UNIVERSITÄT
KOBLENZ · LANDAU

Entwicklung und Anwendung einer Multi-Agenten-Umgebung zur Simulation des Entstehungsprozesses von Normen

Development and application of a multi-agent-environment to simulate the process of norm-appearance

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers im Studiengang Informatik

vorgelegt von
Frank Klingert
frank.klingert@uni-koblenz.de

Betreuer: Prof. Dr. Klaus G. Troitzsch
Dr. Michael Möhring
(Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich 4)

Koblenz, im Juni 2007

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

(Ort, Datum) (Unterschrift)

Danksagung

Ich möchte mich bedanken

bei Klaus G. Troitzsch und Michael Möhring für die gute Betreuung sowie das große Maß an Freiheit bei der Ausgestaltung meiner Diplomarbeit.

bei Ulf Lotzmann für die fortwährend konstruktive Kritik.

bei Alexander Probst, Andreas Langs, Jürgen Starek, Markus Pinl und Tim Keupen für das Korrekturlesen.

und nicht zuletzt bei meinen Eltern für Ihre unermüdliche Unterstützung vor und während meines Studiums.

Kurzfassung

Sozialwissenschaftler können ihre Annahmen nicht wie Mathematiker oder Wissenschaftler der theoretischen Informatik beweisen. Deshalb ist Simulation umso wichtiger und nützlicher, um Annahmen zu überprüfen. In der Vergangenheit mussten sich Modellierer zwischen einer selbst programmierten Lösung mit Unterstützung von Bibliotheken auf einer logisch niedrigen Ebene, was einen hohen Freiheitsgrad bei der Modellierung erlaubt, und einer logisch höheren, weniger programmiernahen Lösung entscheiden. Weit verbreitete Programme wie NetLogo stehen für die zweite Lösung und akzeptieren einen kleineren Freiheitsgrad sowie die Notwendigkeit, fortgeschrittene Simulationsmodelle an die Bedingungen des Programms anzupassen.

Diese Diplomarbeit analysiert die Voraussetzungen für eine Simulation mit Normen, schlägt einen Coderahmen vor und stellt ein Modellierungs- und Codegenerierungs-Plugin für Eclipse, das auf einem Metamodell für Multi-Agenten-Simulationen basiert, zur Verfügung. Mit Hilfe des Plugins werden zwei Beispiele, die Normen aus verschiedenen Blickwinkeln betrachten, implementiert und diskutiert. Das erste ist das Kirk-Coleman-Modell, welches sich mit dem Verhalten von Menschen in einer 3-Personen-Gruppe beschäftigt. Das zweite Modell beinhaltet den fördernden Effekt von Sanktionierung auf Kooperation. Abschließend werden die erarbeiteten Lösungsansätze und Alternativen diskutiert und Vorschläge für das EU-Projekt „Emergence in the Loop" (EMIL) abgeleitet.

Abstract

Social scientists can not proof their assumptions like mathematicians or theoretical computer scientists. Therefore, simulation is an even more important and valuable way of analysing assumptions than in other sciences. Up to now, modellers had to decide between a self-programmed solution with a low-level support of libraries combined with a high freedom of modelling or a high-level, less programming skills demanding solution. Popular tools like NetLogo choose the last way and accept a smaller degree of freedom and a need to adapt advanced simulation models to the environment those tools offer.

This thesis analyses the needs for a simulation including norms, proposes a code frame and provides a modelling and code generation plugin for Eclipse. This plugin is based on a meta-model for multi-agent simulations in a network. Using this plugin, two examples dealing with different views on norms are implemented and discussed. The first one is the Kirk-Coleman-Model which deals with the behaviour of people in a 3-person-group. The second one is dealing with the advantage of sanctioning in a public goods game. Finally, this thesis provides a discussion of the developed solution and alternatives and makes proposals for the EU-project "Emergence in the Loop" (EMIL).

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Sozialwissenschaftler haben in der Regel nicht die Möglichkeit ihre Annahmen, wie es in der Mathematik oder der Theoretischen Informatik möglich ist, zu beweisen. Sie können ihren Forschungsgegenstand zudem nicht, wie in den Ingenieurwissenschaften, selbst neu konstruieren und auch nicht, wie beispielsweise in den Naturwissenschaften, in nahezu perfekten und wiederholbaren Experimenten testen. Mit Hilfe von Simulation kann der Modellierer ein besseres Verständnis von der Welt erhalten[1]. Die Möglichkeit, Annahmen nicht nur mit der Erstellung eines Modells diskutieren, sondern auch simulieren zu können, stellt also eine interessante und umso wichtigere Möglichkeit sozialwissenschaftlicher Forschung dar.

Normen bestimmen das soziale Zusammenleben der Menschen. So gibt es auf der einen Seite relativ weit verbreitete Normen. „Wort halten" ist zumindest in der westlichen Welt eine wichtige Norm. „Pünktlichkeit" ist eine davon abgeleitete Norm, die insbesondere Menschen aus dem deutschsprachigen Raum zugeschrieben wird. Auf der anderen Seite gibt es auf kleinere Gruppen bezogene Normen, wie die Kleidungsordnung einer Firma oder die Essgewohnheiten innerhalb einer Familie.

Normen entstehen neu, gehen zum Teil in rechtlich bindende Gesetze über und verfallen wieder. Aufgrund der hohen Bedeutung von Normen gibt es zu diesem Thema wissenschaftliche Veröffentlichungen aus verschiedenen Blickwinkeln. Beispiele hierfür sind die am „Homo oeconomicus" orientierte Spieltheorie[2] oder weniger auf monetären Aspekten fußende sozialwissenschaftliche Untersuchungen, wie dem Verhalten in 3-Personen-Gruppen[3].

Der Prozess der Normenentstehung und des Verfalls der Normen ist in zahlreichen Umsetzungen simuliert worden. Beispiele hierfür sind in [Cas98], [Gal05] und zahlreichen weiteren Veröffentlichungen zu finden. Trotzdem gibt es keine einheitliche Umgebung, die die Simulation von Normen unterstützt. So wurden die Modelle meist selbst und von Grund auf, teilweise mit Hilfe von verschiedenen Frameworks, implementiert. Das von der EU geförderte Projekt „Emergence in the Loop" (EMIL) soll hier Abhilfe schaffen, indem zunächst Anforderungen an eine Umgebung zur Simulation von Normen gestellt werden und anschließend eine solche entwickelt wird.

1 *„Simulation is a particular type of modelling. Building a model is a well-recognized way of understanding the world: [..]"* [Gil05, S. 2]
2 Eine Veröffentlichung, die sich mit Normenentstehung durch Sanktionierung beschäftigt, ist [Gür06a].
3 Eine Veröffentlichung zum Verhalten in 3-Personen-Gruppen ist [Kir67].

Im Rahmen dieser Arbeit wird im Hinblick auf EMIL eine prototypische Umgebung für die Simulation von Normen entwickelt, diskutiert und daraus Anforderungen für EMIL abgeleitet. Zur Umgebung gehören ein Modellierungs-Plugin, ein darin integrierter Codegenerator und neben weiteren Bausteinen graphische Darstellungsformen. Ergänzt wird diese um eine Diskussion von Alternativen sowie zwei ausgearbeitete Beispiele.

Diese Arbeit beginnt mit der Einordnung in den Kontext von EMIL, einer Definition verwendeter Begriffe und einer kurzen Einführung der zwei verwendeten Beispiele in Kapitel 1. Es folgt eine Anforderungsanalyse in Kapitel 2. Darauf aufbauend werden in Kapitel 3 möglicherweise hilfreiche Frameworks evaluiert und eine grobe Architektur festgelegt. In Kapitel 4 folgen dann der detaillierte Entwurf und die Umsetzung. Kapitel 5 stellt zwei Simulationen vor, die aus verschiedenen Anwendungsbereichen kommen und sich an der vorher entwickelten Struktur orientieren. Im Kapitel 6 folgt ein Fazit, insbesondere mit Blick auf EMIL.

1.1 Einordnung in das Projekt EMIL

Das von der EU geförderte Projekt „Emergence in the Loop: Simulating the two way dynamics of norm innovation" (EMIL) (siehe [Con05]) ist zunächst auf den Zeitraum September 2006 bis August 2009 angelegt. An dem Projekt sind folgende Einrichtungen beteiligt:

- Institute for Cognitive Science and Technology, CNR
- Universität Bayreuth, Departement of Philosophy
- University of Surrey, Centre for Research on Social Simulation
- University of Koblenz-Landau, Departement of Computer Science
- Manchester Metropolitan University, Centre for Policy Modelling
- AITIA International Inc.

EMIL besteht aus folgenden Aufgabenpaketen:

- EMIL-M hat die Erforschung der Theorie zum Ziel. Dazu gehören sowohl ein allgemein gültiges Modell zur Normen-Anpassung als auch das Bereitstellen von empirischen Beispielen.
- EMIL-S hat die Erstellung eines Simulators zum Ziel. Dieser wird Vorlagen für verschiedene Arten von Agenten und deren Umwelt enthalten. Mit ihm sollen anschließend die Ergebnisse von EMIL-M überprüft werden.
- EMIL-T hat die Evaluation der vorherigen Ergebnisse zum Ziel. Dabei wird die erarbeitete Theorie mit den Ergebnissen der Simulation sowie den empirischen Daten verglichen.

Die vorliegende Arbeit hat den Zweck, erste Erkenntnisse über das Projekt zu gewinnen. Hierbei liegt der Fokus insbesondere auf dem Aufgabenpaket „EMIL-S", welches unter der Leitung der Arbeitsgruppe „Empirische Methoden, Modellbildung und Simulation" der Universität Koblenz-Landau bearbeitet wird. Die Erstellung eines Programms innerhalb eines Forschungsprojekts ist aufgrund der Einmaligkeit und Neuartigkeit mit schwer kalkulierbaren Risiken verbunden. Um Erkenntnisse über die Entwicklung und die damit verbundenen Risiken zu gewinnen, wird eine Simulationsumgebung entwickelt und mittels einiger Beispiele aus der Literatur getestet. Die Anforderungen an dieses Programm werden, soweit möglich, aus der Projektbeschreibung entnommen und um Annahmen aus der Forschungsgruppe ergänzt. Die Simulationsumgebung dient dem Projekt EMIL als Prototyp, der, falls sinnvoll, zum Teil im Projekt EMIL weiterverwendet wird.

1.2 Terminologie

In diesem Unterkapitel werden wichtige Begriffe im Zusammenhang mit dieser Arbeit erläutert. Zunächst werden jeweils Definitionen sozialwissenschaftlicher Begriffe aus der Literatur zitiert und aus diesen eine übersetzte und zusammenfassende Definition abgeleitet.

Agent

- *„[...] there is **no generally agreed definition** of what an 'agent' is."*[4] [Gil05]
- *„Essentially, while there is a general consensus that **autonomy is central** to the notion of agency, there is **little agreement beyond** this."* [Wool02]
- *„Perhaps the most general way in which the term agent is used is to denote a hardware or (more usually) software-based computer system that enjoys the following properties: **[A]utonomy** [...], **social ability** [...], **reactivity** [...], **pro-activeness**."* [Wool95]
- Ein Agent ist im Kontext dieser Arbeit eine Software, die autonom ist, mit anderen Agenten interagiert, auf Ereignisse reagiert und ziel-orientiert vorgeht.

Emergenz

- *„Bei der Beobachtung von Tiergruppen, besonders bei staatenbildenden Insekten, stößt man oft auf den Effekt, daß sich durch die Interaktion der einzelnen Tiere untereinander, bzw. mit ihrer Umwelt, **komplexe Verhaltensmuster bilden**, die aus den **Aktionen eines einzelnen Tieres***

4 Der Fettdruck in diesem und den folgenden Zitaten ist in dem zitierten Dokument nicht vorhanden. Er wurde zum Herausheben wichtiger Stellen ergänzt.

***nicht** apriori **ersichtlich** sind. Dieses Phänomen nennt man Emergenz."* [Klü96]

- *„More precisely, a phenomenon is emergent if it **requires new categories to describe it** which are not required to describe the behaviour of the underlying components."* [Gil05]
- *„Hence, innovation in social systems is a bidirectional process: bottom-up: **emergence of new entity or phenomenon** (e.g., a given behavioural regularity, or agent property) at the aggregate level **from interaction among agents** [...]"* [Con05]
- Emergenz ist also das Auftreten von Phänomenen auf der Makroebene durch das Zusammenspiel einzelner Agenten auf der Mikroebene, die sich nicht alleine durch die Mikroebene erklären lassen.

Immergenz

- *„Hence, innovation in social systems is a bidirectional process: [...] top-down: immergence of the entity or **phenomenon in the minds of the agents** [...]"* [Con05]
- Immergenz ist also die Beeinflussung der Agenten auf der Mikroebene durch Phänomene auf der Makroebene.

Norm

- *„In the **Artificial Intelligence** literature a norm is treated as a **behavioural constraint**, that is, a reduction of the action repertoire and therefore of the actions physically available to the system. [...] The function of these norms is essentially that of permitting or improving co-ordination among participants. However, we claim that norms of co-ordination are only a subset of possible norms."* [Cas98]
- *„The difference between norms and rules is only factual: a normative strategy is one which **may be disadvantageous** to the agent that applies it [...]"* [Cas98]
- *„norms often arise from conflicts of utilities among agents, and sometimes they provide what economists call Pareto-inferior solutions; that is, solutions which produce a state of affairs such that **some agents are worse** off than they were before."* [Con95]
- *„norms [...] are also imposed by **explicit prescriptions, directives and commands**."* [Con95]
- *„**not all norms serve to improve coordination**, reduce accidental collisions and other negative interferences. [...] For example, the norm of reciprocation is essentially orientated to control cheating."* [Con95]
- *„Social norms are **prescriptions** according to which **agents are expected to act**."* [Con05]
- Im Kontext dieser Arbeit sind Normen Vorschriften, denen Agenten ihr Handeln unterordnen, unabhängig davon, ob sie davon profitieren (Norm im

Sinne der Spieltheorie), gegebenenfalls nicht davon profitieren (weiter gefasster Norm-Begriff inklusive ethischer Aspekte wie Gewaltreduzierung [Cas98]) oder ob die Norm von außen (zum Beispiel als Konstante der Umwelt) festgelegt wird. Eine Norm befindet sich auf der Makroebene und wird durch das Verhalten der einzelnen Agenten, beeinflusst von Veränderung der Regeln der Umwelt, emergent gebildet. Die Regeln der Agenten auf der Mikroebene werden durch die Normen eingeschränkt.

Im Folgenden werden weitere Begriffe definiert, die nicht aus dem sozialwissenschaftlichen Anwendungsgebiet kommen, sondern für den Informatik-Anteil der Arbeit notwendig sind. Es handelt sich um die Begriffe Simulationsmodell, Simulationsinstanz, Simulationsausführung und Simulationsrunde, die hier voneinander abgegrenzt werden sollen. Eine Darstellung des Zusammenhangs zwischen diesen findet sich in Abbildung 1.

Simulationsmodell

Ein Simulationsmodell ist ein im Hinblick auf die zu untersuchenden Sachverhalte hin vereinfachtes Abbild der Realität. Das Simulationsmodell besteht zum einen aus den Klassen, die für die Erstellung einer Simulationsinstanz notwendig sind. Hierzu gehören insbesondere die Klassen Agent und Umwelt sowie davon abgeleitete Klassen. Zum anderen besteht es aus dem Verhalten der beteiligten Agenten und der Reihenfolge möglicher Interaktionen innerhalb eines Simulationsschrittes.

Simulationsinstanz

Die Simulationsinstanz besteht aus den für die Simulationsdurchführung notwendigen Objekten, die aus den Klassen des Simulationsmodells erstellt werden, sowie deren Startwerte. Hierzu gehören insbesondere die von der Klasse Agent und den davon abgeleiteten Agentenklassen instanziierten Objekte.

Simulationsausführung bzw. -durchführung

Eine Simulationsausführung ist die konkrete Ausführung einer Simulationsinstanz. Diese ist bei der Verwendung von stochastischen Elementen vom Zufall abhängig. Mit dem selben Satz von stochastischen Werten liefern zwei auf der selben Simulationsinstanz basierende Simulationsausführungen das gleiche Ergebnis. Eine Simulationsausführung besteht aus einer bis endlich vielen Simulationsrunden.

Simulationsrunde

Eine Simulationsrunde ist Teil einer Simulationsausführung. Sie ist die konkrete Ausführung des im Simulationsmodell beschriebenen Ablaufs. Sie enthält zum Beispiel die konkrete Reaktion der Agenten auf Veränderungen in der Umwelt und die konkrete Kommunikation der Agenten. Eine Simulationsrunde entspricht in der Regel einem festen Zeitintervall. In jedem Fall folgt eine Simulationsrunde mit einer höheren Rundenzahl einer mit einer niedrigeren.

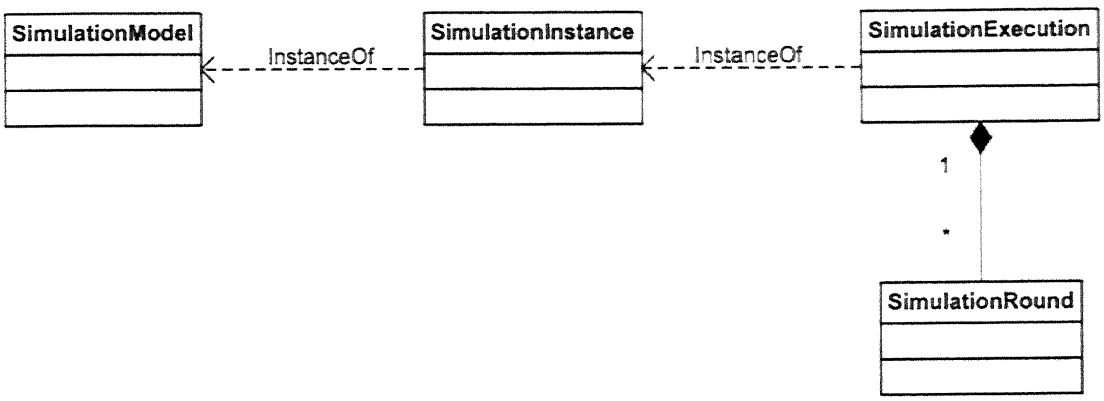

Abbildung 1: Zusammenhang zwischen Simulationsbegriffen

1.3 Anwendungsbeispiele

Im Rahmen dieser Arbeit wird nicht nur eine Simulationsumgebung erstellt, sondern diese um zwei Beispiele ergänzt. Im Folgenden werden die beiden bereits in der Einführung angesprochenen Beispiele kurz vorgestellt:

- Kirk und Coleman simulieren in Ihrem Simulationsmodell aus [Kir67] die Interaktion in einer Drei-Personen-Gruppe. Die Grundannahme hierbei ist, dass sich Interaktion und Beliebtheit gegenseitig verstärken. Diese Idee geht zurück auf Georg Simmel, der aus Beobachtungen die Annahme formulierte, dass sich aus einer Drei-Personen-Gruppe immer ein Ungleichgewicht, d.h. ein starkes Paar sowie eine isolierte Person bildet. *„Es gibt kein noch so inniges Verhältnis zwischen dreien, in dem nicht jeder einzelne gelegentlich von den beiden andren als Eindringling empfunden würde [...]"* [Sim08][5]. In Ihren Simulationen konnten Kirk und Coleman die Beobachtungen von Georg Simmel nachbilden.

- Die Entstehung von Kooperation ist Grundlage zahlreicher wirtschafts- und sozialwissenschaftlicher Untersuchungen. In [Gür06a] wird ein solches spieltheoretisches

5 Im Kapitel: „Die quantitative Bestimmtheit der Gruppe", auch verfügbar unter http://socio.ch/sim/soziologie/soz_2.htm, Stand 06.02.2007

Experiment, das sich mit der Bedeutung von Sanktionierung im Bezug auf Kooperation beschäftigt, beschrieben. Die Teilnehmer konnten sich dabei zunächst für eine Organisation mit oder ohne Sanktionierung entscheiden. Anschließend investierten die Teilnehmer eine individuell verschiedene Geldsumme. Der Ertrag kam der Gruppe gleichermaßen zu Gute. Im Experiment wurde beobachtet, dass in der Gruppe mit Sanktionierung bereits nach wenigen Runden Kooperation entstand und diese deshalb besser abschnitt.

Beide Beispiele werden im Rahmen dieser Arbeit simuliert, modifiziert und analysiert. Weiteres dazu findet sich in Kapitel 5.

2 Anforderungsanalyse

„Die Analyse als erster Schritt der Systementwicklung entscheidet maßgeblich über den Erfolg oder Misserfolg eines Projektes." [Rupp04]

„Software-Prototypprojekte dienen dazu, die Anforderungen für ein neues Softwareprodukt auszuloten. Ihr Ausgangszustand ist eine vage Vision. [...] Sie gleichen Expeditionen ins Ungewisse. Wer also solche Projekte betreibt, muss bereit sein, viel zu investieren, ohne ein sicheres Ergebnis zu erwarten." [Sne05]

Die hier aufgelisteten Anforderungen sind auf die im Rahmen dieser Arbeit entstandene Simulationsumgebung bezogen. Diese stellen einen Kompromiss zwischen schwierig planbarem Prototypprojekt und notwendiger Anforderungsanalyse dar. Die Anforderungen sind deshalb zum Teil als Vision formuliert.

Der Kern der Arbeit wird mit der Bedingung „muss" und weitere Anforderungen mit der Bedingung „soll" gekennzeichnet. Wichtige Anforderungen an die Simulationsumgebung des Projekts EMIL, nicht aber für die Simulationsumgebung zu dieser Arbeit (letztere folgend Simulationsumgebung), enthalten die Wörter „kann" oder „wird".

Die Aufteilung in funktionale Anforderungen, technische Anforderungen, Anforderungen an die Benutzerschnittstelle sowie qualitative Anforderungen ist aus [Rupp04, S.140ff] entnommen. Das Glossar findet sich unter „Terminologie" im vorhergehenden Kapitel. Die Anforderungen selbst sind zum Teil von [Wool02] inspiriert.

Die Strukturierung in [Rupp04] wurde vermutlich nicht im Hinblick auf die Entwicklung von Forschungsprototypen entwickelt. Deshalb sind einzelne Aspekte wie die Benutzerschnittstelle und die Anforderungen an die Qualität weniger detailliert behandelt als die Funktionalität. Die Anforderungen wurden, soweit möglich, aus der Projektbeschreibung entnommen und um Annahmen aus der Forschungsgruppe ergänzt.

2.1 Funktionalität

Simulation

- Die Simulation muss aus Agenten bestehen.
- Die Simulation muss diskret in Zeit, Zustand und Aktionen der Agenten sein.
- Die Simulation muss mindestens zwei Ebenen, eine Mikroebene und eine Makroebene, enthalten können.

- Die Simulationsumgebung muss mit Hilfe von mindestens zwei verschiedenen Simulationsmodellen getestet werden.
- Die Simulation muss deterministisch umgesetzt werden können. Eine festgelegte Aktion bei einem festgelegten Zustand muss demnach immer das gleiche Ergebnis liefern. Sie sollte allerdings auch indeterministische Elemente, d.h. stochastische Einflüsse, enthalten können.
- Es kann eine Umwelt ermöglicht werden.

