhne
TZI, Universität Bremen

TZI-Bericht Nr. 53
2009

TZI-Berichte

Herausgeber:
Technologie-Zentrum Informatik und Informationstechnik
Universität Bremen
Am Fallturm 1
28359 Bremen
Telefon: +49-421-218-7272
Fax: +49-421-218-7820
E-Mail: info@tzi.de
http://www.tzi.de

ISSN 1613-3773

Zustandsbasierte Modellierung zur vereinfachten Implementierung von Simulationsmodellen

Jörg Höhne
Digitale Medien in der Bildung
Universität Bremen
Postfach 330440
28334 Bremen, Deutschland
E-Mail: hoehne@informatik.uni-bremen.de

18. August 2009

Zusammenfassung

Von einem Problem zu einer fertigen Simulation sind diverse, aufeinander aufbauender, Arbeitsschritte notwendig. Der Artikel konzentriert sich auf den Schritt der Modellierung, der direkt vor der Erstellung eines Simulationsmodells liegt. Ein Simulationsmodell ist die Implementierung des Modells in ein ausführbares Modell und markiert gleichzeitig den Punkt, ab dem zusätzliches Wissen über das eigentliche Problem hinaus notwendig ist.

Erfolgt die Simulation auf dem Rechner, dann muss ein Modell in einem durch den Rechner ausführbaren Simulationsmodell vorliegen. Der dafür notwendige Umsetzungsschritt erfordert Programmierkenntnisse, die einen Nutzerkreis potentiell einschränken, wenn der Aufwand zur Umsetzung zu groß wird. Um diesen Schritt zu vereinfachen, soll ein passendes Modell gewählt werden, das eine einfache Implementierung erlaubt, weil es etablierte Methoden der Informatik verwendet.

Als geeignete informatorische Technik wird die Automatentheorie herangezogen, weil sie in einer graphischen Notation auf einfache Weise ein Modell beschreiben kann und gleichzeitig eine Implementierung durch eine Programmiersprache erlaubt. Die Demonstration der Technik wird anhand der sogenannten Mikrosimulation gezeigt. Die Mikrosimulation erlaubt dem Anwender eine algorithmische Beschreibung simulierter Individuen und Objekten innerhalb einer Simulation. Es wird dargelegt, dass die zustandsbasierte Formulierung gegenüber anderen Darstellungsformen Vorteile besitzt und eine graphische Repräsentation eines zustandsbasierten Modells ebenfalls vorteilhaft ist.

Inhaltsverzeichnis

1 Einleitung

Simulationstechnik erlaubt die Untersuchung von Systemen, die sich normalerweise nicht oder nur unter unverhältnismäßig hohem Aufwand untersuchen lassen. Gleichzeitig entfaltet die Simulationstechnik erst in Verbindung mit einer Problemlösung einen wirklichen Nutz- bzw. Mehrwert. Dies bedeutet, dass für eine Problemlösung Fachwissen über das ursprünglich (zu lösende) Problem sowie Simulationswissen, d.h. Fachwissen zur Erstellung einer Simulation, benötigt werden.

Unter dem Aspekt der möglicherweise notwendigen Wissensakquirierung bietet beispielsweise der Schulunterricht mit seinen unterrichteten Fächern in der Sekundarstufe II eine interessante Lehrplattform: die Informatik liefert die theoretischen und praktischen Grundlagen zur Formulierung von Modellen, Simulationsmodellen und deren Ausführung (Simulationswissen), während andere Fachgebiete die zu untersuchenden Probleme und das notwendige Fachwissen (Problemwissen) bereitstellen.

Die Simulationstechnik verlangt nach interdisziplinären Arbeiten und Problemlösungsansätzen, ein Potential, das im Schulunterricht vorhanden ist. Im Unterricht könnten zum Beispiel Modelle in der Biologie und Chemie entwickelt, demonstriert und untersucht werden, die Phänomene wie Selbstorganisation und Emergenz zeigen, vgl. (Höhne 2007). Solche Systeme, die ein Erklärungsmodell für viele Prozesse bieten können, lassen sich ohne Hilfsmittel nicht oder nur wenig verstehen. Eine mathematisch analytische Herangehensweise bietet sich nicht

an, weil diese in der Regel komplexe mathematische Grundlagen voraussetzt, die wiederum eine (unverhältnismäßig) hohe Zugangshürde setzt. Einige Problemstellungen lassen sich mit Hilfe mathematisch analytischer Verfahren auch nur inadäquat abbilden, so dass sich der Einsatz von Simulationstechnik empfiehlt.

Die Simulationstechnik soll durch ihren Einsatz das Verstehen von Systemen ermöglichen. Hierbei steht das Erfahren eines Systems im Vordergrund, d.h. um das Erkennen von Zusammenhängen durch das Ausprobieren eines Systems. Mit Hilfe der Simulationstechnik können Änderungen am System vorgenommen und die Resultate betrachtet werden, so dass die Zusammenhänge erkennbar werden und im Rahmen der Beschäftigung mit Systemen das systemische Denken gefördert wird.

Die besondere Anforderung an die Simulationstechnik beruht im schulischen Anwendungsfall in der zu begrenzenden Komplexität, denn Schüler (und Lehrer) sind in der Regel keine Informatiker oder Experten in der Simulationserstellung. Daraus folgt, dass der simulationstechnische Aspekt in der Simulationserstellung möglichst einfach sein muss, damit er den Erstellungsprozess einer Simulation nicht dominiert. Die Simulationstechnik soll als Hilfsmittel zur Verfügung stehen und nicht als Problembereiter.

Um die Erstellung von Simulationen einfach zu gestalten, bleibt als ein Ansatzpunkt den Simulationsprozess zu vereinfachen, und somit das Simulationswissen zu reduzieren. Das Problemwissen bleibt unberührt, denn dieses muss unabhängig von einer Simulation erarbeitet werden. Dieser Artikel konzentriert sich auf den Modellierungsprozess als einen der ersten Schritte in dem Entwicklungsprozess zu einer Simulation. Es soll eine Formulierung von Modellen gefunden werden, die sich aus informatorischer Sicht gut für eine Umsetzung in ein Simulationsmodell eignen. Mit der Eignung soll erreicht werden, dass der Umsetzungprozess von einem Modell zu einem Simulationsmodell vereinfacht wird, damit dieser Schritt sich für einen Anwender möglichst einfach gestaltet und daher Anteil des Simulationswissens am Gesamtlösungsprozess reduziert werden kann.

Dieser Artikel behandelt daher zunächst den Gesamtprozess der Simulationserstellung und einige theoretische Grundlagen. Es wird dann der Schwerpunkt auf die individuenbasierte Modellierung gelegt und für diesen Modellierungsansatz verschiedene Notationen untersucht, welche die Beschreibung des Verhaltens der Individuen erlauben. Aus diesen Erkenntnissen ergeben sich dann Anforderungen an Simulationsumgebungen.

2 System, Modell und Simulationsmodell

2.1 Arbeitsschritte zur Erstellung einer Simulation

Ausgehend von einem Problem erfolgt die Erstellung einer Simulation über die folgenden abgrenzbaren Arbeitsschritte:

1. Problem: Eine Fragestellung über einen zu untersuchenden Gegenstand.

2. System: Der abgegrenzte Problembereich, der die interagierenden Systemkomponenten enthält.

3. Modell: Eine Abbildung des Systems, das die wesentlichen Systemkomponenten und deren Interaktion sowie die (algorithmische) Beschreibung beider enthält.

4. Simulationsmodell: Die Umsetzung (Implementierung) eines Modells in ein ausführbares Simulationsmodell. Für die Umsetzung werden die Möglichkeiten der Simulationsumgebung genutzt.

