Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Christian Maschmann

Genetische Lernverfahren zur Optimierung einer Multiagentensimulation für marktwirtschaftliche Vorgänge

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Christian Maschmann

Genetische Lernverfahren zur Optimierung einer Multiagentensimulation für marktwirtschaftliche Vorgänge

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Kai von Luck
Zweitgutachter : Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 11. Juli 2007

Christian Maschmann

Thema der Bachelorarbeit

Genetische Lernverfahren zur Optimierung einer Multiagentensimulation für marktwirtschaftliche Vorgänge

Stichworte

Genetische Lernverfahren, Multiagentensystem, Optimierung, Zusammenstellungen, Fitness

Kurzzusammenfassung

In dieser Arbeit wurde ein Multiagentensystem in einem marktwirtschaftlichen Kontext erstellt. Dieses Multiagentensystem mit seinen Akteuren, den Agenten, bildete eine Handelskette aus vier Schichten. Jeder Akteur war entweder Rohstofflieferant, Spotmarkt, Hersteller oder Endkunde. Akteure besitzen außerdem einen Charakter: passiv, moderat oder aggressiv, der das Verhalten steuert. Der ablaufende Handel fand nur zwischen benachbarten Schichten statt. Dieses System diente als ein Ensemble und wurde mit Hilfe von genetischen Lernverfahren versucht zu optimieren. Die Optimierung zielte dabei auf einen maximalen und unter den Akteuren ausgewogenen verteilten Gewinn ab.

Christian Maschmann

Title of the paper

Genetic learning for optimizing a multi-agent-simulation of market economy acts

Keywords

Genetic learning, multi-agent system, optimisation, ensembles, fitness

Abstract

In this thesis a multi-agent system in a free market context was created. This multi-agent system with his participants, the agents, formed a chain of four layers. Each participant was either a resource supplier, spot market, manufacturer or customer. In addition participants possess a character: passively, moderately or aggressively, which rules the behavior. The trade running off took place only between neighbouring layers. This system served as an ensemble and was tried to optimze by genetic learning procedures. The optimisation aimed thereby at a maximum and evenly distributed profit among the participants.

Meiner Frau Viktoria
und der Familie...

Danksagung

In erster Linie danke ich Kai von Luck für die erstklassige Betreuung. Danke dafür, dass ich ihn sogar in seinen eigenen vier Wänden mit Besprechungen heimsuchen durfte und seinen nicht enden wollenden Enthusiasmus für das Themengebiet.

Gunter Klemke, wie auch Kai von Luck, danke ich dafür, dass ich das Kolloquium auch in ihrer vorlesungsfreien Zeit abhalten darf.

Ein großer Dank geht an Waldemar Spät, dafür dass er alle(!) Praktika mit mir ausgehalten hat.

Auch die Lerngruppe bestehend aus Stefan Gehn, Manuel Schneider, Stefan Tamm, Sven Tennstedt und Boris Woscidlo hat ihren Teil zu diesem Abschluss beigetragen.

Nicht vergessen möchte ich meine Familie, die mir immer unterstützend zur Seite stand.

Ein besonderer Dank geht an meine Frau Viktoria für ihr Verständnis, ihre Unterstützung, ihre Geduld, ihre Ermutigung, ihre Fürsorge und dafür, dass sie einfach da ist.

Inhaltsverzeichnis

1. Einleitung

Der moderne Mensch hat sich daran gewöhnt, dass elektronische Helfer seinen Alltag vereinfachen. Der Wecker erinnert ans Aufstehen, der PDA[1] hilft beim Organisieren des Tagesablaufes und komplexere Aufgaben erledigen Computerprogramme.

Dabei versucht die Computerindustrie immer komplexere Probleme durch die Maschinen lösen zu lassen. Dies stellt wiederum höhere Anforderungen an die Programmierer, welche ausgeklügeltere Algorithmen entwickeln müssen.

Es gibt Aufgaben, bei denen verschiedene Komponenten so zusammengesetzt werden sollen, dass aus ihnen eine akzeptable Lösung entsteht. Das Kochen eines Gerichts wäre eine solche Aufgabe. Die Zutaten bilden die verschiedenen Komponenten. Eine akzeptable Lösung wäre dann eine Zusammensetzung von Zutaten, die bestimmte Kriterien erfüllt. Zum Beispiel: schmackhaft, preislich im Budget und gesund.

Der Lösungsraum, in dem sich auch die akzeptablen Lösungen neben dem Optimum befinden, ergibt sich aus den zur Verfügung stehenden Kombinationen der Komponenten miteinander.

Das Durchprobieren aller Möglichkeiten des Lösungsraums, um eine oder mehrere akzeptable Lösungen zu finden, wäre in den meisten Fällen zu teuer. Deshalb wird bei solchen Problemen häufig nur approximiert. In diesen Fällen kann durch heuristische Verfahren versucht werden eine gute Lösung (ein schmackhaftes, preislich im Budget liegendes und gesundes Gericht) im Lösungsraum zu finden. Die Suche im Lösungsraum, spiegelt in diesen Fällen das heuristische Durchprobieren verschiedener komplexer Parameter-Settings wieder. Lernverfahren benutzen heuristische Methoden, um sich im Lösungsraum fortzubewegen.

Es stellt sich nun die Frage, ob eine Optimierung einer Zusammenstellung mit Hilfe eines genetischen Lernverfahrens möglich ist. Die Lösung dieser Frage soll exemplarisch und experimentell durchgeführt werden.

Ein Beispiel für einen erfolgreichen Lernprozess ist Dexter von Anybot. Dexter ist ein Roboter, der sich das Laufen selbst beigebracht hat. Dexter balanciert sich dynamisch aus und kann somit auch kaum umgestoßen werden. Das erste was von Dexter zu sehen war, sind

[1] Personal Digital Assistant

ein paar wackelige Schritte. Inzwischen kann man schon Luftsprünge von ihm bei YouTube[2] bewundern. Dies verdankt er nicht zuletzt seinen Eigenschaften sich dynamisch auszubalancieren und zu lernen. Eines Tages soll Dexter in der Lage sein, durch unebenes Gelände zu laufen.

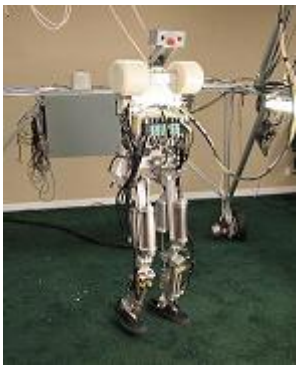


Abbildung 1.1.: Dexter von Anybot

Da jeder Mensch das Laufen lernt, scheint es offensichtlich, dass das Laufen gelernt werden kann. Aber könnten Lernverfahren auch zum Ermitteln einer guten Kombination aus verschiedenen Komponenten dienen? Dies soll in dieser Arbeit an Hand einer kleinen und stark vereinfachten marktwirtschaftlichen Simulation untersucht werden.

Diese Simulation dient als zu optimierende Zusammenstellung und besteht aus vier Schichten von Akteuren: Rohstofflieferanten, Spotmärkte, Hersteller und Endkunden. Diese Akteure bilden eine strikte Handelskette, jede Schicht handelt nur mit den benachbarten Schichten. Die Akteure besitzen außerdem eines von drei möglichen Verhalten: passiv, moderat oder aggressiv. Diese Verhalten bestimmen ihre Handelsstrategie.

Diese Arbeit ist wie folgt aufgebaut: In Kapitel 2 (Grundlagen) werden die verwendeten Techniken, wie genetische Lernverfahren, genetische Algorithmen und (Multi-)Agentensysteme, beschrieben.
In Kapitel 3 (Analyse) wird aufbauend auf den Vorgaben aus der Einleitung analysiert, wie die zu optimierende Zusammenstellung konkret aussehen soll, in welchem Szenario diese Zusammenstellung agieren soll, wie der Ablauf der Handlungen im Szenario aussieht und wie Kommunikation zwischen den Akteuren stattfinden kann.
Durch die Vorgaben aus Kapitel 3 (Analyse) und den Erkenntnissen aus Kapitel 2 (Grundlagen) kann nun in Kapitel 4 (Design und Realisierung) ein konkretes System entworfen und realisiert werden.

[2] Videoportal im Internet

Das in Kapitel 4 (Design und Realisierung) entwickelte System wird anschließend in Kapitel 5 (Evaluation) für Versuchsläufe verwendet. Die Ergebnisse dieser Versuche könnten Aufschluss geben, ob eine Optimierung einer Zusammenstellung mit genetischen Lernverfahren sinnvoll ist. Außerdem zeigt sich, ob die Umsetzung der Vorgaben aus Kapitel 3 (Analyse) in Kapitel 4 (Design und Realisierung) erfolgreich waren.
Abschließend wird in Kapitel 6 (Zusammenfassung und Ausblick) der Verlauf der Arbeit resümiert und ein Ausblick gegeben.

Zeitgleich ist die thematisch verwandte Arbeit von Sven Tennstedt [Tennstedt (2007)] entstanden. Besonders über die Grundlagen fand ein intensiver Austausch von Wissen und Erfahrungen statt. Es können daher in manchen Bereichen Überschneidungen mit seiner Arbeit auftreten.

2. Grundlagen

2.1. Genetische Lernverfahren

Als der Mensch den ersten Flugversuch unternahm, studierte er das Vorbild in der Natur, den Vogel, um eine Vorstellung davon zu entwickeln, wie man diese schwierige Aufgabe in Angriff nehmen könnte. Bemerkenswerte Fehlversuche, wie der von Daedalus, und mißlungene Versuche mit zahlreichen anderen vogelähnlichen Vehikeln (Ornithopter), schienen anfangs in die falsche Richtung zu weisen, aber schließlich führten Ausdauer und die Abstraktion des entsprechenden Wissens (Auftrieb mit Hilfe eines Tragflügels) zu erfolgreichen Segel- und Motorflugzeugen. Ist es in Hinblick auf diese Leistungen nicht seltsam, dass der Mensch beim Versuch, denkende, lernende und sich anpassende Maschinen zu bauen, eines der großartigsten Beispiele für Anpassung in der Natur, die Genetik und natürliche Auslese, ignorierte und weiterhin ignoriert?
David Goldberg
Goldberg (1986)

2.1.1. Die Grundidee

David Goldberg stellte im einleitenden Zitat die Frage, warum Genetik und natürliche Auslese nicht zum Entwickeln von denkenden, lernenden und sich anpassenden Maschinen benutzt wird. Und genau das ist passiert. Genetik und natürliche Auslese sind die Grundelemente der genetischen Lernverfahren.

Das genetische Lernverfahren entspricht einer heuristischen Suche in einem Lösungsraum. Zweidimensional besteht die Fläche des Lösungsraums aus Punkten, die jeweils für eine Lösung stehen. Diese Fläche muss noch in die dritte Dimension gehoben werden. Die Z-Koordinate gibt die Qualität der Lösung an. So ergibt sich eine Fläche mit unterschiedlich ausgeprägten Erhöhungen. Der Startpunkt in diesem Raum ist in der Regel zufällig. Das perfekte Ergebnis des Lernverfahrens wäre die Lösung (Punkt auf den X-Y-Achsen) mit der höchsten Qualität (die größte Erhebung auf der Z-Achse).

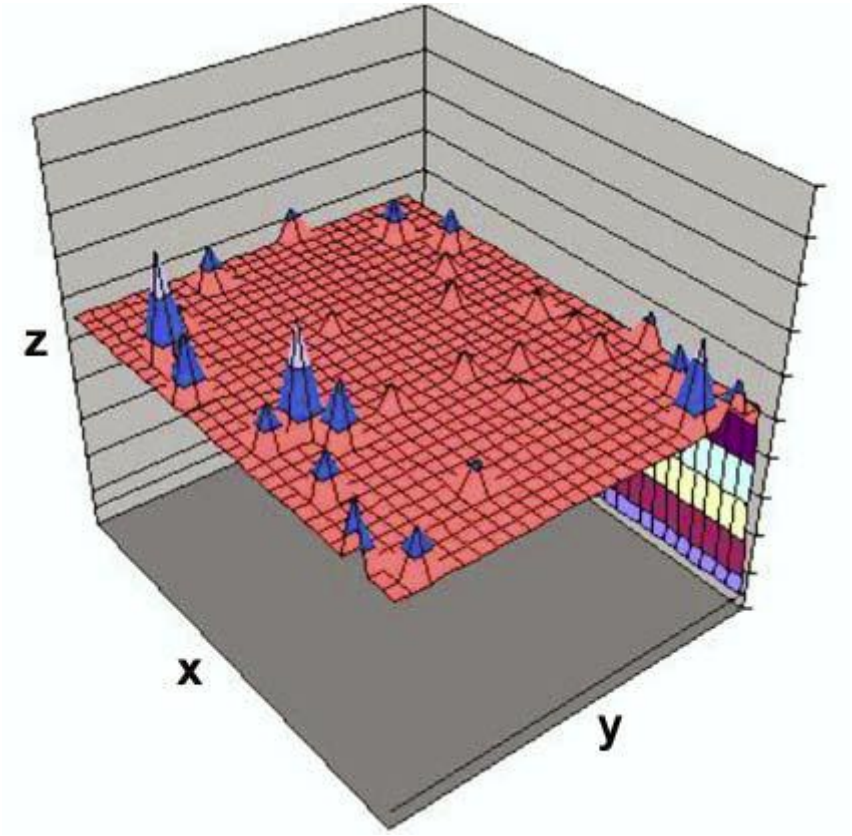


Abbildung 2.1.: Dreidimensionaler Lösungsraum, Quelle: [Porschke (2002)]

Der Ablauf solcher Lernverfahren sieht in der Regel wie folgt aus: An jedem Anfang eines genetischen Verfahrens steht eine Startpopulation, die Generation 0. Diese Generation besteht aus beliebig vielen Individuen. Nachdem von jedem Individuum die Fitness in einem Durchlauf berechnet wurde, kommen verschiedene Prozeduren zum Bilden einer neuen Generation zum Einsatz. Hier wären vor allem verschiedene Selektionsverfahren, wie die Roulette-Selektion oder die Schöpfkellen-Methode zu nennen. Diese werden mit Kreuzungs- und Mutationsmethoden kombiniert. Diese neue Generation 1 durchläuft die gleiche Prozedur. Dies geschieht so lange, bis eine gewisse Anzahl von Generationen durchlaufen wurden oder eine bestimmte Qualität der Lösung erreicht wurde.

Das Vorgehen des eben beschriebenen Lernverfahrens klingt kreativ, ist aber eine Suche in dem oben beschriebenen Lösungsraum.

Allgemein liegt die Schwierigkeit nicht in den Prozeduren zur Optimierung, sondern im Ermitteln der Qualität. Es muss ein meistens komplexer Sachverhalt vergleichbar gemacht werden. Gelingt es nicht den Sachverhalt so aufzubereiten, ist meistens auch das Lernverfahren erfolgsarm. Üblicherweise wird versucht die Qualität auf Zahlen abzubilden.

2.1.2. Die Evolutionstheorie

Die genetischen Lernverfahren basieren auf Genetik und natürliche Auslese, wie die Evolutionstheorie, die als weiteres Vorbild herangezogen wurde. Deshalb sollen in diesem Abschnitt die Komponenten der modernen Evolutionstheorie nach Charles Darwin[1] kurz in ihrer Bedeutung in der Biologie und der Informatik erläutert werden.
Für eine Vertiefung der Begriffe aus dem Bereich der Genetik siehe auch [Janning und Knust (2004)]. Evolutionäre Verfahren und ihr biologischer Hintergrund können bei [Gerdes u. a. (2004)] nachgeschlagen und vertieft werden.

Biologie

In der Biologie werden genetische Informationen in Genomen gespeichert. Genomen enthalten mehrere Gene und bestehen aus DNS[2], die mit Proteinen verpackt sind. Genome haben eine feste länge. Gene sind ein Teilstück eines Genoms. Jedes Gen ist für eine Teileigenschaft verantwortlich. So legt, zum Beispiel, ein Gen die Augenfarbe eines Menschen fest. Die Ausprägungen dieser Gene nennen sich Allele. Auf das Beispiel mit der Augenfarbe bezogen wäre ein Allel, zum Beispiel, blau. Der Begriff Locus gibt in der Biologie den physikalischen Ort eines Genes im Genom an.

Der Phänotyp gibt das Erscheinungsbild eines Lebewesens (Größe, Gewicht, Augenfarbe) an. Dazu zählen auch erworbene Eigenschaften[3]. Neben dem Phänotyp gibt es den Genotyp, der das Erscheinungsbild eines Genoms definiert. Er repräsentiert seine exakte genetische Ausstattung. Ein Genotyp verändert sich im Leben eines Organismus nicht. Ausnahmen bilden Unfälle mit radioaktiver α, β oder γ-Strahlung oder Temperaturschocks. Den durchschnittlichen Anteil eines Genotyps an der nächsten Generation bezeichnet die Fitness. Dieser Anteil wird rechnerisch mit dem Anteil anderer Genotypen verglichen. Die Fitness

[1] Britischer Naturforscher (1809 - 1882) und einer der bedeutendsten Naturwissenschaftler
[2] Desoxyribonukleinsäure
[3] Beispiele für erworbene Eigenschaften: Vergrößerte oder verkümmerte Muskelgruppen - je nach Gebrauch, Zwergwuchs durch widrige Umweltbedingungen, etc.

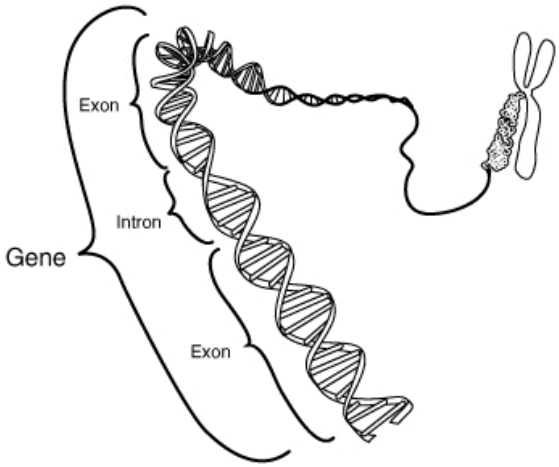


Abbildung 2.2.: Schematische Darstellung eines Gens auf einem DNS-Strang

ist ein wichtiges Element der Evolutionstheorie. Wird die Fitness im evolutionären Zusammenhang betrachtet, so wird sie nur an der Vermehrungsrate eines Genotyps zu anderen gemessen.