Agent

- Die Simulation muss aus autonomen, interagierenden, lernfähigen, und zielgerichtet agierenden Agenten bestehen können.
- Ein Agent muss eigene Ziele und Regeln besitzen können und kann auf einer Hierarchieebene angeordnet sein.
- Ein Agent muss mit Agenten auf gleicher Ebene interagieren können.
- Ein Agent muss mit seinen Aktionen gegebenenfalls die Umwelt, die Normen auf der darüber liegenden Makroebene sowie den Zustand der anderen Agenten beeinflussen.
- Die Regeln eines Agenten müssen von den Normen der Makroebene beeinflussbar sein.
- Ein Agent muss ein gegebenenfalls unvollständiges Bild der Struktur und der Art der Beziehungen, die zwischen den beteiligten Agenten bestehen, besitzen können. Gleiches gilt gegebenenfalls für die Umwelt.
- Ein Agent muss aufgrund seiner eigenen Regeln, seinem Bild der Gesamtstruktur aller sozialen Beziehungen, eventuell auch der Umwelt und weiteren zufälligen Einflüssen entscheiden können, welche Aktion er zur Erreichung seiner Ziele im nächsten Schritt ausführt.
- Ein Agent soll seine Regeln an die Erfahrung aus vergangenen Runden anpassen können.
- Ein Agent kann idealerweise auch eine langfristige Planung vornehmen.

Umwelt

- Es kann die Modellierung einer statischen Umwelt ermöglicht werden, die die Aktionen der Agenten beeinflusst und selbst von den Agenten über Ihre Aktionen verändert werden kann. Statisch heißt eine Umwelt, wenn diese sich nicht ohne Einfluss der Agenten ändert.
- Es kann zudem die Modellierung einer einfachen dynamischen Umwelt ermöglicht werden, die ihre Eigenschaften auch unabhängig von den Agenten ändert.

- Die Umwelt kann eine Topographie enthalten, auf der sich die Agenten bewegen.

2.2 Technik

Programmiersprache

- Die Simulationsumgebung muss in Java implementiert werden. Java ist, wie vom Projekt gefordert (siehe dazu [Con05, S. 28]), plattformunabhängig, objekt-orientiert und ermöglicht die Erstellung einer erweiterbaren Simulationsumgebung. Zudem sind existierende Frameworks für sozialwissenschaftliche Simulation, wie zum Beispiel Repast und MASON, in Java geschrieben.

2.3 Benutzerinteraktion

Benutzergruppe

- Die Simulationsumgebung muss von den Mitgliedern der Arbeitsgruppe Troitzsch und sollte von den Beteiligten im Projekt EMIL verwendet werden können.

Benutzerschnittstelle

- Die Benutzerschnittstelle sollte graphisch umgesetzt werden.
- Die Benutzerschnittstelle sollte verschiedene Möglichkeiten der statistischen Auswertung bieten.

2.4 Qualität

Zuverlässigkeit

- Die Simulationsumgebung muss korrekt funktionieren.
- Die Simulationsumgebung kann ausfallsicher und robust gegenüber Fehlern erstellt werden.

Wartbarkeit

- Die Simulationsumgebung muss mittels Blackbox-Tests auf ihre Funktion getestet werden.
- Die graphische Benutzerschnittstelle, die Dokumentation des Codes und die Bezeichner im Code müssen, insbesondere im Hinblick auf die Benutzergruppe, in Englisch gehalten sein.
- Der Kern der Simulationsumgebung kann mit JUnit-Testklassen getestet werden.

Portabilität

- Es sollten nur Frameworks verwendet werden, die ihrerseits unter den gängigen Betriebssystemen lauffähig sind.

Effizienz

- Die Simulationsumgebung muss 100 Runden eines kleinen Simulationsbeispiels von bis zu 5 Agenten in angemessener Zeit ausführen und die Informationen dazu anzeigen. Angemessen ist hier eine Zeit von maximal 10 Sekunden.
- Die Simulationsumgebung kann dahingehend optimiert werden, dass eine Runde einer größeren Simulation von zum Beispiel 1.000 Agenten innerhalb einer angemessenen Zeit von 10 Sekunden ausgeführt werden kann.

Benutzerfreundlichkeit

- Die Einarbeitungszeit sowie der Aufwand zur Nutzung des Prototyps sollten angemessen sein. Angemessen heißt hier, dass ein mit dem Thema vertrauter wissenschaftlicher Mitarbeiter an Hand eines mitgelieferten Beispieles die wesentliche Funktionsweise der Simulationsumgebung innerhalb einer Stunde verstehen und einige Parameter innerhalb einer Minute verändern kann.

2.5 Betrachtete Arten von Modellen

Da die Anforderungen aus dem Projekt EMIL an die zu erstellende Simulationsumgebung erst nach Abschluss dieser Arbeit vorliegen werden und eine Abdeckung aller denkbaren Arten von Modellen im Prototyp zu aufwändig und zudem nicht notwendig ist, wurden in der Koblenzer Arbeitsgruppe des EMIL-Projekts zunächst einige einfache Beispiele diskutiert. Besonders häufig waren Modelle mit folgenden Eigenschaften vertreten:

- Es gibt in der Regel nur eine Art (Klasse) von Agenten.
- Alle Agenten sind in einem Netzwerk mit jedem anderen Agenten verbunden.
- Es kann eine Stärke der Beziehung angegeben werden, die in der Regel ungerichtet ist.
- Auf eine Grid-basierte Umwelt, ähnlich eines zellulären Automaten, wird verzichtet.
- Die Eigenschaften der Umwelt werden als Variablen modelliert.

Die hier entwickelte Simulationsumgebung wird auf diese Art von Modellen beschränkt, eine mögliche Weiterentwicklung aber mit berücksichtigt.

3 Architektur

„Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung, sowie die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen." [Reu06]

In diesem Kapitel werden die architektonischen Entscheidungen bezüglich der hier zu entwickelnden Simulationsumgebung vorgestellt und begründet. Es beginnt mit einer Evaluation von Simulations-Frameworks als Basis für die Umgebung. Darauf aufbauend wird die grundlegende Architektur beschrieben und mögliche Varianten bei der Umsetzung diskutiert. Anschließend werden weitere hilfreiche Frameworks vorgestellt. Zuletzt wird das Vorgehen bei der Erstellung des Plugins zur Modellierung und Codegenerierung beschrieben.

3.1 Evaluation von Simulations-Frameworks

Im Projektantrag von EMIL heißt es zum Simulationsprogramm bezüglich des Requirements Engineerings: *„An evaluation of available simulators (REPAST, MASON, Swarm) [...] will open this phase."* [Con05] Da zum Zeitpunkt dieser Arbeit diese Phase noch bevor steht, werden hier die wichtigsten Funktionen der Frameworks im Hinblick auf diese Arbeit sowie das Projekt EMIL evaluiert. Eine vollständige Evaluation ist aufgrund der zeitlichen Beschränkung nicht das Ziel. Die abschließende Bewertung ist mit Vorbehalt zu sehen. *„The difficulty in evaluating these simulation frameworks lies mainly in the fact that it is practically impossible to gain extensive practical experience with the frameworks within a period of a few weeks."* [Tob04] Eine umfangreichere Evaluation innerhalb des Projekts EMIL ist notwendig.

Da von den drei genannten Frameworks Swarm das Älteste ist, Ideen daraus in die Entwicklung von Repast eingeflossen sind[6] und es in Objective-C geschrieben wurde, wird sich im Folgenden auf Repast [Rep07] und MASON [Luke04] beschränkt. Es wird dabei jeweils die letzte stabile Version verwendet. Das ist zur Zeit[7] die Version 3.1 bei Repast und bei MASON die Version 10. Bei Repast wird aufgrund der längeren Zeit zwischen der letzten stabilen und der bereits veröffentlichten Alpha-Version zudem das neue Repast Simphony mit Repast 3.1 verglichen.

6 *„Repast borrows many concepts from the Swarm agent-based modeling toolkit"* [Rep07]
7 Stand 02.01.2007

Repast wurde unter anderem[8] bereits in [Tob04] gegen Swarm, Quicksilver und VSEdit evaluiert. Die Autoren schreiben dazu: „*The most suitable simulation framework for applications-oriented theory and data based modeling is clearly RePast.*" [Tob04] Dieser Evaluation lag die Version 2.01 von Repast zu Grunde und MASON wurde nicht berücksichtigt, sodass eine erneute Evaluation sinnvoll ist.

Beim Vergleich der beiden Frameworks wird wie folgt vorgegangen: Zunächst werden die Angaben auf der jeweiligen Webseite[9] berücksichtigt. Im zweiten Schritt werden bereits implementierte Beispiele im „Journal of Artificial Societies and Social Simulation" (JASSS)[10] gesucht. Drittens werden die ersten Kapitel des jeweiligen Tutorials nachvollzogen, und schließlich wird ein mitgeliefertes Beispiel von Agenten in einem Netzwerk verändert.

Bei der Evaluation wird auf folgende Kriterien Wert gelegt:

- **Verbreitung:** Wie viele Arbeitsgruppen verwenden das Simulationsframework?
- **Benutzbarkeit:** Wie lange dauert die Einarbeitungszeit? Wie viel Aufwand ist für ein einfaches Beispiel nötig? Ist es möglich, einfache Agenten ohne Programmierung auf der Ebene des Programmcodes zu erstellen?
- **Netzwerkfähigkeit:** Wie umfangreich werden Netzwerke mit Agenten als Knoten und Beziehungen als Kanten bei der Erstellung, der Durchführung und der Darstellung der Simulation unterstützt? Die in den Tutorials überrepräsentierten 2D-Topographien sind dagegen weniger wichtig.
- **Wissensspeicherung:** Wie umfangreich wird die Wissensspeicherung und -generierung innerhalb der Agenten unterstützt? Ist es möglich den Agenten ein „Gedächtnis" mitzugeben, in dem sie sich ein Bild der Struktur aller Agenten machen können?
- **Lernverfahren:** Gibt es Module, auf Basis derer die Agenten auf Grundlage ihres Wissens lernen, d.h. ihre Regeln anpassen können?
- **Weitere auffällige Aspekte**

3.1.1 MASON

MASON [Luke04] ist ein Framework für ereignis-diskrete Multi-Agenten-Simulation, das von der Fakultät für Informatik und dem Zentrum für soziale Komplexität an der George Mason Universität in Fairfax entwickelt wird. Die erste Version wurde 2003[11]

8 Es gibt ältere Artikel als [Tob04] über den Vergleich von Frameworks zur Multi-Agenten-Simulation, wie zum Beispiel [Gil02]. Auf diese wird aufgrund der seitdem neu erschienen Versionen von Frameworks zur Multi-Agenten-Simulation hier nicht weiter eingegangen.

9 Zu finden unter [Mas07] und [Rep07].

10 Zu finden unter http://jasss.soc.surrey.ac.uk/, Stand 02.01.2007

11 Die erste Version von MASON wurde erstmals auf der Konferenz „Agent 2003" präsentiert [Mas07].

veröffentlicht. Zur Zeit ist die stabile Version 10 sowie das „previous release" 11 auf der Webseite verfügbar. Da die neue Version ohne weitere Änderung der Konfigurationsdateien nicht sofort lauffähig ist und am Umfang der Standardnetzwerkkomponente offensichtlich nicht viel verändert wurde (im Paket sim.field.network befinden sich, genauso wie in der Version 10, nur 3 Klassen), wird im Folgenden nur die Version 10 evaluiert.

Auf der Webseite von MASON sind einige Projekte, die mit MASON erstellt wurden, verlinkt. Diese führen allerdings alle auf Unterseiten der George Mason Universität[12]. In dem Journal of Artificial Societies and Social Simulation (JASSS) ist zur Zeit eine Veröffentlichung auf der Grundlage von MASON auffindbar[13]. Der Autor von [Dun05] arbeitet auch an der George Mason Universität. Somit wird MASON bisher vor allem an der George Mason Universität verwendet und ist offenbar noch nicht so weit verbreitet wie vergleichbare Frameworks.

Nach eigenen Angaben auf der Webseite des Projekts [Mas07] sowie [Luke04] wendet sich MASON an erfahrene Java-Programmierer. Entsprechend ist es für einen Anfänger ohne Programmiererfahrung nicht möglich, ein Modell auf Grundlage von MASON zu entwerfen. Das Ziel von MASON ist es, eine schnelle und kompakte Bibliothek zur Verfügung zu stellen, um insbesondere viele Ausführungen großer Simulationen mit vielen Agenten in einem möglichst kleinen Zeitraum durchführen zu können. Um dies zu erreichen, ist MASON mit einem kleinen, schnellen Kern ausgestattet, zu dem weitere Module sowie die auch von MASON zur Verfügung gestellte GUI hinzugefügt werden können. Es ist zudem möglich, Ausführungen von Simulationen anzuhalten, den Zustand zu speichern und auf anderen Computern weiterzuführen.

Die mit MASON im Paket sim.field.network ausgelieferte Netzwerkkomponente bietet nur rudimentäre Funktionen in den Klassen Edge und Network. Mehr Komfort im Hinblick auf den Umfang bereits implementierter Methoden wäre wünschenswert. In einer Instanz der Klasse Edge kann neben den Start und Zielknoten (from und to) noch auf ein weiteres Objekt verwiesen werden, in dem zum Beispiel die Gewichtung der Kante übergeben werden kann. Um diese abzufragen, ist die Methode getStrength() implementiert, die, falls es sich bei dem Objekt um eine Instanz der Klasse Number handelt, die Gewichtung der Kante zurückgibt. Eine Methode setStrength() ist jedoch selbst zu implementieren. Auch eine Visualisierung der Stärke einer Beziehung fehlt. Die Klassen Edge und Network enthalten einen Bruchteil der Anzahl der Methoden anderer Werkzeuge. Die im Paket sim.portrayal.network vorhandenen Klassen sind die

12 http://gmu.edu/

13 Gesucht wurde mit der in JASSS integrierten Suchmaschine nach dem Begriff „MASON". Danach wurden alle Fundstellen mit „Mason University" sowie Artikel, denen ihrerseits eine Evaluation zu Grunde liegt, ausgeschlossen.

Grundlage für die graphische Benutzerschnittstelle. Auf der Webseite des Projekts MASON[14] ist zudem eine hier nicht weiter betrachtete Erweiterung um Statistiken bzgl. Netzwerke verfügbar.

Eine Wissensspeicherung ist selbst zu implementieren. Auf der Webseite ist eine Erweiterung zur evolutionären Programmierung verlinkt. Diese konnte aus Zeitgründen nicht getestet werden.

Die Evaluation hat ergeben, dass die Trennung von GUI und Simulation konsequent umgesetzt wurde. Die Informationen zu den beiden Teilen werden in verschiedenen Klassen getrennt gespeichert. Die GUI ist bei der Ausführung der Simulation optional und kann auch während der Simulation beliebig aktiviert oder deaktiviert werden.

MASON verletzt ein grundlegendes Prinzip der Softwaretechnik. Um das Prinzip der „Trennung der Belange" zu erfüllen, gibt es in Java unter anderem eine Programmierrichtlinie, nach der der direkte Zugriff auf Variablen mittels der Sichtbarkeit „private" zu verbieten und nur indirekt über Methoden der Sichtbarkeit „public" zu ermöglichen ist. In der Klasse Edge wird in 5 von 6 Fällen die Sichtbarkeit nicht explizit angegeben[15], in einem Fall wird explizit die Sichtbarkeit „public" angegeben. Ein direkter Zugriff auf Klassenvariablen kann die Geschwindigkeit erhöhen, erhöht aber zudem die Fehleranfälligkeit und ist deshalb zu vermeiden.

3.1.2 Repast

Repast [Rep07] ist ebenfalls ein Framework zur ereignis-diskreten Multi-Agenten-Simulation, das seinen Ursprung an der Universität von Chicago hat. Zur Zeit ist sowohl die stabile Version 3.1 als auch die Version Simphony im Zustand „Alpha 1" über die Webseite verfügbar. Da der Unterschied zwischen Repast 3.1 und Simphony größer als zwischen MASON 10 und 11 erscheint, werden hier beide Versionen untersucht.

Repast scheint von mehr unterschiedlichen Arbeitsgruppen genutzt zu werden als MASON. So befindet sich in JASSS mehr als ein Artikel, dessen Forschung auf Repast als Framework aufbaut. Artikel gibt es zum Beispiel zu den Themen „*Network and Terrain Landscape*" [Dib04], „*Global Reputation Systems*" [Schl06] und „*Axelrod's 'Evolutionary Approach to Norms'*" [Gal05]. Damit kann davon ausgegangen werden, dass das Framework Repast nicht nur älter, sondern auch weiter verbreitet als MASON ist.

14 Siehe http://cs.gmu.edu/~eclab/projects/ecj/, Stand 02.01.2007

15 Damit ist die Sichtbarkeit auf den Standard „paketsichtbar" gesetzt. Siehe dazu auch [Ull07, Kap. 6.7.15].

Wie bei MASON wird ein Modell in Java-Code beschrieben. Java-Erfahrung ist also Voraussetzung für die Entwicklung mit Repast. Tutorials und Beispiele zur Modellerstellung sind zahlreich vorhanden. So wird in einem der Beispiele auch ein neuronales Netz verwendet.

Die Unterstützung von Simulationen mit Agenten in Netzwerken ist bei **Repast 3.1** etwas komfortabler als bei MASON 10. Die Standardklassen bzgl. Knoten (DefaultNode) und Kanten (DefaultEdge) enthalten auch Methoden zur Gewichtung der Kanten. In dem Paket uchicago.src.sim.network sind insgesamt 36 Klassen enthalten.

Der Webseite [Rep07] nach enthält Repast unter anderem Bibliotheken zu genetischer Programmierung, neuronalen Netzen und der Modellierung sozialer Netze. Über neuronale Netze beispielsweise können Agenten ihr Wissen über die Welt zunächst über Training des Netzes speichern und folgend auch anwenden.

Es werden keine Bestandteile neuer Java-Versionen ab 1.5, zum Beispiel generische Datentypen, verwendet, da Repast 3.1 vor Veröffentlichung derselben entstanden ist.

Repast Simphony alpha 1 befindet sich zur Zeit noch in der Entwicklung. Es nimmt bei der Erstellung eines neuen Simulationsmodells Teile aus der prozeduralen Java-Code-Schicht in eine höhere, deklarative Schicht. So wird beispielsweise die Art der GUI-Ausgabe später in der GUI selbst bestimmt. Zwischen GUI und Code befindet sich eine XML-basierte Zwischenschicht. Die GUI selbst ist im Gegensatz zur Vorgängerversion nur in einem einzigen Anwendungsfenster integriert, erscheint kompakt und aufgeräumt und ist deshalb näher am Design der graphischen Benutzerschnittstellen von Windows-Standardapplikationen. In späteren Versionen sollen einfache Modelle vollständig ohne Java-Kenntnisse erstellt werden können[16].

Die Programmierung selbst wird durch ein Tutorial, das die Einbindung in Eclipse beschreibt, sowie ein Plugin für diese Entwicklungsumgebung unterstützt.

Das Tutorial ist in den ersten Schritten verständlich, bei späteren Schritten, wie zum Beispiel über das Thema Netzwerke, aber weniger ausführlich. Material zu Repast auf der zugehörigen Webseite und im Internet allgemein bezieht sich vor allem auf die Version 3.1. Zu Repast Simphony selbst werden nur drei Beispiele mitgeliefert, was deutlich weniger als bei Repast 3.1 und MASON 10 ist. Repast Simphony liegt zudem

16 „A later release of Repast Simphony will offer analysts the ability to create basic models without needing to know Java." [Rep07, http://repast.sourceforge.net/download.html, Stand 03.01.2007]

erst in der sehr frühen Version „alpha 1" vor, was unter anderem eine niedrigere Qualität vermuten lässt.

3.1.3 Fazit: Repast mit leichten Vorteilen

Aufgrund der Evaluation hat Repast 3.1 für diese Arbeit leichte Vorteile gegenüber den anderen Frameworks.

Gegenüber MASON erscheint Repast als das zum Teil umfangreichere, ausgereiftere und mit dem Schritt in Richtung Repast Simphony auch benutzerfreundlichere Programm, was wesentlich an dem früheren Beginn der Entwicklung liegen dürfte. MASON hingegen erscheint als das agilere, kleinere Projekt, welches die Belange Visualisierung und Simulationsablauf klar trennt und einen schnelleren Simulationsablauf ermöglicht. Wichtige Vorteile von Repast sind die Verbreitung, die vermutlich daraus resultierende größere Akzeptanz und das bei einem Projektpartner vorhandene Repast-Wissen eines Entwicklers.

Die neueste Version von Repast, Repast Simphony, liegt zur Zeit noch in der sehr frühen Version „alpha 1" vor und ist demnach bezüglich Dokumentation und vermutlich auch Funktion nicht vollständig. Auf Anfrage empfahl ein Repast-Entwickler[17] Repast 3.1 aufgrund von Qualität und Stabilität. Somit ist im Rahmen dieser Arbeit Repast 3.1 erste Wahl, eine spätere Verwendung von Repast Simphony im Projekt EMIL jedoch nicht ausgeschlossen.

3.2 Architektur aus Modulsicht

Eine Basis der Simulationsumgebung dieser Arbeit ist das Framework Repast. Deshalb werden zunächst die möglicherweise interessanten Module aus Repast beschrieben und um die notwendigen Erweiterungen ergänzt.

Der Kern von Repast besteht aus einem Scheduler, der die einzelnen Simulationsschritte und deren Abfolge enthält, und der Benutzerschnittstelle, über die die Simulationen gesteuert werden können. Letzteres schließt das Laden, Initialisieren und Anhalten der Simulationen ein. Zudem werden Module zur graphischen Visualisierung der Objekte einer Simulation über Graphen, Netzwerke und Grids zur Verfügung gestellt. Auf Parameter des Modells kann über die Benutzerschnittstelle Einfluss genommen werden. Zudem gibt es die Möglichkeit über eine willkürliche Zahl, einem sogenanten „RandomSeed", und einen von Repast zur Verfügung gestellten Zufallszahlengenerator wiederholbare Sequenzen von Zufallszahlen zu erzeugen. Simulationsdurchläufe eines

17 Laszlo Gulyas, zudem ein Projektpartner in EMIL (Email vom 22.12.2006)

unveränderten Simulationsmodells mit gleichen Startwerten und gleichem RandomSeed verlaufen auch mit stochastischen Anteilen immer gleich.

Zudem werden über Adapterklassen weitere Pakete eingebunden, von denen hier nur wenige vorgestellt werden sollen:

- Colt[18] stellt Bibliotheken für schnelle wissenschaftliche und technische Berechnungen zur Verfügung. In Repast kommt es unter anderem bei der Auswahl von Zufallszahlen innerhalb verschiedenster Verteilungen zum Einsatz.
- Joone[19] ist ein Framework zur Erstellung, dem Training und der Anwendung künstlicher neuronaler Netzwerke, in dem verschiedene Lernverfahren zur Auswahl gestellt werden.
- Plot[20] ist Teil eines Frameworks und dient zur Erstellung von Graphen. Es wird in den Modulen von Repast verwendet.

Alle externen Bibliotheksdateien finden sich im lib-Verzeichnis von Repast[21].

Im Rahmen dieser Arbeit wird das Framework Repast um ein davon unabhängiges Modellierungs-Plugin mit integrierter Codegenerierung für Eclipse ergänzt, welches zudem in der Lage ist, den Coderahmen aus den Simulationsmodellen zu generieren. Eine genauere Beschreibung dazu findet sich im folgenden Teilkapitel. Eine Übersicht zur Gesamtarchitektur findet sich in Abbildung 2.

18 Webseite http://dsd.lbl.gov/~hoschek/colt/index.html, Stand 12.04.2007

19 Die zugehörige Webseite findet sich unter http://www.jooneworld.com, Stand 12.04.2007. Eine Anleitung zu Joone findet sich in [Mar07].