5. Simulation: Ausführung des Simulationsmodells als Simulation.

Die genannten Schritte bauen aufeinander auf, wobei die letzten beiden Schritte im Wesentlichen von der verwendeten Simulationsumgebung abhängen.

2.2 Der Systembegriff

Mit dem Begriff *Simulation* ist der Begriff *System* eng verbunden. Eine einheitliche Definition über die verschiedenen Fachgebiete ist nicht vorhanden, so dass ein System an dieser Stelle wie folgt definiert wird:

> *Ein System ist eine Menge miteinander in Beziehung stehender Systemkomponenten, die interagieren und ein Systemverhalten aufrecht erhalten.*

Diese Definition nimmt an, dass ein System aus einzelnen Komponenten besteht, die wiederum interagieren und gleichzeitig durch die Interaktion eine *Funktion* erfüllen. Die Annahme einer Funktion ist sinnvoll, weil sich Systeme einem Betrachter möglicherweise erst dann erschließen, wenn man eine Funktion annimmt. In (Page 1991) wird ein System wie folgt definiert:

> „Unter dem Begriff System versteht man eine Menge miteinander in Beziehung stehender Komponenten, die zu einem gemeinsamen Zweck interagieren.“ (Page 1991)

Die Annahme eines Zwecks ist eine stärkere Akzentuierung des Begriffs Funktion. Diese Sichtweise ist bei künstlichen Systemen sinnvoll, denn sie wurden zu einem Zweck erschaffen. Bei Systemen aus der Natur ist diese Annahme jedoch falsch, weil sie einen teleonomischen Charakter impliziert, der nicht gegeben ist. Bei Systemen aus der Natur entspricht der Zweck die beobachteten Homöostase, die auf einer probabilistischen Grundlage (Dawkins 2001; Eigen u. Schuster 1979) beruht. Ein System besteht bei dieser Sichtweise aus evolvierten Regel- und Steuerkreisläufen, die sich idealerweise gegenseitig zum Vorteil aller Systemkomponenten positiv beeinflussen und stabilisieren und somit eine Resilienz[1] aufweisen. Die Annahme eines Zwecks darf in diesem Fall nur als eine reine Verständnishilfe fungieren; die Annahme einer Funktion unterstellt eine weniger teleonomische Perspektive und wäre somit die geeignetere Betrachtungsweise.

2.3 Modell und Modellierung

Beiden Definitionen ist gemeinsam, dass ein System aus interagierenden Komponenten besteht. Für eine Simulation muss das System zunächst in einem Modell abgebildet werden, wobei die einzelnen Systemkomponenten eine geeignete Modellierung erfahren. Im Rahmen der Modellierung kommt es zu Abstraktionen und Vereinfachungen, so dass das entwickelte Modell lediglich ein Abbild des Systems ist und naturgemäß nicht alle Eigenschaften repräsentieren kann. Ein Simulationsmodell ist ein implementiertes Modell, d.h. ein ausführbares

[1] Die Fähigkeit Abweichungen auszugleichen und somit zu tolerieren.

Modell; beide Begriffe werden synonym gebraucht, denn ein Simulationsmodell setzt bei valider Implementierung lediglich ein Modell in eine äquivalente (algorithmische) Form um. Ein Modell und das Simulationsmodell bedingen sich gegenseitig, denn die Implementierungsmöglichkeiten beeinflussen den eigentlichen Modellierungsprozess, weil sich ansonsten ein Modell nicht in ein Simulationsmodell überführen lässt.

Die geeignete Modellierung ist primär von dem abgebildeten System und dem Zweck der Simulation abhängig und unterliegt unter Umständen subjektiven Kriterien. Dennoch kann generell zwischen zwei Gruppen von Modellen, den kontinuierlichen und diskreten, differenziert werden.

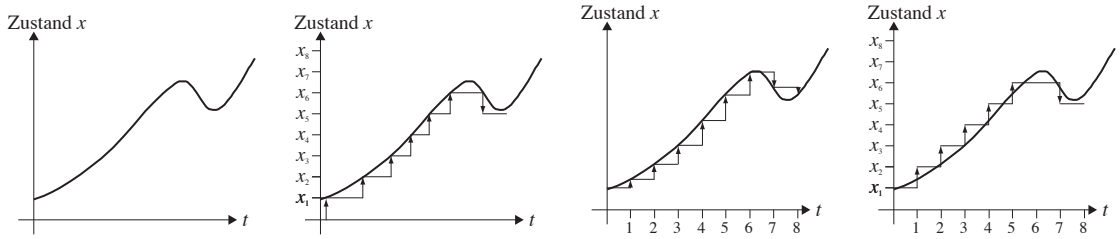
2.3.1 Kontinuierliche Modelle

In Modellen wird die Entwicklung über die Zeit betrachtet, wobei die Messgrößen als Werte abgebildet werden. In kontinuierlichen Modellen ist die Differenz zwischen zwei aufeinanderfolgenden Zeitpunkten t_i und t_{i+1} und zwei aufeinanderfolgenden Werten x_i und x_{i+1} mit $x \in \mathbb{R}$ unendlich klein. Ein kontinuierliches Modell wird daher als zeit- und wertkontinuierlich bzeichnet. Kontinuierliche Modelle sind mathematisch über stetige Funktionen beschrieben, daher lassen sich die gewünschten Größen direkt aus dem formulierten Gleichungssystem exakt berechnen.

2.3.2 Diskrete Modelle

Ein diskretes Modell wird gewählt, wenn beispielsweise ein mathematisches Modell einen Sachverhalt nicht ausreichend exakt beschreibt. Gleiches gilt für den Fall, dass die Lösung des Gleichungssystems eines kontinuierlichen Modells nur über eine numerische Integration möglich ist, so dass daraus ein diskretes Modell resultiert. Bei einem diskreten Modell werden die Zeit und/oder die Werte in diskreten Wertfolgen abgebildet:

- Zeitdiskretes Modell: Die Zeitachse ist in diskrete Abstände $t_0, t_1, \ldots, t_{n-1}, t_n$ unterteilt. Ausschließlich zu einem jeden Zeitpunkt t_i mit $0 \leq i \leq n$ zu t_{i+1} können sich die Zustände und Werte in dem Modell ändern.
- Wertdiskretes Modell: Die Wertänderungen erfolgen nur in konstanten Minimaldifferenzen bzw. deren Vielfache. Für zwei direkt aufeinanderfolgende Werte v_i und v_{i+1} ist die Differenz $\Delta v = |v_i - v_{i+1}| > 0$.



(a) Zeit- und wertkontinuierlich.
(b) Zeitkontinuierlich und wertdiskret.
(c) Zeitdiskret und wertkontinuierlich.
(d) Zeit- und wertdiskret.

Abbildung 1: Klassifikation Ereignisdiskreter Systeme nach (Kiencke 1997).

Die kontinuierliche bzw. diskrete Darstellung auf der Zeit- und Wertachse führt zu den Kombinationen in Abbildung 1 der Modelleigenschaften[2]. In der Abbildung ist die zeitdiskreten Modelleigenschaft durch „Treppenstufen“ gleicher Länge und die wertdiskrete Modelleigenschaft durch „Treppenstufen“ gleicher Höhe dargestellt.

Simulationsmodelle sind in der Regel zeit- und wertdiskrete Modelle, denn die Darstellung von Werten auf Rechnersystemen erfolgt üblicherweise in diskreter Darstellung, so dass die Zeit- und Wertachse zwangsweise diskret sind. Eine alternative Abbildung von Werten, z.B. in Form von Termen, ist möglich. Die Terme werden erst bei der Wertermittlung aufgelöst und in eine wertdiskrete, numerische Darstellung überführt. Diese Form der exakten Berechnung ist ressourcenintensiv, so dass auf diese Art der Repräsentation üblicherweise verzichtet wird.