Populationen sind aus biologischer Sicht eine Menge von Lebewesen der gleichen Art, die eine Fortpflanzungsgemeinschaft bilden und sich in einheitlichen Arealen aufhalten. Populationen ändern sich durch Reproduktion und Todesfälle ständig. Eine Population zu einem bestimmten Zeitpunkt ist eine Generation. Die Reproduktion ist die Erzeugung von Nachkommen von einem oder mehreren Lebewesen.

Die Selektion ist ein normaler Prozess, den man bei allen Fortpflanzungsgemeinschaften beobachten kann. Gut an die Umgebung und ihre Bedingungen angepasste Lebewesen pflanzen sich mit großem Erfolg fort. Lebewesen hingegen, die schwach oder sogar krank sind, weil ihr Immunsystem nicht ausreichend angepasst ist, haben nicht so viel Erfolg. Statistisch gesehen haben angepasstere Lebewesen eine höhere Fortpflanzungswahrscheinlichkeit als die übrigen. So kommt es, dass ihre Gene vermehrt in späteren Generationen vertreten sind und die Gene der übrigen langsam verschwinden. So überleben und vermehren sich die angepasstesten Lebewesen (survival of the fittest) und dominieren die folgenden Generationen.

Wenn sich zwei Lebewesen miteinander reproduzieren, entsteht durch die Rekombination der zwei Genotypen ein neuer dritter Genotyp, der Nachkomme. Bei der Rekombination vermischen sich die Gene der beiden Lebewesen: Die eine Hälfte des DNA-Strangs des einen Lebewesens kombiniert sich mit der anderen Hälfte des DNA-Strangs des anderen Lebewesens. Die Mutation bildet eine weitere Möglichkeit neue Genotypen zu erhalten. Bei

der Mutation werden einzelne Gene mit anderen Allelen besetzt. Mutationen treten in der Regel bei Unfälle mit radioaktiver α, β oder γ-Strahlung oder Temperaturschocks auf.

Informatik

In der Informatik werden die Begriffe aus der Biologie typischerweise nicht so streng übernommen. In dieser Arbeit werden die Begriffe wie folgt gehandhabt:

Genome beinhalten, analog zur Biologie, mehrere Gene. Das Genom kann, zum Beispiel, eine Zeichenkette sein und die Gene sind Teile dieser Zeichenkette. Diese Zeichenkette kann von einem Interpreter zu einem Phänotyp interpretiert werden. Die Begriffe Allel und Locus werden nicht weiter verwendet. Genotyp und Genom werden synonym verwendet.

Eine Population entspricht in der Informatik einer Menge von Genomen. Diese Population bleibt konstant bis die Bewertung jedes Genoms vollzogen ist. Erst dann wird die Population modifiziert. Eine Generation ist wie in der Biologie eine Population zu einem bestimmten Zeitpunkt. Eine Generation reicht von einem Modifikationsprozess der Population bis zum nächsten. Generationen werden allgemein von Null an durchgezählt.

Die Fitness eines Genoms wird von der Fitnessfunktion des genetischen Lernverfahrens bestimmt. Die ermittelte Fitness bestimmt in der Informatik den Anteil an der nächsten Generation, und nicht, wie in der Biologie, der Anteil an der nächsten Generation die Fitness.

Unter Reproduktion wird die Erzeugung eines neuen Genoms aus einem oder mehreren Genomen verstanden. Bei der Selektion ist die Fitness jedes Genoms entscheidend. Je höher die Fitness, desto größer ist die Chance eines Genoms über Mutation, Kreuzung oder Elitismus (direkte Emigration) einen Anteil an der nächsten Generation zu Bilden. Mit der Rekombination (Crossover) ist des Erstellen eines neuen Genomes aus bereits vorhandenen gemeint. Der einfachste Fall ist in der Informatik, dass die Eltern-Genome an einem festen Punkt getrennt, von Genom A der vordere und von Genom B der hintere Teil genommen wird. Aus diesen beiden Teilen wird dann Genom C gebildet.

Die Mutation funktioniert in der Informatik etwas anders als in der Biologie. Während in der Biologie die Gene irgendwann im Leben eines Lebewesens mutiert werden, werden in der Informatik bei der Reproduktion zufällig Gene ausgewählt und durch ein neues zufälliges Gen ausgetauscht. Ist die Kodierung des Genoms binär, so wird an diesem Gen einfach nur ein Bit-Flip durchgeführt. Das Maß der Mutation pro Reproduktion wird entweder durch eine feste Anzahl an Mutationen oder durch eine Wahrscheinlichkeit bestimmt.

2.1.3. Genetische Algorithmen

Die Strukturen und Verfahren der Evolutionstheorie, welche im vorherigen Abschnitt erläutert wurden, sind das Material mit dem genetische Algorithmen arbeiten. Wie die Verfahren in den genetischen Algorithmen umgesetzt werden können, soll in den folgenden Sektionen gezeigt werden.

Allgemeiner Ablauf

Auch wenn genetische Algorithmen in vielen Formen auftreten und ganz verschiedene Operatoren verwenden, so ist ihr prinzipieller Ablauf doch meistens gleich.

Das folgende Schema spiegelt den Ablauf der meisten Genetischen Algorithmen wieder:

1. Initialisierung Erstellen der Urpopulation (Generation 0).

2. Evaluation Berechnen der Fitness jedes Individuums der aktuellen Generation.

3. Selektion Zufällige Auswahl von Individuen der aktuellen Generation. Individuen mit höherer Fitness haben eine größere Chance ausgewählt zu werden.

4. Rekombination Genome verschiedener selektierter Individuen der aktuellen Generation werden gekreuzt und bilden eine neue Generation.

5. Mutation Zufällig werden einzelne Gene von Individuen der neuen Generation mutiert.

6. Fertigstellung Die neue Generation wird um weitere Individuen ergänzt, bis sie die vorher bestimmte Populationsgröße erreicht hat. Hier werden häufig die Gewinner der alten Generation mutiert und unmutiert genommen.

7. Abbruchbedingung Wenn eine entsprechende Abbruchbedingung, wie zum Beispiel: Generationsanzahl oder beste Fitness, nicht erreicht wurde, so fängt der Genetische Algorithmus wieder bei der Evaluation an.

Fitnessfunktion

Die Fitnessfunktion oder auch Zielfunktion ist für den Lernerfolg entscheidend. In ihr ist die Zielsetzung des Lernverfahrens festgehalten. Sie berechnet die Qualität jedes Genotyps. Je näher ein Genotyp dem Lernziel ist, desto größer muss der ermittelte Fitnesswert sein. Kann die Fitnessfunktion diesem Anspruch nicht gerecht werden, so ist es möglich, dass gute Lösungen gar nicht gefunden werden können oder gefunden, aber dennoch verworfen werden. Die Fitnessfunktion bestimmt somit zum größten Teil, welche Lösung das Ergebnis des genetischen Lernverfahrens wird.

Häufig müssen bestimmten Zusatzbedingungen erfüllt werden, ohne die eine Lösung wertlos ist. So ist eine Lösung beim TSP[4], bei der nicht alle Städte besucht werden, wertlos. Für solche Lösungen gibt es drei mögliche Verfahrensweisen:

- Fester Strafwert, der Penalty, pro verletzter Bedingung wird von der Qualität des Genotyps abgezogen.
- Variabler Strafwert, je nach Ausmaß des Fehlers, wird von der Qualität des Genotyps abgezogen.
- Der Genotyp erhält die Qualität $-\infty$

Eine weitere interessante Form der Fitnessfunktion ist die geteilte Fitness. Sie wird im folgenden Abschnitt genauer erklärt.

Geteilte Fitness Die geteilte Fitness entstand aus dem Problem der vorzeitigen Konvergenz durch ein dominierendes Individuum. Um dem entgegenzuwirken wird jedes Individuum mit den anderen einer Population auf Ähnlichkeit verglichen. Je mehr und desto stärker sich Individuen ähneln, desto mehr wird die normale Fitness abgewertet.

Eine solche Fitnessfunktion könnte folgendermaßen aussehen:

$$fit(g) = \frac{fit^*(g)}{\sum_{i=1}^{popsize} Sh(d(g, g_i))}$$

Dabei sei $d(g, g_i)$ die Ähnlichkeit als Abstand zweier Genotypen. $Sh(d(g, g_i))$ ist eine Sharingfunktion, die diesen Abstand auf das Interval[0,1] abbildet. Hierbei gilt je größer der Abstand zweier Genotypen, desto größer der Wert, den die Sharingfunktion zurückliefert. Der Ausdruck unter dem Bruchstrich wird auch als “Nischenzähler” (Niche Count) bezeichnet.

Selektionsverfahren

Das Selektionsverfahren wählt die Individuen zum Mitwirken an der nächsten Generation aus. Um eine ständig verbesserte Anpassung zu erlangen, sollten die Individuen mit höherer Fitness bevorzugt werden. Drei Selektionsverfahren sollen im Folgenden näher erläutert werden: Die Rouletterad-Selektion, die Ranking-Methode und die Schöpfkellen-Methode (best of *n*).

[4] Traveling Salesman Problem (Problem des Handlungsreisenden)

Die Roulette-Selektion ist das wohl verbreitetste Selektionsverfahren. Hierbei ist die Wahrscheinlichkeit der Selektion eines Individuums proportional zur relativen Fitness. Jedes Individuum erhält entsprechend seiner Fitness einen Anteil am Rouletterad, ähnlich eines Kreisdiagramms. Anschließend wird *n*-mal Roulette gespielt. Das Individuum, in dessen Rouletteradteil die Kugel liegen bleibt, kommt in die nächste Generation.

Das folgende Beispiel soll den Zusammenhang zwischen Fitness und Wahrscheinlichkeit verdeutlichen.

Genom	Absolute Fitness	Relative Fitness
1	30	0,1
2	150	0,5
3	15	0,05
4	75	0,25
5	30	0,1
	300	1,00

Abbildung 2.3.: Aufteilung eines Rouletterads

Ablauf der Roulette-Selektion:

- Ermitteln der absoluten Fitness f jedes Individuums i an Hand der Fitnessfunktion $fit()$.

$$f_i = fit(i)$$

- Ermitteln der gesamten Fitness F der Population.

$$F = \sum_{i=1}^{popsize} f_i$$

- Für jedes Individuum die Selektionswahrscheinlichkeit p ermitteln.

$$p_i = \frac{f_i}{F}$$

- Ermitteln der kumulativen Selektionswahrscheinlichkeit q für jedes Individuum.

$$q_i = \sum_{j=1}^{i} p_j$$

- n-mal erzeugen einer reellen Zufallszahl r aus dem Interval $[0, 1]$. Zu jedem r das passende Individuum über $q_{i-1} < r \leq q_i$ ermitteln. Für $i = 1$ gilt es $r \leq q_i$ zu erfüllen.

Die Ranking-Methode gewichtet die Fitnessdifferenzen zwischen den Individuen nicht so stark. Es kommt nur darauf an, welches Individuum eine bessere Fitness hat, wie viel besser, ist unerheblich. Dazu werden zunächst alle Individuen absteigend nach ihrer Fitness sortiert und von eins an indiziert. Nun spielt nur noch der Rang eine Rolle bei der Berechnung der Selektionswahrscheinlichkeit.

Ablauf der Ranking-Methode:

- Ermitteln der absoluten Fitness f jedes Individuums i an Hand der Fitnessfunktion $fit()$.

$$f_i = fit(i)$$

- Für jedes Individuum die Selektionswahrscheinlichkeit p ermitteln. Die Konstante s aus dem Interval $\left[\frac{1}{popsize}, \frac{2}{popsize}\right]$ steuert den Selektionsdruck. r ergibt sich aus $\frac{2(s*popsize-1)}{(popsize(popsize-1)}$.

$$p_i = s - r * (index_i - 1)$$

- Ermitteln der kumulativen Selektionswahrscheinlichkeit q für jedes Individuum.

$$q_i = \sum_{j=1}^{i} p_j$$

- n-mal erzeugen einer reellen Zufallszahl z aus dem Interval $[0, 1]$. Zu jedem z das passende Individuum über $q_{i-1} < z \leq q_i$ ermitteln. Für $i = 1$ gilt es $z \leq q_i$ zu erfüllen.

Die Schöpfkellen-Methode (best of *n*) , oder auch Turnier-Prinzip, ist ein recht einfaches Selektionsverfahren. Wie mit einer Schöpfkelle werden aus dem großen Behälter einer Generation *n* Individuen zufällig entnommen. Das beste der *n* Individuen wird für die nächste Generation selektiert. Die Größe der Schöpfkelle wird über eine Konstante festgelegt, diese Konstante bestimmt den Selektionsdruck. Je größer die Schöpfkelle ist, desto geringer die Chance, dass ein Individuum schlechterer Fitness selektiert wird. Übliche Größen der Schöpfkelle sind zwei oder drei.

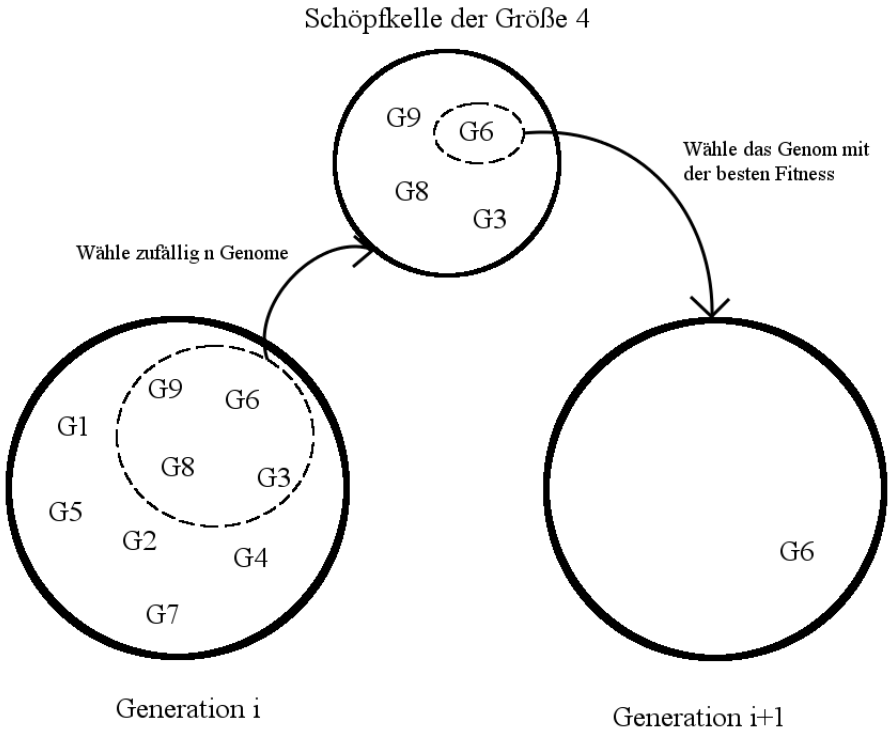


Abbildung 2.4.: Schöpfkellen-Methode

Genetische Operatoren

Die Rekombination (Crossover) , oder auch Kreuzung, bildet aus mehreren (in der Regel zwei) Eltern-Genomen ein oder mehr Kind-Genome für die nächste Generation. Hier sollen das *k*-Point-Crossover und das parametrisierte Uniform Crossover näher erläutert werden.

Das *k*-Point-Crossover teilt das Genom zufällig an *k* Kreuzungspunkten in mehrere Teile und kombiniert diese Teile neu.

Die einfachste Variante ist das One-Point-Crossover. Es wird zufällig eine Stelle im Genome definiert und der Teil hinter diesem Punkt zwischen den Genomen vertauscht. Empirischen Resultaten zufolge liefert das One-Point-Crossover die schlechtesten Resultate.

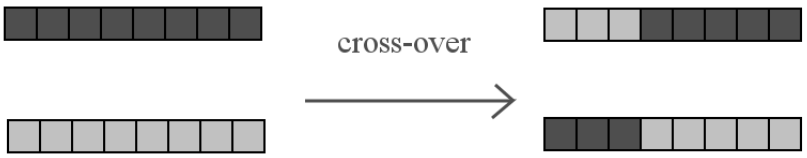


Abbildung 2.5.: One-Point-Crossover

Der am weitesten verbreitete Crossover-Operator ist der Two-Point-Crossover. Hier werden zwei Kreuzungspunkte definiert und die zwischen diesen Punkten liegenden Gene werden vertauscht.

Das parametrisierte Uniform Crossover bestimmt über eine Konstante p_c, die typischerweise zwischen $0,5$ und $0,8$ liegt, mit welcher Wahrscheinlichkeit ein Gen zwischen den Genomen vertauscht wird. Dadurch ergeben sich gut durchmischte neue Genome. Der Fall $p_c = 0.5$ heißt Uniform Crossover.

Abbildung 2.6.: Uniform Crossover

Die Mutation bewirkt kleinere Änderungen am Genom. Im zweidimensionalen Lösungsraum wird dadurch die nähere Umgebung des Genoms durchsucht. Bei einem hohen Maß an Mutationen kann es auch zu großen Sprüngen im Lösungsraum kommen. Die Größe der Sprünge im Lösungsraum ist proportional zum Mutationsgrad. Die Anzahl der Mutationen kann entweder über eine feste Anzahl pro Genom festgelegt werden oder man definiert eine Wahrscheinlichkeit, mit der ein Gen mutiert. Wird eine Wahrscheinlichkeit gewählt, so muss für jedes Gen erwürfelt werden, ob es mutiert. Mutiert ein Gen so wird das vorhandene Gen durch ein anderes zufällig ermitteltes ersetzt.

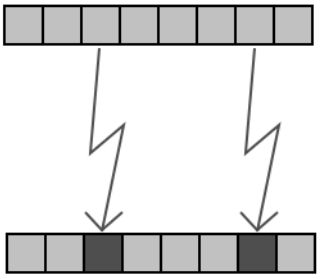


Abbildung 2.7.: Mutation eines Genoms

2.2. Agenten

Der Agenten-Begriff ist sehr vage. Allgemein lässt sich sagen, dass Agenten im Auftrag und Interesse anderer handeln. Agenten gibt es für zahlreiche Bereiche. Unter anderem handeln sie im Interesse von Staaten (Geheimdienst), Firmen (Industriespionage) und Privatpersonen (z.B. Booking Agent bei Künstlern). Aus dem Computerbereich ist wohl der Biet-Agent, der das Gebot kurz vor Auktionsende bei Onlineauktionshäusern abgibt und so den Zuschlag sichern soll, der bekannteste. Wie unterschiedlich der Agenten-Begriff verwendet wird, zeigt sich wenn ein James Bond mit einem Biet-Agenten verglichen wird. Trotzdem sollen anschließend Definitionen für unterschiedliche technische Agententypen dargestellt werden. Die folgenden Definitionen richten sich nach [Ferber (1999)]. Zur Vertiefung des Themas Multiagentensysteme wird hier außer auf [Ferber (1999)] auch auf [Wooldridge (2002)] verwiesen. Bereits in den 1970ern wurde an Agentensystemen geforscht. Aus dieser Zeit sind die Arbeiten von Carl Hewitt und Marvin Minsky, wie [Minsky (1988)], besonders zu empfehlen.