20 Plot ist Teil des Frameworks Ptolemy II. Dessen Webseite ist unter folgender Adresse zu finden: http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm, Stand 12.04.2007

21 In der in dieser Arbeit eingesetzten Repast 3.1-Distribution finden sich folgende Bibliotheken: asm.jar, beanbowl.jar, colt.jar, commons-collections.jar, commons-logging.jar, geotools_repast.jar, ibis.jar, jakarta-poi.jar, jep-2.24.jar, jgap.jar, jh.jar, jmf.jar, jode-1.1.2-pre1.jar, joone.jar, JTS.jar, junit.jar, log4j-1.2.8.jar, OpenForecast-0.4.0.jar, openmap.jar, plot.jar, ProActive.jar, trove.jar, violinstrings-1.0.2.jar

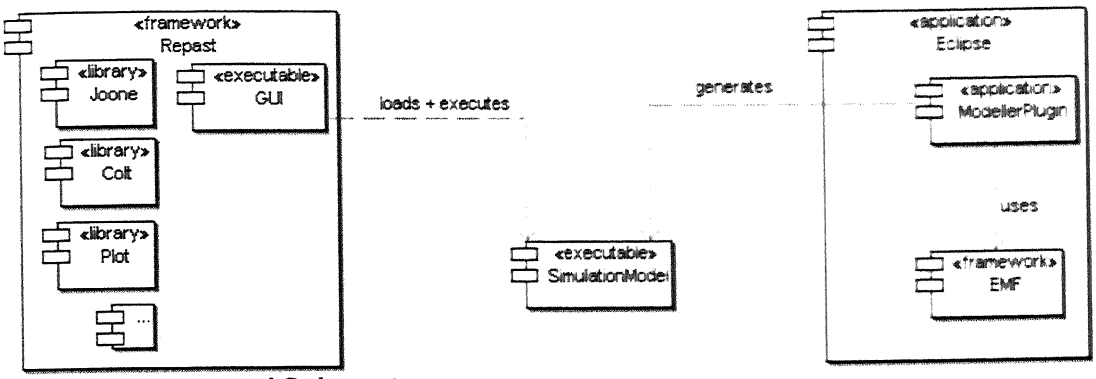

Abbildung 2: Repast und Codegenerierung

3.3 Modellierung und Codegenerierung

Um die relativ breiten Möglichkeiten von Repast auf die notwendigen zu beschränken, für den Anwendungsfall spezifische Erweiterungen einzuarbeiten und dabei die Benutzbarkeit zu vereinfachen, gibt es viele Möglichkeiten.

Die Grundidee, die hier verfolgt werden soll, ist mit vertretbarem Aufwand modellierbare Teile auf Modellierungsebene zu erstellen. Die im Modell enthaltenen Informationen werden dabei möglichst vollständig für eine Generierung von Code genutzt, damit gleiche Informationen nicht doppelt gepflegt werden müssen und bei Beginn der Programmiertätigkeit des Anwenders bereits ein Coderahmen zur Verfügung steht.

Das Konzept, Modelle für Multi-Agenten-Simulationen ganz oder teilweise visuell zu erstellen, ist nicht völlig neu. Eine solche auf Aktivitätsdiagrammen basierende Umgebung, bei der die Agenten in einem Grid angeordnet sind, ist SeSAm [Klü06]. SeSAm ist kein Plugin für eine Entwicklungsumgebung, sondern ein eigenständiges Programm. Es wurde, vermutlich mit Hilfe von Bibliotheken, von Grund auf neu programmiert. In der vorliegenden Arbeit liegt der Fokus auf Agenten in einem Netzwerk und nicht auf einem Grid. Hier soll zudem aus mehreren Gründen ein anderer Ansatz gewählt und ein Plugin für Eclipse [Ecl07] entwickelt werden. Eclipse selbst ist für die in den Anforderungen geforderte Programmiersprache Java als Entwicklungsumgebung geeignet.

Gegen Eclipse als Teil einer Simulationsumgebung spricht die etwas längere Einarbeitung der späteren Nutzer aufgrund vieler zusätzlicher Funktionen. Diese müssen zu Beginn zwar nicht verwendet werden, die Umgebung ist durch sie und die zugehörigen Buttons und Menu-Einträge jedoch zunächst verwirrend. Zudem ist Eclipse

mit einer Downloadgröße von zur Zeit ungefähr 120 MB[22] alleine schon wesentlich größer als ein selbstentwickeltes Programm. Hinzu kommt für eine aktuelle Eclipse-Version die Anforderung an einen aktuellen Rechner mit zur Zeit[23] mindestens 512 MB RAM. Diese Anforderung kann im wissenschaftlichen Umfeld in der Regel als gegeben angenommen werden. Ein wichtiger negativer Punkt ist, dass für die beteiligten Entwickler ein erheblicher Einarbeitungsaufwand in das Plugin-Konzept von Eclipse beziehungsweise verwendete Frameworks besteht.

Für Eclipse hingegen sprechen die breite Entwicklergemeinde, das eingebrachte Wissen und professionelle Management von IBM[24], die weite Verbreitung und die zahlreichen Tutorials rund um Eclipse. Aufgrund des Haupteinsatzzwecks, der Java-Programmierung, ist mit der Eclipse-Plattform der Umstieg vom Modellierer zum Programmierer reibungsloser möglich. Benutzer, die bereits Simulationen mit Hilfe von Eclipse und Java erstellt haben, haben bei der Benutzung eines Eclipse-Plugins zumindest keine längere Einarbeitungszeit als bei einem zusätzlichen Programm. Der größte Vorteil eines Eclipse-Plugins gegenüber einer Eigenentwicklung ist jedoch die gute Integration von Modellierung und Programmierung, da der Programmcode nicht in einem zusätzlichen Programm, sondern in einer dafür vorgesehen integrierten Entwicklungsumgebung erstellt wird. Dies bringt als weiteren Vorteil mit sich, dass bereits in Eclipse vorhandene Module nicht neu entwickelt werden müssen. So ist zum Beispiel die Überprüfung des Codes auf eine korrekte Syntax und die Verwaltung des Codes über SVN entweder bereits in Eclipse vorhanden oder über ein zusätzliches Modul leicht erweiterbar.

Für diese Arbeit wird also ein Eclipse-Modellierungs-Plugin entwickelt, welches Teile des Codes aus dem Modell generiert. Auch wenn dieser Ansatz später nicht weiter verfolgt werden sollte, entsteht daraus für das Projekt EMIL ein Wissensgewinn.

Ein Modellierungsframework zur Erstellung von Plugins in Eclipse ist mit dem Eclipse Modelling Framework (EMF) [EMF07] bereits vorhanden. Mit ihm ist es möglich, aus Metamodellen ein Modellierungs-Plugin für Eclipse zu erzeugen. Es enthält zudem ein Tool namens Java Emitter Templates (JET) [JET07][25], mit dem aus an Java Server

22 http://www.eclipse.org/downloads/, Stand 17.04.2007 für die Windows-Version Eclipse SDK 3.2.2

23 Stand 17.04.2007

24 Eclipse basiert auf IBM Visual Age for Java 4.0, das von IBM im Jahr 2001 freigegeben wurde. IBM ist zudem nach wie vor an der Weiterentwicklung von Eclipse beteiligt. Mehr Informationen hierzu finden sich unter http://de.wikipedia.org/wiki/Eclipse_(IDE) oder http://www.research.ibm.com/eclipse/, Stand 10.04.2007

25 Ein Tutorial hierzu findet sich auf der Eclipse Webseite unter http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html, Stand 23.04.2007

Pages (JSP)[26] orientierten Templates Code erzeugt werden kann. Es liegt nahe, beide Teile des EMFs zu nutzen, um aus erstellten Modellen Code zu generieren.

Das Modellierungs-Plugin, das mit EMF erstellt werden kann, visualisiert die in einer XML-Datei abgelegten Modelle zunächst baumartig. Es kann mit dem Graphical Eclipse Framework (GEF) [GEF07] um eine weitere Darstellungsform erweitert werden. Da das GEF sehr generisch gehalten ist und zum Beispiel auch als Grundlage für Eclipse-Plugins zur graphischen Erstellung von Benutzerschnittstellen verwendet wird, wird im Rahmen dieser Arbeit auf eine solche Umsetzung verzichtet.

Einen leichteren Einstieg verspricht das Graphical Modelling Framework (GMF) [GMF07]. Dieses setzt auf GEF auf und ist für die Erstellung von graphischen Modellierungsoberflächen auf Grundlage eines auf EMF basierten Modellierungstools entwickelt worden. Es scheint einzelnen Internetseiten beziehungsweise Artikeln aus dem „eclipse Magazin" zu Folge[27] noch nicht voll ausgereift zu sein, sodass fundiertes GEF-Wissen für einen sinnvollen Einsatz notwendig ist. Eine spätere graphische Visualisierung ist sinnvoll. Dabei sind GEF und GMF Frameworks erster Wahl.

3.4 Vorgehen bei der Plugin-Entwicklung

Bei der Entwicklung des Modellierungs- und Codegenerierungs-Plugins wird wie folgt vorgegangen (siehe dazu auch Abbildung 3):

- Es wird zunächst ein generisches Simulationsmodell (Kapitel 4.4.1) entwickelt. Dieses enthält die in Kapitel 4.1 beschriebenen Klassen und Methoden für Repast-Simulationen sowie die in Kapitel 4.2 ergänzten Erweiterungen für Simulationen in Netzwerken.
- Als nächstes wird ein Metamodell (Kapitel 4.3) erstellt. Aus diesem wird mit EMF ein Modellierungsplugin generiert.
- Auf Grundlage des Metamodells und des generischen Simulationsmodells werden auf JET basierende Templates entwickelt (Kapitel 4.4.2).
- Die Klassen zur Codegenerierung, die mit JET aus den Templates generiert wurden, werden anschließend in das Modellierungs-Plugin integriert.

26 Java-basierte Technologie für die Erstellung dynamisch generierter Webseiten

27 Zum Beispiel „Magische Momente mit GMF" (Ausgabe 8, August 2006), auch zu finden unter http://eclipse-magazin.de/itr/online_artikel/psecom,id,841,nodeid,230.html, Stand 23.04.2007.

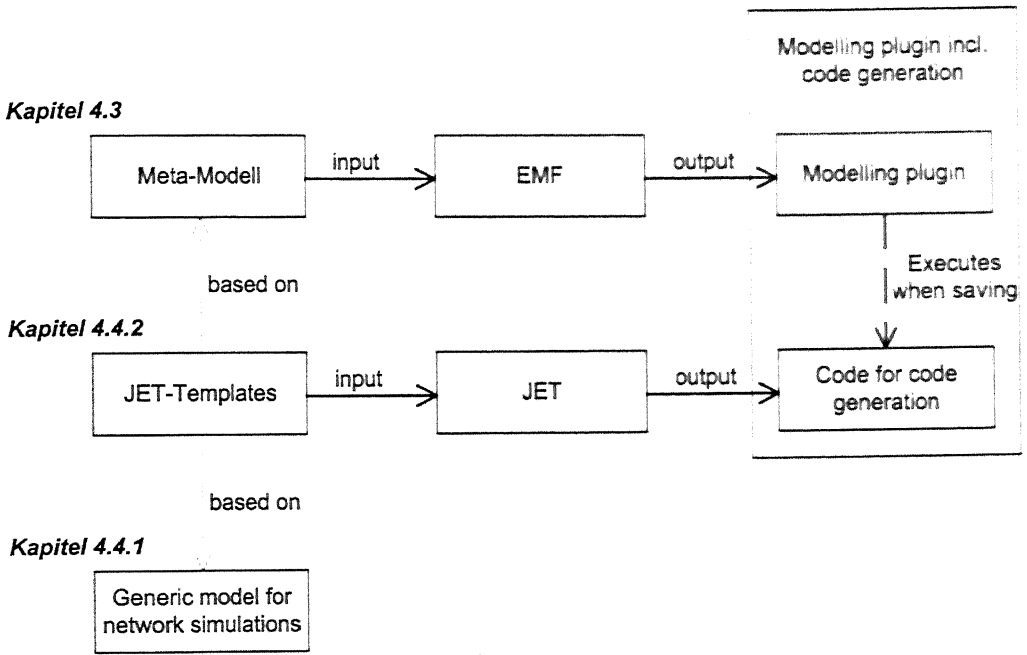

Abbildung 3: Implementierung des Eclipse-Plugins

Der Nutzer kann mit dem Plugin Modelle graphisch erstellen und die Informationen daraus für die Generierung eines Coderahmens für Repast-Simulationen in Netzwerken nutzen. Die Simulationsmodelle können dann im Code vervollständigt werden.

4 Entwurf und Implementierung

Repast stellt als Framework für Multi-Agenten-Simulationen alle Kernfunktionen zur Erstellung einer solchen Simulation zur Verfügung. Dazu gehören folgende Teile:

- Das Management des Schedulers, einschließlich dem Zählen der Runden
- Eine graphische Benutzerschnittstelle, über die Parameter auch während einer Simulationsausführung geändert werden können
- Die Visualisierung des Simulationsmodells sowie der Daten in Graphen und Klassen, die bestimmte Agentenstrukturen, zum Beispiel Netzwerke oder Grids, unterstützen
- Weitere Frameworks, die über Wrapper-Klassen dem Repast-Nutzer zur Verfügung gestellt werden

Für diese Arbeit ist es also zunächst wichtig, die von Repast vorgegebene Struktur von Klassen, Methoden und Attribute vorzustellen. In einem zweiten Schritt werden dann weitere Elemente ergänzt, die für die hier betrachteten Simulationsmodelle notwendig sind. Anschließend wird ein Metamodell für die Modellierung von Simulationsmodellen entworfen. Darauf aufbauend wird ein Plugin zur Modellerstellung und Code-generierung entwickelt und ein Beispiel implementiert. Abschließend wird eine zusätzliche Form der Darstellung über die Benutzerschnittstelle vorgestellt.

4.1 Grundlegende Repast-Architektur aus Klassensicht

Die grundlegenden Bestandteile eines auf Repast basierenden Simulationsmodells sind zum Ersten eine Modellklasse, mit der die Simulationsinstanz erstellt wird und in der Methoden und Attribute der Umwelt des jeweiligen Simulationsmodells abgelegt sind. Zum Zweiten ist es die Agentenklasse (beziehungsweise bei klar zu unterscheidenden Agenten auch mehrere Agentenklassen, die von dieser abgeleitet sind), die die Methoden und Attribute der einzelnen Agenten enthält. Ein Klassendiagramm dazu ist in Abbildung 4 zu finden.

In dem Klassendiagramm finden sich sowohl notwendige als auch sinnvollerweise hinzugefügte Attribute und Methoden. Zu letzteren gehört zum Beispiel buildDisplay(), da ein Simulationsmodell auch nur durch einen Funktionsgraphen und ohne zum Beispiel eine Netzwerkdarstellung ausreichend visualisiert sein könnte. In der Regel sind jedoch beide Visualisierungsformen sinnvoll, sodass auch beide im Klassendiagramm eine zugehörige Initialisierungs-Methode besitzen.

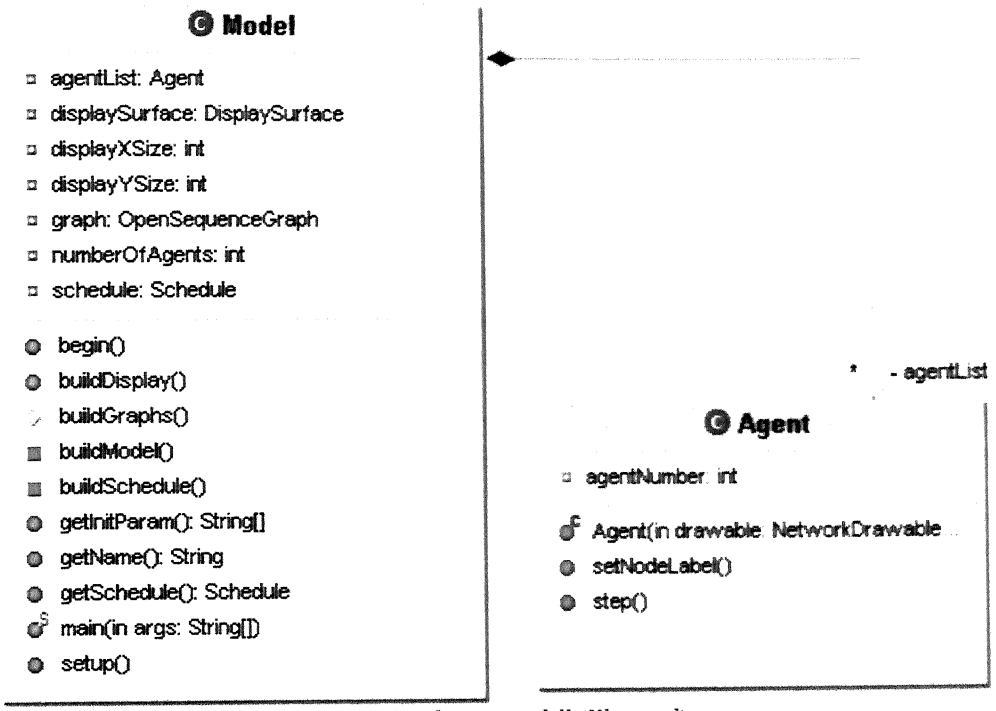

Abbildung 4: Grundlegendes Repast-Simulationsmodell (Klassendiagramm)

Zu Beginn der Simulation wird die Methode begin() aufgerufen, die wiederum die folgenden Methoden aufruft:

- buildModel()
  erstellt die für die konkrete Simulationsausführung eines Simulationsmodells notwendige Simulationsinstanz, bestehend aus den zu den Klassen des Simulationsmodells zugehörigen Objekte, zum Beispiel Instanzen der Klasse Agent.
- buildDisplay()
  erstellt die graphische Repräsentation des Simulationsmodells (zum Beispiel Knoten und Kanten).
- buildGraphs()
  erstellt den oder die Graphen, die in Abhängigkeit von Variablen der Agenten oder des Simulationsmodells Sequenzen über den Simulationsverlauf enthalten. Graphen, die nur auf Methoden der Modell-Klasse basieren, können auch nach Abschluss der Programmierung über die GUI von Repast hinzugefügt werden.
- buildSchedule()
  legt den Ablaufplan für das Simulationsmodell fest.

Die Methode getInitParam() enthält die Parameter des Simultionsmodells, die über die GUI von Repast verändert werden können. Neben dem Eintrag in dem von

getInitParam() zurückgegebenen Array muss zu jedem Parameter in der Modell-Klasse Model eine get- sowie set-Methode definiert sein[28].

Das Attribut agentList enthält die Agenten der Simulation, die in buildModel() erstellt worden sind. Die Agenten selbst besitzen sinnvollerweise zumindest eine step()-Methode, die vom in buildSchedule() erzeugten Ablaufplan in jeder Simulationsrunde einmal ausgeführt wird. Zudem besitzen sie auch eine Darstellungsform, die im Konstruktor festgelegt werden muss. In dem Beispiel aus Abbildung 4 handelt es sich bei der Darstellung eines Agenten um einen NetworkDrawableNode.

Auf Basis dieser Architektur und Ergänzungen in der step()-Methode sind bereits einfache Simulationen in Repast möglich.

4.2 Agentensimulation innerhalb von Netzwerken

Das grundlegende Repast-Simulationsmodell wird im nächsten Schritt um die Elemente erweitert, die notwendig sind, um Agenten innerhalb von Netzwerken, in denen jeder mit jedem Agenten über eine Kante verbunden ist, zu simulieren. Zudem werden einige weitere Erweiterungen ergänzt. Die Architektur ist dabei teilweise an der des Netzwerkbeispiels „Jiggle" von der Repast-Webseite [Rep07] beziehungsweise den Vorgaben von Repast selbst orientiert.

Eine mögliche Darstellung dieses grundlegenden Simulationsmodells in der GUI findet sich in Abbildung 5. Jeder Agent ist dort als Knoten abgebildet. Dieser enthält zudem den aktuellen Wert der Variable result, in der ein Fließkommawert zu dem Agenten abgelegt werden kann. Jeder der hier fünf Agenten ist mit allen anderen über eine Kante verbunden. Auch wenn zur Übersichtlichkeit jeweils nur eine Linie zwischen den zwei Agenten dargestellt wird, ist dieses Netzwerk intern als gerichteter Graph realisiert worden, sodass auch zwei unterschiedliche Werte für die eingehende und ausgehende Kante von einem Agenten zu einem anderen möglich sind.

28 Die Architektur aus Abbildung 4 enthält keine über die GUI änderbaren Parameter.

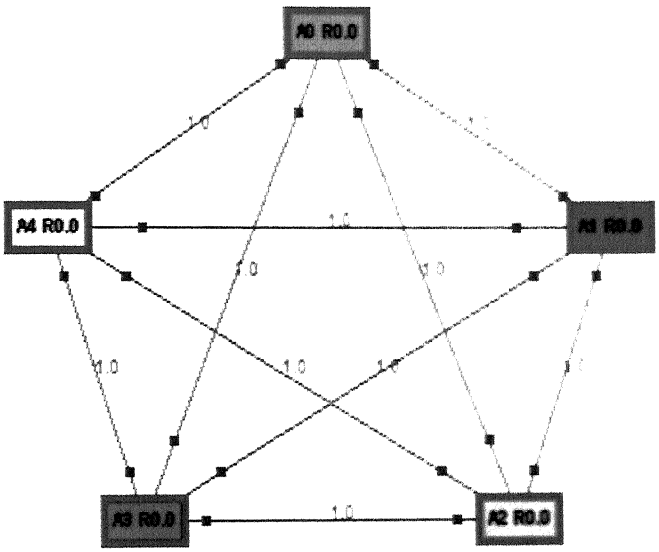

Abbildung 5: Darstellung von Agenten in einem Netzwerk

Die hierfür notwendigen Klassen finden sich in Abbildung 6. Im Vergleich zu dem grundlegenden Klassendiagramm aus Abbildung 4 sind die Klasse NetworkAgentEdge sowie einige Attribute und Methoden hinzugekommen.

Von der Klasse NetworkAgentEdge werden die Kanten zwischen den Agenten instanziiert. Über die Vorfahren DefaultDrawableNode, DefaultNode und das Interface Node wird der Instanz der Klasse NetworkAgent über die Methode addOutEdge(edge) die ausgehenden und über addInEdge(edge) die eingehenden Kanten hinzugefügt. Die in Abbildung 5 zu sehende Kantengewichtung ist über die Elternklasse von NetworkAgentEdge, DefaultEdge und den Methoden getStrength() und setStrength() realisiert. Die draw()-Methode regelt die graphische Visualisierung der Kante in der GUI.

Im Netzwerksimulationsmodell selbst wurden einige Erweiterungen zum Komfort, beispielsweise die Methode getAllEdgeWeights(), ergänzt. Hinzu kommen die Methoden getNumberOfAgents() sowie setNumberOfAgents() und eine Ergänzung der Variable numberOfAgents in der Methode getInitParam(), um die Anzahl der beteiligten Agenten komfortabel über die GUI von Repast steuern zu können.

Der Klasse NetworkAgent wurde eine Variable result hinzugefügt, die ein Ergebnis des Agenten aufnimmt und während der Simulationsausführung über die GUI ausgibt.