Aufgrund der fehlenden kontinuierlichen Zeitachse bei zeitdiskreten Modellen spricht man von Ereignissen, an denen eine Berechnung des Systemzustandes, d.h. die Bestimmung aller Systemvariablen, stattfindet. Diese Systeme werden daher als *Ereignisdiskrete Systeme* (EDS) bezeichnet. Die Ereignisse werden dem System extern zugeführt oder entstehen aufgrund interner Berechnungsvorgänge. Mit der Zuführung externer Ereignisse ist in dem Modell die Basis für Anwenderinteraktion vorgesehen.

2.4 Individual based modeling

Dieser Artikel konzentriert sich auf diskrete Simulationsmodelle mit einem individual based modeling-orientierten (IBM) Ansatz. Bei dem IBM-Ansatz besteht eine Simulation aus einer Menge von Objekten (Individuen), die ein bestimmtes (situationsabhängiges) Verhalten zeigen und durch Interaktion das abzubildende System darstellen. Die Interaktion der Objekte bestimmt die Systementwicklung (Prozess) und auf höherer Betrachtungsebene können Regelkreisläufe beobachtet werden. Andere Simulationsansätze hingegen setzen die Regelkreisläufe voraus um die Systementwicklung abzubilden.

Jede Komponente des Systems wird in dem Modell durch ein eigenes Objekt mit einem eigenen Satz an Regeln dargestellt. Diese regelbasierte Darstellung wird in (Koschwitz u. Wedekind 2004) als *Mikrosimulation* bezeichnet, weil die Simulation durch die (vielen) einzelnen Objekte geprägt ist und keine hochaggregierten[3] Variablen verwendet werden. Hochaggregierte Variablen versuchen das statistische Verhalten von vielen Objekten zu beschreiben um das durchschnittliche Verhalten abzubilden. Diese regelbasierte Formulierung der Eigenschaften ist laut (Koschwitz u. Wedekind 2004; Klopfer 2003; Klopfer u. a. 2002; Colella u. a. 2001) positiv zu beurteilen, weil:

- Das Regelwerk realitätsnah und alltagssprachlich formuliert ist,
- die Einzelereignisse in fachlich interpretierbaren Begriffen formuliert sind und
- weil durch sukzessive Erweiterung des Modells in Form von einsichtigen Erweiterungsschritten auch komplexere und realitätsnähere Modelle ohne die Vorgabe einer mathematischen Einstiegsschwelle[4] erzeugt werden können.

[2] Der Begriff System und Modell wird in der Simulationsliteratur gelegentlich synonym verwendet, denn ein Modell bildet ein System ab, daher beschreibt ein diskretes Modell letztendlich ein diskretes System.

[3] Mit diesen Variablen sollen Eigenschaften eines Systems beschrieben werden, die sich statistisch aus der Summe von Objekten ergeben. So ist beispielsweise die Sterberate in einem Simulationsmodell die statistische Wahrscheinlichkeit auf eine Population bezogen und nicht auf ein konkret simuliertes Individuum.

[4] Werden Regelkreisläufe vorausgesetzt, dann müssen die (hochaggregrierten) Systemvariablen zunächst formuliert und berechnet werden; dieses kann ein sehr komplexes Verfahren bedingen.

Mit der Mikrosimulation lassen sich Modelle realisieren, die zu neuen Erkenntnissen führen, weil sich beispielsweise räumliche Eigenschaften innerhalb der Simulation einfacher abbilden lassen, als es durch Differentialgleichungen möglich wäre. Beispiele hierfür sind der Hyperzyklus nach (Eigen u. Schuster 1979), dessen originale Formulierung als Differentialgleichungssystem ein grundlegendes Problem (Smith 1979) aufwarf, wodurch das gesamte Modell in Frage gestellt wurde. Erst eine einfache Simulation unter Berücksichtigung der spatialen Umgebungsbedingungen durch (Boerlijst u. Hogeweg 1991) löste den aufgeworfenen Widerspruch auf.

2.5 Anforderungen an die Modellierung

Der Ansatz der Mikrosimulation und die damit verbundene Modellierung mittels Individuen wird als ein geeigneter Lösungsansatz für die Erstellung von Simulationen betrachtet. Für die Anwendung dieser Modellierungsart muss eine geeignete Darstellungsform gefunden werden, die es einem Anwender erlaubt, ein Modell in ein Simulationsmodell umzusetzen. Erforderlich ist eine Darstellungsform, die dem Anwender eine einfache Formulierung des Individuenverhaltens erlaubt und gleichzeitig den Implementierungsschritt vom Modell zum Simulationsmodell einfach gestaltet. Diese beiden Forderungen müssen im Fokus bleiben, denn sie dienen der Konzentration auf die Lösung der ursprünglichen Problemstellung und sollen verhindern, dass der Aufwand für die Simulationserstellung dominant wird.

3 Zustandsbasierte Modellierung

3.1 Die Beschreibung von Modellen

In (Koschwitz u. Wedekind 2004) wird die regelbasierte Beschreibung des Verhaltens als ein großer Vorteil der Mikrosimulation gegenüber der Makrosimulation aufgeführt. Eine regelbasierte Beschreibung ähnelt bereits einer algorithmischen Beschreibung, wie sie in gängigen (imperativen) Programmiersprachen verwendet werden. Die nachfolgende Auflistung zeigt eine nach (Schmickl u. Crailsheim 2004) regelbasierte Darstellung des Verhaltens einer Honigbiene bei der Nahrungssuche, vergleiche hierzu auch (Camazine u. a. 2001; Seeley 1995):

1. The behaviour of the honey bee starts at item 2.
2. The honey bee is in the hive without any info of a food (nectar) source.
3. If the honey bee is in the hive without any info of a food (nectar) source then with a probability of p(follow) it will follow a dancing bee that describes by dancing the direction and distance to a food source, refer to 6.
4. If the honey bee is in the hive without any info of a food (nectar) source then with a probability of p(scout) the bee will become a bee scouting for food, refer to 19.
5. Go to 2.
6. If the honey bee is following a dancing bee then with a probability of p(hear) the bee learns about the direction and distance of a food source and becomes a bee with information about a food source, refer to 8.
7. If the honey bee is following a dancing bee then if the bee didn't learn about the food source in 6 (probability is $1 - p$(hear)) it remains as a bee without any information of a food source, refer to 2.

8. If the bee is a bee with information about a food source and the bee is (still) in the hive then with a probability of p(forget) the bee forgets about the food source and it becomes a bee without any information about a food source, refer to 2.
9. If the bee is a bee with information about a food source and the bee is (still) in the hive then with a probability of p(dance) the bee will start dancing for providing information about the food source to other bees, refer to 12.
10. If the bee is a bee with information about a food source and the bee is (still) in the hive then with a probability of p(forage) the bee starts foraging, refer to 13.
11. Go to 8.
12. If the bee a is a dancing bee and it has danced long enough it becomes a bee with information about a food source in the hive, refer to 8.
13. If the bee is foraging and has no success finding food it becomes a bee with information about a food source outside the hive, refer to 15.
14. If the bee is foraging and has success finding food it becomes a outside the hive bee with collected food and information about the food source, refer to 16.
15. If the bee is a bee with information about a food source and outside the hive it returns after a short delay to the hive. Now it is a bee with information about a food source inside the hive, refer to 8.
16. If the bee is a outside the hive bee with collected food and information about the food source it returns to the hive and queue in the unload queue, refer to 17.
17. If the bee is in the unload queue it abandons the information about the food source with the probability of p(abandon) and becomes a bee in the hive without any information about the food source, refer to 2.
18. If the bee is in the unload queue and didn't abandon the information about the food source (probability is $1 - p$(abandon)) it becomes a bee in the hive with information about the food source, refer to 8.
19. If the bee is scouting and has no success it returns to the hive without any information about a food source, refer to 21.
20. If the bee is scouting and has success it becomes a bee outside the hive with information about a food source and collected food, refer to 16.
21. If the bee is returning to the hive and has no information about a food source it arrives after a delay at the hive and becomes a bee in the hive without any information about a food source, refer to 2.