Es sollte noch gesagt sein, dass dies keine strikten Definitionen sind. Sie enthalten mehrere Punkte und es ist nicht gesagt, wieviele von diesen Punkten erfüllt sein müssen, um der Definition gerecht zu werden.

2.2.1. Definitionen

Es gibt eine Grundunterscheidung der Agenten in physikalische und virtuelle Einheiten. Eine physikalische Einheit ist etwas, das in der realen Welt agiert: ein Roboter, ein Flugzeug oder ein Auto sind ein paar Beispiele. Software oder ein Programmteil hingegen sind virtuelle Einheiten. Sie haben keine physikalische Existenz.

Trotz dieser Unterscheide gibt es eine allgemeine Definition, die folgendermaßen aussieht:

Ein Agent ist eine physikalische oder virtuelle Einheit,
- die in einer Umgebung agieren kann,
- die direkt mit anderen Agenten kommunizieren kann,
- die von einem Satz Tendenzen angetrieben wird: individuelle Ziele oder eine Zufriedenheits-/Überlebensfunktion, die versucht wird zu erfüllen,
- die eigene Ressourcen besitzt,
- die ihre Umgebung in einem gewissen Maß wahrnehmen kann,
- die eine teilweise Vorstellung der Umgebung hat (oder auch gar keine),
- die Fertigkeiten besitzt und Dienstleistungen anbieten kann,
- die eventuell fähig ist sich zu reproduzieren,
- deren Verhalten zur Erfüllung ihrer Ziele tendiert und hierfür verfügbare Ressourcen und Fähigkeiten abhängig von ihrer Wahrnehmung, ihrer Weltvorstellung und Kommunikation nutzt.

Multiagentensysteme

Ein Multiagentensystem (MAS) besteht aus mehreren (unterschiedlichen) Agenten, die gemeinsam ein Problem lösen.

Der Begriff Multiagentensystem (MAS) entspricht einem System, das die folgenden Elemente beinhaltet:
- Eine Umgebung (environment), E, die in der Regel ein Volumen besitzt.
- Einen Satz von Objekten, O. Diese Objekte sind situiert, so dass zu jedem Zeitpunkt jedes Objekt eine Position in E besitzt. Diese Objekte sind passiv, d. h. sie können von den Agenten wahrgenommen, erzeugt, zerstört und modifiziert werden.
- Eine Gruppe von Agenten, A, welche spezielle Objekte darstellen ($A \subseteq O$). Sie repräsentieren die aktiven Objekte des Systems.
- Eine Menge von Relationen, R, die Objekte (also auch Agenten) miteinander verbinden.
- Eine Sammlung von Operationen, Op, die es den Agenten von A möglich machen die Objekte von O zu bemerken, produzieren, konsumieren, transformieren und manipulieren.

Allgemein werden vier MAS-Typen unterschieden:

Kommunizierendes MAS (*communicating MAS*): Die Agenten im System tauschen Nachrichten direkt untereinander aus.

Rein kommunizierendes MAS (*purly communicating MAS*): Ein kommunizierendes MAS, in dem alle Objekte Agenten sind, die nichts anderes machen als Nachrichten auszutauschen. Außerdem ist keine Umgebung vorhanden.

Situiertes MAS (*situated MAS*): Allen Agenten ist in einer Umgebung eine Position zugewiesen.

Rein situiertes MAS (*purly situated MAS*): Ein situiertes MAS ohne Kommunikation über Nachrichten. Agenten können höchstens (ungerichtete) Signale aussenden, die von anderen über eventuelle Sensoren aufgefangen werden können.

Situierte Agenten

Als situierte Agenten (*situated agents*) werden Einheiten bezeichnet, die ihre Umgebung wahrnehmen und mit ihr agieren. Wenn dabei kein Nachrichtenaustausch mit anderen Agenten stattfindet, so spricht man von einem rein situierten Agenten (*purly situated agent*).

In Anlehnung an die oben genannte allgemeine Definition, lässt sich für einen rein situierten Agenten folgende Definition aufstellen:

Ein rein situierter Agenten ist eine physikalische Einheit (oder wenn simuliert, eine Recheneinheit),

- die in einer Umgebung eingebettet ist,
- die von einer Überlebens-/Zufriedenheitsfunktion getrieben wird,
- die Ressourcen in Form von Energie und Werkzeugen besitzt,
- die ihre Umgebung im gewissen Maße wahrnehmen kann,
- die praktisch keine Vorstellung von der Umgebung hat,
- die Fähigkeiten besitzt,
- die sich eventuell reproduzieren kann,
- dessen Verhalten zur Erfüllung ihrer Überlebens-/Zufriedenheitsfunktion tendiert. Hierzu werden verfügbare Ressourcen, Fähigkeiten und die Wahrnehmung der Umgebung verwendet.

Kommunizierende Agenten

Die kommunizierenden Agenten (*communicating agents*) können mit anderen Agenten Nachrichten austauschen und diese auch interpretieren. Ein rein kommunizierender Agent (*purly communicating agent*) ist außerdem in keiner Umgebung positioniert. Dieser wird auch Software-Agent (*software agent*) genannt.

Ein rein kommunizierender Agent (oder Software-Agent) ist als eine Recheneinheit definiert,

- die sich in einem offenen Computersystem befindet (Gruppe von Anwendungen, Netzwerke oder heterogene Systeme),
- die mit anderen Agenten kommunizieren kann,
- die von eigenen Zielsetzungen angetrieben ist,
- die eigene Ressourcen besitzt,
- die nur eine teilweise Vorstellung anderer Agenten hat,
- die Fähigkeiten (Dienstleistungen) besitzt, die angeboten werden können,
- dessen Verhalten zur Erfüllung ihrer Ziele tendiert. Hierzu werden verfügbare Ressourcen, Fähigkeiten und die Kommunikation mit anderen Agenten verwendet.

3. Analyse

In diesem Kapitel wird die Zielsetzung aus der Einleitung weiter vertieft. Es wird analysiert, wie man die Optimierung eines Ensembles durchführen kann. Hierfür muss stellvertretend ein Szenario entwickelt werden. Im Folgenden wird geklärt, wie das zu optimierende Ensemble aussieht. Die zu erfüllenden Anforderungen, die sich im Laufe dieses Kapitels ergeben, werden am Ende zusammengefasst.

Im gesamten Kapitel werden neben den textuellen Beschreibungen immer wieder formalere Beschreibungen auftauchen. Dieses dienen dem besseren Verständnis und sollen die Migration der Funktionalität von der Analyse ins Design unterstützen. Diese formaleren Beschreibungen werden *anders formatiert* dargestellt.

3.1. Szenario

Als Szenario dient eine stark vereinfachte marktwirtschaftliche Simulation. Diese Simulation erhebt nicht Anspruch die marktwirtschaflichen Aspekte naturgetreu wiederzugeben. Sie dient lediglich als Kulisse für das zu optimierende Ensemble. Die Simulation stellt mit verschiedenen Rollen ein Handelssystem dar. Diese Rollen bilden eine Handelskette vom Rohstofflieferanten bis zum Endkunden. Der Einfachheit halben gibt es nur zwei Güter: Rohstoff und Produkt. Um einen geregelten Ablauf zu gewährleisten wird das Szenario in Runden ablaufen. Eine Protokollierung der Vorgänge in eine oder mehrere Textdateien, zum Prüfen der Richtigkeit des Verhaltens und der Implementierung, sollte außerdem durchgeführt werden.

3.1.1. Rollen

In diesem Szenario sind vier Rollen vorgesehen, die anschließend erläutert werden.

Die vier Rollen im Detail:

Der Rohstofflieferant R fördert Rohstoffe und bietet diese zu einem beliebigen Preis auf dem Markt an. Die maximale Menge der förderbaren Rohstoffe schwankt. Der Rohstofflieferant kann je nach Lagerbestand selbst bestimmen, ob er die maximale Menge, einen Teil davon oder gar keine Rohstoffe fördert.

Der Spotmarkt S handelt mit Rohstoffen. Er kauft sie von Rohstofflieferanten und bietet sie auf dem Markt an. Auf welche und wie viele Angebote er von Rohstofflieferanten eingeht und zu welchem Preis er die Rohstoffe weiterverkauft, ist ganz allein ihm überlassen.

Der Hersteller H fertigt aus Rohstoffen ein Produkt. Ein Rohstoff ergibt ein Produkt. Dieses Produkt bietet er wiederum zu einem beliebigen Preis auf dem Markt an.

Der Endkunde K kauft ausschließlich Produkte. Er verfügt über ein Budget, das regelmäßig um die gleiche Summe aufgestockt wird. Den Endkunden kann man als Schnäppchenjäger im Kaufrausch betrachten. Er kauft so lange die günstigsten Produkte auf dem Markt, bis keine mehr vorhanden sind oder sein Budget erschöpft ist.

Die Rohstofflieferanten, Spotmärkte und Hersteller unterliegen Lagerhaltungskosten für ihre zu verkaufenden Waren. Wir sind in der Lage für unterschiedliche Rollen auch unterschiedliche Lagerhaltungskosten festzusetzen.

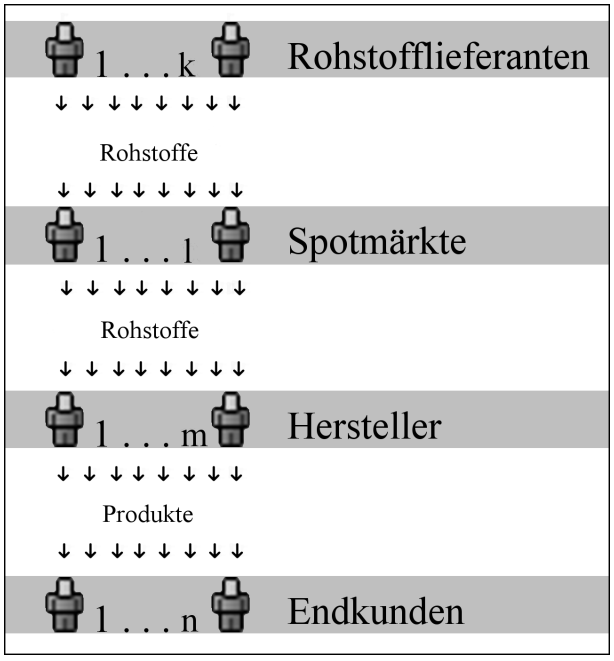


Abbildung 3.1.: Warenfluss zwischen den Rollen

3.1.2. Charaktere

Um eine größere Variation an Ensembles zu erhalten, haben die Rollen unterschiedliche Charaktere (Verhalten). Der Endkunde ist der einzige der vier Rollen, der nur einen Standard-Charakter d besitzt. Ansonsten gibt es drei verschiedene Charaktere:

Passiv p **(*vorsichtig*):** Nur kleine Preisanpassungen ($\pm 5\%$). Kauft oder fördert nur dann, wenn er keinen Lagerbestand mehr hat. Kauft nur das günstigste Angebot.

Moderat m**:** Moderate Preisanpassungen ($\pm 10\%$). Kauft oder fördert nur dann, wenn er keinen Lagerbestand mehr hat. Versucht günstig einzukaufen, aber sein Hauptziel ist es das Lager wieder zu füllen. Deshalb greift er ggf. auch auf teurere Angebote zurück.

Aggressiv a **(*riskant*):** Starke Preisanpassungen ($\pm 20\%$). Versucht möglichst viele Güter in seinen Besitz zu bringen und dadurch den Preis zu bestimmen. Er kauft/fördert so viel er kann. Hierbei besteht die Gefahr, dass die Lagerhaltungskosten das Guthaben dezimieren.

3.2. Das Ensemble

In der Einleitung wurde beschrieben, dass ein Ensemble (eine Zusammenstellung unterschiedlicher Komponenten) optimiert werden soll. Hier muss zunächst entschieden werden, ob das Ensemble eine feste oder variable Größe besitzen soll. Aufgrund der besseren Vergleichbarkeit ist die Entscheidung auf eine feste Größe gefallen.

Aus dem Szenario ergibt sich folgender Aufbau eines Ensembles: Als Komponenten werden Agenten verwendet, denen jeweils eine Rolle mit einem Charakter zugewiesen werden. Die Agenten jedes Ensembles bilden so ein kommunizierenden MAS (siehe Seite 23).

3.2.1. Formal

Für ein besseres Verständnis soll an dieser Stelle der Aufbau des Ensembles nochmal in einer formaleren Form zusammengefasst werden.

Das Ensemble E besteht aus den Rollen Rohstofflieferanten R, Spotmärkten S, Herstellern H und Endkunden K.

$$E = \{R, S, H, K\}$$

Jede dieser Rollen kann verschiedene charakteristische Ausprägungen besitzen. R, S und H können einen passiven p, moderaten m oder aktiven a Charakter annehmen. K besitzt

hingegen immer einen Standard-Charakter d. Jede Rolle kann beliebig oft, unabhängig von der Anzahl anderer Rollen vertreten sein.

$$R = R_p \cup R_m \cup R_a$$
$$\emptyset = R_p \cap R_m \cap R_a$$
$$r_i \in R, i \in \mathbb{N}$$

$$S = S_p \cup S_m \cup S_a$$
$$\emptyset = S_p \cap S_m \cap S_a$$
$$s_j \in S, j \in \mathbb{N}$$

$$H = H_p \cup H_m \cup H_a$$
$$\emptyset = H_p \cap H_m \cap H_a$$
$$h_k \in H, k \in \mathbb{N}$$

$$K = K_d$$
$$k_l \in K, l \in \mathbb{N}$$

3.3. Ablauf

Jedes Ensemble durchläuft im Szenario die folgenden Phasen. Die unten aufgelisteten Punkte werden von oben nach unten linear durchlaufen. Unter den einzelnen Punkten stehen formalere Ausdrücke, diese sollen für eine bessere Übertragung der hier beschriebenen Funktionalität ins Design dienlich sein. Parallel zu den Punkten in der Liste bearbeitet jeder Agent permanent die eingehenden Nachrichten. Dadurch braucht man die Kommunikation innerhalb des linearen Ablaufs nur einmal durch eine Nachricht anzustoßen - und der Rest des Handelns läuft parallel.

Damit nicht innerhalb einer Rolle ein Agent durch seine Position im Ablauf (Erster oder Letzter) Vor- oder Nachteile erfährt, muss die Reihenfolge der Agenten bei jedem Durchlauf zufällig gewählt werden.

- Rohstofflieferanten
  - Rohstoffe fördern
    $lagerbestand_{neu}(r_i) = lagerbestand_{alt}(r_i) + foerdern(r_i)$
  - Lagerhaltungskosten zahlen
    $budget_{neu}(r_i) = budget_{alt}(r_i) - (lagerbestand_{neu}(r_i) \times lagerkosten(R))$
  - Verkaufspreis anpassen
    $preis_{neu}(r_i) = preis\_kalkulieren(preis_{alt}(r_i), lagerbestand_{neu}(r_i), r_i)$
  - Altes Angebot entfernen
    $angebote(R) = \{x \mid x\ ist\ angebot(r \in R,\ r \neq r_i)\} = angebot\_zurueckziehen(r_i)$
  - Neues Angebot erstellen
    $angebot(r_i) = angebot\_erstellen(lagerbestand_{neu}(r_i), preis_{neu}(r_i), r_i)$
  - Neues Angebot aushängen
    $angebote(R) = angebote(R) \cup angebot(r_i) = angebot\_aushaengen(angebot(r_i))$
- Spotmärkte
  - Angebote von Rohstofflieferanten für Rohstoffe einholen
    $angebote(R) = \{x \mid x\ ist\ angebot(r \in R)\} = hole\_angebote(R)$
  - Angebote von Rohstofflieferanten annehmen
    $lagerbestand_{neu}(s_i) = lagerbestand_{alt}(s_i) + angebot(r_i)_{stückzahl}$
    $lagerbestand_{neu}(r_i) = lagerbestand_{alt}(r_i) - angebot(r_i)_{stückzahl}$
    $budget(s_i)_{neu} = budget(s_i)_{alt} - (angebot(r_i)_{stückzahl} \times angebot(r_i)_{stückzahl})$
    $budget(r_i)_{neu} = budget(r_i)_{alt} + (angebot(r_i)_{stückzahl} \times angebot(r_i)_{stückzahl})$
  - Lagerhaltungskosten zahlen
    $budget_{neu}(s_i) = budget_{alt}(s_i) - (lagerbestand_{neu}(s_i) \times lagerkosten(S))$
  - Verkaufspreis anpassen
    $preis_{neu}(s_i) = preis\_kalkulieren(preis_{alt}(s_i), lagerbestand_{neu}(s_i), s_i)$
  - Altes Angebot entfernen
    $angebote(S) = \{x \mid x\ ist\ angebot(s \in S,\ s \neq s_i)\} = angebot\_zurueckziehen(s_i)$
  - Neues Angebot erstellen
    $angebot(s_i) = angebot\_erstellen(lagerbestand_{neu}(s_i), preis_{neu}(s_i), s_i)$
  - Neues Angebot aushängen
    $angebote(S) = angebote(S) \cup angebot(s_i) = angebot\_aushaengen(angebot(s_i))$
- Hersteller

  - Angebote von Spotmärkten für Rohstoffe einholen
    $angebote(S) = \{x \mid x\ ist\ angebot(s \in S)\} = hole\_angebote(S)$
  - Angebote von Spotmärkten annehmen
    $lagerbestand_{neu}(h_i) = lagerbestand_{alt}(h_i) + angebot(s_i)_{stückzahl}$
    $lagerbestand_{neu}(s_i) = lagerbestand_{alt}(s_i) - angebot(s_i)_{stückzahl}$
    $budget(h_i)_{neu} = budget(h_i)_{alt} - (angebot(s_i)_{stückzahl} \times angebot(s_i)_{stückzahl})$
    $budget(s_i)_{neu} = budget(s_i)_{alt} + (angebot(s_i)_{stückzahl} \times angebot(s_i)_{stückzahl})$
  - Lagerhaltungskosten zahlen
    $budget_{neu}(h_i) = budget_{alt}(h_i) - (lagerbestand_{neu}(h_i) \times lagerkosten(H))$
  - Verkaufspreis anpassen
    $preis_{neu}(h_i) = preis\_kalkulieren(preis_{alt}(h_i), lagerbestand_{neu}(h_i), h_i)$
  - Altes Angebot entfernen
    $angebote(H) = \{x \mid x\ ist\ angebot(h \in H,\ h \neq h_i)\} = angebot\_zurueckziehen(h_i)$
  - Neues Angebot erstellen
    $angebot(h_i) = angebot\_erstellen(lagerbestand_{neu}(h_i), preis_{neu}(h_i), h_i)$
  - Neues Angebot aushängen
    $angebote(H) = angebote(H) \cup angebot(h_i) = angebot\_aushaengen(angebot(h_i))$

- Endkunden
  - Budget aufstocken
    $BUDGET \in \mathbb{N}$
    $budget_{neu}(K_i) = budget_{alt}(K_i) + BUDGET$
  - Angebote von Herstellern für Produkte einholen
    $angebote(H) = \{x \mid x\ ist\ angebot(h \in H)\} = hole\_angebote(H)$
  - Angebote von Herstellern annehmen
    $lagerbestand_{neu}(k_i) = lagerbestand_{alt}(k_i) + angebot(h_i)_{stückzahl}$
    $lagerbestand_{neu}(h_i) = lagerbestand_{alt}(h_i) - angebot(h_i)_{stückzahl}$
    $budget(k_i)_{neu} = budget(k_i)_{alt} - (angebot(h_i)_{stückzahl} \times angebot(h_i)_{stückzahl})$
    $budget(h_i)_{neu} = budget(h_i)_{alt} + (angebot(h_i)_{stückzahl} \times angebot(h_i)_{stückzahl})$

3.4. Kommunikation

Die Kommunikation ist ein wichtiger Bestandteil dieses Systems. Im Folgenden soll die Struktur und Ablauf der Kommunikation näher erläutert werden.