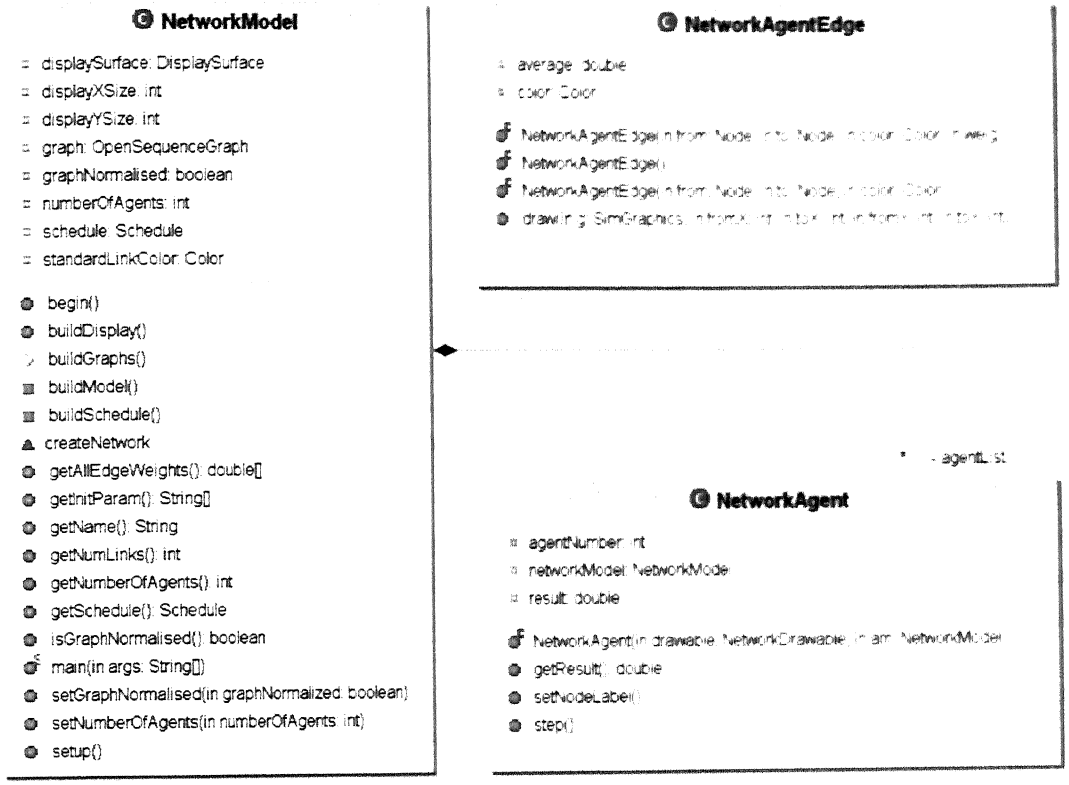

Abbildung 6: Erweitertes Netzwerk-Simulationsmodell (Klassendiagramm)

Eine Simulation mit Agenten in einem Netzwerk ist bereits mit diesen drei Klassen als Grundgerüst sowie dem Framework Repast möglich. Die in der Anforderungsanalyse erwähnten Umweltbedingungen können, auch wenn eine zusätzliche Klasse hier aus softwaretechnischen Gründen sinnvoller wäre, in der Klasse NetworkModel realisiert werden. Über die Kanten können die Agenten zum Beispiel Nachrichten austauschen. Die Regeln der Agenten würden hier prozedural in Methoden der Klasse NetworkAgent programmiert werden.

4.3 Metamodell zur Modellierung

Bei der Suche nach einem angemessenen Metamodell zur Erstellung eines Simulationsmodells ist die Frage zu stellen, welche Teile des Simulationsmodells graphisch umgesetzt werden und deshalb automatisch generiert werden können, und welche Teilstücke programmiert werden müssen. Um das graphische Simulationsmodell einfach und intuitiv zu halten und zudem die Möglichkeiten des Modellierers nicht aufgrund einer beschränkten Anzahl an Bausteinen einzuschränken, verbleiben all die Teile im Code, die Teil des individuellen Verhaltens der beteiligten Elemente (Umwelt, Agenten, Kanten) sind. Eine spätere Erweiterung um Bausteine für das zunächst vollständig im Code realisierte Verhalten ist möglich.

Im graphischen Teil des Simulationsmodells werden alle Aspekte umgesetzt, die das Simulationsmodell als Ganzes betreffen. Das Metamodell zum hier umgesetzten Modellgenerator findet sich in Abbildung 7. Ein Simulationsmodell (SimulationModel) besteht zum Ersten aus der Struktur der Simulation (SimulationStructure), die ihrerseits wiederum aus den beteiligten Elementen (ElementClass) und deren Attributen (Attribute) und Methoden (Method) besteht, und zum Zweiten aus dem Verhalten der Simulation (SimulationBehaviour). Letzteres enthält die Simulationsschritte (SimulationStep), die innerhalb einer Runde schrittweise abgearbeitet werden. Nach der Festlegung der Struktur (SimulationStructure) und des Verhaltens (SimulationBehaviour) der Simulation können dann beide Teile verknüpft werden, indem bei der Ausführung der Simulationsschritte (SimulationStep) jeweils einzelne Methoden (Method) innerhalb der Agenten angestoßen werden.

Das Verhalten der Simulation (SimulationBehaviour) enthält neben den Simulationsschritten (SimulationStep) auch einen Zeiger auf den ersten Schritt in der Simulation (firstStep). Jeder Simulationsschritt enthält seinerseits die Information des ihm nachfolgenden Schrittes (nextStep). Jeder Schritt kann ein oder mehrere Male hintereinander ausgeführt werden (timesRepeated). Erst nachdem alle Wiederholungen durchlaufen sind, wird die Simulation mit dem nächsten Schritt fortgeführt.

Es gibt verschiedene Typen von Schritten. Ein Alle-Objekte-Schritt (AllObjectStep) triggert alle Objekte der betroffenen Klasse an, ein Ein-Objekt-Schritt (OneObjectStep) hingegen nur ein Objekt. Letzterer wählt das Objekt der betroffenen Klasse aufgrund einer Bedingung (selectionCondition) aus. Die Wahl kann hier zufällig (Random) oder auch aufgrund einer Variable geschehen. Beispiele hierfür sind die Auswahl nach Maximum (Maximum) beziehungsweise Minimum (Minimum) eines Wertes, zum Beispiel dem Erfolg eines Agenten. Die Auswahl nach BooleanIsTrue triggert das Element, bei dem die entsprechende Boolean-Variable den Wert „true" besitzt. Wenn mehrere Agenten einen Wert „true" besitzen, wird der erste Agent der Liste ausgewählt. Weitere Unterklassen der Klasse Simulationsschritt (SimulationStep), zum Beispiel Einige-Objekte-Schritt (SomeObjectStep), genauso wie eine Zuordnung der Bedingungen zu den erlaubten Attributtypen, wären hier zukünftig wünschenswert.

Die Struktur der Simulation besteht aus den Elementklassen Agent (AgentClass), Umwelt (EnvironmentClass) und Kante (EdgeClass). Diese wiederum enthalten Eigenschaften (über Attribute) und Verhalten (über Method). Beide können und sollten um eine natürlichsprachliche Spezifikation (naturalLanguageSpecification) ergänzt werden. In der konkreten Durchführung wird das Verhalten der beteiligten Agenten, falls vorhanden auch der Kanten und der Umwelt, von der Simulation angestoßen.

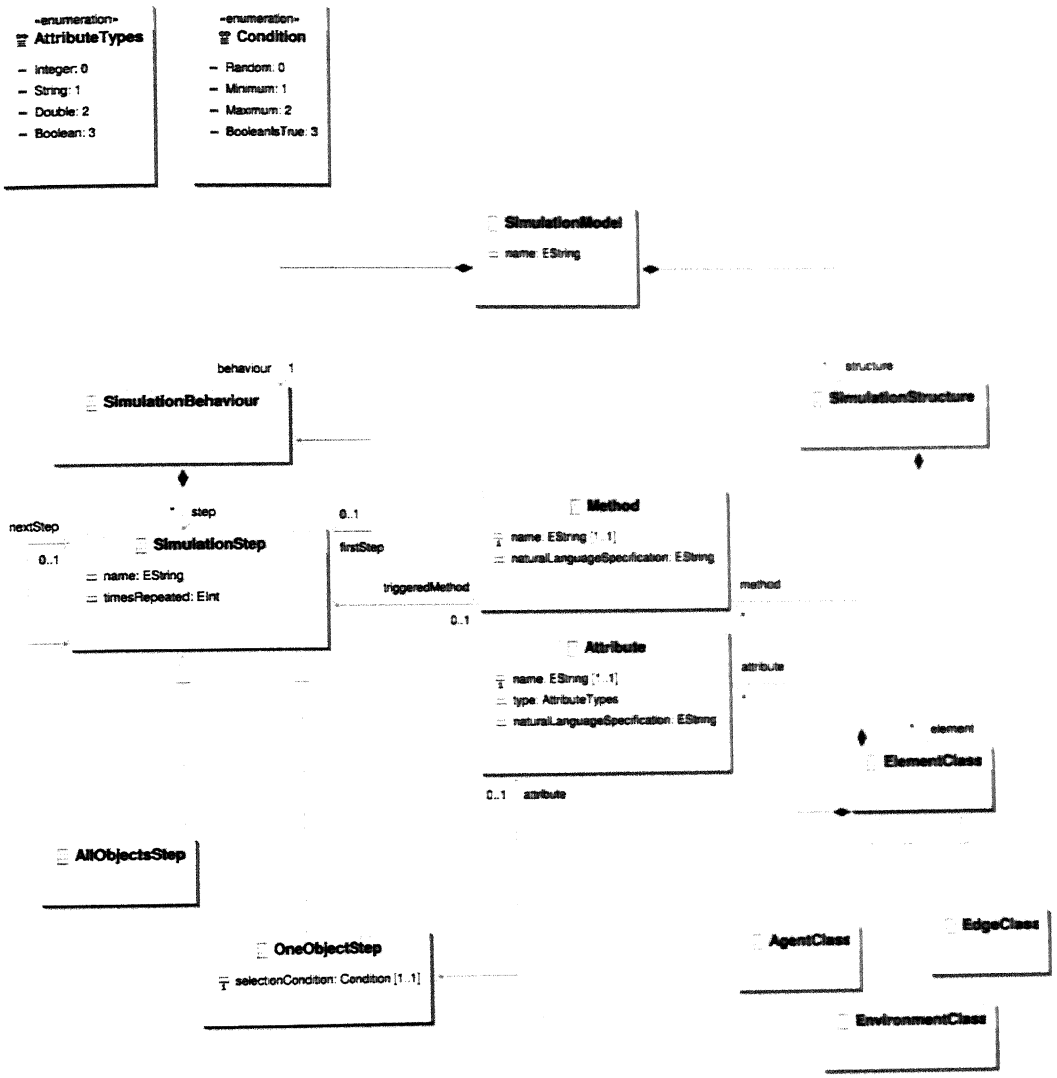

Abbildung 7: Konkretes Metamodell

Auf Basis dieses Metamodells können Simulationsmodelle entworfen werden, die, wie in der Anforderungsanalyse gefordert, autonome, zu Interaktion fähige, auf Ereignisse reagierende und zielorientiert agierende Agenten enthalten. Zudem ist die Umsetzung einer Umwelt möglich. Auch verschiedene Typen von Agenten innerhalb eines Simulationsmodells können mit diesem Metamodell nach einer kleinen Anpassung prinzipiell modelliert werden, sollen aber zunächst nicht in der Codegenerierung realisiert werden.

Eine Schwäche des Modells ist die limitierte Anzahl an Bedingungen und Typen. Diese sollte zukünftig vergrößert werden. Des weiteren ist dieser Ansatz auch hinsichtlich der Tiefe der graphischen Modellierung ausbaubar. So ist die Unterstützung bezüglich der Regeln oder eines Gedächtnisses der Agenten bisher nur teilweise und indirekt über in

Repast enthaltene Bibliotheken vorhanden. Der Einbau eines Regelinterpreters würde hier die deklarative Umsetzung des Agentenverhaltens ermöglichen. Zudem sollten weitere Module wie neuronale Netze dem Nutzer einfacher als bisher zugänglich sein.

4.4 Plugin zur Modellierung und Codegenerierung

In diesem Unterkapitel wird die Umsetzung des Plugins zur graphischen Modellierung und Codegenerierung eines Teils des Simulationsmodells beschrieben.

Das Modellierungs- und Codegenerierungs-Plugin wird auf Grundlage des Frameworks EMF entwickelt. Zunächst wird mit dem Eclipse-Plugin Omondo[29] das in Kapitel 4.3 beschriebene Metamodell modelliert und daraus mittels EMF das Modellierungs-Plugin generiert. Darin enthalten sind die Modellklassen, die die Struktur des graphischen Modells enthalten. Mit diesem Plugin wiederum können in einer anderen Eclipse-Instanz graphische Modelle erstellt werden. Aus diesen wird beim Abspeichern ein Coderahmen für die Simulation generiert.

Die Codegenerierung geschieht auf Basis eines generischen Simulationsmodells für Netzwerke, das im folgenden Unterkapitel vorgestellt wird. Praktisch umgesetzt wird dieses mit Hilfe von Templates, die anschließend beschrieben werden. Aus den Templates wird mittels JET [JET07] Code erzeugt, der auf Basis eines im Modellierungs-Plugin erstellten graphischen Modells konkrete Klassen zur modellierten Simulation erstellen kann.

4.4.1 Generisches Simulationsmodell für Netzwerke

Für die Umsetzung in die Codegenerierung wird das in Kapitel 4.2 beschriebene Repast-Simulationsmodell für Netzwerke erweitert, um die Benutzbarkeit für den Anwender zu erhöhen. Dieser soll die generierten Teile der Simulation möglichst einfach und unabhängig vom Rest des notwendigen Codes erweitern können.

In Abbildung 8 findet sich das generische Simulationsmodell. Wie in Kapitel 4.2 enthält es Klassen zum Netzwerkmodell, zu den darin enthaltenen Agenten und zu Kanten, die die Beziehungen zwischen den Agenten darstellen. Hinzugekommen sind zum einen die Klassen, die mit dem Präfix „Concrete“ beginnen. Zum Zweiten sind Attribute und Methoden zur Umwelt nicht mehr in der Klasse NetworkModel, sondern in zwei davon getrennten Klassen, NetworkEnvironment und ConcreteEnvironment, enthalten. Beide Änderungen wurden zur Vereinfachung der Programmierung des Verhaltens vorgenommen. Diese findet somit nur noch in den Klassen ConcreteEnvironment, ConcreteAgent und ConcreteEdge statt. In diesen sind nach der Codegenerierung die

29 http://www.omondo.de/, Stand 24.04.2007

graphisch modellierten Methoden, Attribute und zugehörigen Beschreibungen vorhanden. Die Klassen NetworkModel, NetworkEnvironment, NetworkAgent, NetworkEdge enthalten den bei Netzwerksimulationsmodellen gleich bleibenden Teil, der weder durch die Codegenerierung noch durch den Implementierer des Verhaltens verändert werden muss. Dazu gehört unter anderem die Darstellung der einzelnen Elemente in der GUI, Graphen sowie Standard-Attribute. In der Klasse NetworkModel wird der Scheduler aus dem „Behaviour"-Teil des graphischen Modells generiert.

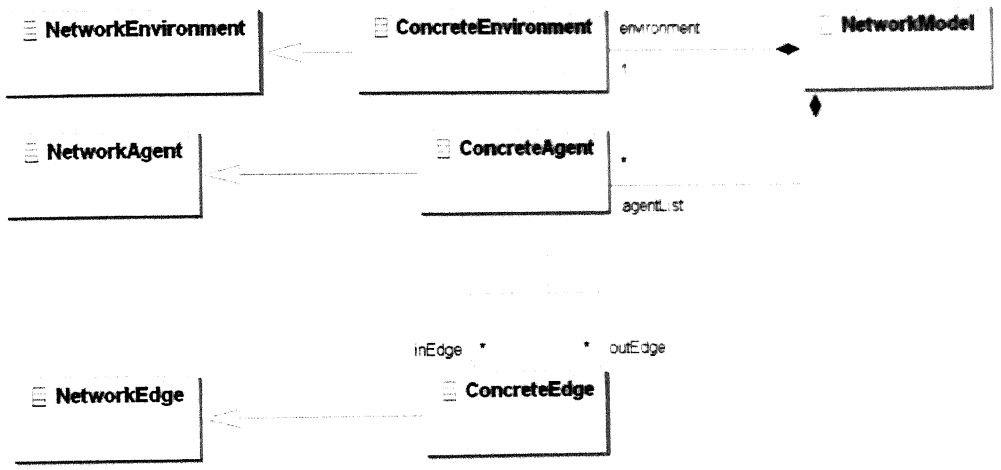

Abbildung 8: Generisches Modell für Simulationen in Netzwerken

4.4.2 Simulationstemplates

An Hand des generischen Simulationsmodells werden sieben Templates zur Codegenerierung entworfen. Hierbei dienen die Templates NetworkEnvironment.javajet, NetworkAgent.javajet und NetworkEdge.javajet der bloßen Erstellung der drei zugehörigen Klassen. Diese sind unabhängig vom graphischen Modell, das der Anwender erstellt hat.

Bei den Templates ConcreteEnvironment.javajet, ConcreteAgent.javajet und ConcreteEdge.javajet ist das anders. Dort müssen gegebenenfalls folgende Teile aus dem „Structure"-Teil des graphischen Modells generiert werden:

- Methoden
- Attribute
- get- und set-Methoden zu den Attributen
- Dokumentation zu den Attributen und Methoden

Das Template NetworkModel.javajet muss den Scheduler aus dem „Behaviour"-Teil des graphischen Modells generieren. Beim Aufrufen der Methoden müssen unter anderem folgende Varianten unterschieden werden:

- Die einzelnen Schritte müssen gemäß Ihrer Reihenfolge, festgelegt über firstStep und nextStep, vom Scheduler abgearbeitet werden.
- Handelt es sich bei dem SimulationStep um einen AllAgentStep, so muss über alle Elemente iteriert werden.
- Handelt es sich um einen OneObjectStep, so müssen die verschiedenen Bedingungen und gegebenenfalls damit zusammenhängende Attribute berücksichtigt werden.
- Je nachdem, ob die aufzurufende Methode der ElementClass zu einer Klasse AgentClass, Environment oder EdgeClass gehört, muss anders reagiert werden.
- Die jeweilige ElementClass muss dem Scheduler verfügbar gemacht werden.

Der folgende Ausschnitt aus dem Code zeigt beispielhaft die Umsetzung der Generierung der Methoden und der dazugehörigen Dokumentation im Template ConcreteAgent.javajet:

```
[..] <% [..]

for (Iterator i = methods.iterator(); i.hasNext();){
      Method method = (Method)i.next();
      if (!method.getNaturalLanguageSpecification().equals("") &&
            method.getNaturalLanguageSpecification()!=null){
      %>
/**
 * <%= method.getNaturalLanguageSpecification() %>
 */<%
      }
%>
public void <%= method.getName()%>() {
      }
<%    }

[..] %> [..]
```

Hierbei sind die Anteile zur Codegenerierung durch die Klammern <% und %> eingegrenzt. In <%= und %> enthaltene Variablen werden im generierten Code als String ausgegeben.

In der Variable methods sind die Methoden der Klasse Agent aus dem graphischen Modell enthalten. Diese wurden zu Beginn des Templates aus dem Simulationsmodell extrahiert. Bei der Generierung des Codes wird über alle Methoden iteriert. Bei jeder Methode wird zunächst die zugehörige Spezifikation als Javadoc-Dokumentation

hinzugefügt. Auf diese folgt der Rahmen der zugehörigen Methode. Dieser kann dann vom Benutzer manuell vervollständigt werden.

4.4.3 Umsetzung der Codegenerierung

Aus den Templates werden mit Hilfe von JET Klassen zur Codegenerierung erzeugt. Diese werden mit Hilfe der Klasse CodeGeneration und der darin enthaltenen Methode generateCode aufgerufen. Die Methode generateCode übergibt hierbei das vom Benutzer graphisch erstellte Modell in einem Objekt an die einzelnen Generatoren und erhält den Code der generierten Klassen in einem String als Rückgabewert. Daraus wiederum erstellt die Methode saveInFile einzelne Dateien im Eclipse-Projekt der Modelldatei mit dem graphischen Modell. Hierzu ist es sinnvoll, dass das Java-Projekt bereits einen „src"-Ordner besitzt. In diesem wird dann zunächst, falls noch nicht vorhanden, das Paket repastSimulation erstellt.

Die Methode generateCode wird beim Abspeichern des graphischen Modells ausgeführt. Dafür ist ein Aufruf in der Methode doSave() der Klasse CodeGenEditor, welche von EMF als Teil des Plugins zur Modellierung generiert wurde, hinzugefügt worden.

4.4.4 Anwendung des Plugins

Um ein vollständiges Simulationsmodell mit der Codegenerierung zu erzeugen und auszuführen, sind als Anwender die in Abbildung 9 zu findenden Schritte notwendig. Teilweise können hierbei Schritte übersprungen oder Schritte parallel bearbeitet werden.

Einzig der vierte Schritt, bei dem das Verhalten der Simulationselemente festgelegt wird, muss von einer Person mit Programmierkenntnissen vorgenommen werden. Dieser wird dabei durch die natürlichsprachliche Beschreibung des Modellierers auf graphischer Ebene, der nur Kenntnisse zur Modellierung und zur Bedienung von Computern benötigt, unterstützt.

Beim Modellieren ist zu beachten, dass die Aspekte, die die jeweiligen Elemente betreffen, auch dort modelliert werden sollten, damit diese später im Code auch an der richtigen Stelle implementiert werden können.

Bei der Modellierung müssen in diesem Prototypen vom Benutzer noch folgende, von der Anforderungsanalyse geforderten Elemente auf einer niedrigen logischen Ebene hinzugefügt werden:

Abbildung 9: Schritte zur Simulationsausführung

- Die **Kommunikation** kann über in Kanten enthaltene Attribute und Methoden erfolgen. Eine explizite und an Standards orientierte Einbeziehung in die Modellierung ist bei einer Weiterentwicklung wünschenswert.
- Die Fähigkeit der **Wissensspeicherung und -auswertung** ist mit den in Repast enthaltenen Bibliotheken, zum Beispiel über neuronale Netze, möglich, muss jedoch noch selbst hinzugefügt werden. Eine bessere Einbindung wäre wünschenswert.
- **Normen** können, wie in dem Kapitel über Anwendungsbeispiele gezeigt wird, entstehen, sind jedoch im generischen Simulationsmodell nicht über eine Variable explizit enthalten.
- Die **Umwelt** besteht hier zunächst nur aus einem Objekt und kann nur direkt in der Klasse mit Hilfe von objektorientierter Programmierung erweitert werden. Die Berücksichtigung komplexerer Umweltstrukturen in der Codegenerierung ist für die Zukunft wünschenswert.

4.4.5 Anwendung am Beispiel des Kirk-Coleman-Modells

Um die Verwendbarkeit des Plugins zu demonstrieren, wird mit diesem die in Kapitel 5 ausführlich beschriebene Standardvariante des Kirk-Coleman-Modells implementiert. Sämtliche Erweiterungen des Kirk-Coleman-Modells werden in Kapitel 5 ausführlich beschrieben und hier nicht berücksichtigt, um den Fokus auf die Umsetzung zu legen.