Bei der regelbasierten Darstellung beginnt die Modellierung des Individuenverhaltens mit dem ersten Eintrag und der sukzessiven Abarbeitung der Regeln. Trifft bei einer Regel die (optionale) Bedingung am Anfang zu, dann kommt es zur Ausführung der aufgeführten Anweisungen. Die regelbasierte Darstellung lässt sich in einen Programmtext einer regulären Programmiersprache überführen, weil sich der Regelaufbau an einem Bedingungskonstrukt wie IF *Bedingung erfüllt* THEN *Ausführung von Befehlen* orientiert. Die regelbasierte Darstellung scheint aus genannten Gründen eine geeignete Beschreibung von Objektverhalten zu sein.

3.2 Zustandsbasierte Modellierung

Die im vorherigen Text präsentierte regelbasierte Darstellung lässt noch Struktur und Übersichtlichkeit vermissen, denn beispielsweise müssen im schlechtesten Fall alle Regeln untersucht werden um zu entscheiden, welche Regel zur Anwendung kommt. Dieser Aufwand kann sich bei der Formulierung von komplexen Modellen negativ auswirken, denn ein Anwender muss im schlechtesten Fall alle Regeln untersuchen um eine für die Situation passende zu finden.

Eine Betrachtung der Regeln zeigt, dass einige die gleiche Bedingung besitzen und insgesamt die Bedingungen einen Zustand definieren, der gerade für das modellierte Individuum gilt. Registrierte man den aktuellen Zustand eines Individuums, dann bräuchten nur noch die Regeln für den aktuellen Zustand untersucht werden, so dass sich der Aufwand verringert. Dieses machte sich der Autor in (Schmickl 2009) zunutze und formulierte ursprünglich das Verhaltensmodell wie folgt:

> Each single actor (bee) can be in one of the following states, following a special set of behavioral rules then:
>
> **in-hive** (yellow bee): Moves around in the hive randomly. With a given probability the bee can switch to scouting and leave the hive. If the bee return successfully from a source, she can dance, if the (inverse) dancing threshold isn't already lowered to 0. With a given probability the bee can just forget about her source. With a given probability she can just fly out to her last source to forage again. If she meets a dancing bee, she switches to this source and leaves the hive.
>
> **dancing** (color-of-the-source): Dancing starts if a random variable falls below the current dancing threshold. This threshold is than lowered by 20%. The nearer the location of the nectar source was and the higher the sugar concentration was, the higher is the initial dancing threshold. Also the duration of the dances is longer then. During the dances, the bee switches to the color of the source, so we can follow the sharing of information (amounts of informational pieces) visually on screen. The source information can be gathered by followers (state: in-hive), but we can adjust the noisiness of this process. After the dancing time is elapsed, the bee falls back to state "in-hive" (yellow). But she might dance again ...
>
> **scouting** (green): Bees leave the hive and fly around randomly. If they find a source, they take the color of the source and fly home directly (state: returning). With a given probability, they just stop scouting and fly back without anything (white bee).
>
> **foraging** (color-of-the-source): A bee that had already successfully foraged or scouted or got information via a dance flies directly to the direction for the distance she knows. This information might be a little wrong ((noisiness !). If she finds something there, she switches to returning. If not, she switches to searching for a given time.
>
> **searching** (color-of-the-source): The bee came with information about the location of the source but missed it. She flies circles and searches for some time. If she doesn't find the source, she returns without success (white bee)
>
> **returning** (color-of-the-source or white). If the bee returns from a successful foraging trip or from successful scouting, it has the source color. If she simply gave up searching due to quite empty honey reserves, she is white. In both cases, she flies back to the hive entrance directly.

In der zustandsbasierten Formulierung sind die Regeln entsprechend der Zustände, die die Biene einnimmt, gruppiert. Zusätzlich spezifiziert der Autor in (Schmickl 2009) das Aussehen

der Bienen in Abhängigkeit vom Zustand. Der nachfolgende Abschnitt zeigt eine modifizierte Variante mit weniger Implementierungsdetails und einer feineren Strukturierung hinsichtlich der Zustände:

initialState The behaviour of the honey bee starts at state INHIVEWITHOUTINFO.

inHiveWithoutInfo The honey bee is in the hive without any info of a food (nectar) source and moves around randomly.

1. With a probability of p(follow) the bee obtains state DANCEFOLLOWING.
2. With a probability of p(scout) the bee obtains state SCOUTING.

danceFollowing The bee follows a dancing bee to learn about the direction and distance to a food source.

1. With a probability of p(hear) the bee learns and obtains state INHIVEWITHINFO.
2. The bee didn't learn (probability is $1 - p$(hear)) and obtains state INHIVEWITHOUTINFO.

inHiveWithInfo The bee has information (direction and distance) about a food source.

1. With a probability of p(forget) the bee forgets and obtains INHIVEWITHOUTINFO.
2. With a probability of p(dance) the bee obtains state DANCING.
3. With a probability of p(forage) the bee obtains state FORAGING.

dancing The bee dances to provide information (direction and distance) about a food source to other bees.

1. If the bee has danced long enough the bee obtains state INHIVEWITHINFO.

foraging The bee is foraging and tries to find a food source.

1. If the bee has no success finding food it obtains state SEARCHING.
2. If the bee has success finding food it obtains state RETURNWITHINFO&LOAD.

searching The bee flies in circles to find the food source the location it has.

1. If the food source is found it obtains state RETURNWITHINFO&LOAD.
2. If the bee has no success finding food and some time elapsed it obtains state INHIVEWITHINFO.

returnWithInfo The bee has information about a food source.

1. After a delay the bee obtains state INHIVEWITHINFO.

returnWithInfo&Load The bee is outside the hive, has collected food and possesses information about the food source (direction and distance).

1. After a delay it returns to the hive and obtains state UNLOADQUEUE.

unloadQueue The bee is in the unload queue and will unload the food.

1. With a probability of p(abandon) the bee will abandon the information about the food source and obtains state INHIVEWITHOUTINFO.
2. If the bee didn't abandon the information about the food source (probability is $1 - p$(abandon)) it obtains state INHIVEWITHINFO.

scouting The bee is scouting (for food).

1. If it has no success it obtains state RETURNWITHOUTINFO.
2. If it has success (found food source) it obtains state RETURNWITHINFO&LOAD

returnWithoutInfo The bee is returning to the hive and has no information about a food source.

1. It will arrive after a delay at the hive and obtains state INHIVEWITHOUTINFO.

Diese Form der Beschreibung definiert das (modellierte) Verhalten einer Honigbiene und bietet eine ausführliche Struktur, die mit Hilfe einer Programmiersprache umgesetzt werden kann. Das Regelwerk für die Verhaltensmodellierung der Biene ist zwar strukturiert, doch bedarf es einer gewissen Einarbeitung, bis es für einen Anwender verständlich ist. Der folgende Abschnitt soll daher die Möglichkeit einer besseren Darstellungsform untersuchen, damit Modelle für den Anwender einfacher zu verstehen und konstruieren sind.