3.4.1. Das schwarze Brett

Im Szenario sollen die Agenten Angebote auf dem Markt zur Verfügung stellen. Hierfür bietet sich eine Art schwarzes Brett an, an dem jeder Agent Angebote veröffentlichen kann ($angebot\_hinzufuegen(angebot)$) oder seine Angebote wieder zurückziehen kann ($angebot\_entfernen(agent)$). Da jeder Agent nur entweder Rohstoffe oder Produkte anbietet, kann auch nur jeder Agent ein Angebot zur Zeit am schwarzen Brett aushängen. Die Agenten können von diesem schwarzen Brett alle Angebote für eine bestimmte Ware erfragen ($\{angebot_0, ..., angebot_i\} = hole\_angebote(ware)$).

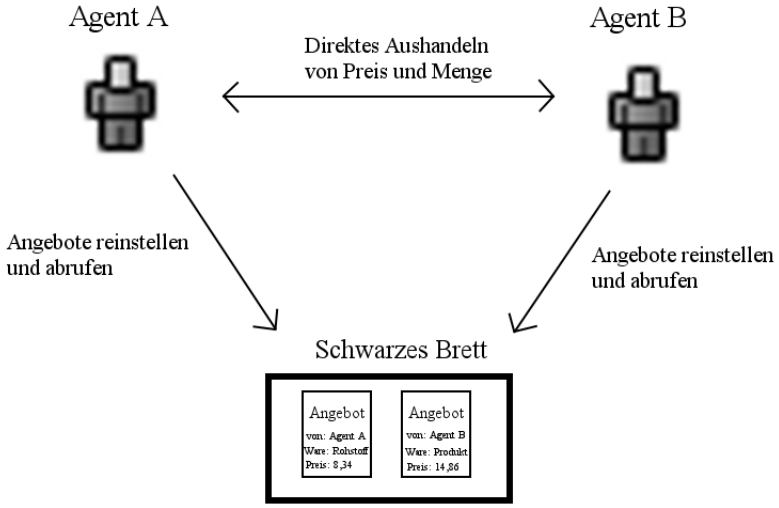


Abbildung 3.2.: Das schwarze Brett

3.4.2. Preisbörse

Über eine Preisbörse können sich alle Agenten die aktuellen Verkaufspreise für verschiedene Waren abrufen ($\{preis(r_0), ..., preis(r_i)\} = hole\_preise(rohstoff)$). Hierfür muss immer dann, wenn ein Agent seinen Verkaufspreis ändert, er seinen neuen Preis an der Preisbörse notieren ($benachrichtigen(ware, preis(r_i), r_i)$). Die Agenten haben so einen Vergleich zu den Konkurrenten und können je nach Charakter ihren Verkaufspreis entweder im oberen oder unteren Bereich der Preisspanne ansiedeln. Des Weiteren kann die Preisbörse die Funktionalität besitzen, die Preisentwicklung von einer beliebigen Anzahl Runden wie-

derzugeben $(\langle \{preis(r_0, t_x), ..., preis(r_i, t_x)\}, ..., \{preis(r_0, t_y), ..., preis(r_i, t_y)\}\rangle = hole\_preise(rohstoff, runden))$.

3.4.3. Nachrichtenaustausch

Jedes Ensemble bildet ein kommunizierendes MAS, in dem die Agenten über Nachrichten Handelsverträge abschließen. Dies setzt voraus, dass die Agenten die physikalische Möglichkeit haben, zu kommunizieren, und dass sie die Nachrichten des jeweils anderen auch interpretieren können.

3.4.4. Sprechakte

Damit sich die Agenten gegenseitig verstehen, benötigen sie eine Sprache aus bekannten Sprechakten. Sprechakte werden im gegenseitigen Wechsel ausgetauscht.

In der folgenden Tabelle werden die unterschiedlichen Sprechakte vorgestellt. Ist in den Spalten der möglichen vorherigen und folgenden Sprechakte ein oder mehrere Sprechakte angegeben, so muss einer von ihnen vorangegangen sein oder folgen.

Sprechakt (SA)	**mögl. vorherige SA**	**mögl. folgende SA**	**Beschreibung**
Angebot anfordern		-Angebot machen -Abbrechen	Agent A fordert von Agent B ein Angebot für eine bestimmte Ware an.
Angebot machen	-Angebot anfordern	-Angebot ablehnen -Angebot annehmen -Abbrechen	Agent B macht Agent A ein Angebot für eine bestimmte Ware mit Preis und *maximaler Stückzahl*.
Angebot ablehnen	-Angebot machen		Agent A lehnt das Angebot von Agent B ab.

Angebot annehmen	-Angebot machen	-Zustimmen -Abbrechen	Agent A nimmt das Angebot von Agent B an und teilt die *Stückzahl* ($\leq$ *maximaler Stückzahl* von *Angebot machen*) mit.
Zustimmen	-Angebot annehmen		Agent B besiegelt den Handel und sendet die endgültige von Agent A erworbene Stückzahl ($\leq$ *Stückzahl* von *Angebot annehmen*) mit.
Abbrechen	-Angebot anfordern -Angebot machen -Angebot annehmen		Agent A bricht den Dialog mit Agent B oder andersherum ab.

3.5. Der Suchraum und das Optimum

Alle möglichen Ensembles bilden den Suchraum, in dem sich das Optimum (Maximum) befindet. Dieses Optimum gilt es zu finden. Das einfachste Vorgehen wäre hierbei das Durchprobieren aller Möglichkeiten. Dies ist häufig, wie in dem oben beschriebenen Szenario, zu Zeitaufwändig. So ergeben sich bei einem Ensemble mit einer Größe von 15 Rollen 490.314 Kombinationen.[1] Mit 16 Rollen sind es schon 735.471 Kombinationen. Und bei einer Ensemblegröße von 40 ergeben sich 377.348.994 Kombinationen.
Das Durchprobieren bietet hierbei eine schlechte Skalierbarkeit. Der Ressourcenbedarf steigt so schnell, wie die Anzahl der Kombinationen. Eine wesentlich besser Skalierbarkeit bieten in der Regel chaotische Suchverfahren, die den Lösungsraum durchlaufen und versuchen, suboptimale Lösungen zu umgehen.

Das Optimum ist das beste Ensemble nach bestimmten Kriterien. Die Erfüllung dieser Kriterien bildet die Qualität eines Ensembles. Das Ensemble mit der höchsten Qualität ist das Optimum. Dabei muss zwischen lokalem und globalem Optimum unterschieden werden. Da

[1] Zur Berechnung wurde die Repetition verwendet: $\frac{(n+k-1)!}{k!(n-1)!}$, bei $n = 9$ (Charaktere) und $k = 15$ (Ensemblegröße).

nicht der vollständigen Suchraum durchlaufen wird, könnte es passieren, dass nicht das globale Optimum gefunden wird, sondern die Suche bei einem lokalen Optimum stagniert.

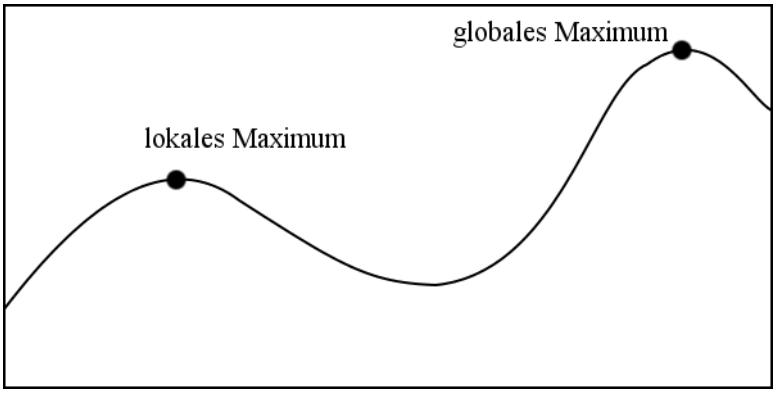


Abbildung 3.3.: Lokales und globales Maximum (Optimum)

3.6. Anforderungskatalog

In diesem Abschnitt sollen die sich aus den vorherigen Abschnitten ergebenen Anforderungen nochmal tabellarisch zusammengefasst werden. Hierbei wird in drei Kategorien unterteilt. Jede Anforderung bekommt eine der drei folgenden Kategorien zugeordnet:

must have: Die Anforderung ist elementar und muss auf jeden Fall erfüllt werden.

should have: Die Anforderung ist wichtig und sollte erfüllt werden. Es geht aber auch ohne sie.

nice to have: Die Anforderung bietet eine nette Ergänzung und sollte, wenn alle wichtigeren Anforderungen erfüllt sind, bei ausreichenden Ressourcen (Zeit und Budget) bearbeitet werden.

Anforderung	must have	should have	nice to have
Rollen	×		
Charaktere		×	
Technische Möglichkeit der Kommunikation	×		
Sprechakte	×		
Schwarze Brett		×	
Simulator mit GUI[a]		×	
Preisbörse			×
Protokollierung (Logging)		×	

[a]Graphical User Interface

4. Design und Realisierung

In diesem Kapitel wird die Architektur der zu entwickelnden Komponenten und deren Realisierung erläutert. Die Erläuterung erfolgt vom großen Überblick über alle Komponenten zu den einzelnen Details hin.

4.1. Überblick über das System

Das System verfügt über zwei große essentielle Komponenten: die virtuelle Maschine und die Evolutionsmaschine. Jede dieser Komponenten ist absolut notwendig für das gesamte System. Zu diesen zwei Komponenten kommt noch eine optionale dritte: die graphische Benutzerschnittstelle.

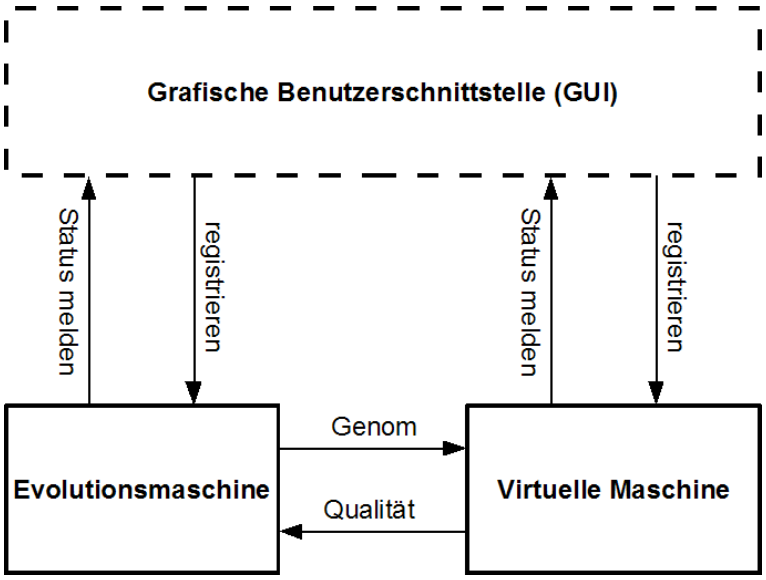


Abbildung 4.1.: Systemüberblick

Das Zusammenspiel der Komponenten sieht folgendermaßen aus: Die Evolutionsmaschine verwaltet die Populationen und lässt die Genome der Populationen von der virtuellen Maschine (VM) auf ihre Qualität hin bewerten. Die virtuelle Maschine lädt dazu die Genome in

das Szenario aus Kapitel 3.1 ein. Einen Überblick über das aktuell von der virtuellen Maschine zu bewertende Genom und den Fortschritt der Evolutionsmaschine liefert die graphische Benutzerschnittstelle.

4.2. Evolutionsmaschine

Die Evolutionsmaschine verwaltet die Generationen. Dazu gehört das Senden jedes Genoms einer Generation an die virtuelle Maschine, um es von ihr auf dessen Qualität hin bewerten zu lassen. Sind alle Genome der aktuellen Generation bewertet, wird mit Hilfe eines genetischen Optimierers die nächste Generation erzeugt. Dies geschieht so häufig, bis ein Grenzwert an Generationen oder eine bestimmte Qualität erreicht wurde.
Zur Erschaffung der ersten Generation werden zufällig Genome erzeugt. Ist ein Genom von der Qualität her besser als alle vorherigen, so wird es in eine Datei exportiert. Aus dieser Datei kann es zu einem späteren Zeitpunkt manuell erneut in die virtuelle Maschine geladen und begutachtet werden.

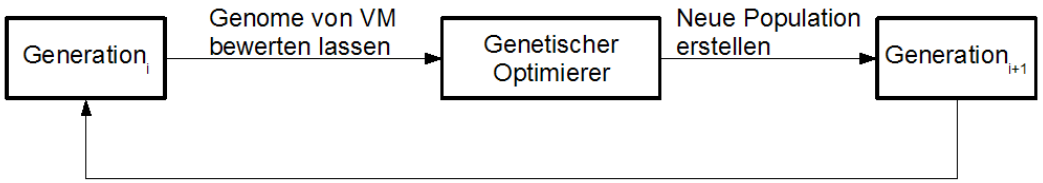


Abbildung 4.2.: Ablauf der Generationen

4.2.1. Genetische Optimierer

Der genetische Optimierer wendet einen Teil der in Kapitel 2.1 vorgestellten Verfahren an. Mit Hilfe dieser Verfahren werden aus bestehenden Generationen neue, besser angepasste Generationen erschaffen.

Der genetische Optimierer in dieser Arbeit läuft wie folgt ab, um eine neue Generation zu erstellen:

1. Elitismus: Eine vorher definierte Prozentzahl von Genomen wird unverändert in die nächste Generation übernommen.
2. Rekombination mit Heiratsschema: Eine vorher definierter Prozentsatz der alten Generation wird paarweise miteinander kombiniert. Hierfür wird das sehr einfach zu implementierende One-Point-Crossover verwendet.

3. Mutation mit Schöpfkelle: Die restlichen freien Plätze in der neuen Generation werden über die Schöpfkelle aufgefüllt. Die Schöpfkelle hat eine vordefinierte Größe. Das durch die Schöpfkelle ermittelte Genom wird anschließend mutiert in die neue Generation gegeben. Auch bei der Mutation gibt es einen vordefinierten Prozentsatz. Dieser gibt die Wahrscheinlichkeit an, mit der ein Gen durch ein zufälliges ersetzt wird.

Es stehen also folgende Stellschrauben des genetischen Optimierers in der Evaluation zur Auswahl:

- Prozentzahl für den Elitismus
- Prozentzahl für das Heiratsschema
- Größe der Schöpfkelle
- Wahrscheinlichkeit in Prozent für die Mutation einzelner Gene

Der genetische Optimierer hat nach außen folgende Methoden:

- `Generation createGeneration(Generation oldGeneration)` erzeugt aus `oldGeneration` unter Verwendung genetischer Verfahren eine neue Generation und gibt diese zurück.
- `void setElitismPercent(double percent)` setzt die Prozentzahl für den Eilitismus. $0 \leq$ `percent` ≤ 100.
- `void setWeddingshemePercent(double percent)` setzt die Prozentzahl für das Heiratsschema. $0 \leq$ `percent` ≤ 100.
- `void setScoopSize(int size)` setzt die Größe der Schöpfkelle. `size` ≥ 1.
- `void setMutationChance(double percent)` setzt die Wahrscheinlichkeit für die Mutation eines Genes. $0 \leq$ `percent` ≤ 100.

4.3. Die virtuelle Maschine

Die virtuelle Maschine dient der Bewertung der Genome. Die Genome werden im Kontext des Szenarios auf ihr Potenzial hin überprüft. Das Potenzial eines Genoms im Startzustand ergibt sich aus der Qualität des Genoms im Endzustand. Jedes Genom durchläuft die gleiche Anzahl von Schritten im Szenario und wird anschließend auf seine Qualität hin bewertet. Aus dieser Qualität ergibt sich sein Potenzial.

Der Ablauf in der VM wird durch künstlich herbeigeführte Zufälligkeiten beeinflusst: Die Menge der geförderten Rohstoffe schwankt und die Reihenfolge in der die Agenten agieren wird auch jedes Mal neu gemischt. Dies hat zur Folge, dass der gleiche Startzustand in der Regel nicht den gleichen Endzustand zur Folge hat. Dies bedeutet auch, dass die Qualität des Genoms im Endzustand schwankt. Dies hat zur Folge, dass je nach Ablauf in der VM dem gleichen Genom ein anderes Potenzial zugeschrieben wird.

Man kann auf drei Arten mit diesem nicht deterministischen Verhalten umgehen:

- Es wird ein sehr genaues Ergebnis für das Potential eines Genoms angestrebt. Man errechnet die Qualität aller möglichen Endzustände und mittelt diese. Dies wird je nach Anzahl von Schritten und Anzahl der möglichen Folgesituationen sehr teuer.
- Soll nur ein schneller und nicht so genauer Wert berechnet werden, so kann man einen Endzustand berechnen.
- Die dritte Möglichkeit ist ein Kompromiss aus beiden. Es wird eine feste Anzahl von Endzuständen berechnet und diese gemittelt.