Zunächst muss in Eclipse ein neues Java-Projekt mit einem „src"-Ordner erstellt werden. Danach muss in den „Build Path" dieses neuen Projekts Repast eingebunden werden. Eine genaue Beschreibung zur Einbindung von Repast in Eclipse findet sich unter [Lyt06] und soll hier nicht weiter erläutert werden. Als nächsten Schritt wird über einen Rechtsklick auf das Projekt eine neue „CodeGen Model"-Datei hinzugefügt. Nach Angabe des Namens mit der Endung „.codegen" wird das Modellobjekt „Simulation Model" ausgewählt.

Danach muss, wie in Abbildung 10 gezeigt, ein graphisches Modell erstellt werden. Für die Standardversion des Kirk-Coleman-Modells sind folgende Schritte notwendig:

- Herabsetzen der Stärke aller Kanten, da die Beziehung sich über die Zeit abschwächt (AllObjectStep in ConcreteEdge).
- Die Agenten müssen Ihre individuelle Beliebtheit ausrechnen (AllObjectStep in ConcreteAgent).
- Jeder Agent wählt seinen bevorzugten Kommunikationspartner zufällig, aber gewichtet nach der Stärke seiner Beziehungen (AllObjectStep in ConcreteAgent).
- Ein Agent wird zufällig ausgewählt (defineCommunicationLeader), der seinerseits mit seinem bevorzugten Kommunikationspartner eine Kommunikation (executeCommunication) vollzieht (OneObjectStep in ConcreteAgent, bei dem das Objekt zufällig ausgewählt wird).

Anschließend müssen die Schritte um eine Beschreibung ergänzt, der Agentenklasse drei und der Kantenklasse eine Methode zugeordnet und ein Verweis auf diese innerhalb der Schritte hinzugefügt werden. Zum Speichern der bevorzugten Kante wird in der Agentenklasse das Integer-Attribut preferredCommunicationEdge ergänzt. Die Stärke der Kanten wird über das in den Elternklassen bereits vorhandene Attribut strength beziehungsweise die zugehörigen get/set-Methoden gespeichert. Ein zusätzliches Attribut ist hier also nicht notwendig. Gleiches gilt für die Beliebtheit des Agenten, die in der Variable result der Elternklasse NetworkAgent abgespeichert wird. Die Stärke der Kanten und das Ergebnis eines Agenten werden somit bei der Ausgabe als Netzwerk oder Graph in der Benutzerschnittstelle berücksichtigt. Auf beide Attribute wird in einem Kommentar zu Beginn der Datei hingewiesen.

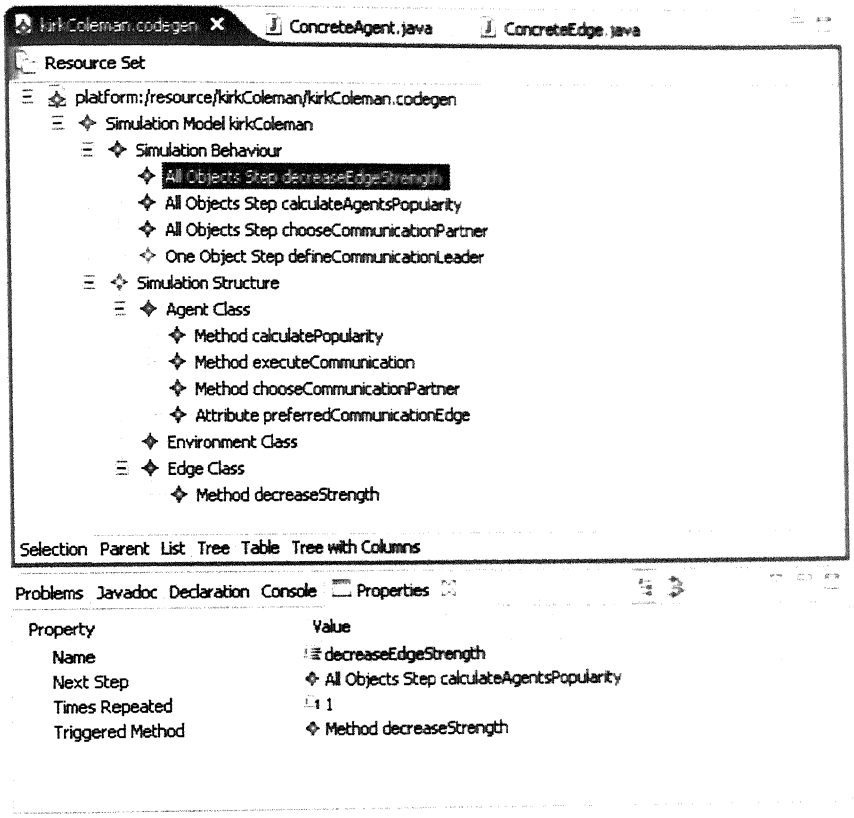

Abbildung 10: Plugin in Eclipse

Aus diesem graphischen Modell wird ein Coderahmen im „src"-Ordner generiert. Dieser muss um konkrete Befehle ergänzt werden. Ein Beispiel hierfür ist die Methode calculatePopularity:

```
/**
 * Calculates the Popularity (= result) by summing up all edge-weights
 */
public void calculatePopularity() {
        ArrayList outEdges = this.getOutEdges();
        this.result = 0;

        for(Iterator i = outEdges.iterator(); i.hasNext();) {
                this.result +=
        ((ConcreteEdge)i.next()).getStrength();
        }
}
```

In diesem Ausschnitt sind die kursiv dargestellten Teile automatisch erzeugt worden (Dokumentation und Methodenrahmen). Wenn die Methode aufgerufen wird, wird zunächst die Variable result auf Null gesetzt und anschließend durch die Addition der Gewichtungen aller auslaufenden Kanten neu berechnet.

Vom Scheduler, der in der Klasse NetworkModel enthalten ist. wird diese Methode durch folgenden, vollständig generierten Teil aufgerufen:

```
// calculateAgentsPopularity
case 2: {
      for (Iterator i = agentList.iterator(); i.hasNext();)
            ConcreteAgent networkAgent =
                  (ConcreteAgent) i.next();
            networkAgent.calculatePopularity();
      }
      break;
}
```

Dabei wird über alle Agenten iteriert und jeweils die Methode calculatePopularity() der Agenten aufgerufen.

Anschließend kann die Simulation über den „Run"-Button von Eclipse gestartet und über den „Run"-Button von Repast ausgeführt werden. Eine entsprechende Darstellung findet sich in Abbildung 11.

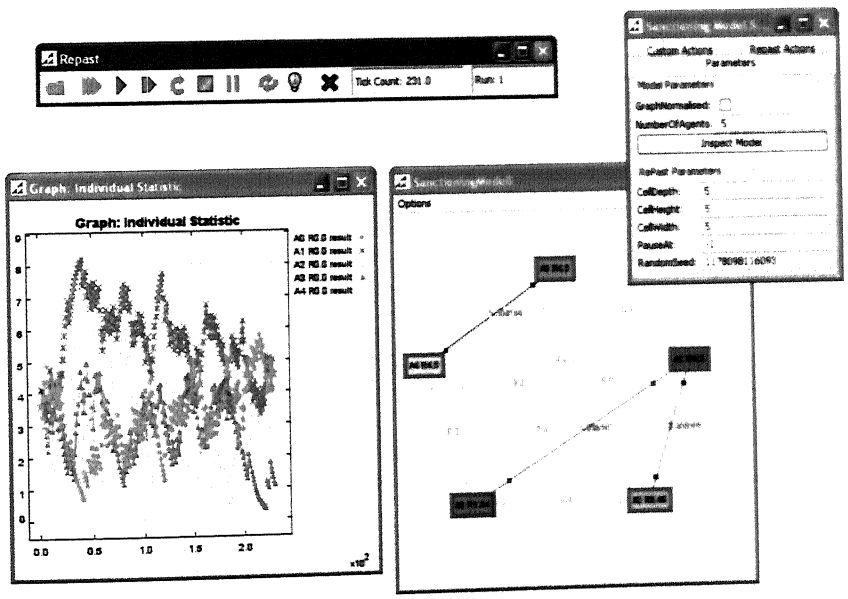

Abbildung 11: Repast mit Kirk-Coleman-Modell

4.5 Visualisierung einer großen Anzahl von Agenten

Neben der in Abbildung 14 auf Seite 41 gezeigten Darstellung mittels Graphen können in Repast Kanten und Knoten visualisiert werden. Ein Beispiel hierfür ist Abbildung 5 auf Seite 26, die sich auf das Kirk-Coleman-Modell bezieht.

Die Standarddarstellung der Netzwerkknoten, die sich insbesondere für eine ausführliche Darstellung weniger Agenten eignet, ist für Simulationen mit 10 oder mehr Agenten ungeeignet, da sie dabei wegen der zahlreichen Verbindungen und der dazugehörigen Verbindungsbeschreibungen unübersichtlich wird. Deshalb sind andere Darstellungsformen notwendig.

Denkbar wäre eine Darstellung der Agenten als Kreis und der zugehörigen Informationen über die Dicke des Randes oder die Größe oder Farbe des inneren Kreises.

Eine weitere Darstellungsmöglichkeit ist über die Position der Agenten gegeben. In Abbildung 12 sind Bilder aus einer Simulationsdurchführung des in Kapitel 5 beschriebenen Sanktionierungsmodells nach der ersten, fünfzehnten und dreißigsten Runde zu sehen. Die Agenten werden an der x-Achse nach ihrer Investitionsfreudigkeit und an der y-Achse nach ihrer Sanktionierungsfreudigkeit ausgerichtet. In der ersten Runde sind die Agenten noch relativ gleichmäßig verteilt. Bis zur Runde 15 vereinheitlicht sich das Investment auf einen Mittelwert, während die Sanktionierungsfreudigkeit bereits deutlich abnimmt. Bis zur Runde dreißig ist das Investment fast aller Agenten in diesem Simulationsdurchlauf bereits bis auf Null abgefallen.

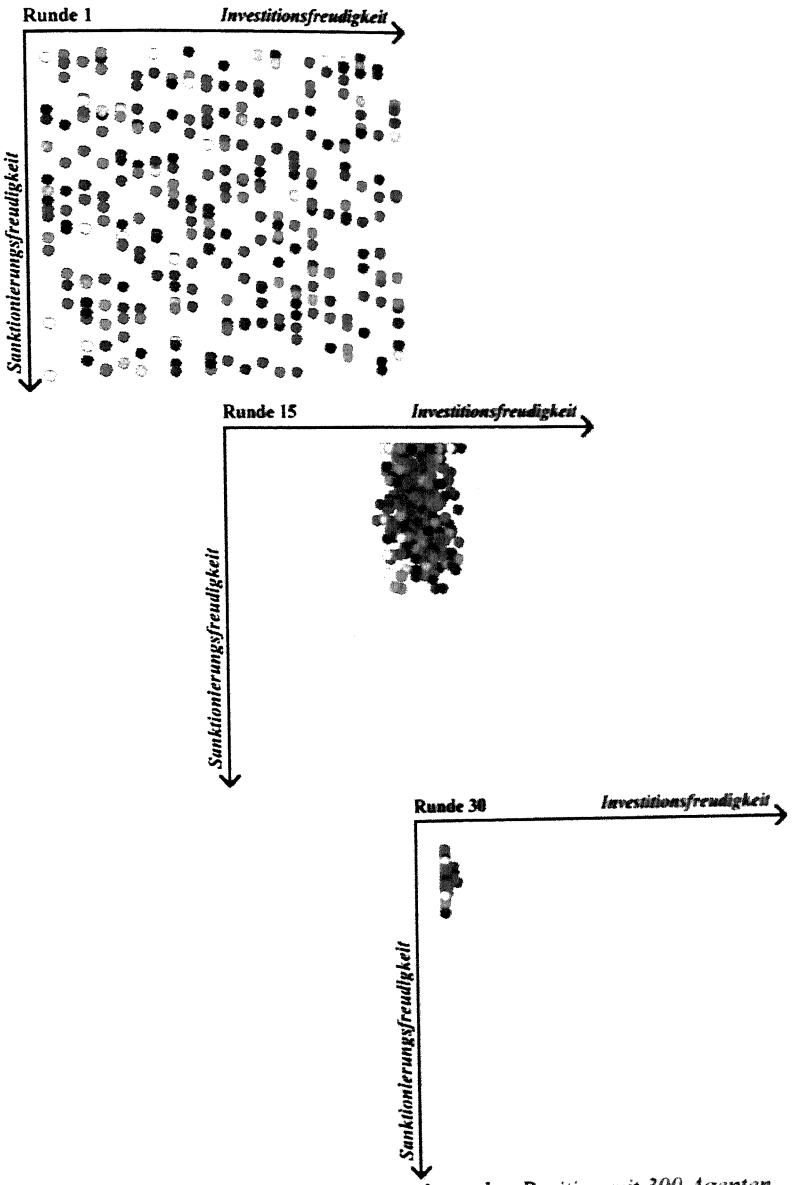

Abbildung 12: Alternative Darstellungsform über Position mit 300 Agenten

5 Anwendungsbeispiele

Um einen tieferen Einblick in die Simulation sozialwissenschaftlicher Prozesse zu erhalten, ist neben der informatiknahen Sicht auch eine sozialwissenschaftliche Sichtweise erforderlich. Dementsprechend werden in diesem Abschnitt gemäß der Anforderungsanalyse zwei Beispiele nicht nur implementiert, sondern mit Hilfe einiger Tests justiert und mögliche Varianten diskutiert. Im Hinblick auf das Projekt EMIL wird dabei auf die Voraussetzungen zur Entstehung von Normen eingegangen. Die Entwicklung dieser Anwendungsbeispiele fand hierbei parallel zu dem vorherigen Kapitel statt, sodass sich Ergebnisse aus beiden Teilen gegenseitig befruchten konnten.

5.1 Kirk-Coleman-Modell

Das Simulationsmodell von Kirk und Coleman (folgend Modell) wurde in [Troi99] aufgegriffen und um die Untersuchung von Fünf-Personen-Gruppen erweitert. Im Rahmen dieser Diplomarbeit wird das grundlegende Modell reimplementiert, um die Möglichkeit der Untersuchung von n-Personen-Gruppen sowie um weitere Änderungen und Erweiterungen der zu Grunde liegenden Regeln ergänzt.

5.1.1 Standardmodell

Das grundlegende Modell besteht aus drei oder mehr Personen, die mit den anderen Beteiligten in Beziehung stehen. In jeder Runde wird eine Person zufällig ausgewählt (Kommunikationsführer). Diese entscheidet mit wem sie kommunizieren möchte (Kommunikationspartner) und bevorzugt dabei die Person, zu der sie die stärkste Beziehung hat. Durch eine zu Stande gekommene Kommunikation verstärkt sich die Beziehung zwischen den beiden beteiligten Personen. In der Implementierung wird dies mit Hilfe einer Addition um einen Summanden (increaseSummand) umgesetzt. Zu Beginn jeder Runde wird zudem die Beziehung um einen Faktor kleiner Eins (decreaseFactor) abgeschwächt, sodass weit zurück liegende Kommunikationsvorgänge die Beziehung weniger beeinflussen als näher in der Gegenwart liegende. Die Agenten agieren hierbei ziellos, sodass sich ihr Verhalten nicht ändert.

In Abbildung 13 findet sich ein möglicher stabiler Zustand (RandomSeed ist hier 5) der Simulation mit Standardwerten. Im konkreten Fall bilden sich eine 3er-Gruppe sowie eine 2er-Gruppe. Das Simulationsmodell zeigt bei jedem Simulationsverlauf mit fünf Agenten dieses Verhalten. Agent 1 ist, wie in Abbildung 14 zu sehen ist, aufgrund seiner beiden Beziehungen die Person mit der höchsten Beliebtheit.

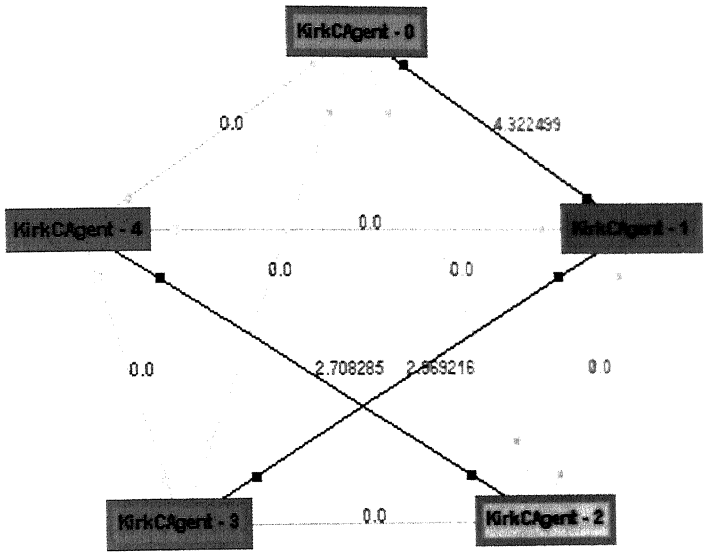

Abbildung 13: Netzwerk-Darstellung mit Standardregeln (KC)

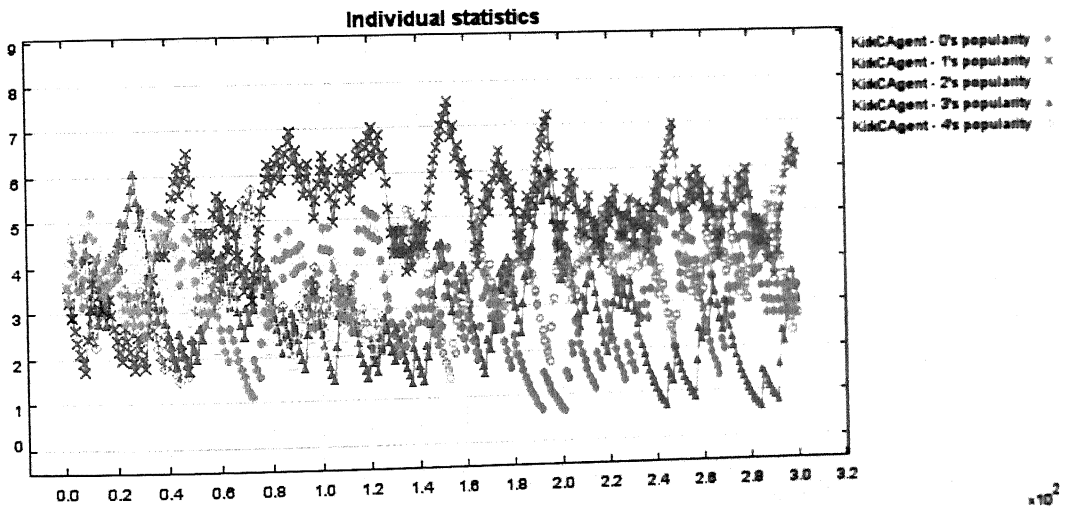

Abbildung 14: Graph-Darstellung mit Standardregeln (KC)

Bei sechs Personen ist die Anzahl möglicher Strukturen bereits größer. Häufig (in mehr als 10 % der Fälle) bilden sich drei Paare oder zwei 3er-Gruppen. Deutlich seltener ein Paar und eine 4er-Gruppe mit einer Person als Hub, die Verbindungsperson zu den anderen drei Personen ist. Dieser seltene Fall ist beispielsweise in Abbildung 15 (hier mit RandomSeed 1172827872281) zu finden.

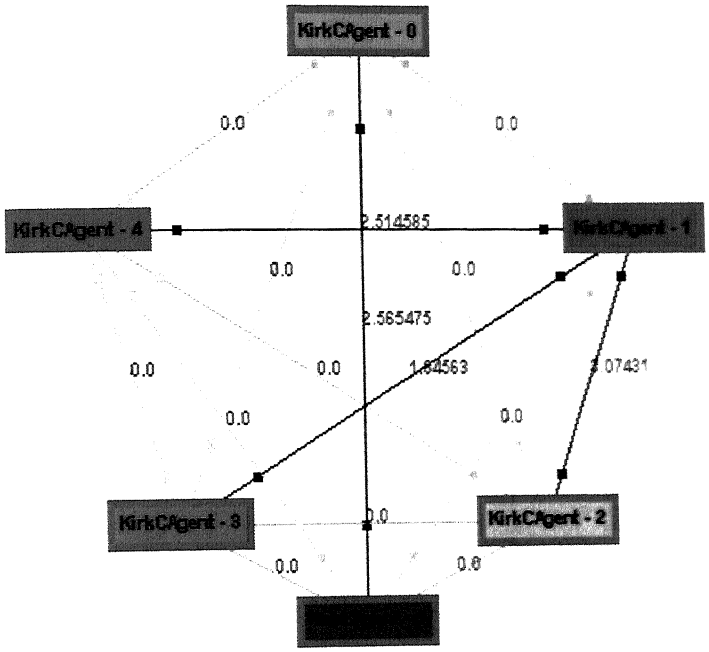

Abbildung 15: 6 Personen incl. 4er-Gruppe (KC)

Bei einer größeren Anzahl an beteiligten Personen ist eine ähnlich strukturierte Gruppenbildung zu erwarten. Eigene stichprobenartige Tests mit 7 bis 15 Personen bestätigen diese Annahme. Es bilden sich zum einen Paare und zum anderen Gruppen mit 3, seltener mehr Personen, bei denen nur eine Person mit allen anderen Personen verbunden ist. Die anderen Personen der Gruppe kommunizieren ihrerseits nur mit diesem Vermittler. „Dreiecks"-Beziehungen, in denen alle drei Paarbeziehungen zwischen den drei Agenten der Gruppe vorhanden sind, bleiben nicht stabil, da die schwächste Verbindung zunehmend vernachlässigt wird.

5.1.2 Erweiterungen

Zur tiefergehenden Untersuchung des Kirk-Coleman-Modells wurden einige Erweiterungen implementiert, von denen einige interessant erscheinende hier vorgestellt werden.

Wird zusätzlich zur Wahl des Kommunikationspartners auch die des Kommunikationsführers der jeweiligen Runde nicht zufällig, sondern an die jeweilige Beliebtheit gekoppelt getroffen, entwickeln sich zunächst meist zwei starke Paare. Bei fortschreitender Simulation behält eines der beiden Paare die Oberhand und wird zunehmend stärker. Die anderen drei Personen haben so nicht mehr die Chance Kommunikationsführer zu werden. Eine möglicher stabiler Zustand findet sich in Abbildung 16.

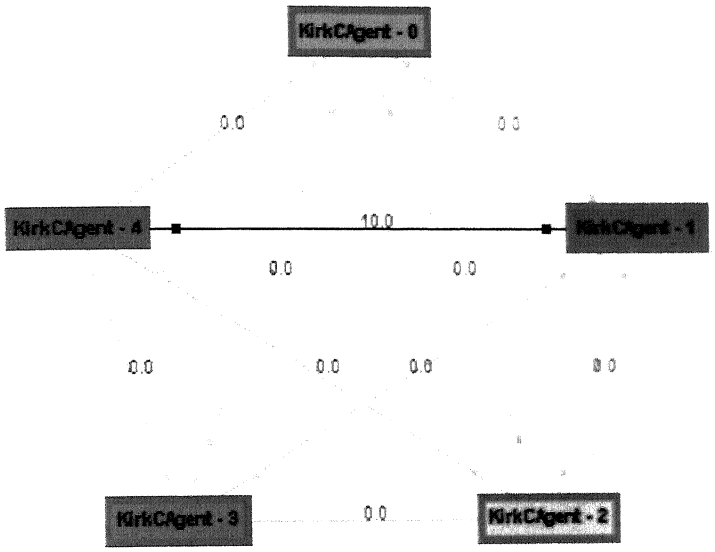

Abbildung 16: Kommunikationsführer nach Beliebtheit (KC)

Eine weitere mögliche Änderung ist die Ergänzung eines „Gehirns" über ein neuronales Netz. In der Implementierung wurde dazu in der Klasse AgentBrain das neuronale Netz aus dem „XOR"-Beispiel[30] des Frameworks Joone beziehungsweise dem technisch gleich aufgebauten Beispiel „Neural Office" aus Repast als Grundlage verwendet.