3.3 Deterministische Endliche Automaten

Das in (Schmickl 2009) formulierte Modell basiert auf Zuständen um das Verhalten der simulierten Honigbienen abzubilden. Die Automatentheorie (Hopcroft u. a. 2002) bildet die theoretische Grundlage für die Programmierung mit Zuständen, so dass diese kurz Erwähnung finden soll.

Das Modell für ein Individuum beruht auf einem *Deterministischen Endlichen Automaten* (DEA), dessen Definition als ein 5-Tupel $(Q, \Sigma, \delta, q_0, F)$ sich an (Hopcroft u. a. 2002) anlehnt:

- Die endliche nicht leere Menge Q mit Zuständen.
- Das Eingabealphabet Σ als eine endliche nicht leere Menge von Symbolen.
- Der Anfangszustand $q_0 \in Q$.
- Übergangsfunktion $\delta(q, a) \rightarrow Q$ mit $a \in \Sigma$
- Die Menge finaler oder akzeptierender Zustände F mit $F \subseteq Q$.

Neben DEA sind noch *Nichtdeterministische Endliche Automaten* (NEA) beschrieben, die sich von einem DEA lediglich in der Übergangsfunktion δ unterscheiden. Bei einem DEA berechnet die Übergangsfunktion δ genau *einen* Folgezustand, während δ bei einem NEA eine *Menge* von Folgezuständen berechnet. Bei der Ausführung kann ein NEA infolgedessen gleichzeitig mehrere Zustände, die Autoren in (Hopcroft u. a. 2002) sprechen von „Vermutungen“, annehmen. Eine Konvertierung von DEAs und NEAs ist in beide Richtungen möglich. Die Konvertierung eines NEAs mit k Zuständen kann im schlechtesten Fall zu einem DEA mit 2^k Zustände führen, während ein NEA im schlechtesten Fall gleich viele Zustände wie der zugrunde legende DEA besitzt (Hopcroft u. a. 2002, S. 70ff.).

3.4 Die Notation von Deterministischen Endlichen Automaten

3.4.1 Der Automat

Die Definition eines DEA als ein 5-Tupel $(Q, \Sigma, \delta, q_0, F)$ ist bezüglich der Verständlichkeit optimierbar, wie es anhand eines Beispiels aus (Hopcroft u. a. 2002, S. 55ff.) dargelegt werden soll. Das Beispiel aus Abbildung 2 zeigt einen DEA, der alle aus Nullen und Einsen bestehende Zeichenfolgen akzeptiert, die die Folge „01“ enthalten. Die Akzeptanz einer Zeichenkette, d.h. wenn in dieser die Folge „01“ ergibt sich daraus, dass der Automat in den Zustand q_1 gelangt. Für die Analyse der Arbeitsweise des definierten Automaten erfordert die Notation

Automat A

- Die Menge $Q = \{q_0, q_1, q_2\}$.
- Das Eingabgealphabet $\Sigma = \{0, 1\}$.
- Der Anfangszustand $q_0 \in Q$.
- Die Übergangsfunktion $\delta(q_0, 0) \to q_2$, $\delta(q_0, 1) \to q_0$, $\delta(q_1, 0) \to q_1$, $\delta(q_1, 1) \to q_1$, $\delta(q_2, 0) \to q_2$, $\delta(q_2, 1) \to q_1$.
- Die Menge $F = \{q_1\}$.

Abbildung 2: Definition eines Deterministischen Endlichen Automaten (DEA) A zur Akzeptanz aller Zeichenketten, die die Folge „01“ enthalten.

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\}) \text{ mit } \delta \text{ als } \begin{array}{ll} \delta(q_0, 0) \to q_2 & \delta(q_1, 1) \to q_1 \\ \delta(q_0, 1) \to q_0 & \delta(q_2, 0) \to q_2 \\ \delta(q_1, 0) \to q_1 & \delta(q_2, 1) \to q_1 \end{array}$$

Abbildung 3: Tupeldarstellung eines DEA A zur Akzeptanz aller Zeichenketten, die die Folge „01“ enthalten.

einen gewissen Aufwand zum Nachvollziehen der Regeln. Selbst bei diesen kleinen Automaten ist der Zusammenhang der Zustände schwer ersichtlich und muss vom Anwender erarbeitet werden.

3.4.2 Die Darstellung als 5-Tupel

Der gleiche Automat ist in Abbildung 3 in der korrespondierenden Tupelnotation dargestellt. Die Tupelnotation ist kompakter als die vorherige Notation und der Abstraktionsgrad ist mit der vorherigen Notation vergleichbar.

3.4.3 Die Darstellung als Übergangstabelle

Eine Übergangstabelle als alternative Notation stellt die Übergangsfunktion δ dar. Die Tabelle ist so strukturiert, dass die Zeilen die Zustände q_i und die Spalten die Eingabe a von $\delta(q, a)$ enthalten. In der für den Beispielautomaten resultierenden Tabelle 1 ist der Startzustand mit dem Pfeil „$\to$“ und der finale Zustand mit „*“ markiert.

	0	1
$\to q_0$	q_2	q_0
$^\star q_1$	q_1	q_1
q_2	q_2	q_1

Tabelle 1: Die Übergangstabelle eines DEA A zur Akzeptanz aller Zeichenketten, die die Folge „01“ enthalten.

3.4.4 Die Darstellung als Übergangsdiagramm

Die Abbildung 4 zeigt den Beispielsautomaten als Übergangsdiagramm in einer graphischen Notation. Diese Notation ist äquivalent zu der vorherigen tabellarischen bzw. formalen Notation als 5-Tupel. Im Gegensatz zu den anderen Notationen ist dieser Darstellung übersichtlicher, denn die Abhängigkeiten zwischen den Zuständen sind bereits visualisiert: Die Anzahl der Zustände ist sichtbar und die Kanten vermitteln einen Eindruck über den Zusammenhang zwischen den Zuständen. Die Autoren in (Hopcroft u. a. 2002) schlagen für die graphische Notation eines Automaten folgende folgende Regeln vor:

- Jeder Zustand $q \in Q$ wird in Form eines Knotens dargestellt.
- Für jeden $q \in Q$ und jedes Eingabesymbol $a \in \Sigma$ sei $\delta(q, a) = p$. Das Diagramm enthält einen Pfeil von Knoten q zu Knoten p. Der Pfeil ist mit der Bedingung c von a beschriftet.
- Ein mit *Start* beschrifteter Pfeil auf den Knoten q_0; dieser Pfeil beginnt nicht an einem Knoten.
- Die Menge der finalen oder akzeptierenden Knoten F werden mit doppelter Kreislinie dargestellt.

Abweichend von den vorgeschlagenen Regeln wird der Pfeil, der zu dem Startknoten führt, nicht mit *Start* beschriftet.

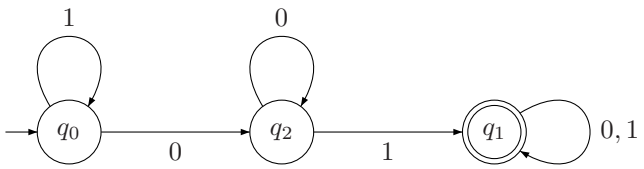


Abbildung 4: Übergangsdiagramm eines DEA A zur Akzeptanz aller Zeichenketten, die die Folge „01“ enthalten.