Als praktische Lösung könnte man die Anzahl der zu berechnenden Endzustände von den zur Verfügung stehenden Ressourcen abhängig machen. So könnte, wenn mehrere Prozessorkerne zur Verfügung stehen, jeweils einer mit einem Thread, der einen Endzustand errechnet, belegt werden.
In dieser Arbeit wurde die schnelle und ungenauere Berechnung eines Endzustands verwendet, da weniger das eigentliche Ensemble als mehr die Möglichkeit der Optimierung von Interesse ist.

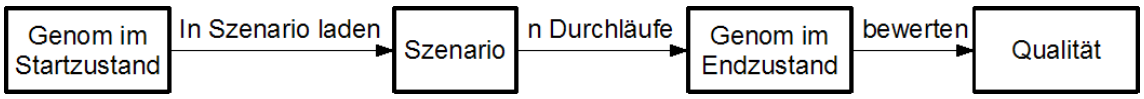


Abbildung 4.3.: Ablauf in der virtuellen Maschine

Ist die virtuelle Maschine mit einer graphischen Benutzerschnittstelle verbunden, hält sie die Benutzerschnittstelle inkrementell aktuell. Um die Aktualisierung möglichst ressourcensparend und zeitnah durchzuführen wurde das Observer-Pattern angewandt. Dadurch wird umgehend nur der Teil der Benutzerschnittstelle aktualisiert, der sich geändert hat.

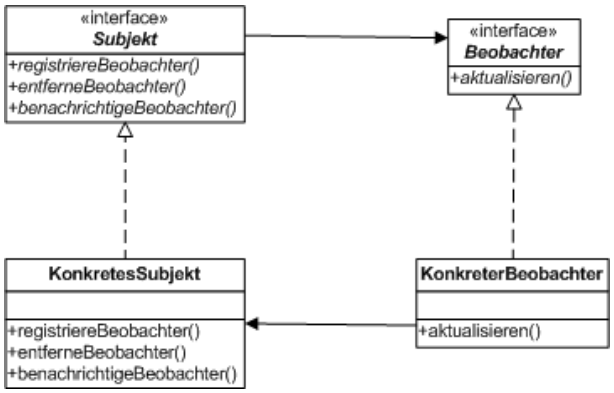


Abbildung 4.4.: Observer-Pattern

4.3.1. Szenario

Das Szenario ist ein Multiagentensystem (MAS) in einem marktwirtschaftlichen Kontext. Von hier an wird das Szenario als MAS betrachtet. An dieser Stelle soll auf die [FIPA] verwiesen werden, die eine ausführliche Definition für die Realisierung eines MAS bietet. Das Design dieser Arbeit wurde von den FIPA-Definitionen stark beeinflusst. Das MAS besteht aus folgenden Komponenten: Dem Proposal Broker (PB), dem Agent Management System (AMS) und dem Agent Communication Channel (ACC). Mit diesen Komponenten agiert eine Menge von Agenten.

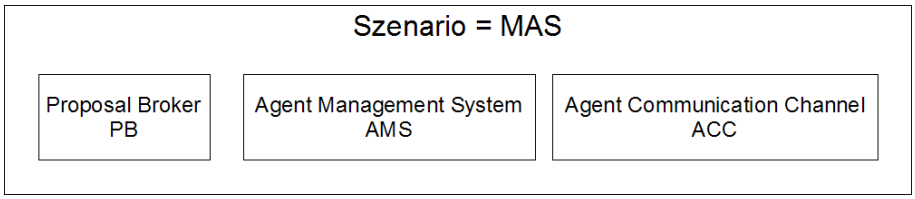


Abbildung 4.5.: Szenario-Komponenten

Proposal Broker

Der Proposal Broker übernimmt die Funktion des schwarzen Bretts, wie in der Analyse beschrieben (Kapitel 3.4.1). Er nimmt Angebote (Proposals) entgegen und stellt diese, nach Waren-Typen sortiert, wieder zur Verfügung. Der Proposal Broker wird als Remote Object über eine Registry publiziert.
Es wurde das Singleton-Pattern angewandt, um zu gewährleisten, dass es nur einen Proposal Broker pro MAS (Szenario) gibt.

Aus der formalen Bezeichnung in der Analyse ergibt sich in der Realisierung eine Klasse mit folgenden Methoden:

Formale Bezeichnung	Methode	Beschreibung
$\{angebot_0, ..., angebot_i\} = hole\_angebote(\ ware\ )$	`List<Proposal> getProposals (Units unit)`	Liefert eine Liste der Angebote, die `unit` zur Verfügung stellen. `Units` ist ein Enum mit den möglichen Werten Rohstoff und Produkt.
$angebot\_hinzufuegen(\ angebot\ )$	`void addProposals (Proposal p)`	Fügt das Angebot `p` dem Angebot-Pool hinzu.
$angebot\_entfernen(\ agent\ )$	`void removeProposals (String offerer)`	Entfernt Angebote von `offerer` aus dem Angebot-Pool. `offerer` ist der eindeutige Bezeichner eines Agenten.

Die Proposals werden in einer Kollektion verwaltet. Des weiteren wurde das Singleton-Pattern angewandt, da es in einem Szenario nur einen Proposal Broker geben kann.

Proposal

Ein Proposal (Angebot) beinhaltet die handelsrelevanten Informationen, wie Anbieter, Ware und Preis, außerdem eine für den Anbieter eindeutige Bezeichnung. Diese Bezeichnung wird bei Anfragen anderer Agenten mit angegeben, damit der Anbieter sie richtig zuordnen kann. Das Proposal dient lediglich als Behälter für Informationen und hat keine eigene Funktionalität. Proposals werden über den Proposal Broker publiziert.

Die Informationsfelder des Proposals:

Typ	Name	Beschreibung
`String`	`offerer`	Der eindeutiger Bezeichner des Anbieters
`Units`	`unit`	Die angebotene Ware
`double`	`price`	Der Stückpreis
`int`	`quantity`	Stückzahl der Ware, die zum Verkauf steht
`String`	`replyTo`	Eindeutige Bezeichnung des Angebots, die bei Rücksprache mit dem Anbieter angegeben wird.

Agent Management System

Das Agent Management System (AMS) stellt Agenten Telefonbuch-Dienstleistungen[1] zur Verfügung. Die Agenten melden sich dort mit ihrer Adresse und ihrer Rolle an. Das AMS ordnet ihnen daraufhin einen eindeutigen Bezeichner zu. Diese Informationen werden offen zur Verfügung gestellt. Alle Agenten können eine Liste mit Agenten einer bestimmten Rollen abfragen, um sie zu kontaktieren. Der AMS ist die einzige Komponente, die die Adresse dem eindeutigen Bezeichner eines jeden Agenten zuordnen kann - und umgekehrt.

Das AMS hat nach außen folgende Methoden:

`String registerAgent(AgentTypes type, String address)` wird von einem Agenten aufgerufen, um sich beim AMS zu registrieren. `type` (seine Rolle) und seine Adresse `address` werden dazu an das AMS übermittelt. Das AMS vergibt darauf hin als Rückgabewert den eindeutigen Bezeichner.

`void deregisterAgent(String aid, AgentTypes type)` deregistriert den Agenten mit dem eindeutigen Bezeichner `aid` (Agent ID) und der Rolle `type`.

`String getAddress(String aid)` liefert die Adresse zum eindeutigen Bezeichner `aid`.

`List<String> getAIDs(AgentTypes type)` liefert eine Liste von eindeutigen Bezeichnern der Agenten mit der Rolle `type`.

Intern werden die eindeutigen Bezeichner und physikalischen Adressen in einer Kollektion mit Schlüssel-Werte-Paaren einander zugeordnet. Der typische Fall sieht so aus, dass zu einem Bezeichner die Adresse ermittelt werden muss, deshalb ist der Schlüssel der Bezeichner und der zugeordnete Wert die Adresse.

[1] angelehnt an den FIPA-Begriff *white pages services*

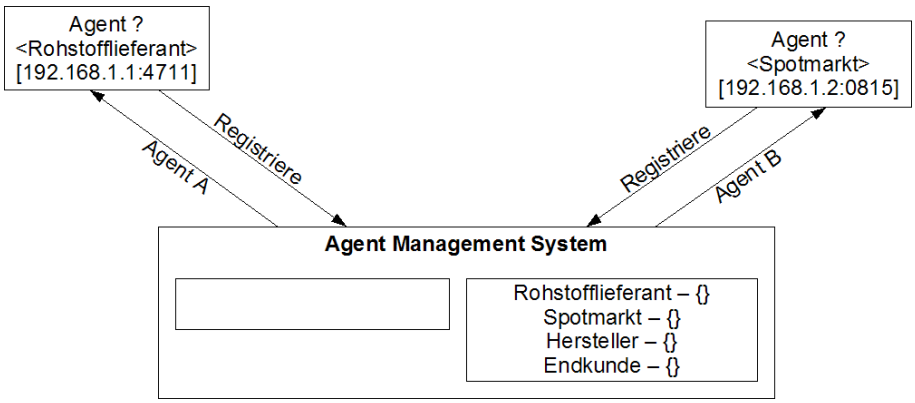


Abbildung 4.6.: Registrierungsvorgang am AMS

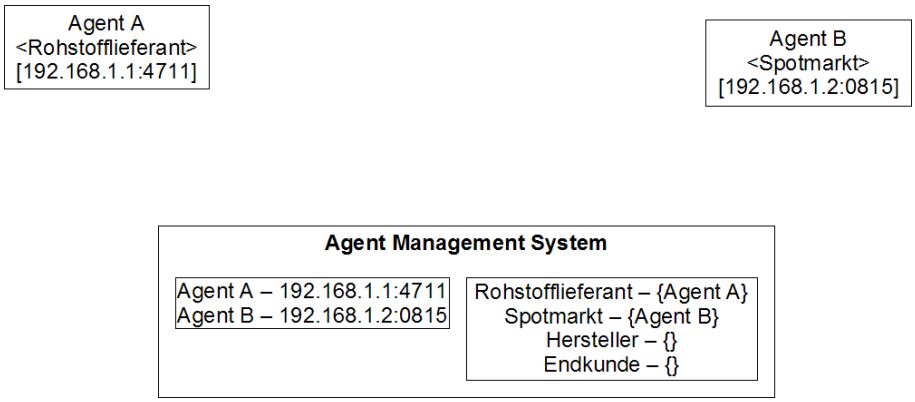


Abbildung 4.7.: Zustand nach der Registrierung am AMS

Agent Communication Channel

Der Agent Communication Channel (ACC) ist für die Übertragung aller Nachrichten von Agent zu Agent zuständig. Die Agenten kennen von den Proposals und den Nachrichten lediglich den eindeutigen Bezeichner der anderen Agenten. Wenn sie eine Nachricht an einen Agenten schicken, erfragen sie zuerst die Adresse beim AMS. Die eigentliche Nachricht wird in eine Transport-Nachricht mit der Adresse des Empfängers verpackt. Diese Nachricht wird dann an den ACC gesendet, der diese weiterleitet.
Der Vorteil dieser Struktur liegt in der niedrigen Anzahl an Verbindungen. Jeder Agent hat genau eine Verbindung: Die zum ACC. Lediglich ein entfernter Aufruf am AMS zum Erfragen der Adresse ist noch nötig.

Der ACC ist als Thread realisiert, um den Nachrichtenverkehr zeitnah und parallel bewältigen zu können.

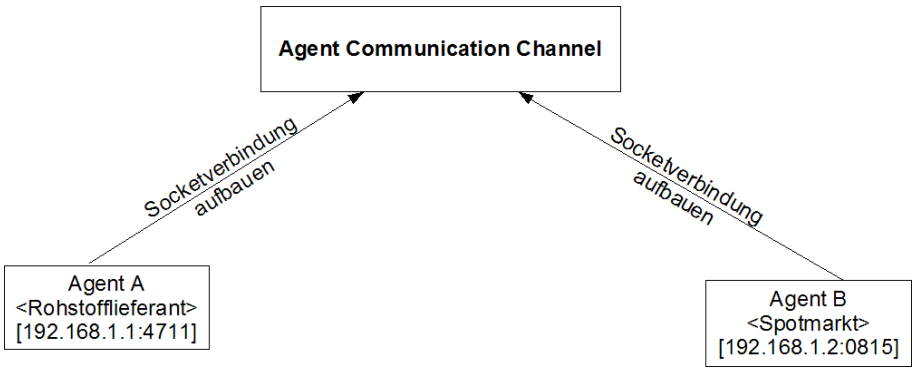


Abbildung 4.8.: Verbindungsaufbau zum ACC

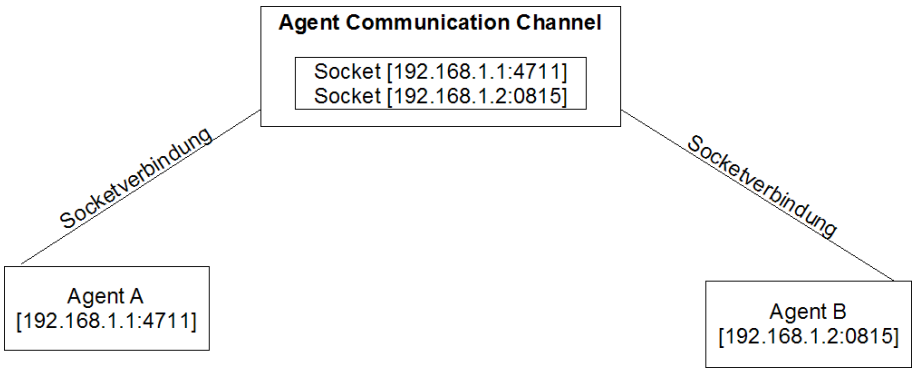


Abbildung 4.9.: Zustand nach dem Verbindungsaufbau zum ACC

4.3.2. Agent

Der Agent ist das Produkt des Ladens eines Genes in das Szenario. Die Agenten sind die Akteure im Szenario und führen die Aktionen zum Handeln aus.
Der Agent besteht aus drei Komponenten: seiner Rolle, seinem Charakter und einem Verbindungsmodul.

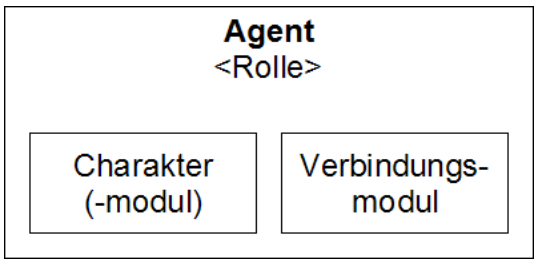


Abbildung 4.10.: Der Agent und seine Module

Die Rolle definiert seine Kosten für die Lagerhaltung und die Waren, die er ein- und verkauft. Die Rolle ist fester Bestandteil des Agenten und ist zur Laufzeit nicht änderbar.

Sein Charakter bestimmt sein Verhalten auf eingehende Nachrichten, die Preiskalkulation für zu verkaufende Waren und das Einkaufverhalten. Der Charakter ist modular in dem Agenten integriert und kann nach belieben zur Laufzeit ausgetauscht werden.

Das Verbindungsmodul verschachtelt die technische Komponente der Kommunikation nach draußen. Der Charakter gibt seine Nachrichten an das Verbindungsmodul und erhält auch die eingehenden Nachrichten von ihm. Das Verbindungsmodul sendet und empfängt Nachrichten je nach implementierter Technik. Außerdem führt es auch die Aufrufe von Methoden entfernter Objekten, wie AMS oder Proposal Broker, durch. Das Verbindungsmodul ist wie der Charakter zur Laufzeit austauschbar.[2]

Durch die lose Kopplung der Komponenten entsteht ein flexibler Agent.

Der Agent an sich dient nur als Behälter für Felder, wie zum Beispiel: Kontostand, seine Rolle oder sein eindeutiger Bezeichner. Die beiden Module (Charakter und Verbindung) wissen nichts voneinander und agieren immer nur mit dem Agenten. Deshalb werden auch beim Verbindungsmodul einkommende Nachrichten an den Agenten geleitet, der diese an den Charakter weiterleitet, um charakterspezifisch zu reagieren. Das Verbindungsmodul und der

[2] Module zur Laufzeit auszutauschen ist keine Voraussetzung, brachte aber kaum Mehraufwand in der Realisierung und wurde deshalb als Feature beibehalten.

Charakter agieren in gewisser Form selbst als Agenten in der Hülle des eigentlichen Agenten. In der jetzigen Konstellation ist der Charakter sehr maßgebend und aktiv. Er leitet fast den gesamt Ablauf aller Handlungen. Diese Verteilung der Aktivität hätte auch im Agenten angesiedelt werden können, der dann einzelne Handlungen des Charakters abruft.

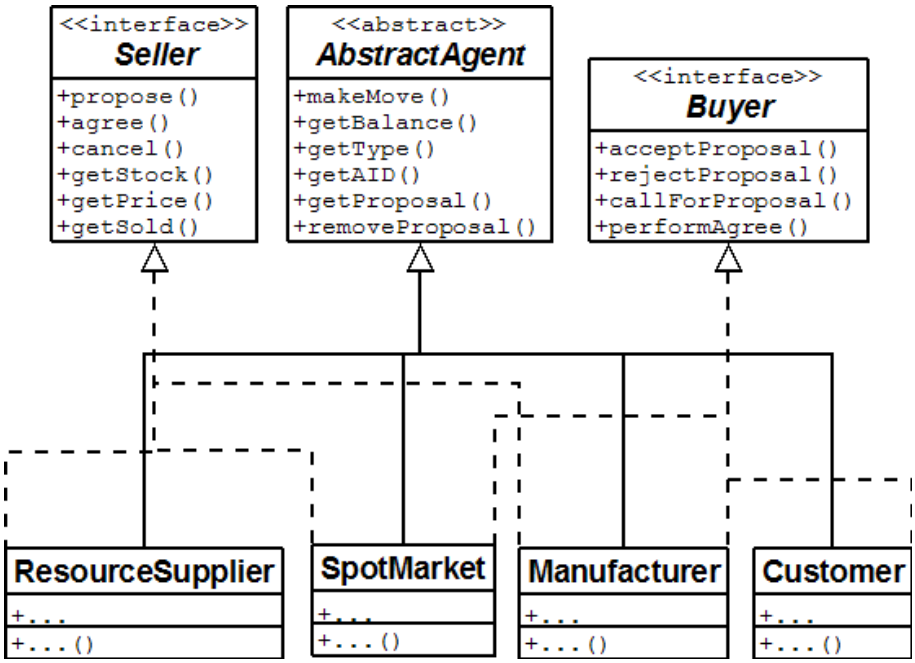


Abbildung 4.11.: UML-Klassendiagramm: Vererbung der Agenten

Ein ausführlicheres UML-Diagramm findet sich im Anhang A.