Mögliche Strategien waren:

- Je schwächer die Beziehung, desto eher wähle ich die Person als Kommunikationspartner.
- Je stärker die Beziehung, desto eher wähle ich die Person als Kommunikationspartner (Standardregel).

Das Ziel jeder Person ist es, möglichst beliebt zu sein. Als Eingabewert des neuronalen Netzes wurde die Beliebtheit genommen. Wenn die Beliebtheit seit der letzten Kommunikationsführerschaft gestiegen ist, wurde die Strategie beibehalten, andernfalls gewechselt.

Die Simulationen haben ergeben, dass das neuronale Netz die Personen dazu anleitet, über längere Zeit eine bestimmte Strategie zu wählen. Damit bewirkt es eine gewisse Stabilität in ihrer Entscheidung.

30 In diesem Beispiel muss das neuronale Netz eine „XOR"-Funktion an Hand von Beispielen lernen. Das verwendete neuronale Netz besteht dabei aus einer Eingabeschicht mit zwei Neuronen, einer versteckten Schicht mit drei Neuronen sowie einer Ausgabeschicht mit einem Neuron. Eine genauere Beschreibung des Beispiels findet sich in [Mar07].

Die gewählte Strategie bei der Kommunikation hat direkt keine Auswirkungen auf die Beliebtheit, da diese für die unmittelbar auf eine Kommunikationsführerschaft folgende Runde in beiden Fällen wie folgt berechnet werden kann:

$$b_{t+1} = b_t + 0{,}9$$

Somit gibt es nur indirekte, längerfristige Einflüsse durch eine mögliche Beeinflussung der Strategie anderer Agenten, die zu unscharf für das neuronale Netz sind, da die Simulation in mehreren Dimensionen vom Zufall abhängig ist. Deshalb reagiert das neuronale Netz völlig unabhängig vom Eingabewert Beliebtheit.

Auch zusätzlich eingeführte Regeln haben hier keine Veränderung bewirkt:

- Erstes Beispiel hierfür ist die Herabsetzung des Beziehungsverfalls bei kleinen Werten[31]. Diese Regel müsste die oben zuerst genannte Strategie, schwächere Beziehungen zu stärken, begünstigen.
- Zweites Beispiel ist die Einführung eines Freundschaftsfaktors bei gegenseitiger Wahl als Wunschpartner und überdurchschnittlich starker Beziehung. Diese Regel müsste die oben als zweites genannte Strategie, stärkere Beziehungen zu stärken, begünstigen.

Letztere Regel ergab starke Beziehungen über einen längeren Zeitraum, von denen beide Agenten profitierten. Nach einiger Zeit verschwanden diese wieder. Das getestete neuronale Netz änderte die Ergebniswerte aber auch hier wegen unzureichender direkter Einflüsse unabhängig vom Eingabewert. Wenn sowohl Anwendung als auch Training in die Simulation eingebettet werden, ist es deshalb für das Kirk-Coleman-Modell und die vorgestellten Varianten aufgrund dessen starker Zufallsabhängigkeit ungeeignet. Wird das neuronale Netz vor Beginn der Simulation mit idealen, fehlerfreien Werten[32] trainiert, so differenziert es die Ergebnisse abhängig vom Eingabewert.

Bei sehr langsamem Verfall der Beziehungen bei einer Rate von 0,99 hat sich ein weiterer interessanter Effekt ergeben. So ist bei gleich bleibendem Summanden zum einen die absolute Stärke der Beziehungen größer, zum Zweiten bilden sich nach einer langen Simulationszeit von 20.000 Runden wieder eine 2er- und eine 3er-Gruppe. Im Gegensatz zum Verlauf im grundlegenden Modell mit einer Rate von 0,9 bleiben alle drei Beziehungen in der 3er-Gruppe erhalten. Zudem gibt es keine Person, die dauerhaft beliebter ist als alle anderen Beteiligten der Gruppe. Der Zustand nach 20.000 Durchläufen (RandomSeed 1172840479515) findet sich in Abbildung 17.

31 Die Idee dabei ist, dass Menschen in der Realität auch nach vielen Jahren völlig ohne Kontakt manche Personen nicht vergessen.

32 In dem XOR-Standardbeispiel von Joone beziehungsweise dem Neural-Office-Beispiel von Repast wird das Training auch mit fehlerfreien Werten durchgeführt.

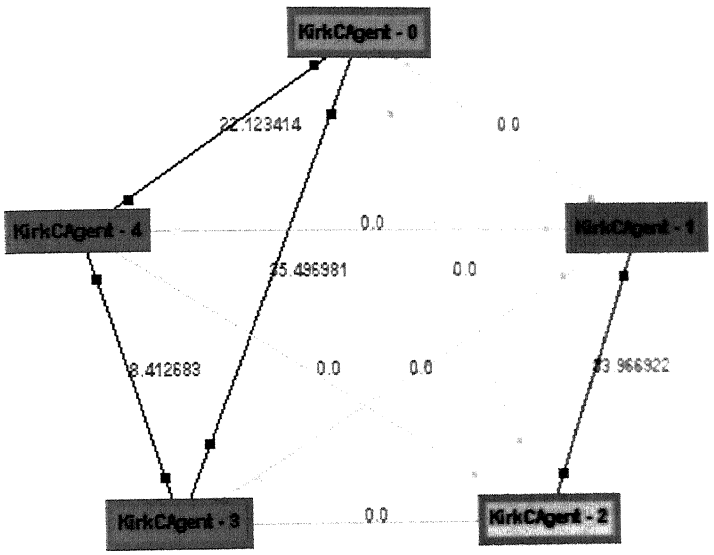

Abbildung 17: Runde 20.000 bei langsamen Beziehungsverfall (KC)

5.1.3 Interpretation der Ergebnisse und Ausblick

Kommunizieren Menschen aufgrund der Annahme „je stärker die Beziehung zu einer Person, desto eher kommuniziere ich mit ihr", dann gibt es in der Regel Personen, die die Gruppeninteraktion dominieren. Dominante Personen in Gruppen können auch in der Realität beobachtet werden, wohingegen die Beziehungen zwischen den anderen Gruppenmitgliedern nach einer gewissen Zeit durchaus noch existent sein können. Um eine realistischere Simulation zu erhalten, könnte es helfen, den Faktor zum Abschwächen der Beziehungen näher an 1,0 zu wählen.

Bei Einführung eines „Freundschaftsfaktors" bilden sich länger andauernde, stärkere Freundschaften, die nach einiger Zeit wie in der Realität verfallen. Der Verfall tritt auf, wenn beide Freunde längere Zeit nicht mehr Kommunikationsführer waren. In der Realität lässt sich das auf die Tatsache abbilden, dass Personen nach einer starken Freundschaft weniger Zeit für soziale Beziehungen haben und sich später neu orientieren. Die Gründe für die Bildung und den Verfall von Freundschaften sind in der Realität jedoch vielschichtiger als ein Freundschaftsfaktor und vom Zufall bestimmte Kommunikationspausen.

In den Simulationsdurchläufen ist die Norm, eher starke als schwache Beziehungen zu pflegen, erkennbar. Auch diese Beobachtung ist in der Realität möglich. Die Norm bildet sich in der Simulation nicht emergent, sondern ist indirekt durch die zu Grunde liegenden Regeln vorgegeben. Sollen die Agenten immergent über ein neuronales

Netzwerk die Beziehungsstruktur verstehen, sind neben einer sinnvollen Lernregel weniger vom Zufall abhängige Simulationsdaten notwendig.

Im Verlauf der Experimente ließen sich mit kleinen Änderungen neue und interessante Phänomene erzeugen, sodass das Kirk-Coleman-Modell auch 40 Jahre nach seiner Veröffentlichung Potential für Erweiterungen erwarten lässt. So ist eine Änderung des Graphen, zum Beispiel durch Weglassen einiger Verbindungen, denkbar. Durch komplexere Regeln und intelligentere Agenten wäre auch komplexeres Verhalten möglich. Sozial schwache Agenten könnten zusammen die Vormachtstellung der starken Personen brechen, indem sie in gegenseitigem Einvernehmen neue Untergruppen bilden.

Mit dem Blick auf die Möglichkeit, das Kirk-Coleman-Modell als konzeptionelle Grundlage für ein Beziehungsverhältnis zwischen Menschen in einer größeren Simulation zu nutzen und der Existenz weiterer darauf basierender Veröffentlichungen[33] hat es zudem nicht an Aktualität verloren.

5.2 Einfluss von Sanktionen und Normen auf Kooperation

Bei dem in [Gür06a] beschriebenen Experiment nahmen 84 Studenten der Universität Erfurt teil. Das Spiel dauerte über 30 Perioden an. Jede Periode bestand aus drei Teilen[34]:

- Die Teilnehmer konnten sich zunächst für die Teilnahme an einer von zwei Organisationen entscheiden. In der einen Organisation war es möglich, das Verhalten der anderen Teilnehmer zu sanktionieren, in der anderen nicht.
- Nach Auswahl der Organisation konnten die Teilnehmer 0 bis 20 virtuelle Geldeinheiten in ein gemeinsames Projekt investieren. Der Ertrag der Gruppe aus diesem Projekt errechnet sich durch den Einsatz aller Teilnehmer multipliziert mit dem Faktor 1,6 und wird direkt nach der Entscheidung aller Teilnehmer an diese zu gleichen Teilen verteilt. Das jeweils individuell behaltene, nicht investierte Geld wird jedem Teilnehmer neben dem Ertrag der Gruppe gutgeschrieben.
- Im letzten Schritt hat jeder Teilnehmer in der Organisation mit Sanktionierung die Möglichkeit, weitere bis zu 20 Geldeinheiten zur Sanktionierung beliebig vieler anderer Teilnehmer einzusetzen: *„[...] each player in SI may positively or negatively sanction other members of SI [...]"* [Gür06a]. Setzt ein Teilnehmer eine Geldeinheit zur negativen Sanktionierung ein, so verliert der sanktionierte Teilnehmer drei Geldeinheiten. Beide Geldmengen werden aus dem Spiel

33 Siehe zum Beispiel [Schw04]
34 Eine ausführliche Beschreibung des Simulationsablaufs findet sich in [Gür06b]

genommen. Bei der positiven Sanktionierung verliert der sanktionierende Teilnehmer eine Geldeinheit, die der sanktionierte gutgeschrieben bekommt. Der nicht eingesetzte Anteil (bei der Organisation ohne Sanktionierung also alle 20 Geldeinheiten) verbleibt im Besitz des jeweiligen Teilnehmers.

Nach Abschluss aller Runden wurde den Teilnehmern für Ihre im Spiel erworbenen Geldeinheiten reales Geld ausgezahlt: *„At the end of the experiment your total payoff will be converted into Euro with an exchange rate of 1 € per 100 tokens."* [Gür06b] Nach dieser Regel bekommt jeder Teilnehmer abhängig von seinen Geldeinheiten und unabhängig vom Erfolg der anderen Teilnehmer einen Geldbetrag.

In den Veröffentlichungen [Gür06a] sowie [Gür06b] sind einige Statistiken enthalten. An Hand dieser ist erkennbar, dass zu Beginn nur ein Drittel, zum Ende jedoch über 90 Prozent der Teilnehmer die Organisation mit Sanktionierung gewählt haben. Dies ist begründet durch den auch in der Organisation ohne Sanktionierung beobachtbaren Erfolg der Teilnehmer in der Organisation mit Sanktionierung. Ab Periode 15, also der Mitte des Experiments, erzielten die Teilnehmer in der Organisation mit Sanktionierung fast immer 50 oder mehr Geldeinheiten. 40 Geldeinheiten entspricht dem Gewinn ohne Kooperation und Sanktionierung. Beide letztgenannten Beobachtungen finden sich in Abbildung 18.

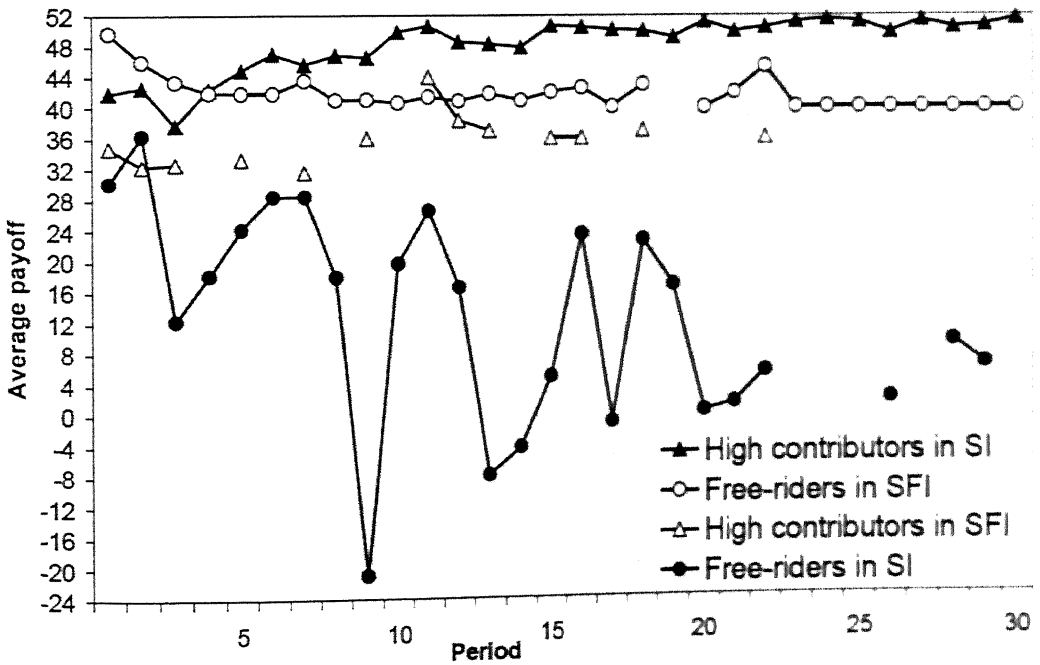

Abbildung 18: Ergebnis des Experiments; Grafik aus [Gür06b] (Sanktionierung)

5.2.1 Umsetzung als Simulation

Da dieses Experiment grundsätzlich Normenbildung ermöglicht, wird es hier zunächst formalisiert und anschließend genauer untersucht. Ideen für die technische Realisierung einer Simulationsumgebung zur Simulation des Entstehungsprozesses von Normen sind bereits in die vergangenen Kapitel eingegangen und werden hier nicht weiter erläutert.

Die Regeln des Experiments sind in [Gür06a] sowie [Gür06b] exakt beschrieben. Da demgegenüber die Regeln, nach denen die Personen im Experiment interagiert haben, nicht explizit aus den Ergebnissen des Experiments hervorgehen und deshalb verständlicherweise auch nicht detailliert beschrieben sind, wird das Simulationsmodell zur Verminderung der Komplexität zunächst vereinfacht.

Zum ersten wird eine Dimension der Entscheidung der Teilnehmer, nämlich die Möglichkeit des Gruppenwechsels, nicht ins Simulationsmodell übernommen. Eine Runde besteht deshalb nur noch aus der Auswahl des Investments sowie der Bestrafung. Im vereinfachten Standardfall gibt es zudem nur eine Organisation. In einer Erweiterung sind zwei Organisationen vorhanden, wobei sich die Teilnehmer, wie im Experiment, über das Ergebnis der anderen Gruppe informieren können.

Zudem wird auf die Möglichkeit der positiven Sanktionierung verzichtet, da diese zum einen weniger genutzt wurde als die negative Variante und zum Zweiten einen eher schwachen Effekt hatte: *„1,66 negative sanction points (SE = 0.60) are allocated per positive sanction point. [...] positive sanctions have a slightly negative but rather insignificant effect."* [Gür06a].

Die technische Umsetzung erfordert weitere Aktivitäten, die in Abbildung 19 zu den beiden oben genannten hinzugefügt wurden. Nach der Festlegung des Investments eines jeden Agenten wird auf Modellebene der individuelle Ertrag aus dem gemeinsamen Investment berechnet. Dieser wird im darauf folgenden Schritt dann zu dem nicht eingesetzten Kapital hinzu addiert. Nach dem zweiten Schritt, in dem das Ausmaß der Bestrafung nicht kooperativer Agenten festgelegt wird, berechnen die Agenten ihr Ergebnis der gesamten Runde. Folgend wird das Gruppenergebnis aller an der Simulation beteiligten Agenten berechnet. Abschließend verändert der Agent seine Regeln mit Hilfe einer Lernregel.

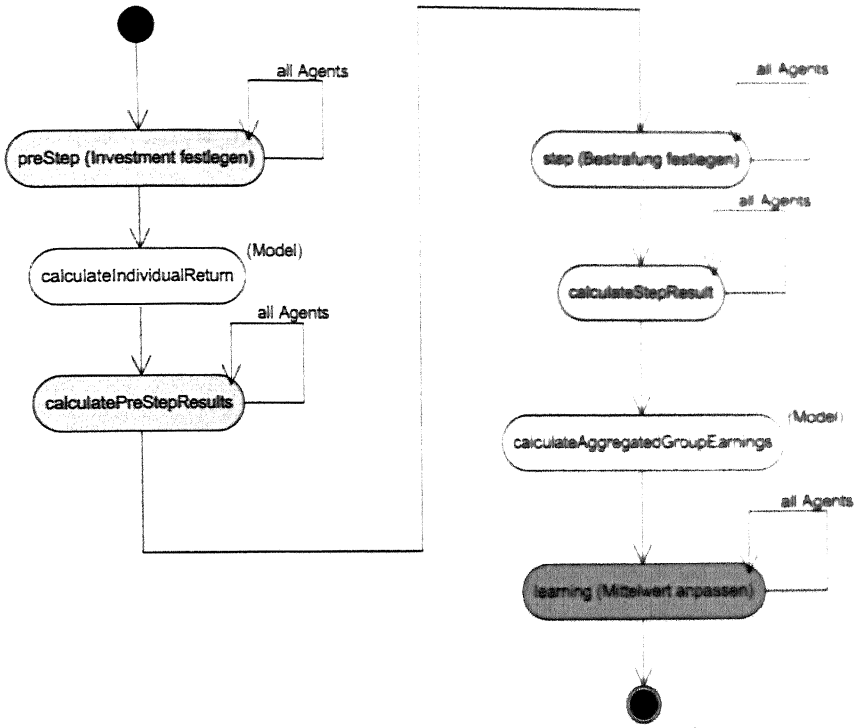

Abbildung 19: Aktivitätsdiagramm zu einem Simulationsschritt (Sanktionierung)

Maßgeblich für den Verlauf der Simulation ist das individuelle Verhalten der Agenten beim Investieren, Bestrafen sowie bei der Anpassung der Regeln.

Die erfolgreichste Strategie wäre, genug zu investieren, um der Strafe zu entgehen. Gleichzeitig sollte der Teilnehmer sich bei der Bestrafung zurückhalten, da diese persönlich einen Verlust darstellt und unter Umständen zwar einen Vorteil gegenüber anderen Gruppen verschafft, nicht-bestrafende Teilnehmer aus der eigenen Gruppe mit gleich hoher Investition aber bevorzugt: „*The most successful behavior would be to contribute in SI and hence avoid being punished, but refrain from the costly punishment of others*" [Gür06a].

Da die Veröffentlichungen [Gür06a] und [Gür06b] nicht im Detail auf das Verhalten der beteiligten Personen eingehen, werden hier zunächst einige Grundannahmen gemacht.

Aufgrund des Experiments und dessen Beschreibung in [Gür06a] wird angenommen, dass die zunächst nicht rational erscheinende Möglichkeit zur Bestrafung Verwendung findet. Dies ist der Fall, da sie zum einen längerfristig durch den resultierenden Lerneffekt für den Einzelnen im Vergleich zu Teilnehmern anderer Organisationen und für die Organisation als Ganzes rational sinnvoll ist. Zum Zweiten ist es ein bei Menschen bekanntes Verhalten, das auf dem Gerechtigkeitsgefühl beruht. Das Experiment hat

ergeben, dass einige Teilnehmer zu Beginn bestrafen und sich dauerhaft ausreichend viele Teilnehmer anschließen, um einen kooperationsfördernden Effekt zu erzielen: *„This results in a quite stable proportion of ~40% (mean = 42.1%, SE = 5.9%) of subjects who both contribute highly punish during the last 20 periods"* [Gür06a]

Weiterhin wird angenommen, dass die Neigung zur Bestrafung dadurch verstärkt wird, dass sich auch andere Teilnehmer daran beteiligen: *„Hence, group members punish because it is common to do so."* [Gür06a] Durch die Nachahmung des für die Gruppe sinnvollen Verhaltens anderer bildet sich hier eine Art „Norm", die der rational-egoistischen Handlungsweise teilweise entgegen steht.

Zudem wird angenommen, dass die Agenten, die mehr als andere Agenten investiert haben, diesen im Durchschnitt in der Größenordnung Geldeinheiten entziehen möchten, in der sie in der ersten Runde schlechter als diese gewesen sind. Bei Organisationen mit mehr als zwei Teilnehmern wird das Engagement in der Bestrafung Einzelner der Gruppenzahl angepasst. Das heißt, dass zum Beispiel bei drei Teilnehmern für jeden anderen nicht-kooperativen Agenten nur die Hälfte an Geldeinheiten zur Bestrafung aufgewendet wird, da sich vermutlich mehrere Teilnehmer an der Bestrafung von Trittbrettfahrern beteiligen.

Technisch realisiert wird die Wahl des Investments sowie der Sanktionierung über die Fließkommavariablen averageInvestment und averagePunishment. Die Entscheidung eines Agenten für ein konkretes Investment beziehungsweise einer konkreten Sanktionierung geschieht zufällig als Normalverteilung mit dem Mittelwert der oben genannten Variablen innerhalb der erlaubten Spanne von 0 bis 20. Als Standard-abweichung wird 10 % der möglichen Spanne, beim Investment also beispielsweise 2, gewählt.

Das Lernen erfolgt über die Anpassung der beiden Variablen averageInvestment sowie averagePunishment. Eine streng egoistische Lernregel ist zum Beispiel die Anpassung in Richtung des besten Agenten der aktuellen Runde.

5.2.2 Standardmodell

Das grundlegende Modell besteht aus fünf beziehungsweise beliebig vielen Personen, die gemeinsam einer Organisation zugeordnet sind. Ein Simulationsschritt enthält die im vorigen Kapitel beschriebenen Aktivitäten eingeschlossen der Möglichkeit zu sanktionieren. Während sich das individuelle durchschnittliche Investment eines Agenten verändert, bleibt der Faktor für die Bestrafung unverändert. Das konkrete Investment in einer Runde wird mit einer Zufallszahl aus einer Normalverteilung um das durchschnittliche Investment bestimmt.