3.5 Die Bewertung der Notationen

Die aufgeführten Notationen lassen eine erste Bewertung über die Ergonomie zu. Die formale Definition eines Automaten in Abbildung 2 strukturiert anhand der Zustände. Der Startzustand q_0 und akzeptierende Zustand q_1 sind separat aufgeführt. Die Übergangsfunktionen sind ebenfalls gelistet und zeigen für jeden Zustand q_i das jeweilige akzeptierende Symbol (Bedingung), die zu einem Folgezustand führen.

Diese Darstellung ist kompakt, aber liefert einem Anwender jedoch zunächst nur die Zustände. Weitere Informationen, z.B. Zusammenhang der Zustände untereinander, muss sich ein Anwender anhand der Übergangsfunktionen erarbeiten.

Die Tupeldarstellung aus Abbildung 3 besitzt den gleichen Informationgehalt wie Abbildung 2 bei gleichzeitiger Verdichtung der Darstellung. Es gelten für den Anwender im wesentlichen die gleichen Eigenschaften, denn es sind auf den ersten Blick nur die Zustände sichtbar und der interne Zusammenhang zwischen den Zuständen muss erarbeitet werden.

Die Übergangstabelle verdichtet die Information weiterhin und hat ähnliche Eigenschaften wie die Tupeldarstellung. Eine Tabellendarstellung wird jedoch „lichter“, wenn mehr Übergangsbedingungen existieren und nicht mehr jeder Zustand von jedem anderen erreicht werden kann.

Allen drei Darstellungsformen ist gemeinsam, dass für einen Zustand q – ausgenommen dem Startzustand – nicht sofort ersichtlich ist, aus welchen Zuständen er erreicht werden kann. Daher lässt sich aus den bisher bewerteten Darstellungsformen das abgebildete Verhaltensmodell für einen Leser nur mit Aufwand erfassen, weil für jeden Zustand q alle anderen Zustände untersucht werden müssen, ob von diesen aus der Zustand q direkt erreicht werden kann. Diese Untersuchung ist notwendig, damit der Zusammenhang der Zustände erfasst werden kann, denn zum Verständnis ist es durchaus hilfreich zu wissen, von welchem Zustand aus ein Zustand erreicht werden kann. Um die Rückverfolgung zu möglichen Vorgängerzuständen von einem einem Zustand q zu erleichtern, wäre beispielsweise die Konstruktion einer zusätzlichen Tabelle mit den Vorgängerzuständen von q und den zugehörigen Bedingungen ein geeigneter Weg.

Die graphische Notation zeigt diese Schwäche nicht, denn wie in Abbildung 4 ersichtlich, können die eingehenden Übergänge zu einem Zustand *ausgehend von diesem Zustand* direkt rückverfolgt werden. In der graphischen Notation sind alle Zustände eindeutig unterscheidbar und zu jedem Zustand zeigen die Transitionen mit Pfeilen mögliche Folgezustände an. Diese Informationen reichen zunächst für eine einfache Erfassung des Modells, eine tiefere Analyse, wie die Bewertung der Bedingungen an den Transitionen, kann in einem zweiten Schritt durch den Anwender erfolgen.

Die graphische Notation stellt demnach alle Informationen zur Verfügung und erlaubt gleichzeitig einen abgestuften Zugang zu den enthaltenen Informationen. Der Vorteil für einen Leser liegt darin, dass die Informationen sofort und einsichtlich verfügbar sind. Ohne die Darstellungsform zu wechseln oder zusätzliche Quellen heranzuziehen können die Informationen *nacheinander* betrachtet werden.

3.6 Die Anwendung der Automatentheorie auf ein Modell

Die graphische Notation scheint besser geeignet zu sein als die anderen vorgestellten Notationen. Für die Verdeutlichung der Notation soll ein Übergangsdiagramm für das Verhalten der Honigbiene aus Unterabschnitt 3.1 erzeugt werden.

Eine Überführung des verbal (textuell) formulierten Modells in die graphische Notation verlangt in der Regel strukturierende Zwischenschritte:

Strukturierung nach Zuständen Ein verbales Modell wird in seine Zustände strukturiert. Dieser Schritt erfordert entweder die Einführung von Zuständen, also eine Umformulierung des Textes, oder eine einfache Umstellung. In diesem Artikel wurde der Schritt von einer algorithmischen Beschreibung in Unterabschnitt 3.1 zu einer zustandsbasierten algorithmischen Beschreibung in Unterabschnitt 3.2 vollzogen.

Tabellarische Darstellung Wenn gewünscht kann ein zusätzlicher Zwischenschritt erfolgen, nämlich eine tabellarische Darstellung zur zusätzlichen Informationsverdichtung und -strukturierung. In dieser Tabelle werden alle Zustände und ihre abgehenden Transitionen zusammengefasst. Die zu jeder Transition gehörende Bedingung wird ebenfalls aufgeführt.

state $q \in Q$	#	$\delta(q, \sigma) \rightarrow$ TARGET	$\sigma \in \Sigma$
START	1	$\delta(q, \sigma) \rightarrow$ INHIVEWITHOUTINFO	ε
DANCEFOLLOWING	1	$\delta(q, \sigma) \rightarrow$ INHIVEWITHINFO	p(hear)
	2	$\delta(q, \sigma) \rightarrow$ INHIVEWITHOUTINFO	$1 - p$(hear)
DANCING	1	$\delta(q, \sigma) \rightarrow$ INHIVEWITHINFO	delay
INHIVEWITHOUTINFO	1	$\delta(q, \sigma) \rightarrow$ DANCEFOLLOWING	p(follow)
	2	$\delta(q, \sigma) \rightarrow$ SCOUTING	p(scouting)
INHIVEWITHINFO	1	$\delta(q, \sigma) \rightarrow$ DANCING	p(dance)
	2	$\delta(q, \sigma) \rightarrow$ INHIVEWITHOUTINFO	p(forget)
	3	$\delta(q, \sigma) \rightarrow$ FORAGING	p(forage)
UNLOADQUEUE	1	$\delta(q, \sigma) \rightarrow$ INHIVEWITHOUTINFO	p(abandon)
	2	$\delta(q, \sigma) \rightarrow$ INHIVEWITHINFO	$1 - p$(abandon)
SCOUTING	1	$\delta(q, \sigma) \rightarrow$ RETURNWITHOUTINFO	no success
	2	$\delta(q, \sigma) \rightarrow$ RETURNWITHINFO&LOAD	success
FORAGING	1	$\delta(q, \sigma) \rightarrow$ RETURNWITHINFO&LOAD	success
	2	$\delta(q, \sigma) \rightarrow$ SEARCHING	no success
SEARCHING	1	$\delta(q, \sigma) \rightarrow$ RETURNWITHINFO&LOAD	success
	2	$\delta(q, \sigma) \rightarrow$ RETURNWITHOUTINFO	no success
RETURNWITHOUTINFO	1	$\delta(q, \sigma) \rightarrow$ INHIVEWITHOUTINFO	delay
RETURNWITHINFO&LOAD	1	$\delta(q, \sigma) \rightarrow$ UNLOADQUEUE	delay
RETURNWITHINFO	1	$\delta(q, \sigma) \rightarrow$ INHIVEWITHINFO	delay

Tabelle 2: Zustände und Transitionen für das Modell aus Unterabschnitt 3.2 ab Seite 12. Der Eintrag delay zeigt das Verstreichen einer angemessenen Zeit an.

Erzeugung von Determinismus Der Determinismus in einem Deterministischen Endlichen Automaten liegt darin, dass jede von einem Zustand q ausgehende Transition mit $\delta(q, \star) \rightarrow q_i$ eindeutig ist. Das Symbol $\star$ steht für eine Zeichenkette beliebiger Länge, die für q und einer Übergangsfunktion mit $\delta(q,)$ nur exakt einmal auftritt. Durch das exakt einmalige Auftreten einer Zeichenkette für einen Zustand q ist sichergestellt, dass maximal nur eine Transition zur Ausführung kommen kann und daher nur ein Folgezustand erreicht wird.