4.3.3. Agenten-Kommunikation

Allgemein zur Kommunikation lässt sich sagen, dass das Handeln zwischen den Agenten, abgesehen vom schwarzen Brett, von Punkt zu Punkt läuft. D. h. die Agenten adressieren mit ihren Nachrichten direkt den Agenten, mit dem sie kommunizieren. Aus Ressourcen-Gründen werden die Nachrichten aber über den ACC versendet.

Technische Komponente

Die Agenten sollen auch in der Lage sein, sich in einem Netzwerk zu verständigen. Um Nachrichten zwischen den Agenten auszutauschen standen in dieser Arbeit deshalb folgende Möglichkeiten zur Auswahl:

- XML[3]-formatierte (oder anders formatierte) Strings über TCP-[4] oder UDP[5]-Sockets versenden
- Entfernte Methodenaufrufe wie RMI[6] oder CORBA[7] verwenden

Bei RMI und CORBA werden Objekte an einer Registry oder ORB[8] angemeldet. Über diese Objekte können entfernte Methodenaufrufe stattfinden. Soll der Austausch von formatierten Strings stattfinden, müssen Protokolle definiert werden. Die Kommunikation soll möglichst einfach sein. Weil Sockets schnell und einfach auf- und abgebaut werden können, ist das Senden von Strings über Sockets am zweckmäßigsten. Die Wahl zwischen TCP und UDP fällt auf das etwas langsamere, aber dafür für Paketverlust nicht so anfällige TCP. XML scheint für die kurzen Nachrichten zu schwergewichtig. Hier wird eine eigene Formatierung gewählt.

Nachrichtenformat

Die Nachrichten orientieren sich an den definierten Sprechakten aus Kapitel 3.4.4. Neben einer Objekt-Form werden die Nachrichten auch als Zeichenkette dargestellt. Das Format dieser Zeichenkette wird hier in der Backus-Naur-Form beschrieben.

[3] Extensible Markup Language
[4] Transmission Control Protocol
[5] User Datagram Protocol
[6] Remote Method Invokation
[7] Common Object Request Broker Architecture
[8] Object Request Broker

<nachricht>	::=	<angebot anfordern> | <angebot machen> | <angebot annehmen> | <zustimmen> | <abbrechen>
<angebot anfordern>	::=	CALL_FOR_PROPOSAL;<ware>
<angebot machen>	::=	PROPOSE;<ware>;<menge>;<stückpreis>;<bezugsnummer>;<anbieter>
<angebot annehmen>	::=	ACCEPT_PROPOSAL;<bezug>;<bezugsnummer>;<menge>;<absender>
<zustimmen>	::=	AGREE;<bezug>;<menge>
<abbrechen>	::=	CANCEL;<bezug>
<ware>	::=	unit:<waren>
<menge>	::=	quantity:<ziffern>
<stückpreis>	::=	price:<ziffern>.<ziffern>
<bezugsnummer>	::=	replyto:<ziffern>
<anbieter>	::=	offerer:<rolle><ziffern>
<bezug>	::=	inreplyro:<ziffern>
<absender>	::=	<rolle><ziffern>
<rolle>	::=	SPOT_MARKET | MANUFACTURER | CUSTOMER
<waren>	::=	RESOURCE | PRODUCT
<ziffern>	::=	<ziffern><ziffern> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

4.3.4. Fitnessfunktion

In der Fitnessfunktion ist die Zielsetzung des genetischen Lernverfahrens definiert. Jedes Genom wird von der Fitnessfunktion bewertet. Ist die Zielsetzung nicht, wie erforderlich, sorgfältig in der Fitnessfunktion kodiert, ist der Erfolg des Lernverfahrens gefährdet.

In diesem Szenario ist das Ziel eine Zusammenstellung von Rollen und Charakteren zu finden, in der jeder Agent möglichst viel Gewinn macht und sich der Gesamtgewinn trotzdem gleichmäßig verteilt. Wie bei jedem genetischen Lernverfahren sollte am Ende der Suche im Lösungsraum das globale Optimum stehen.

Um dies zu erreichen wird zunächst der Gewinn g jedes Agenten (Genes) a, mit Ausnahme der Endkunden, addiert (g_{gesamt}). Die Gewinn eines einzelnen Agenten ist dabei sein Startguthaben ($start()$) abgezogen vom aktuellen Kontostand ($konto()$) geteilt durch die Anzahl der in der VM gelaufenen Runden ($runden$). Es wird durch die Anzahl der Runden geteilt, um eine Vergleichbarkeit zwischen Durchläufen mit unterschiedlichen Rundenzahlen zu erreichen.

$$g_a = \frac{konto(a) - start(a)}{runden}$$

$$g_{gesamt} = \sum_{a=1}^{agenten} g_a$$

Um den Wert g_{gesamt} von der Beeinflussung der Anzahl der Agenten (die Genomgröße) ($agenten$) zu befreien, wird dieser Wert durch die Anzahl der Agenten geteilt.

$$g_{geteilt} = \frac{g_{gesamt}}{agenten}$$

In $g_{geteilt}$ fehlt noch die Berücksichtigung, wie ausgeglichen der Gewinn unter den Agenten ist. Hierzu wird zunächst die Differenz $diff$ des Gewinns zwischen dem besten g_{max} und den schlechtesten g_{min} ermittelt:

$$diff = g_{max} - g_{min}$$

Aus der Differenz $diff$ wird eine durchschnittliche Differenz ($diff_{geteilt}$) pro Agent ermittelt:

$$diff_{geteilt} = \frac{diff}{agenten}$$

Die Fitness des Genoms $genom$ ergibt sich nun aus dem durchschnittlichen Differenz $diff_geteilt$ subtrahiert von dem durchschnittlichen Gewinn $g_{geteilt}$ seiner Gene (Agenten):

$$fitness(genom) = g_{geteilt} - diff_{geteilt}$$

Es wurde versucht, jede Beeinflussung jeglicher Quantität aus der Bewertung der Fitness zu entfernen, in der Hoffnung für möglichst viele Genome, die sich auch in der Größe und evtl. sogar in der Anzahl der durchlaufenen Runden unterscheiden, vergleichbare Werte zu erzielen.

Die Fitnessfunktion kann so angepasst oder ausgetauscht werden, um Formen sozialer Gerechtigkeit aus der Volkswirtschaftslehre abzubilden. Dies ist aber nicht Bestandteil dieser Arbeit. Es wird nur auf das Potential der Fitnessfunktion hingewiesen.

4.4. Protokollierung

Die Protokollierung muss zwei Ansprüchen genügen: Zum Einen muss sie so genau sein, dass auch Fehler im Detail über das Protokoll gefunden werden können und zum Anderen muss der Verlauf des genetischen Lernverfahrens aus dem Protokoll ohne großen Aufwand ersichtlich sein. Um diesen Ansprüchen gerecht zu werden wurde das Protokoll dreigeteilt:

Agenten-Protokoll: Beinhaltet die Aktivitäten der Agenten drin. Zum Beispiel: Wie der Preis angepasst wurde oder welche Nachrichten sie verschickt und erhalten haben.
Beispieleintrag:
INFO: RESOURCE_SUPPLIER4: Aktueller Preis: 11.025 (10.5)

Agenten-Plattform-Protokoll: (De-)Registrierungsvorgänge am AMS und das veröffentlichen, bzw. zurückziehen, von Angeboten am Proposal Broker werden hier Protokolliert.
Beispieleintrag:
INFO: AMS: SPOT_MARKET7 wurde als SPOT_MARKET unter 169.254.178.160:2172 registriert.

Genetischer-Optimierer-Protokoll: Hier wird protokolliert, welche Generation mit welchen Genomen welche Fitness erzielt hat.
Beispieleintrag:
INFO: Genetic Optimizer: Starte Generation 1
...
INFO: Population: Beste Fitness: 48.68

Es wurde der im JDK[9] von [Sun] enthaltene Logger verwendet. Als Alternative stand noch das sehr mächtige Log4j zur Verfügung. Der Standard-Logger des JDK liefert alle benötigte Funktionalität und so wurde Log4j, wegen seines Konfigurationsaufwandes nicht verwendet.

[9] Java Development Kit

4.5. Entwicklung

In diesem Abschnitt soll kurz erläutert werden, welche Programmiersprache und Entwicklungsumgebung ausgewählt und warum ausgerechnet diese verwendet wurden.

4.5.1. Programmiersprache

Entwickelt wurde in der objektorientierten Programmiersprache [Java] von [Sun]. Es wurde das aktuelle JDK in der Version 1.6.0 verwendet. Java wurde vor allem aus folgenden Gründen gewählt:

- Es ist plattformunabhängig. Java-Bytecode kann auf Windows, Linux, Mac OS X und Solaris ausgeführt werden. So kann die Applikation auch in heterogenen Netzwerken ohne großen Aufwand verwendet werden.
- Es bietet mit seinem Exception[10]-Konzept eine komfortable Fehlerbehandlung und dank des jeweils von der Exception mitgelieferten Stacktraces auch eine schnelle Fehlerbehebung.
- Die im JDK enthaltene I/O[11]-Funktionen reichen vollkommen für die Realisierung der Kommunikation in dieser Applikation.

4.5.2. Entwicklungsumgebung

Bei der Wahl der Entwicklungsumgebung (IDE[12]) fiel die Wahl auf [Eclipse] in der Version 3.2. Eclipse ist eine ausgereifte Open-Source-Entwicklungsumgebung. Hauptsächlich wird Eclipse für die Entwicklung von Java-Programmen benutzt. Es besitzt aber eine plugin-basierte Struktur und kann somit mit entsprechenden Plugins auch für andere Programmiersprachen verwendet werden.

Eclipse sticht besonders durch seine große Anzahl möglicher Konfigurationsmöglichkeiten in den Bereichen Compiler-Einstellungen, Code-Formatierung und GUI-Anpassung hervor. Außerdem bietet Eclipse ein ausgereiftes Hot Code Replacement, das den Austausch von Quellcode während der Laufzeit ermöglicht. Dies ist beim Debuggen[13] äußerst hilfreich.

[10] Ausnahme
[11] Input/Output
[12] Integrated Development Environment
[13] Fehlerbehebung

Als Alternative zu Eclipse stand noch die ebenfalls kostenlose Open-Source-Entwicklungsumgebung [NetBeans] von Sun Microsystems zur Auswahl. Diese Entwicklungsumgebung ist ebenfalls bestens für Java geeignet und bietet auch die Erweiterung durch Plugins an. Es hätten beide Entwicklungsumgebungen verwendet werden können. Aus persönlicher Präferenz fiel dann die Wahl aber auf Eclipse.

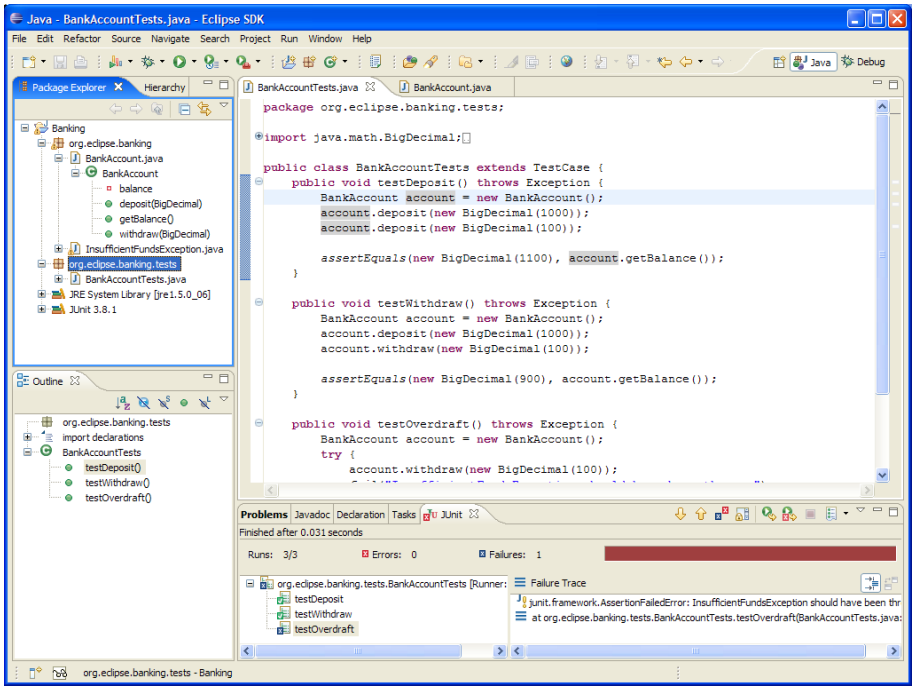


Abbildung 4.12.: Eclipse IDE

4.6. Weitere Realisierungsaspekte

Das Programm hat einen Umfang von 6177 Lines of Code und 157 KB[14] erreicht. Es befindet sich in einem stabilen Stadium und kann als Framework verwendet werden.
Die Ausführung auf einem aktuellen Rechner[15] dauert bei 30 Generationen mit einer Populationsgröße von 20, einer Genomgröße von 15 und über 50 Schritte etwa eine Stunde und 45 Minuten.

Während der Realisierung traten einige Probleme auf. Vor allem die Kommunikation lief nicht reibungslos. Ursprünglich war geplant, die Kommunikation ohne ACC direkt zwischen den Agenten ablaufen zu lassen. Weil nicht jeder Agent eine permanente Socketverbindung zu den anderen Agenten haben sollte, war angedacht jeweils beim Senden einer Nachricht einen Socket aufzubauen und nach dem Senden wieder abzubauen.
Dieses Konzept wurde schnell wieder verworfen, als es einem ersten praktischen Test unterzogen wurde. Zum einen war es recht langsam, aber noch ungünstiger war, dass es regelmäßig Fehler beim Verbindungsaufbau[16] gab.

Ein weiteres Problem trat durch das parallele Bearbeiten von eingehenden Nachrichten auf. Bei eingehenden Nachrichten passiert es unter Anderem, dass sich die Angebote ändern. Dies ist der Fall, wenn ein Kauf erfolgreich abgeschlossen wurde. Es wird dann sofort ein neues Angebot veröffentlicht und das alte wird ungültig. Dies führte dazu, dass die Käufer eine Ablehnung erhielten und der Verkäufer in dieser Runde nichts verkaufte.
Die folgende Rolle in der Handelskette durfte also erst dann kaufen, wenn die vorherige alle Käufe abgeschlossen hatte. Dies wurde vorläufig über eine Zeitpause gelöst. Die machte aber das System unnötig langsam und wurde durch ein Status-Flag ersetzt. Erst wenn alle Agenten einer Rolle ihr Status-Flag gesetzt hatten, führten die Agenten der folgenden Rolle ihren Schritt aus.

Eine kleineres Problem trat beim Warenfluss auf. Der sollte strikt zwischen den benachbarten Gliedern der Handelskette ablaufen. Weil aber Rohstofflieferant und Spotmarkt Angebote mit Rohstoffen als Ware aushingen, kauften die Hersteller von Rohstofflieferanten. Dies wurde unterbunden, indem die Ware Rohstoff in zwei neue Waren geteilt wurde: Rohstoff_Lieferant und Rohstoff_Spotmarkt. Spotmärkte kauften nun Rohstoff_Lieferant und verkauften Rohstoff_Spotmarkt. Hersteller kauften nun Rohstoff_Spotmarkt.

[14] Kilobyte
[15] AMD 64 3000+ mit 1 Gigabyte DDR400-Arbeitsspeicher unter Microsoft Windows XP SP2
[16] z.B. Port schon belegt

5. Evaluation

In der Evaluation werden mehrere Versuchsdurchläufe stattfinden. Weil die Zusammenhänge des Systems so zahlreich und komplex sind, ist eine grafische Benutzeroberfläche notwendig. Zusammen mit dem bisherigen System ergibt dies den Simulator.

5.1. Simulator

Als Simulator versteht sich die grafische Benutzeroberfläche (GUI) im Zusammenspiel mit der VM und der Evolutionsmaschine. Die GUI ist für das Verständnis des Zusammenspiels des gesamten Systems unerlässlich. Schon alleine die Vorgänge in der VM sind so zahlreich und verstrickt, dass dies über Textausgaben nicht nachvollziehbar wäre. Es musste eine Darstellung her, die den Zustand des Ensembles/Genoms klar und deutlich visualisiert.

Die Komponenten GUI und VM sind zeitgleich entstanden. Die GUI war in den ersten Versionen ein Werkzeug zur Überprüfung der Funktionalität der VM. In der Version 0.1 wurden hart codiert Agenten in die VM geladen, die dann jeweils über das Drücken eines Knopfes einen Schritt ausführten. Über den Zustand der einzelnen Agenten war in der GUI nichts zu sehen. In diesem Stadium wurde noch viel im Debugger und über Konsolenausgaben gearbeitet.

Abbildung 5.1.: Simulator v0.1

In der Version 0.2 wurden die Agenten bereits über ein Menü hinzufügt und entfernt. Des weiteren konnten über das Menü *Ausführen* alle Agenten eine beliebige Anzahl von Schritten laufengelassen werden. Durch das Ausführen über mehrere Schritte wurde auch ein Fortschrittsbalken notwendig. Um schneller einen Eindruck über den Zustand eines Agenten zu bekommen, wurden sie grün, gelb oder rot eingefärbt.

grün: $Kontostand > 0$

gelb: $Kontostand \leq 0$ & $Lagerbestand > 0$

rot: $Kontostand \leq 0$ & $Lagerbestand = 0$

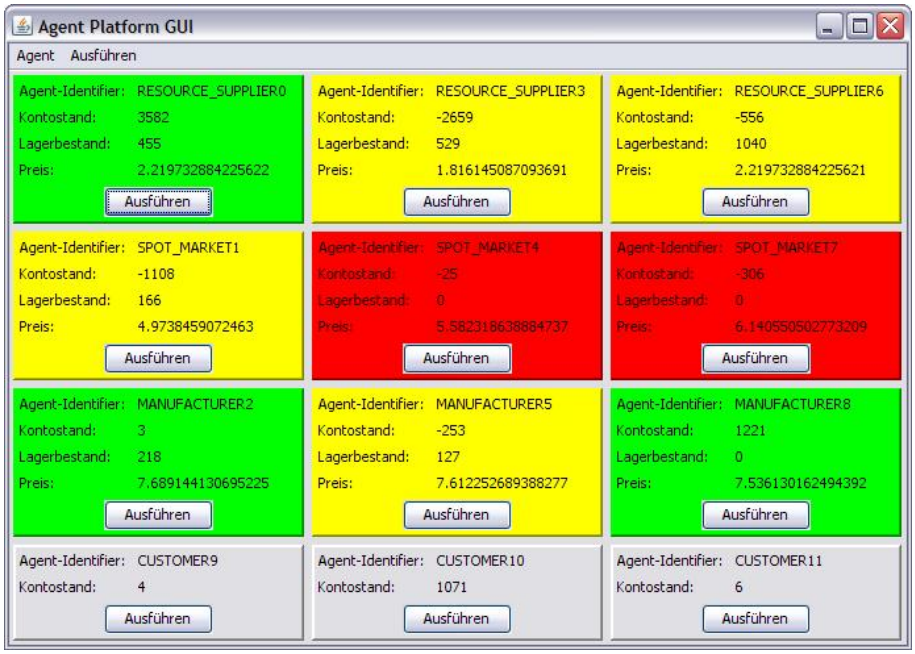


Abbildung 5.2.: Simulator v0.2

In Version 1.0 ist eine Evolutionsmaschine und ein Diagramm hinzugekommen. Das Diagramm bildet auf der X-Achse die Generationen, von Null beginnend, und auf der Y-Achse die beste Fitness der Generation ab. Außerdem wurden zwei Fortschrittsbalken hinzugefügt. Einer für den aktuellen Fortschritt innerhalb der Generation und einer für den Fortschritt des gesamten Lernverfahrens. Der Fortschritt des Lernverfahrens ist bekannt, weil die Evolutionsmaschine immer eine vordefinierte Anzahl von Generationen läuft.