In den einzelnen Teilaktivitäten eines Simulationsschrittes werden die getroffenen Annahmen im Standardmodell wie folgt umgesetzt:

- Zuerst legt jeder Agent ein konkretes Investment fest. Dieses wird von ihm als Zufallswert aus einer Normalverteilung um den Wert des durchschnittlichen Investments mit einer Standardabweichung von 2,0 (10 % aus der erlaubten Spanne von 0 bis 20) gewählt.
- Im zweiten Teil legt jeder Agent einen positiven Wert größer oder gleich Null für die konkrete Bestrafung der anderen Agenten fest. Der durchschnittliche Faktor zur Bestrafung bleibt im vereinfachten Standardmodell konstant bei 1,0. Dieser wird mit dem Wert aus der im vorigen Kapitel beschriebenen Regel multipliziert. Die konkrete Bestrafung eines Agenten ist dann ein Zufallswert aus einer Normalverteilung um das berechnete Ergebnis mit einer Standardabweichung von 0,2 (10 % aus der erlaubten Spanne von 0,0 bis 2,0).
- Im letzten Teil verändert jeder Agent seine Durchschnittswerte für das Investment in einer Lernregel. Im Standardmodell wird von einer egoistischen Lernregel ausgegangen. Hierbei wird der eigene Durchschnittswert auf Grundlage des konkreten Wertes des besten Agenten angepasst. Zu dem eigenen Durchschnittswert der auslaufenden Runde wird der Unterschied zwischen dem konkreten Wert des besten Agenten und dem eigenen konkreten Wert der auslaufenden Runde, dividiert durch einen Divisor, abgezogen beziehungsweise addiert. Nach eigenen Tests hat sich ergeben, dass ein Divisor mit dem Wert fünf verglichen mit den Ergebnissen des Experiments ähnliche Resultate produziert. D.h., dass pro Runde der durchschnittliche Wert ungefähr 20 % in die Richtung des besten Agenten angepasst wird.

Die Simulationsergebnisse sind denen aus dem Experiment ähnlich. Wenn es genügend kooperative Teilnehmer gibt, d.h. Teilnehmer, die mit einem hohen Durchschnittswert für die Investition starten, etabliert sich nach einer Phase starker Bestrafung Kooperation. In Abbildung 20 ist ein möglicher Simulationsverlauf zu finden, bei dem zu Beginn drei der fünf Agenten überdurchschnittlich viel Einsatz zeigen. Das durchschnittliche Investment eines jeden Agenten vor dem Start der Simulation findet sich auf der rechten Seite der Abbildung. Das durchschnittliche Investment aller Agenten liegt bei 12,8. Eine hohe Beteiligung zu Beginn wurde in der Organisation mit Sanktionierung auch im Experiment beobachtet: *„Participants who initially join SI contribute on average 12.7 MUs [...]"* [Gür06a].

Bei dem Beispiel in Abbildung 20 stellt sich Kooperation bereits nach kurzer Zeit ein. Das durchschnittliche Rundenergebnis aller Agenten beträgt zum Ende der Simulation konstant über 45, was nur wenig unter den beobachteten Ergebnissen von mindestens 49

liegt. Erst ab Runde 300 erzielen die Agenten ein durchschnittliches Ergebnis, das auch konstant über 49 ist.

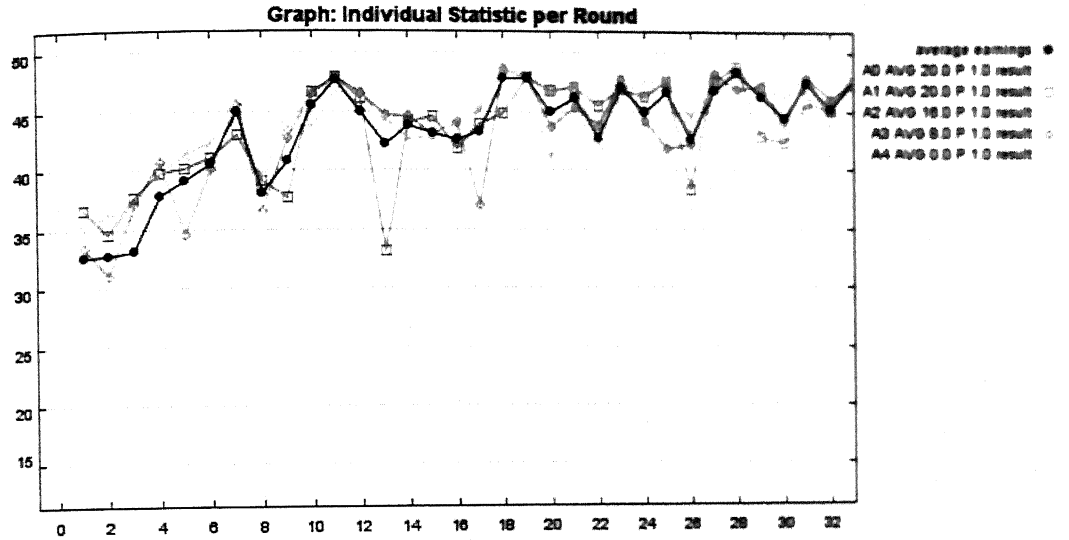

Abbildung 20: Ergebnis Standardregel (Sanktionierung)

Da die Annahme, dass verschiedene Personen eine gleich große Motivation zum Bestrafen haben, unrealistisch ist, wird folgend auch der Startwert der durchschnittlichen Bestrafung bei verschiedenen Agenten auf verschiedene Werte gesetzt. Zudem wird im dritten Schritt nicht nur das durchschnittliche Investment, sondern auch der Faktor für die durchschnittliche Neigung zur Bestrafung verändert. Dies geschieht im Standardmodell nach den oben für das Investment beschriebenen Regeln, ist also auch egoistisch motiviert.

In Abbildung 21 ist ein möglicher Simulationsverlauf zu finden, bei dem die Ausgangswerte für das durchschnittliche Investment dem der Simulation aus Abbildung 20 entsprechen, den Agenten jedoch unterschiedliche Startwerte für die Bestrafung der anderen vorgegeben werden. Während sich zwei Agenten sehr stark und einer stark beteiligen, gibt es zwei, die sich nicht beteiligen.

Das Ergebnis dieser Simulation kommt dem Ergebnis des Experiments wesentlich näher. Schon nach wenigen Runden mit starker Bestrafung etabliert sich Kooperation, die ab Runde sieben konstant Werte über 45, ab Runde 13 um 50 liefert.

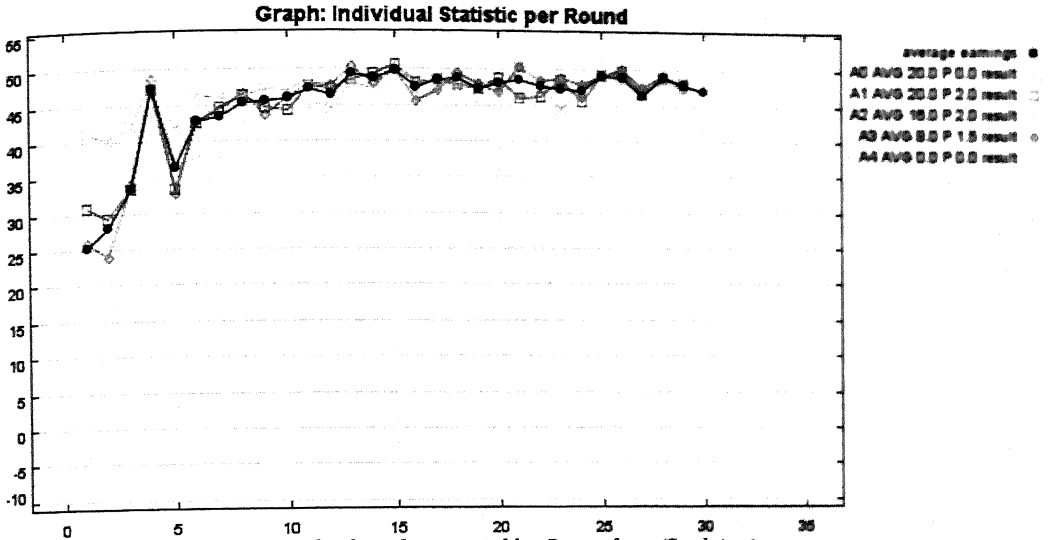

Abbildung 21: Ergebnis Standardregel mit variabler Bestrafung (Sanktionierung)

Das Ergebnis bei einer höheren Anzahl von Teilnehmern ist ähnlich, erreicht aber bei zunehmender Teilnehmerzahl wegen der höheren Wahrscheinlichkeit von Agenten, die von dem Durchschnitt abweichen, und dem kleineren direkten Effekt von Bestrafung nicht ganz den Wert 50 als durchschnittliches Ergebnis. In Abbildung 22 findet sich das Ergebnis eines Simulationsdurchlaufs mit 15 Teilnehmern. Die durchschnittlichen Startwerte bezüglich Sanktionierung und Investment sind, verglichen mit den bisher beschriebenen Simulationsdurchläufen, ähnlich.

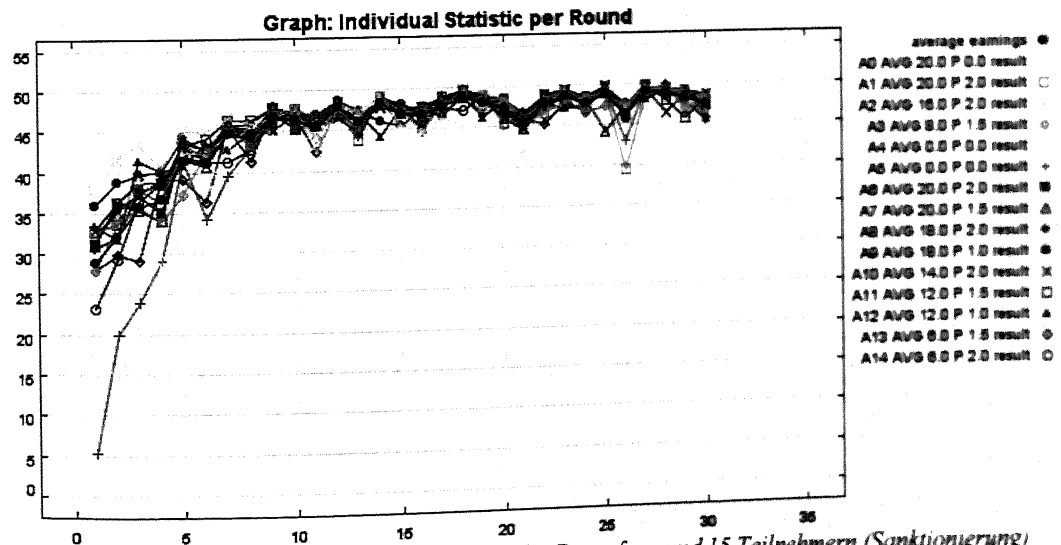

Abbildung 22: Ergebnis Standardregel mit variabler Bestrafung und 15 Teilnehmern (Sanktionierung)

5.2.3 Erweiterungen

Auch das Standardmodell mit egoistischer Lernregel und variablem Bestrafungsfaktor ist trotz im Vergleich zum Experiment ähnlicher Ergebnisse nicht voll zufriedenstellend. Während einer längeren Simulation über 30 Runden hinaus schwächt sich die Kooperation zunehmend wieder ab. Zum Ende investieren alle Agenten keine oder nur wenige Geldeinheiten. Der Grund liegt bei der Orientierung am besten Agenten, der in der Regel ein relativ hohes Investment, aber auch einen niedrigen Bestrafungsfaktor hat. Damit sinkt zunächst die Bestrafung und deshalb einige Runden später auch das Investment. Neben einem möglichen Lösungsansatz wird in diesem Unterkapitel zunächst auf realisierte Erweiterungen und Änderungen des Standardmodells eingegangen. Alle Veränderungen sind im Simulationsprogramm über einen Parameter in die Repast-GUI integriert worden.

Sind die konkreten Werte von Investment und Bestrafung immer gleich den durchschnittlichen Werten, statt über eine Zufallszahl aus einer Normalverteilung gewählt zu werden, konvergiert die Simulation zu einem stabilen Zustand. Abhängig vom Investment und der Bestrafung der Teilnehmer konvergieren beide Werte auf einen stabilen Wert zwischen dem minimalen und maximalen Startwert.

Wird die Bestrafung durch einen weiteren zusätzlichen Faktor verstärkt, sind die Verluste zu Beginn deutlich höher und Kooperation etabliert sich erwartungsgemäß schneller. Im langfristigen Verlauf der Simulation schwächt sich aber auch hier die Kooperation zunehmend ab. Bei geringerer Bestrafung wird die Kooperation schwächer und die Dauer kürzer. Wird die Möglichkeit der Bestrafung nur sehr schwach genutzt, entsteht keine Kooperation.

Ein weiterer veränderbarer Wert ist die Standardabweichung. Liegt sie höher als bei 10 % der möglichen Spanne und weichen damit die konkreten Werte für Investment und Bestrafung vom Mittelwert deutlicher ab als das beim Standardmodell der Fall ist, etabliert sich Kooperation in der Regel etwas langsamer. Die Veränderung des durchschnittlichen Ertrags aller Agenten ähnelt weniger einer linearen Funktion als bei dem Wert der Standardabweichung des Standardmodells.

Das Experiment aus [Gür06a] basiert auf zwei Organisationen. Die Einführung einer zweiten Organisation in das Simulationsmodell liegt also nahe, da der Wettbewerb zwischen diesen die Simulationsergebnisse beeinflusst. Mit dieser Erweiterung erneuert sich die Frage nach einer sinnvollen Lernregel. Mit der gleichen Lernregel, der Orientierung am besten aller im Spiel beteiligten Agenten, gibt es ähnliche Ergebnisse wie bei der Simulationsdurchführung mit nur einer Organisation. Unterschiedlich ist, dass die Teilnehmer von Organisationen mit einer niedrigeren durchschnittlichen

Investition schlechter abschneiden. Zudem kann es sein, dass zu Beginn ein wenig engagierter Teilnehmer einer Organisation mit sehr hart bestrafenden Mitgliedern schlechter abschneidet als ein absolut weniger engagierter Teilnehmer, der seinerseits in seiner Organisation aber an der Spitze steht und deshalb weniger bestraft wird. Dieser Vorteil wird aber in den darauf folgenden Runden aufgrund des erstgenannten Unterschieds zwischen den Organisationen mehr als ausgeglichen.

Orientiert sich der Agent hingegen nur zu einem kleinen Teil an dem besten Agenten insgesamt und dafür zu einem größeren Teil am besten Agenten der Organisation, ist das Ergebnis der zu Beginn weniger engagierten Organisation lange Zeit schlechter als das Ergebnis der anderen Organisation.

Wenn die Teilnehmer von einer der beiden Organisationen entsprechend der Regel im Experiment nicht die Möglichkeit zur Sanktionierung haben, entsteht nur in der anderen Organisation Kooperation. Auch hier fällt die Kooperation mittelfristig ab, da auch im Vergleich mit einer anderen Organisation kurzfristig die beste Strategie den Verzicht auf Sanktionierung einschließt.

Um eine lange bestehende Kooperation zu fördern, sind die bisherigen Erweiterungen nicht ausreichend. Im Groben lassen sich zwei Lösungswege unterscheiden. Eine Möglichkeit ist die Einführung einer zusätzlichen Kraft hin zu „gutem Verhalten", d.h. starker Beteiligung an Sanktionierung, weil es für die Organisation sinnvoll ist und andere Agenten sich auch beteiligen. Eine andere Möglichkeit wäre die Einführung langfristiger und damit auch komplizierterer Lernregeln mit dem Ziel, zwar ein überdurchschnittlich guter, jedoch nicht der absolut beste Mitspieler zu sein. Eine rein egoistische Lernregel, die zum Ziel hat, der absolut beste Agent zu werden, ist auch bei einer langfristigen Lernregel nicht ausreichend, um das beobachtete Verhalten zu konstruieren.

Eine beim Lernen einfließende Kraft hin zu „gutem Verhalten" kann die Orientierung des Durchschnittswertes der Bestrafung am vorbildlichsten Agenten sein. Dieser ist der Agent, der in der letzten Runde am härtesten bestraft und sich damit am uneigennützigsten verhalten hat. Ist der Drang in Richtung des vorbildlichsten Agenten nur halb so groß wie in die Richtung des besten Agenten, ergibt sich eine auch langfristig stabile Kooperation, wie in Abbildung 23 zu sehen ist.

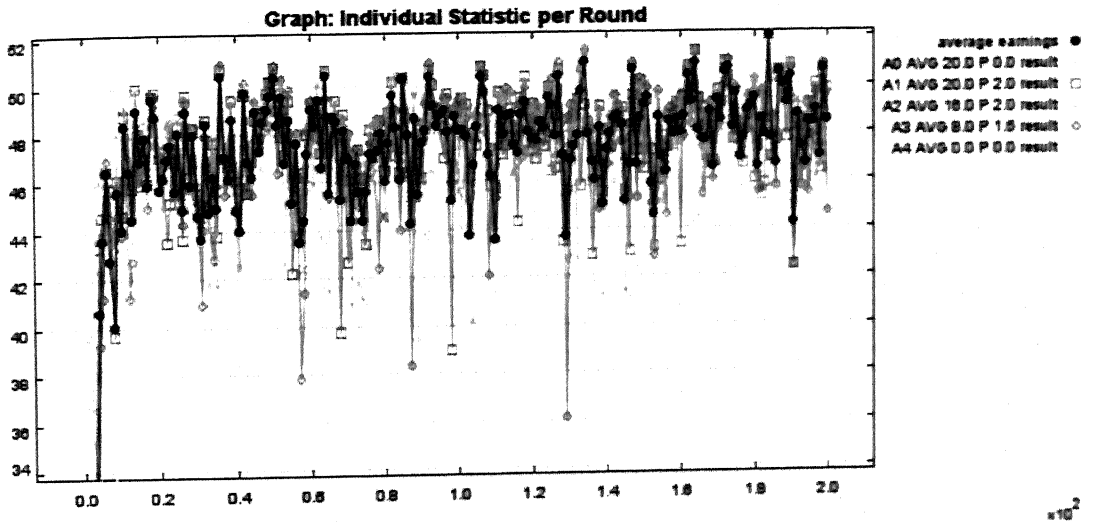

Abbildung 23: Orientierung am engagiertesten Agenten (Sanktionierung)

Weitere Variationsmöglichkeiten werden im folgenden Unterkapitel diskutiert.

5.2.4 Interpretation der Ergebnisse und Ausblick

Mit relativ einfachen, rein egoistisch ausgelegten und nur die kurzfristige Vergangenheit überblickenden Regeln gelingt eine Annäherung an einen großen Teil der Ergebnisse aus dem Experiment. Bei geeigneten Ausgangswerten bildet sich emergent die Norm bei der Investition nicht vom Gruppenverhalten abzuweichen. Nach dem Verfall der Bestrafung verfällt auch die Norm der hohen Investition schrittweise, da es keinen Anreiz mehr für die Umsetzung derselben gibt. Wird hingegen explizit die Norm der Orientierung am engagiertesten Teilnehmer eingeführt, erkennen die beteiligten Agenten überdurchschnittliches Engagement, welches in der Simulation zum Beispiel durch extreme Zufallswerte über der Normalverteilung bedingt sein kann. Daraus resultiert, dass die Norm der hohen Investition dauerhaft bestehen bleibt.

Abgesehen von den guten Ergebnissen der Simulation liegt die Vermutung nahe, dass die Entscheidungen der Teilnehmer aufgrund von komplexeren Regeln getroffen wurden. Eine komplexere Regelstruktur der Agenten scheint für den Entwurf eines Modells, das der Realität näher kommt und gegebenenfalls auch bessere Ergebnisse produziert, unabdingbar. Offen bleibt zudem, wie stark die Norm „to punish low contributors" [Gür06a] bereits in den Köpfen der teilnehmenden Studenten vorhanden war und ob diese der wichtigste Auslöser für die Resultate des Experiments ist[35]. Eine Möglichkeit wäre, das Experiment nicht mit einer homogenen Gruppe von Agenten, die sich nur in den Ausgangseigenschaften unterscheiden, sondern mit einer heterogenen Gruppe abzubilden. Diese Gruppe könnte unterschiedliche Typen von Agenten

35 Die Erkenntnisse in diesem und den folgenden Absätzen sind zum Teil auf Diskussionen auf dem ZUMA Advanced Simulation Workshop vom 19.-23. März 2007 in Koblenz zurückzuführen.

beinhalten. Beispiele hierfür wären uneigennützige Agenten, Agenten mit starkem Gerechtigkeitsempfinden, team-orientierte sowie stark egoistische Agenten. Vor einer weiteren Entwicklung des Simulationsmodells empfiehlt sich in jedem Fall ein weiteres Literaturstudium zum Thema experimentelle Wirtschaftsforschung, insbesondere über das Mikroverhalten der Teilnehmer.

Möglicher Ansatzpunkt für eine weitere Verfeinerung der Lernregel ist die explizite Einführung einer weiteren Kraft innerhalb der Agenten, die der egoistischen entgegen wirkt. Der Vorteil wäre hierbei, dass eine entstehende Norm messbar wäre. Die Existenz und Stärke dieser Kraft müsste hierfür aber begründet werden.

Sollen nicht die Norm des „Nachahmens" oder des „Guten" die Norm der hohen Investition produzieren, sondern ausschließlich die Lernregel, so sollte diese nicht nur kurzfristig, sondern über ein Gedächtnis der Agenten längerfristig ausgerichtet werden. Zudem müsste diese Lernregel ein schwächeres Ziel, zum Beispiel besser als der Durchschnitt zu sein, enthalten. Ein Nachteil ist, dass diese Regel umfangreich und schwer begründbar wäre. Idealerweise passen die Agenten ihr Verhalten nicht nur dem Verhalten anderer an, sondern konstruieren eine Strategie, die sie verfolgen und weiter-entwickeln. In diesen Prozess sollten in jedem Fall möglichst alle verfügbaren Informationen einbezogen werden.

Als Erweiterungen des Simulationsmodells wäre die Einführung eines Beziehungs-geflechts ähnlich dem Kirk-Coleman Modell einschließlich des Konzepts der Reputation der Agenten untereinander denkbar[36]. Reputation fördert im Allgemeinen Kooperation und würde in dem vorliegenden Simulationsmodell die Bildung einer Norm unterstützen. Um das Simulationsmodell dem Experiment weiter anzunähern, sollte auch das Wechseln der Organisation möglich sein. Auch hier würde der zusätzliche Freiheitsgrad in der Entscheidung der Agenten die Komplexität des Simulationsmodells erhöhen.

Interessant könnte auch eine Verallgemeinerung der Simulation auf möglichst viele Experimente der experimentellen Wirtschaftsforschung, welche Normenbildung ermöglichen, sein. Neben der Abstrahierung von spezifischen Regeln ist hierfür auch die Abstraktion von spezifischem Agentenverhalten notwendig. Als Ausgangspunkt kann zum Beispiel die Dissertation von Eva Ebenhöh [Eben06] dienen, die sich mit den genannten Themen auseinander setzt.

36 Auch hierzu existieren bereits Veröffentlichungen aus der experimentellen Wirtschaftsforschung, wie zum Beispiel [Rock06].

Abschließend kann festgestellt werden, dass eine Norm aus verschiedenen mono- oder multikausalen Ursachen entstehen kann. Eine weitere Diskussion hierzu ist im folgenden Kapitel zu finden.

6 Fazit

Diese Arbeit liefert Ideen und einen konkreten Vorschlag für den Weg zu einer generischen Entwicklungsumgebung zur Simulation von Normen. Ein Codegenerator, der auf einem anwendungsspezifischen Metamodell basiert, ist eine Lösung zur Verringerung des Abstandes zwischen high-level Simulationsprogrammen und low-level Frameworks. Diese Arbeit liefert keine ausführlich getestete und zur Anwendung gedachte Umgebung, sondern vielmehr ein Grundgerüst und weitere Anforderungen an eine solche.