Bei der Modellierung von Objekten in Simulationen werden in der Regel keine Zeichenketten als Übergangsbedingung verwendet, sondern algorithmisch formulierte Bedingungen. Diese Bedingungen lassen sich nur mit erhöhtem Aufwand derart formulieren, dass ausschließlich eine Bedingung greift. Aus diesem Grund ist die Festlegung einer Auswertungsreihenfolge sinnvoller, so dass die erste passende Bedingung einer Transition in einen Folgezustand führt oder bei keiner passenden Bedingung der Zustand erhalten bleibt. Die Tabelle 2 zeigt die fertige Tabelle mit Zuständen und Transitionen. In der Spalte mit dem „#“-Symbol ist die Auswertungsreihenfolge aufgeführt.

Graphische Notation In diesem Schritt werden die Zustände aus Tabelle 2 in Form von

Kreisen und die Transitionen mit gerichteten Kanten (Pfeilen) eingezeichnet. An jede Transition wird die Übergangsbedingung eingetragen, siehe Abbildung 5.

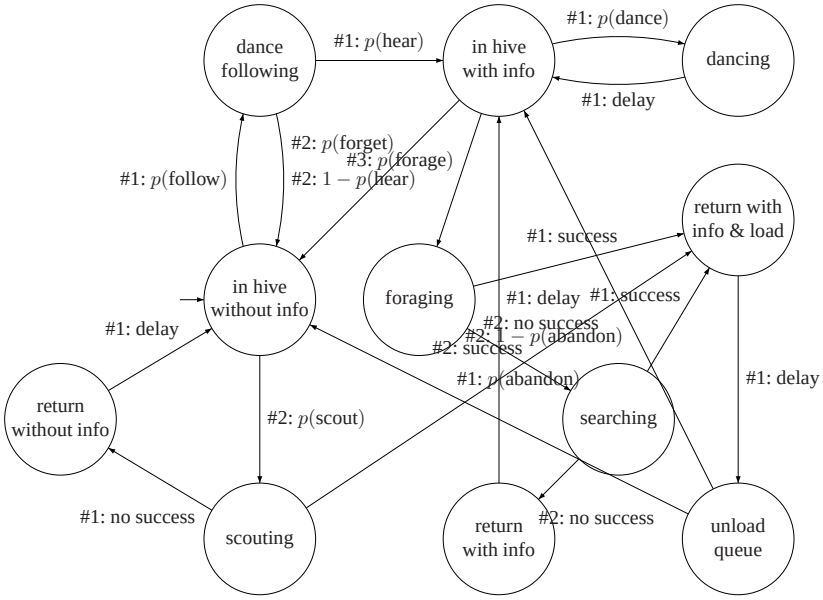


Abbildung 5: Unstrukturiertes Übergangsdiagramm für die Zustände einer Biene bei der Nahrungssuche.

Layout Die Abbildung 5 zeigt das Problem einer graphischen Notation, nämlich die fehlende gute Anordnung der Knoten und Kanten. Ein Anwender muss den Graphen überarbeiten um eine geeignete Anordnung zu treffen, die ein übersichtlicheres Layout, wie in Abbildung 6, realisiert.

Die hier aufgeführten Arbeitsschritte sind insofern notwendig, als dass ein Modell nicht bereits in einer graphischen Notation erstellt wurde.

Die direkte graphische Erstellung könnte sich prinzipiell als eine gute Vorgehensweise erweisen, denn während des Modellierungsprozesses können Fehler besser erkannt werden, weil die Interaktionen zwischen den Zuständen besser sichtbar sind. Als prinzipieller Nachteil bleibt der Aufwand für die Erstellung eines guten Layouts bestehen, weil dessen Konstruktion nicht immer offensichtlich ist.

4 Schlussfolgerung

Die Simulationstechnik kann verwendet werden um das Verstehen von Systemen zu fördern, in dem Systeme *erfahrbar* gemacht werden. Für die Modellierung der Systeme bietet sich ein individual based modeling-basierter Ansatz an, da sich während einer Simulationsausführung im Zusammenspiel der einzelnen Objekte (Individuen) Regelkreisläufe ergeben. Andere Simulationsformen, wie die Makrosimulation, setzen die Regelkreisläufe bereits voraus. In einem solchen Fall besteht keine Möglichkeit die Regelkreisläufe in ihrer Entstehung zu beobachten oder beispielsweise ihre Störanfälligkeit in der Entstehungsphase zu untersuchen.

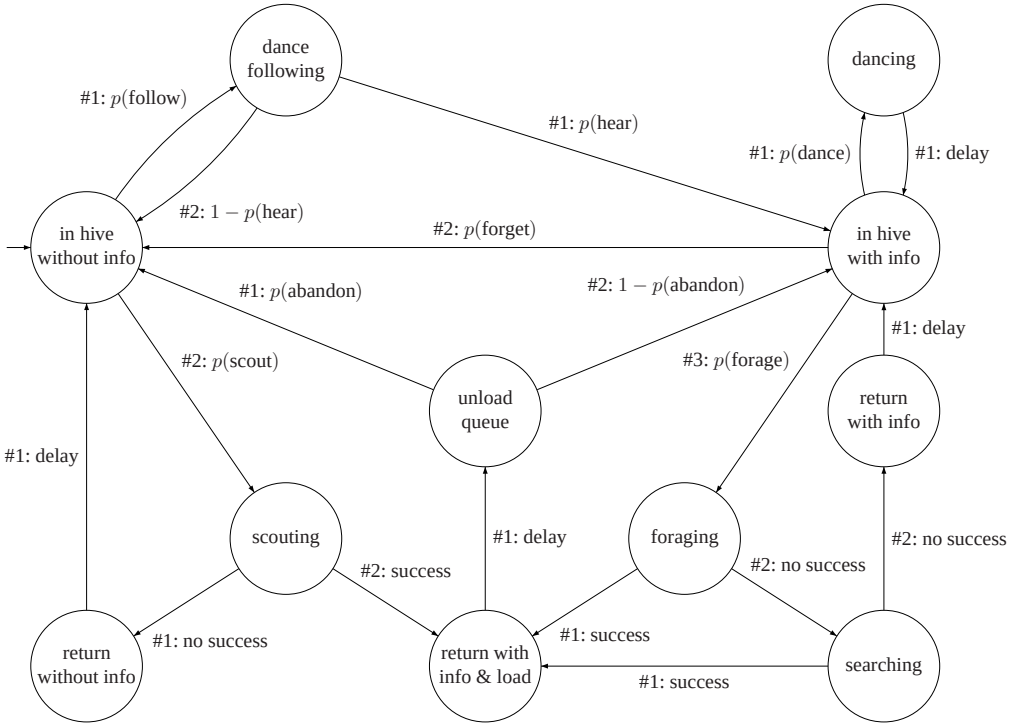


Abbildung 6: Übergangsdiagramm für die Zustände einer Biene bei Nahrungssuche.

Der individual based modeling-Ansatz wird in der so genannten Mikrosimulation verfolgt. In der Mikrosimulation erfolgt die Modellierung des Objektverhaltens in Form von (algorithmischen) Regeln. Der Vorteil der regelbasierten Formulierung ist, dass die Regeln in einer alltagsnahen Notation vorliegen und sich – im Gegensatz zur Makrosimulation – einfacher ändern lassen.