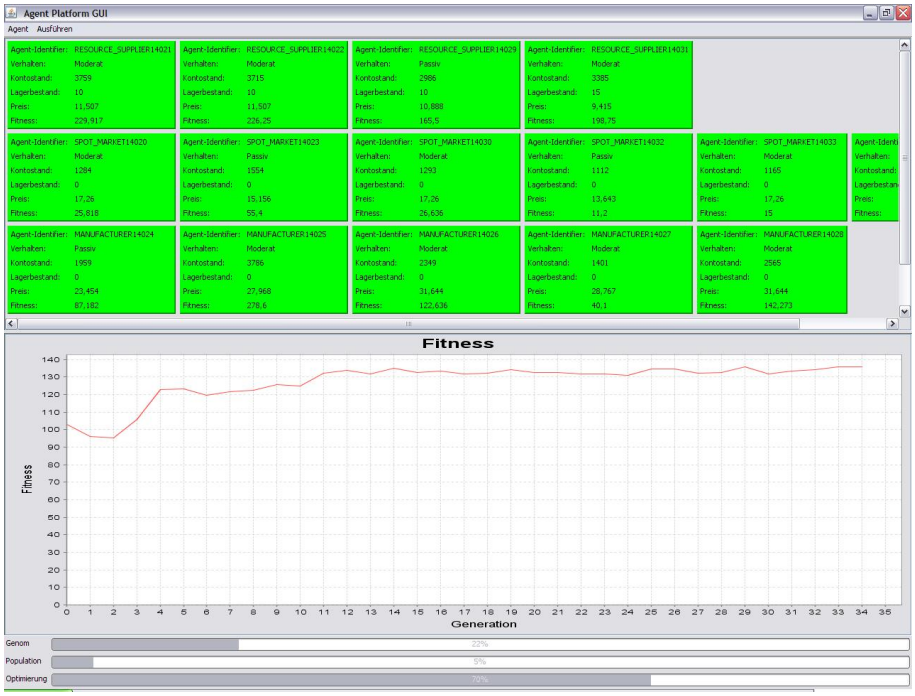


Abbildung 5.3.: Simulator v1.0

Nachdem nun klar war, wie sich die Fitness nach oben entwickelt, stellte sich die Frage, wie sich die Fitness ganz unten entwickelt. Mit der schlechtesten Fitness zusammen lässt sich die Bandbreite der Qualität erahnen. Deshalb wurde in Version 1.1 das Diagramm um die jeweils schlechteste Fitness einer Generation erweitert (blaue Linie).

Abbildung 5.4.: Simulator v1.1

Insgesamt wären mehr statistische Werkzeuge über die Verteilung der Qualität innerhalb einer Population wünschenswert gewesen. Auch ein Stammbaum für Genome wäre sicherlich interessant. So dass man verfolgen kann, wie das beste Genom entstanden ist und sich entwickelt hat. Diese Funktionen hätten aber den Rahmen dieser Arbeit gesprengt.

5.2. Versuchsläufe

Die Ergebnisse der Versuche werden Aufschluss geben, ob das Ergebnis aus Analyse, Design und Realisierung die angestrebte Funktion erfüllt.
Es sollte nun ein System zur Verfügung stehen, mit dem Zusammenstellungen von Komponenten nach bestimmten Kriterien optimiert werden können. Der Kontext der Komponenten und der Zusammenstellung wurde in Kapitel 3.1 beschrieben und die Kriterien befinden sich in Kapitel 4.3.4.

Jeder Versuch besteht aus den folgenden vier Teilen:

Aufbau: Im Aufbau werden kurz die Einstellungen der Evolutionsmaschine (Genomgröße, Populationsgröße, Anzahl der Generationen, etc.) und des genetischen Optimierers (Mutationswahrscheinlichkeit, Größe der Schöpfkelle, etc.) beschrieben.

Motivation & Erwartung: In der Motivation & Erwartung wird beschrieben, was untersucht werden soll und womit als Ergebnis gerechnet wird.

Ergebnis: Im Ergebnis werden die besten und schlechtesten Fitnesswerte der ersten und letzten Generation dargestellt. Hinzu kommt das Diagramm aus dem Simulator, auf dem der Verlauf der besten und schlechtesten Fitness je Generation zu verfolgen ist. Es wird außerdem in einer Tabelle die Zusammenstellung des besten Genoms dargestellt.

Fazit: Im Fazit werden die Ergebnisse analysiert.

5.2.1. Versuch 1

Aufbau

Evolutionsmaschine		**Genetischer Optimierer**	
Populationsgröße:	20	Elitismus:	10 %
Genomgröße:	15	Heiratsschema:	20 %
Endkunden:	5	Schöpfkellengröße:	3
Runden:	50	Mutationswahrscheinlichkeit je Gen:	30 %
Generationen:	30		

Motivation & Erwartung

Für den ersten Versuch wurden Werte gewählt die intuitiv angemessen erscheinen. Untersucht werden soll erstmal das allgemeine Verhalten.
Erwartet wird durch den Elitismus bedingt eine ständige Anhebung der Kurve der besten Fitness. Es lässt sich vermuten, dass sich der Wert der schlechtesten Fitness insgesamt auch verbessert.

Ergebnis

	Generation 0	**Generation 30**
Beste Fitness	55,2	87,6
Schlechteste Fitness	-169,4	-16,1

	passiv	moderat	aggressiv
Rohstofflieferant	5	0	0
Spotmarkt	3	1	1
Hersteller	3	2	0

Abbildung 5.5.: Diagramm Versuch 1

Fazit

Die beste Fitness hat sich in den ersten 15 Generationen fast konstant verbessert. Danach pendelte sich die beste Fitness auf diesem Niveau, um die 90, ein. Das die beste Fitness trotz Elitismus absackt, ist auf das nicht-deterministische Verhältnis zwischen Start- und Endzustand der Genome zu schließen.

Die schlechteste Fitness verhält sich sehr sprunghaft. Dies zeigt, dass die Änderungen an den Genomen, die vom genetischen Optimierer durchgeführt werden, nicht vorhersehbare Auswirkungen zur Folge haben können. Das Endergebnis der schlechtesten Fitness ist nicht aussagekräftig, da es schon in der nächsten Generation komplett anders sein könnte.

In den nächsten Versuchen wird beobachtet, ob die beste Fitness sich immer so schnell einpendelt. Falls dies der Fall ist, kann die Anzahl der Generationen reduziert werden.

5.2.2. Versuch 2

Aufbau

Evolutionsmaschine		**Genetischer Optimierer**	
Populationsgröße:	20	Elitismus:	10 %
Genomgröße:	15	Heiratsschema:	20 %
Endkunden:	5	Schöpfkellengröße:	3
Runden:	10	Mutationswahrscheinlichkeit je Gen:	30 %
Generationen:	30		

Motivation & Erwartung

In diesem Versuch soll untersucht werden, wie sich das System mit einer deutlich niedrigeren Rundenanzahl von 10 pro Genom verhält. Insbesondere wird überprüft, ob die angestrebte Vergleichbarkeit der Fitnesswerte von Genome mit unterschiedlich vielen durchlaufenen Runden, gewährleistet ist. Die Fitnesswerte der besten Genome sollten sich also auch in einem Bereich von etwa 90 einpendeln.

Ergebnis

	Generation 0	**Generation 30**
Beste Fitness	88,7	102,9
Schlechteste Fitness	-152,4	-27,8

	passiv	moderat	aggressiv
Rohstofflieferant	5	0	0
Spotmarkt	3	1	1
Hersteller	3	2	0

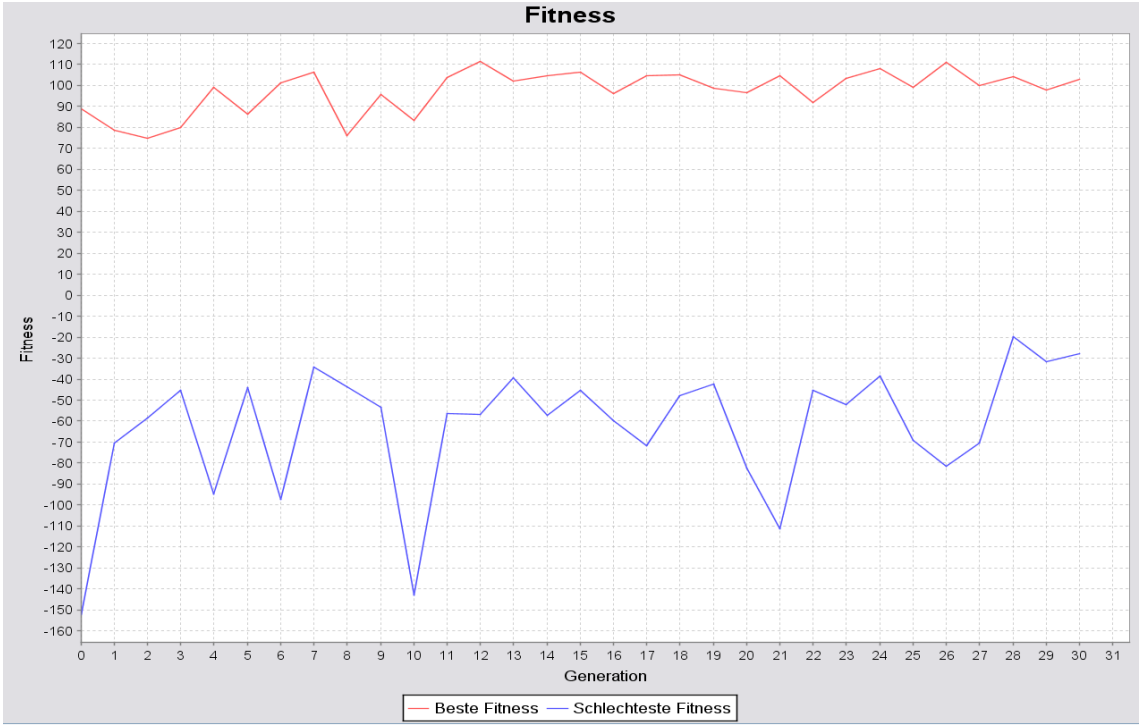


Abbildung 5.6.: Diagramm Versuch 2

Fazit

Die beste Fitness hat sich in einem Bereich von 100 bis 110 eingependelt und liegt damit zu hoch, als das man von einer direkten Vergleichbarkeit reden kann.
Auffallend sind die wesentlich größeren Unregelmäßigkeiten in der Linie der besten Fitness im Vergleich zum ersten Versuch. Vermutlich findet das Ensemble bei einer so einer kurzen Anzahl von Runden noch nicht sein Gleichgewicht.
Die Kurve der schlechtesten Fitness ist wieder sehr sprunghaft, hat aber diesmal aber eine leichte Tendenz nach oben.

5.2.3. Versuch 3

Aufbau

Evolutionsmaschine		**Genetischer Optimierier**	
Populationsgröße:	20	Elitismus:	10 %
Genomgröße:	30	Heiratsschema:	20 %
Endkunden:	10	Schöpfkellengröße:	3
Runden:	50	Mutationswahrscheinlichkeit je Gen:	30 %
Generationen:	30		

Motivation & Erwartung

Dieser Versuch unterscheidet sich vom Aufbau im Vergleich zum ersten Versuch durch die Genomgröße und die Anzahl der Endkunden. Sie wurden verdoppelt. Dies soll zeigen, ob die Fitness ungeachtet der Genomgröße, bei gleichem Verhältnis der Endkunden, vergleichbar ist.
Hier sollte sich die Fitness bei 90 einpendeln.

Ergebnis

	Generation 0	**Generation 30**
Beste Fitness	33,3	90,6
Schlechteste Fitness	-86,3	-53,9

	passiv	moderat	aggressiv
Rohstofflieferant	5	0	0
Spotmarkt	6	2	2
Hersteller	8	2	3

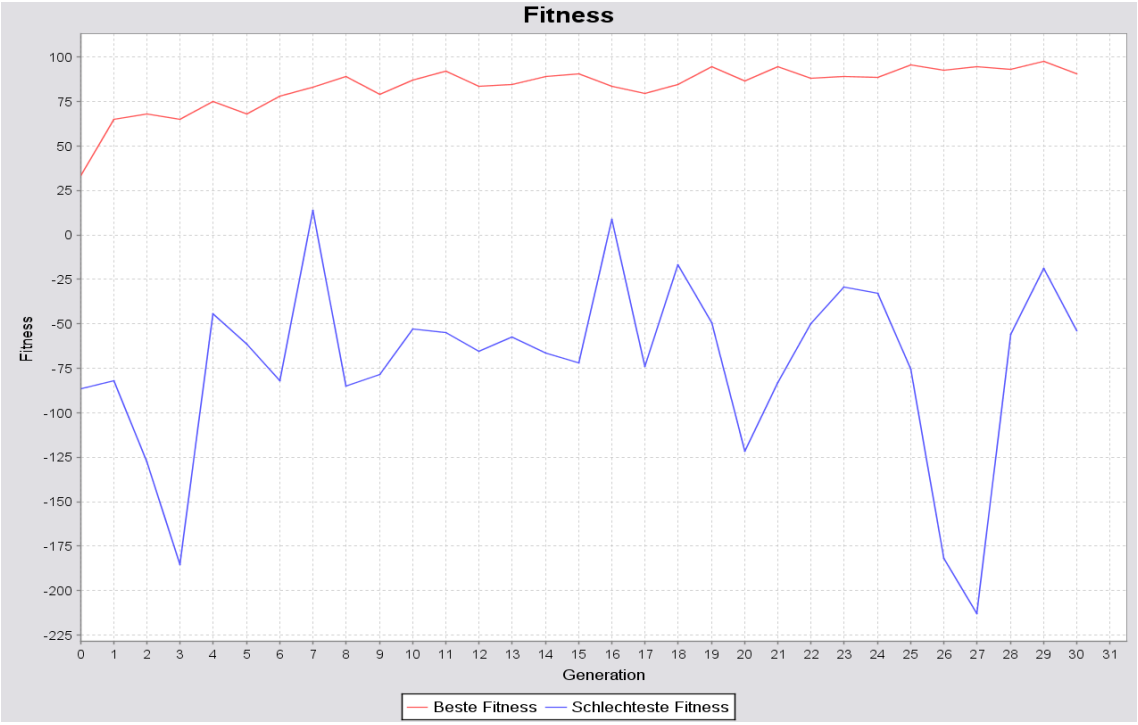


Abbildung 5.7.: Diagramm Versuch 3

Fazit

Die beste Fitness hat sich im gleichen Bereich eingependelt, wie im ersten Versuch. Es kann also von einer Vergleichbarkeit unabhängig von der Genomgröße geredet werden. Das Verhältnis Genomgröße zu der Anzahl der Endkunden sollte aber das gleiche sein.
Ansonsten bringt dieser Versuch keine neuen Erkenntnisse. Die beste Fitness ist wieder, nach einem ersten großen Schub, langsam und schwankend gestiegen. Der große erste Schub lässt auf eine ungünstige erste Population schließen. Der Verlauf der schlechtesten Fitness verdeutlicht, dass der Wert nicht aussagekräftig ist.

5.2.4. Versuch 4

Aufbau

Evolutionsmaschine		**Genetischer Optimierier**	
Populationsgröße:	20	Elitismus:	20 %
Genomgröße:	15	Heiratsschema:	30 %
Endkunden:	5	Schöpfkellengröße:	6
Runden:	50	Mutationswahrscheinlichkeit je Gen:	60 %
Generationen:	30		

Motivation & Erwartung

Dieser Versuch ist von den Evolutionsmaschinenparametern her identisch mit dem ersten, hat aber stark veränderte Parameter des genetischen Optimierers. Es soll festgestellt werden, ob der schwache Lernerfolg auf falsche Parameter des genetischen Optimierers zurückzuführen ist.