Für die Implementierung eines Modells in ein Simulationsmodell müssen die Regeln in einer Programmiersprache abgebildet werden. Aus diesem Grund sollten Modell und die spätere Implementierung möglichst große Ähnlichkeit aufweisen, um den Implementierungsprozess zu vereinfachen. Der geringe Aufwand soll auch verhindern, dass der Implementierungsaufwand dominiert und das eigentliche Problem, den Grund für die Simulationserstellung, in den Hintergrund drängt.

Das Modell für die regelbasierte algorithmische Beschreibung der Mikrosimulation sollte in einem passend strukturierten Format vorliegen. Die unstrukturierte Niederlegung der Regeln in einem Text, wie in einer verbalen Programmiersprache üblich, erschwert die Lesbarkeit des Modells. Das Verhalten eines Objektes erschließt sich für den Anwender lediglich über einem analytischen Mehraufwand.

Für die Verhaltensmodellierung von Objekten bietet sich eine zustandsbasierte Formulierung an, weil ein Zustandsmodell dem Modell eine zusätzliche, übergeordnete Struktur aufprägt. Ein Zustandsmodell besteht aus einer Menge von Zuständen und Transitionen, die den Übergang von einem Zustand zum nächsten regeln. Ein zustandsbasiertes Modellierung lässt sich in ein durch den Computer ausführbares Simulationsmodell überführen. Die theoretischen Grundlagen sind in Form der Automatentheorie vorhanden und gleichzeitig in der Informatik eine etablierte Technik. Auf der Basis von Automaten werden viele Probleme

in der Informatik gelöst, wie zum Beispiel das Parsing (Syntaxanalyse) von Quelltexten von Programmiersprachen. Zustände lassen sich in den regulären Programmiersprachen abbilden, d.h. die Modelle können prinzipiell in Simulationsmodelle überführt werden.

Der Vorteil in einem Zustandsmodell besteht zunächst in der klaren Differenzierung der Zustände, denn allein bei sinnvoller (semantischer) Benennung lässt sich erkennen, welche Funktion ein Zustand in dem Verhaltensmodell innehat.

Eine graphische Visualisierung mittels eines Zustandsgraphen vermittelt zusätzlich die möglichen Zustandsübergänge, d.h. welches Verhalten kann aufgrund eines bestehenden Verhaltens überhaupt erreicht werden. Die genauen Umstände – das wären die Bedingungen der Transitionen – würden zwar vom Anwender untersucht werden müssen, aber die prinzipielle Information über erreichbare Zustände vermittelt einen ersten Eindruck über das gesamte mögliche Verhalten eines modellierten Objektes.

Andere Darstellungsformen als eine graphische Darstellung werden als weniger geeignet angesehen, weil sie die Informationen sehr abstrakt präsentieren, wie z.B. die tabellarische Form in Tabelle 1, oder gar die Tupelnotation in Abbildung 3.

Die Eingangs geforderte einfache Umsetzung eines Modells in ein Simulationsmodell erfordert neben der vorbereitenden Modellformulierung ebenfalls eine Simulationsumgebung, deren Programmierumgebung die zustandsbasierte Implementierung unterstützt.

In Zusammenhang mit der Simulationstechnik ist weiterhin zu beachten, dass neben dem informatischen Wissen (Simulationswissen) für eine erfolgreiche Simulationserstellung und -durchführung Problemwissen erforderlich ist. Die Kombination aus beiden Wissensformen resultiert daraus, dass die Simulationstechnik stets ein Anwendungsfeld benötigt. Für ein Anwendungsfeld bietet sich beispielsweise die Schule an, weil diese das jeweilige Fachwissen vermitteln kann und somit ein fächerübergreifendes Arbeiten erlaubt. Für die Auswahl einer Simulationsumgebung sollten daher die informatorischen Kriterien ebenso wie eine Anwendung im schulischen Bereich berücksichtigt werden.

Literatur

Boerlijst u. Hogeweg 1991
Boerlijst, M. C. ; Hogeweg, P.: Spiral wave structure in pre-biotic evolution: hypercycles stable against parasites. In: *Phys. D* 48 (1991), Nr. 1, S. 17–28. – ISSN 0167–2789

Camazine u. a. 2001
Camazine, Scott ; Franks, Nigel R. ; Sneyd, James ; Bonabeau, Eric ; Deneubourg, Jean-Louis ; Theraulaz, Guy: *Self-Organization in Biological Systems*. Princeton University Press, 2001. – ISBN 0691012113

Colella u. a. 2001
Colella, Vanessa ; Klopfer, Eric ; Resnick, Mitchel: *Adventures in modeling: Exploring complex, dynamic systems with StarLogo*. New York [u.a.]: Teachers College Press, 2001. – ISBN 0807740829

Dawkins 2001
Dawkins, Richard: *Das egoistische Gen*. 3. Auflage. Rowohlt, 2001

Eigen u. Schuster 1979
Eigen, Manfred ; Schuster, Peter: *The Hypercycle - A Principle of Natural Self-Organization*. Springer, 1979

Hopcroft u. a. 2002
Hopcroft, John E. ; Motwani, Rajeev ; D.Ullman, Jeffrey: *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. 2., überarb. Aufl. München: Pearson Studium, 2002. – ISBN 3827370205

Höhne 2007
Höhne, Jörg: *Ein zustandsinhärentes Objekt- und Programmiermodell zur Simulation natürlicher emergenter Prozesse: Nutzungsmöglichkeiten von Multiagententechnik für die schulische Bildung*, Universität Bremen, Bremen, Deutschland, Diss., 2007. `http://nbn-resolving.de/urn:nbn:de:gbv:46-diss000108546`

Kiencke 1997
Kiencke, Uwe: *Ereignisdiskrete Systeme: Modellierung und Steuerung verteilter Systeme*. Oldenbourg, 1997. – ISBN 3486241508

Klopfer 2003
Klopfer, E.: Technologies to support the creation of complex systems models-using StarLogo software with students,. In: *Biosystems* 71 (2003), Nr. 1-2, S. 111–122. – ISSN 03032647;

Klopfer u. a. 2002
Klopfer, Eric ; Colella, Vanessa ; Resnick, Mitchel: New paths on a StarLogo adventure. In: *Computers and Graphics* 26 (2002), Nr. 4, S. 615–622. – ISSN 00978493

Koschwitz u. Wedekind 2004
Koschwitz, Horst ; Wedekind, Joachim: Künstliches Leben im Biologieunterricht: Mikrosimulationen mit Multi-Agenten-Systemen. In: *LOG IN* 130 (2004), November, S. 28–34

Page 1991
PAGE, Bernd: *Diskrete Simulation. Eine Einführung mit Modula-II.* Springer, 1991

Schmickl 2009
SCHMICKL, Thomas: *Bee foraging.* Version: 2009. `http://zool33.uni-graz.at/schmickl/Self-organization/Collective_decisions/Bee_foraging/bee_foraging.html`, Abruf: 9. Aug. 2009

Schmickl u. Crailsheim 2004
SCHMICKL, Thomas ; CRAILSHEIM, Karl: Costs of Environmental Fluctuations and Benefits of Dynamic Decentralized Foraging Decisions in Honey Bees. In: *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems* 12 (2004), Nr. 3-4, S. 263–277. – ISSN 1059–7123

Seeley 1995
SEELEY, Thomas D.: *The wisdom of the hive: the social physiology of honey bee colonies,.* Cambridge, Mass. [u.a.]: Harvard Univ. Press, 1995. – ISBN 0674953762;

Smith 1979
SMITH, John M.: Hypercycles and the origin of life. In: *Nature* 280 (1979), Nr. 4, S. 445–446