Ergebnis

	Generation 0	**Generation 30**
Beste Fitness	56,7	119,72
Schlechteste Fitness	-194,4	-76,0

	passiv	moderat	aggressiv
Rohstofflieferant	3	0	0
Spotmarkt	3	0	0
Hersteller	1	6	2

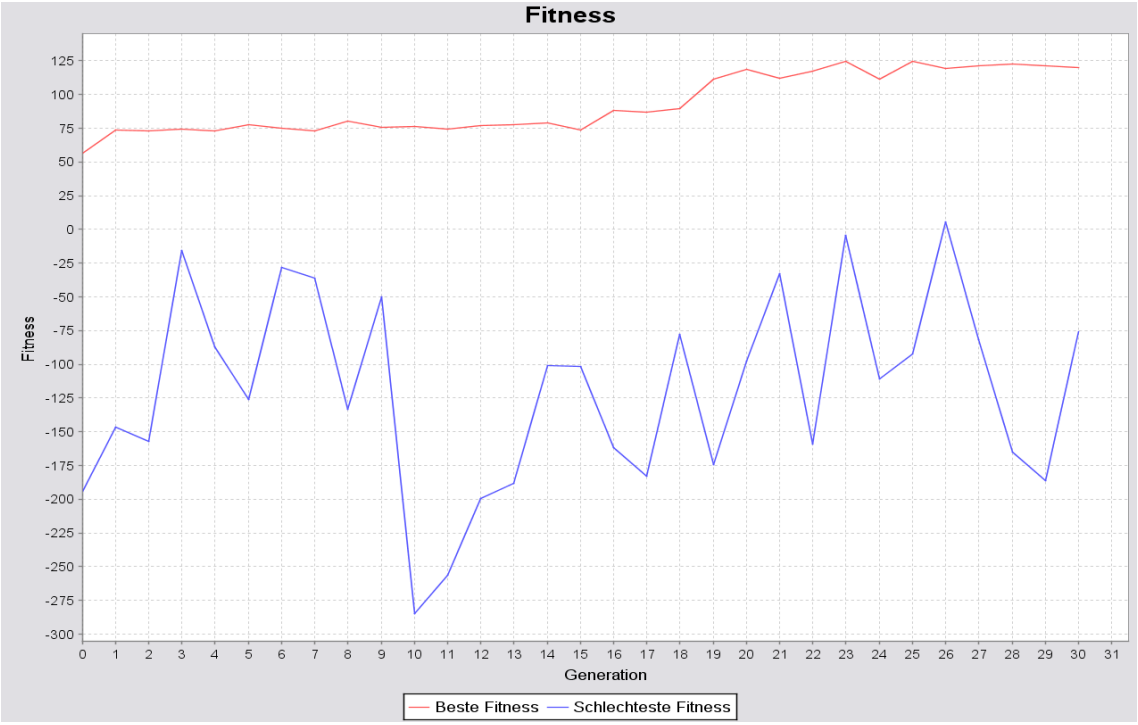


Abbildung 5.8.: Diagramm Versuch 4

Fazit

Die Kurve der besten Fitness zeigt hier einen bemerkenswerten Verlauf. Bis einschließlich der 15. Generation hat die beste Fitness sich um einen Wert von 75 eingependelt. Konnte aber in den folgenden 15 Generation nochmal bis auf knapp 125 steigen.
Bei den Werten um 75 schien es sich um ein lokales Optimum zu handeln, von dem sich aber gelöst werden konnte.
Auch wenn der Lernerfolg hier größer ausfiel, sorgt das anfängliche Stagnieren über 15 Generationen für Skepsis.

5.2.5. Versuch 5

Aufbau

Evolutionsmaschine		**Genetischer Optimierier**	
Populationsgröße:	20	Elitismus:	20 %
Genomgröße:	30	Heiratsschema:	30 %
Endkunden:	10	Schöpfkellengröße:	6
Runden:	100	Mutationswahrscheinlichkeit je Gen:	60 %
Generationen:	50		

Motivation & Erwartung

Nach dem bessern Lernerfolg in Versuch 4 soll nun mit den gleichen Einstellungen für den genetischen Optimierer noch ein großer Versuch gefahren werden. Bis auf die Populationsgröße werden dabei alle Werte der Evolutionsmaschine angehoben. Besonderes die hohe Anzahl an Runden und Generationen dürfte für ein deutlicheres Endergebnis sorgen.

Ergebnis

	Generation 0	**Generation 30**
Beste Fitness	89,3	110,1
Schlechteste Fitness	-88,7	0,4

	passiv	moderat	aggressiv
Rohstofflieferant	6	0	1
Spotmarkt	7	2	5
Hersteller	5	4	0

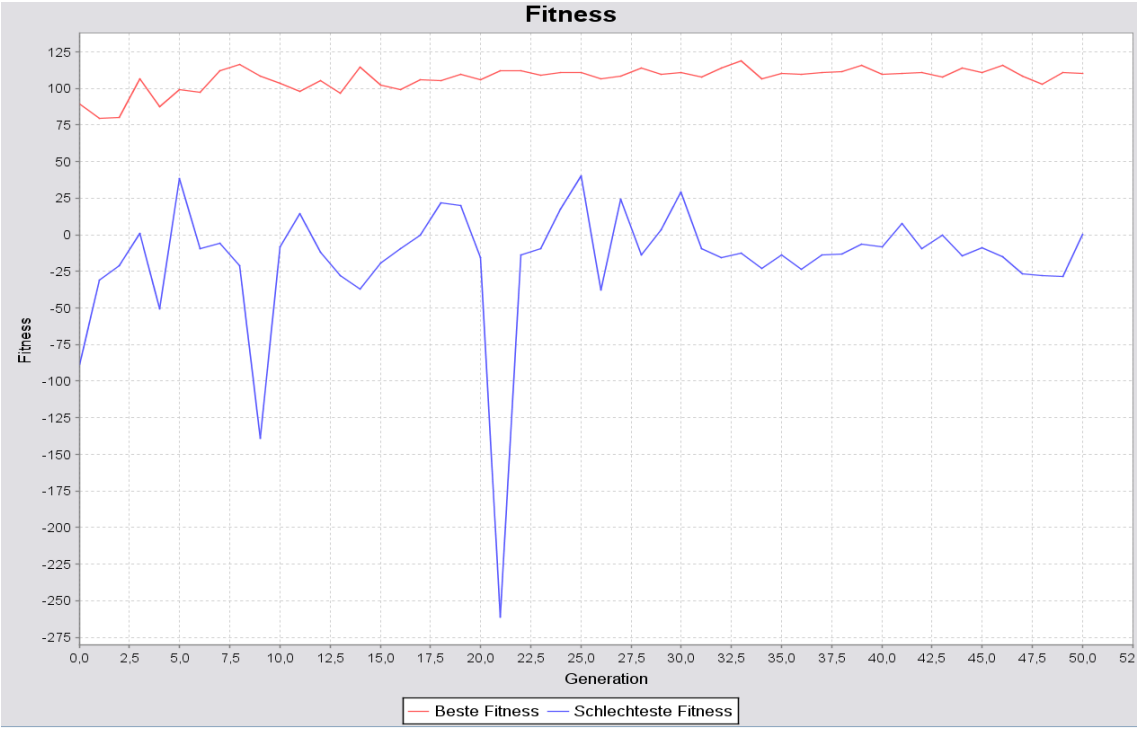


Abbildung 5.9.: Diagramm Versuch 5

Fazit

Die Kurve der besten Fitness zeigt nicht den erhofften Lernerfolg. Insgesamt ist kaum von einer Steigung zu sprechen. Die Einstellungen des genetischen Optimierers aus Versuch 4 konnten keine Besserung des Lernerfolgs bewirken.
Der Lernerfolg scheint bei diesem Szenario mit der angewandten Fitnessfunktion allgemein nicht sonderlich hoch zu sein.

5.3. Gesamtfazit der Versuche

Die Frage, ob es möglich ist eine Zusammenstellung mit einem genetischen Lernverfahren zu optimieren, kann an dieser Stelle nicht eindeutig geklärt werden. Die Versuche zeigen zwar die richtigen Tendenzen: So orientiert sich die beste Fitness stetig nach oben, macht aber nur kleine Fortschritte und erleidet auch häufig Rückschläge.

Ein Grund für das langsame Steigen der Fitness kann eine Verzerrung durch die Fitnessfunktion sein. Sie ist so darauf ausgelegt quantitative Einflüsse zu filtern, dass die Unterschiede kleiner erscheinen können, als sie wirklich sind.
Es kann aber auch sein, dass im Lösungsraum so viele gute Lösungen sind, dass schon die besten Lösungen der ersten Population kaum noch eine Steigerung der Fitness zulassen. In diesem Fall sorgt das genetische Lernverfahren für eine Absicherung der Qualität.

Das zwischenzeitliche Absacken der Fitness trotz Elitismus muss auf das nicht-deterministische Verhalten der virtuellen Maschine zurückgeführt werden.

Insgesamt scheint es möglich zu sein Zusammenstellungen mit Lernverfahren zu optimieren. Um dies abschließend beurteilen zu können, müsste die Fitnessfunktion an Hand klarerer und einfacherer Kriterien bewerten. So wäre eine Verzerrung ihrerseits auszuschließen.
Des weiteren müsste das nicht-deterministische Verhalten der virtuellen Maschine aufgehoben werden. Alternativ kann auch eine mehrfache Bewertung der Genome stattfinden, um einen stabileren Qualitätswert zu erhalten.

6. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Multiagentensystem in einem marktwirtschaftlichen Kontext erstellt. Dieses Multiagentensystem mit seinen Akteuren, den Agenten, bildete eine Handelskette aus vier Schichten. Jeder Akteur war entweder Rohstofflieferant, Spotmarkt, Hersteller oder Endkunde. Akteure besitzen außerdem einen Charakter: passiv, moderat oder aggressiv, der das Verhalten steuert. Der ablaufende Handel fand nur zwischen benachbarten Schichten statt. Dieses System diente als ein Ensemble und wurde mit Hilfe von genetischen Lernverfahren versucht zu optimieren. Die Optimierung zielte dabei auf einen maximalen und unter den Akteuren ausgewogenen verteilten Gewinn ab.

In Kapitel 2 (Grundlagen) wurde die Technik genetischer Verfahren an Hand der Vorbilder aus der Biologie erläutert und kurze Definitionen von unterschiedlichen (Multi-)Agentensystemen vorgestellt.
Kapitel 3 (Analyse) brachte umsetzbare konkrete Anforderungen an das zu entwickelnde System hervor. Nach diesem Kapitel stand fest, wie Szenario, Ensemble und Kommunikation auszusehen haben.
Erfolgreich wurden in Kapitel 4 (Design und Realisierung) die Anforderungen aus der Analyse umgesetzt und ein stabiles System zum Durchführen von Testläufen mit einem einfachen marktwirtschaftlichen Szenario geschaffen. Außerdem können die Komponenten als Framework für andere zu optimierende Ensembles genutzt werden.
Im Kapitel 5 (Evaluation) wurden 5 Versuche erfolgreich durchgeführt und lieferten einige Erkenntnisse über das Lernverhalten des entwickelten Systems.

Abschließend lässt sich sagen, dass in dieser Arbeit gezeigt werden konnte, dass Zusammenstellungen, sprich komplexe Parameter-Settings, über genetische Lernverfahren optimiert werden können. Es wurde außerdem in der Evaluation deutlich, dass bestimmte Randbedingungen erfüllt sein sollten, um zuverlässige Ergebnisse zu erzielen.

Das System in seinem jetzigen Zustand kann für ähnliche kleine marktwirtschaftliche Versuchsläufe benutzt werden. Hierfür können, Rollen, Charaktere, Waren und die Fitnessfunktion geändert oder ausgetauscht werden.

Zukünftig könnte das System um bessere statistische Auswertungen erweitert werden. So kann die Qualitätsverteilung innerhalb der Population erfasst werden. Auch ein Stammbaum

des besten Genoms wäre sicherlich eine sinnvolle Erweiterung. Dieser würde dann Abstammung und Änderungen protokollieren.

Literaturverzeichnis

[Anybots] ANYBOTS: *Dexter*. – URL http://www.anybots.com/abouttherobots.html

[Eclipse] ECLIPSE: *Eclipse Foundation*. – URL http://www.eclipse.org/

[Ferber 1999] FERBER, Jacques: *Multi-agent Systems: Introduction to Distributed Artificial Intelligence*. Addison Wesley, 1999

[FIPA] FIPA: *Foundation for Intelligent Physical Agents*. – URL http://www.fipa.org/

[FIPA 2002] FIPA: *FIPA Communicative Act Library Specification*. 12 2002

[Freeman u. a. 2005] FREEMAN, Eric ; FREEMAN, Elisabeth ; SIERRA, Kathy: *Entwurfsmuster von Kopf bis Fuß*. O'Reilly, 2005

[Gerdes u. a. 2004] GERDES, Ingrid ; KLAWONN, Frank ; KRUSE, Rudolf: *Evolutionäre Algorithmen (Computational Intelligence)*. Vieweg, 2004

[Goldberg 1986] GOLDBERG, David: *The Genetic Algorithm Approach: Why, How, and Waht Next?* K. S. Narendra, New York, 1986

[Horstmann und Cornell 2003] HORSTMANN, Cay S. ; CORNELL, Gary: *Core Java 2, Band 1. Grundlagen*. Addison-Wesley, München, 2003

[Horstmann und Cornell 2005] HORSTMANN, Cay S. ; CORNELL, Gary: *Core Java 2, Band 2. Expertenwissen*. Addison-Wesley, München, 2005

[Janning und Knust 2004] JANNING, Wilfried ; KNUST, Elisabeth: *Genetik. Allgemeine Genetik - Molekulare Genetik - Entwicklungsgenetik*. Thieme, Stuttgart, 2004

[Java] JAVA: *Sun Microsystems*. – URL http://java.sun.com/

[Levy 1993] LEVY, Steven: *KL, Künstliches Leben aus dem Computer*. Droemer Knaur, 1993

[Log4j] LOG4J: *Apache Software Foundation*. – URL http://logging.apache.org/log4j/

[Minsky 1988] MINSKY, Marvin: *Society of Mind*. Simon and Schuster, 1988

[NetBeans] NETBEANS: *NetBeans*. – URL http://www.netbeans.org/

[Porschke 2002] PORSCHKE, Jan P.: *Genetische Programmierung von verhaltensbasierten Agenten*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2002

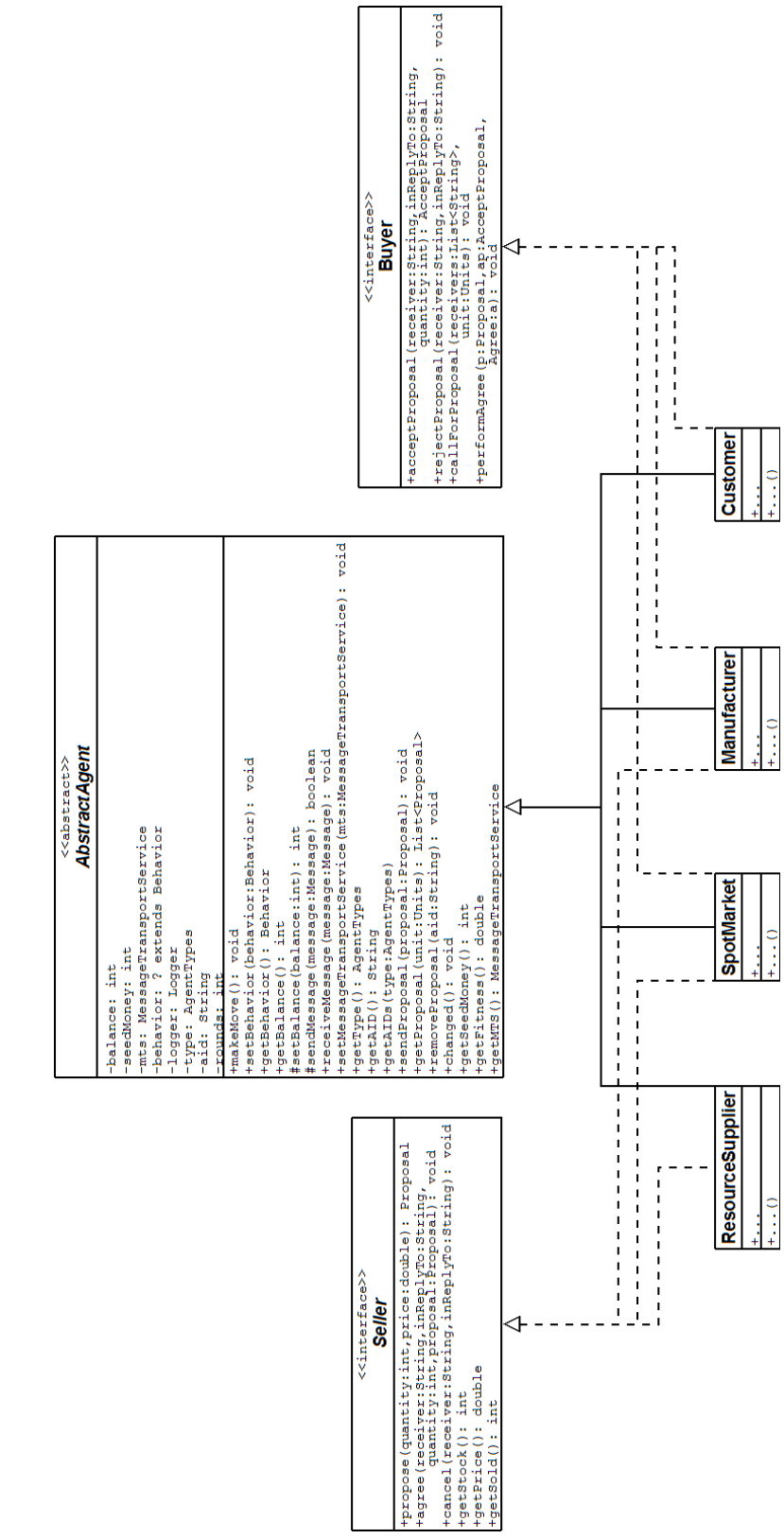
[Sun] SUN: *Sun Microsystems*. – URL http://www.sun.com/

[Tennstedt 2007] TENNSTEDT, Sven: *Genese sozialen Verhaltens in Multiagentensystemen durch genetische Verfahren*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2007

[Wooldridge 2002] WOOLDRIDGE, Michael: *Introduction to MultiAgent Systems*. Wiley, 2002

[YouTube] YOUTUBE: *YouTube, Inc.*. – URL http://www.youtube.com/

A. UML-Diagramm


<<abstract>>
AbstractAgent
-balance: int
-seedMoney: int
-mts: MessageTransportService
-behavior: ? extends Behavior
-logger: Logger
-type: AgentTypes
-aid: String
-rounds: int
+makeMove(): void
+setBehavior(behavior:Behavior): void
+getBehavior(): Behavior
+getBalance(): int
#setBalance(balance:int): int
#sendMessage(message:Message): boolean
+receiveMessage(message:Message): void
+setMessageTransportService(mts:MessageTransportService): void
+getType(): AgentTypes
+getAID(): String
+getAIDs(type:AgentTypes)
+sendProposal(proposal:Proposal): void
+getProposal(unit:Units): List<Proposal>
+removeProposal(aid:String): void
+changed(): void
+getSeedMoney(): int
+getFitness(): double
+getMTS(): MessageTransportService
<<interface>>
Seller
+propose(quantity:int,price:double): Proposal
+agree(receiver:String,inReplyTo:String,
quantity:int,proposal:Proposal): void
+cancel(receiver:String,inReplyTo:String): void
+getStock(): int
+getPrice(): double
+getSold(): int
<<interface>>
Buyer
+acceptProposal(receiver:String,inReplyTo:String,
quantity:int): AcceptProposal
+rejectProposal(receiver:String,inReplyTo:String): void
+callForProposal(receivers:List<String>,
unit:Units): void
+performAgree(p:Proposal,ap:AcceptProposal,
Agree:a): void
ResourceSupplier
+...
+...()
SpotMarket
+...
+...()
Manufacturer
+...
+...()
Customer
+...
+...()

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 11. Juli 2007	
Ort, Datum	Unterschrift