Folgende Elemente der Anforderungsanalyse sind neben den durch die Verwendung von Repast erfüllten Anforderungen (zum Beispiel „Die Simulation muss aus Agenten bestehen") jetzt auf höherer Ebene realisiert:

- Agenten, deren **Eigenschaften und Verhalten** können auf Modellierungsebene **modelliert und spezifiziert** werden. Die **Interaktion** ist über Kanten zwischen den Agenten möglich.
- Es kann sowohl eine **statische** als auch eine **dynamische Umwelt** realisiert werden.
- Das **Verhalten der Simulation** wird vollständig auf Modellierungsebene festgelegt.
- Die Simulationsumgebung wurde vollständig in **Java** erstellt.
- Die Simulationsumgebung wurde an Hand von **zwei Beispielen getestet**.
- Die **Wartbarkeit**, **Benutzerfreundlichkeit** sowie **Zuverlässigkeit** von Simulationsmodellen werden durch eine Codegenerierung erhöht.

Damit sind die Anforderungen, die mit „muss" oder „soll" gekennzeichnet sind, erfüllt.

Im Folgenden werden die Erkenntnisse für das Projekt EMIL und mögliche weitere Schritte beschrieben.

6.1 Erkenntnisse für das Projekt EMIL

In diesem Unterkapitel werden zunächst Aspekte zum Framework Repast als Basis für die Entwicklung einer Multi-Agenten-Umgebung aufgeführt. Anschließend werden die Codegenerierung und die Entstehung von Normen thematisiert.

6.1.1 Repast als Basis von Weiterentwicklungen

Die Ende 2006 publizierte Veröffentlichung [Rail06] stellt den Frameworks Swarm, Repast und MASON das für seine einfache Benutzbarkeit bekannte[37] Tool NetLogo

37 Diese Aussage belegt nicht nur die in [Rail06] beschriebene Evaluation, sondern auch eine Diskussion auf dem ZUMA Advanced Simulation Workshop vom 19.-23. März 2007 in Koblenz

entgegen. Gerade weil beide Herangehensweisen zum Simulationsentwurf sehr unterschiedlich sind, sind die Ergebnisse für das Projekt EMIL interessant, da der Vergleich der Tools und die daraus abgeleiteten Empfehlungen für Entwicklungen Aufschluss über die in der Anforderungsanalyse des Projekts zu treffenden Entscheidungen gibt.

Ein großer Teil der Erkenntnisse aus dieser Veröffentlichung deckt sich auch mit den Erfahrungen bei der Erstellung dieser Arbeit. Die auffälligen Aspekte während der Arbeit mit Repast, die über die in Kapitel 3 enthaltene Evaluation hinaus gehen, sind:

- Repast selbst ist eine Sammlung verschiedener, den Modellierer unterstützende Module. Programmiererfahrung ist für die Erstellung von Simulationsmodellen in jedem Fall notwendig. Diese Anforderung an den Implementierer eines Simulationsmodells wird bei umfangreichen oder ungewöhnlichen Modellen durch eine **fehlende ausführliche**, über die Tutorials hinausgehende **Dokumentation** verstärkt. Für einfache Modelle reichen die Tutorials zur Einarbeitung aus.
- Repast und der darin enthaltene Scheduler sowie die Module zur Benutzerschnittstelle sind eine **gute Grundlage zur Entwicklung einer neuen Simulationsumgebung**, da Repast vor allem aufgrund der längeren Entwicklungszeit mehr Unterstützung bietet als zum Beispiel MASON. *"Repast is certainly the most complete Java platform."* [Rail06] Insbesondere die Integration der graphischen Benutzerschnittstelle ist gut gelungen. Graphen können mit wenig Aufwand im Code oder teilweise auch über die Repast-GUI erstellt werden. Parameter, die über die GUI von Repast anpassbar sein sollen, können relativ einfach gekennzeichnet werden.
- Die in Repast enthaltenen Bibliotheken sind zum Teil bereits gut integriert (wie die Graphenbibliothek Plot), zum Teil eher lose mit Repast gekoppelt. *"Compared to Swarm, Repast and MASON seem more like libraries and less like frameworks, which makes the transition from ideas to working simulator more difficult."* [Rail06] Für die lose gekoppelten Bibliotheken, wie zum Beispiel zu den neuronalen Netzen, ist eine Einarbeitung an Hand der jeweiligen projekteigenen Dokumentationen erforderlich. In jedem Fall gibt die Sammlung **Ideen für die Erweiterung** von Modellen (wie den Einsatz von neuronalen Netzen).
- Messungen von [Rail06] zu Folge ist **Repast nicht wesentlich langsamer als MASON**: *„but Repast execution times were only 1 to 54% longer"* [Rail06]. Bei sich laufend vervielfachenden Rechengeschwindigkeiten ist dies zu vernachlässigen. Entscheidender dürfte aufgrund der aktuellen Entwicklung hin zu Mehrkernprozessoren die Unterstützung von auf mehreren Kernen verteilten Simulationen sein. Eigenen Erfahrungen zu Folge nutzt Repast 3.1 die Möglichkeit eines Zweikern-Prozessors nicht aus. So war ein aktueller Zweikernprozessor[38] nur zu

38 Gearbeitet wurde mit einem Intel® Core™ Duo T2300 mit 1,66 Gigahertz.

circa 50 Prozent ausgelastet, ein Arbeiten neben der Simulation daher problemlos möglich und damit noch Potential für Geschwindigkeitsverbesserungen vorhanden. Angesichts der Fertigstellung von Repast 3.1 im Jahr 2003 bleibt in diesem Punkt die weitere Entwicklung abzuwarten.

- Manchmal hielt die Simulation für 2-3 Sekunden kurz an, seltener stürzte sie aus nicht bekannten Gründen ab. Abgesehen von den **sporadisch auftretenden Abstürzen** arbeiteten die auf Repast basierenden Simulationen stabil.
- Sehr spezielle Fragen scheinen im Forum[39] selten und spät beantwortet zu werden. Bezüglich Einstiegsfragen scheint die Unterstützung besser zu sein. Die **Experten im Forum** sind dort zur Zeit **nicht sehr aktiv**. Anfang Mai 2007 gab es insgesamt nur vier Themen aus April 2007 mit mindestens einer Antwort. Auf der anderen Seite gab es zahlreiche SPAM-Nachrichten.

Alles in allem ist die Verwendung von Repast als Grundlage für eine Entwicklung einer neuen Entwicklungsumgebung eine tragfähige Lösung, da Repast bereits mehr Funktionen als MASON übernimmt und dennoch eine verglichen mit eigenständigen Programmen wie NetLogo freie Entwicklung erlaubt. Idealerweise wird eine in Zukunft fertig gestellte, finale Version von Repast Simphony verwendet, wobei bei dieser Alternative zunächst die weitere Entwicklung abgewartet werden muss.

6.1.2 Modellierung und Codegenerierung

Die Codegenerierung basiert wesentlich auf dem EMF und dem darin enthaltenen JET. In folgenden Quellen finden sich Hilfen zur Einarbeitung in diese Frameworks[40]:

- Kurze Einführung in die Möglichkeiten von EMF und JET an Hand eines kleinen Beispiels
  Schmauder, Ralf; Schill, Philipp: *Codegenerierung mit dem „Eclipse Modeling Framework" und JET.* Objektspektrum 1/2005.
- Einstiegstutorials zu JET
  http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html
  http://www.eclipse.org/articles/Article-JET2/jet_tutorial2.html
- Quellensammlung auf Eclipse.org zum EMF: Überblick, Tutorials und weitere Dokumente
  http://www.eclipse.org/modeling/emf/docs/
- Sehr ausführliche Einführung in das EMF
  Moore, Bill; Dean, David; Gerber, Anna; Wagenknecht, Gunnar; Vanderheyden, Philippe: *„Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework".* IBM Redbook, URL:

39 Das Forum zu Repast findet sich unter: http://www.nabble.com/repast-interest-f3967.html, Stand 03.05.2007.
40 Jeweils Stand 04.05.2007.

http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246302.html, 2004.

Für das Projekt EMIL stellt ein Modellierungs-Plugin für Eclipse, welches Repast-Modelle generiert, einen möglichen Lösungsweg dar. Eine Alternative dazu wäre eine vollständig neu entwickelte Umgebung, deren Programmierung gegebenenfalls durch Bibliotheken unterstützt würde. Vorteile der eigenständigen Entwicklung sind die Unabhängigkeit von anderen Projekten, aufeinander abgestimmte Komponenten sowie eine Beschränkung auf hilfreiche Bibliotheken. Diese Beschränkung ermöglicht eine kompakte Umgebung. Vorteile für ein auf Eclipse und einem Simulationsframework basierenden Modellierungs-Plugin sind Standardisierung und Integration sowie der Zeitgewinn nach der Einarbeitung in das Framework. Zudem gibt es insbesondere in Eclipse eine sehr große Anzahl von den Entwickler unterstützenden Erweiterungen. Eine weitere Möglichkeit wäre eine Weiterentwicklung bereits vorhandener Programme, wie NetLogo, wobei hier zunächst eine Evaluation hinsichtlich der Tauglichkeit zur Simulation von Normen und zur Erweiterbarkeit notwendig ist.

Falls das Modellierungs-Plugin als Alternative ausgewählt wird, sind Erweiterungen des Modellierungs-Plugins notwendig oder zumindest sinnvoll:

- Das Metamodell muss insbesondere im Hinblick auf die Bedingungen für den OneObjectStep und verschiedenen Klassen von Agenten, der Umwelt und deren Vererbungshierarchien erweitert und um dazugehörige JET-Templates ergänzt werden. Weitere Ideen sind:
  - Entsprechend den in Aktivitätsdiagrammen der UML enthaltenen Elemente ist eine Einführung von Start- und Endzuständen sowie von Verzweigungspunkten sinnvoll.
  - Ein SomeObjectStep wäre wünschenswert. Beispielsweise könnten so alle Objekte mit einem bestimmten Wert aufgerufen werden.
  - Gegebenenfalls müssten die Namen der Elemente, wenn geeignetere vorhanden sind, an den Sprachgebrauch der Sozialwissenschaftler angepasst werden.
- Es müssen weitere Komfortmerkmale ergänzt werden. So ist die Generierung des Codes im Eclipse-Workspace bei Abspeicherung des Modells zwar recht komfortabel, es sprechen aber auch Gründe für eine Trennung beider Teile. Der Code wird zur Zeit immer komplett neu generiert und damit bereits manuell ergänzte Abschnitte überschrieben. Mittels JMerge und einer Kennzeichnung der generierten Teile mit „@generated“ in der Javadoc-Dokumentation könnten manuell ergänzte Teile mit dem Löschen des „@generated“-Attributes geschützt werden.

- Zur Zeit findet im Modell keine über die Bedingungen des Metamodells hinausgehende Konsistenzprüfung statt. Mittels der Object Constraint Language (OCL) könnten weitere Integritätsbedingungen ergänzt werden und, falls möglich automatisch, in das Modellierungs-Plugin integriert werden. Erste Schritte wären beispielsweise die Überprüfung der Methodennamen (Verbot von Leerzeichen) oder die feste Verknüpfung der Bedingungen mit sinnvollen Attributen.
- Die Erstellung des Modells sollte nicht nur im von EMF erstellten baumorientierten Editor, sondern auch in einem Editor mit graphischer, näher an der UML-orientierten, intuitiveren und anwenderfreundlicheren Darstellung möglich sein. Eine solche Oberfläche kann mit GEF und GMF entwickelt werden.
- Die Codegenerierung könnte aus Modellen und darin enthaltenem Code nicht nur Code für ein, sondern für mehrere Frameworks, wie MASON, Repast oder andere erzeugen.
- Zu überlegen ist eine Hierarchie von Metamodellen. Ein sehr allgemeines, möglichst alle denkbaren sozialwissenschaftlichen Simulationen umfassendes Metamodell könnte um mehrere anwendungsspezifische Metamodelle ergänzt werden. Das allgemeinere Metamodell sollte idealerweise eng verknüpft mit den dahinter liegenden Frameworks entwickelt werden. Ein entsprechendes Modell für Repast Simphony existiert bereits, wie die Ende 2006 publizierte Veröffentlichung [Par06] zeigt, und soll Grundlage für die später erscheinende finale Version sein. Wenn es der Repast-Entwicklergruppe gelingt, eine für den Anwendungsfall angemessene Codegenerierung mitzuliefern, ist es evtl. möglich, die integrierte Codegenerierung zu nutzen und dieser einzig ein anwendungsspezifisches Metamodell hinzuzufügen.
- Das Verhalten der Agenten sollte auch deklarativ mittels Regeln oder einer einfacheren prozeduralen Sprache beschreibbar sein.
- Die Kommunikation sollte bereits im generierten Simulationsmodell explizit berücksichtigt sein. Eine Definition der Freiheitsgrade während der Kommunikation, beispielsweise erlaubte Wörter, ist denkbar.
- Der Startzustand sollte nicht im Code realisiert werden müssen, sondern Teil des Modells werden. Bei Netzwerken ist beispielsweise eine Graphendarstellung eine sinnvolle graphische Repräsentation. In diesem könnte der Startzustand einzelner Agenten verändert werden.
- Auf Bibliotheken, die „intelligentes Verhalten" erzeugen können, sollte leichter zugegriffen werden können. Beispiele sind neuronale Netze oder genetische Programmierung.
- Zur Geschwindigkeitssteigerung sollten bei großen Netzwerken die Kanten in der Ausgabe der Benutzerschnittstelle nicht dargestellt werden und technisch ohne eigene Objekte realisiert sein.

Zusammenfassend kann festgestellt werden, dass Wissensrepräsentation für Speicherung im Agenten und zur Kommunikation sowie einfache Benutzbarkeit die nächsten Felder zur Weiterentwicklung des Codegenerator-Plugins sein sollten, da damit ein großer Teil der oben genannten Punkte abgedeckt ist.

6.1.3 Arten von Normen während der Simulation

Im Groben sind, technisch gesehen, zwei Arten von Normen zu unterscheiden. Das ist zum einen eine implizite, emergent entstandene Norm, die sich indirekt im Verhalten der Agenten wider spiegelt. Für die beteiligten Agenten ist es hierbei genauso wie dem den Simulationsverlauf verfolgenden Wissenschaftler möglich, durch Muster in den Entscheidungen und den daraus resultierenden Ergebnissen eine neu entstandene Norm zu erkennen. Für den programmierten Agenten ist dies jedoch ungleich schwieriger, sodass im Vorhinein entweder eine klare Zuordnung von vorhandenem Muster zu einer Norm vorhanden oder das Muster zumindest klar erkennbar sein muss.

Zum Zweiten ist es möglich, eine Norm explizit auszudrücken. Dies kann entweder als Variable der Umwelt, die alle Agenten betrifft, oder als individuell unterschiedliche Variable innerhalb der Agenten geschehen. Auch hier besteht das Problem der Berechnung eines solchen Wertes. Der Vorteil ist, dass Normenverhalten für den Wissenschaftler leichter sichtbar wird. Der Nachteil ist, dass die Entstehung einer Norm an eine oder mehrere Variablen gebunden ist und damit bereits vorher dem Simulationsmodell mitgegeben werden muss. Diese Vorgehensweise schränkt den Blick auf mögliche Variationen ein.

Normen können aus mehreren Gründen entstehen. Zum Ersten wie im Kirk-Coleman-Modell direkt aus der Festsetzung der zugrunde liegenden Regeln. Das Kirk-Coleman-Modell fußt im Wesentlichen auf zwei Regeln. Zum einen auf der Tatsache, dass Beziehungen, die stärker sind, eine Kommunikation bevorzugen. Zum Zweiten, dass eine Kommunikation die Beziehung stärkt. Daraus folgt, dass starke Beziehungen ständig gestärkt werden und die Norm entsteht, dass vorhandene Beziehungen den neuen Beziehungen vorgezogen werden.

Eine weitere Möglichkeit ist die im Sanktionierungsmodell vorhandene Kombination aus Startzustand, den zu Grunde liegenden Regeln und dem Zufall. Je nach Startinvestition, der anfänglichen Bestrafung sowie anderen Faktoren, entsteht die Norm der hohen Investition oder nicht. In jedem Fall verfällt sie nach einiger Zeit wieder. Gegen den Verfall der Norm kann eine durch eine Regel explizit eingeführte Norm, beispielsweise sich auch am engagiertesten Agenten der Gruppe zu orientieren, wirken.

Grundsätzlich gilt, dass es mit steigender Komplexität der Agentenentscheidungen zunehmend schwieriger wird, entstandene Normen alleine aus dem Verhalten der Agenten und dem Ergebnis abzuleiten. Letztlich ist hier ein Kompromiss zwischen Einfachheit der Regeln und der Entstehung von ausreichend sichtbaren Normen zu finden.

6.2 Ausblick

Bei der Entwicklung von agentenbasierten Umgebungen zur sozialwissenschaftlichen Simulation ist vor allem die Verknüpfung von schnellem Einstieg auf der einen und größtmöglicher Erweiterbarkeit auf der anderen Seite wichtig. Daraus folgt, dass, wie in anderen Bereichen der IT, zunehmend die Themen Dokumentation und Standardisierung in den Vordergrund rücken werden.

Letztlich lassen sich die Anforderungen an eine Simulationsumgebung von der folgenden Frage ableiten: Welche verschiedenen Arten von Nutzern gibt es und wie kann die Umgebung die Arbeit der Beteiligten optimal unterstützen? Die Veröffentlichung [Par06] identifiziert folgende Rollen von Benutzern: Den Repast Entwickler, der das Metamodell und den Code von Repast entwickelt, den Entwickler, der Repast erweitert und seine Modelle weiterhin in Java entwickelt, den Modellierer, der mit seinem Anwendungswissen Modelle graphisch modelliert und den Wissenschaftler, der die Ergebnisse der Simulation analysiert.

Diese Arbeit legt die Grundlage für eine weitere Rolle, den Entwickler, der Repast oder ein ähnliches Framework beziehungsweise Programm auf eine Anwendungsdomain reduziert. Er ist gleichermaßen mit technischem Wissen und Anwendungswissen ausgestattet. Er ist für das anwendungsspezifische generische Simulationsmodell, das den technischen Rahmen der Klassen des Simulationsmodells bildet, verantwortlich. Zudem erstellt er ein anwendungsspezifisches Metamodell, die Abbildung eines daraus abgeleiteten graphischen Modells in den Code des Simulationsmodells sowie eine dazugehörige Dokumentation. Damit vereinfacht er die Schritte des Anwenders zum Modell erheblich.

Die möglichst generisch entwickelten Frameworks beziehungsweise Programme sollten mit einer zusätzlichen Schicht spezialisiert werden, um die Benutzbarkeit zu vereinfachen.

Literaturverzeichnis

[Cas98] Castelfranchi, Cristiano; Conte, Rosaria; Paolucci, Mario: *Normative reputation and the costs of compliance.* Journal of Artificial Societies and Social Simulation 1(3), URL: http://www.soc.surrey.ac.uk/JASSS/1/3/3.html, 1998.

[Con95] Conte, Rosaria: *Understanding the functions of norms in social groups through simulation.* In: Gilbert, Nigel (Hg.): Artificial societies: the computer simulation of social life, London: UCL Press, 1995.

[Con05] Conte, Rosaria et al.: *Emergence In the Loop: Simulating the two way dynamics of norm innovation.* Projektantrag, 2005.

[Dib04] Dibble, Catherine; Feldman, Philip G.: *The GeoGraph 3D Computational Laboratory: Network and Terrain Landscapes for RePast.* Journal of Artificial Societies and Social Simulation 7(1) , URL: http://jasss.soc.surrey.ac.uk/7/1/7.html, 2004.

[Dun05] Dunham, Jill Bigley: *An Agent-Based Spatially Explicit Epidemiological Model in MASON.* Journal of Artificial Societies and Social Simulation 9(1), URL: http://jasss.soc.surrey.ac.uk/9/1/3.html, 2005.

[Eben06] Ebenhöh, Eva: *Modelling human behaviour in social dilemmas using attributes and heuristics.* Dissertation, URL: http://www.usf.uni-osnabrueck.de/~eebenhoe/forschung/thesis/ebenhoeh-thesis.pdf, 2006.

[Ecl07] Eclipse Developers Group: *Eclipse – an open development platform.* URL: http://www.eclipse.org/, Stand 17.04.2007, 2007.

[EMF07] EMF Developers Group: *Eclipse Modeling Framework Project (EMF).* URL: http://www.eclipse.org/modeling/emf/?project=emf, Stand 18.04.2007, 2007.

[Gal05] Galan, Manual Jose; Izquierdo, Luis: *Appearances Can Be Deceiving: Lessons Learned Re-Implementing Axelrod's 'Evolutionary Approach to Norms'.* Journal of Artificial Societies and Social Simulation 8(3), URL: http://jasss.soc.surrey.ac.uk/8/3/2.html ,2005.

[GEF07] GEF Developers Group: *Eclipse Graphical Editing Framework (GEF).* URL: http://www.eclipse.org/gef/, Stand 17.04.2007, 2007.

[GMF07] GMF Developers Group: *Eclipse Graphical Modeling Framework (GMF).* URL: http://www.eclipse.org/gmf/, 17.04.2007, 2007.

[Gil02] Gilbert, Nigel; Bankes, Steven: *Platforms and methods for agent-based modeling.* Proceedings of the National Academy of Sciences of the USA 99(Anhang 3). Seiten 7197-7198, 2002.

[Gil05] Gilbert, Nigel; Troitzsch, Klaus G.: *Simulation for the Social Scientist.* Maidenhead und New York: Open University Press, 2005.

[Gür06a] Gürerk, Özgür; Irlenbusch, Bernd; Rockenbach, Bettina: *The Competitive Advantage of Sanctioning Institutions.* Science 312(5770), Seiten 108-111, 2006.

[Gür06b] Gürerk, Özgür; Irlenbusch, Bernd; Rockenbach, Bettina: *The Competitive Advantage of Sanctioning Institutions: Supporting Online-Material.* URL: http://www.uni-erfurt.de/mikrooekonomie/downloads/ G%FCrerk_Irlenbusch_Rockenbach_Science2006_SOM.pdf, Stand 07.03.2007.

[Kir67] Kirk, Jerome; Coleman, James: *Formalisierung und Simulation von Interaktionen in einer Drei-Personen-Gruppe.* In: Mayntz, Renate. (Hg.): Formalisierte Modelle in der Soziologie. Soziologische Texte, Ausgabe 39, Seiten 169-190, Neuwied: Luchterhand, 1967.

[Klü96] Klügl, Franziska; Puppe, Frank; Raub, Ulrich; Tautz, Jürgen: *Ein Simulationssystem zur Darstellung emergenten Verhaltens.* In: ASIM (Hg.): Fachtagung Simulation und Animation für Planung, Bildung und Präsentation, Seiten 51-62, 1996.

[Klü06] Klügl, F.; Herrler, R.; Fehler, M.: *SeSAm: Implementation of Agent-Based Simulation Using Visual Programming.* AAMAS 2006, Hakodate, Maz 2006, Seiten 1439f, 2006.

[Luke04] Luke, Sean; Cioffi-Revilla, Claudio; Panait, Liviu; Sullivan, Keith: *MASON: A New Multi-Agent Simulation Toolkit.* SwarmFest Workshop, 2004.

[Lyt06] Lytinen, Steve; Railsback, Steve; Howe, Tom: *How to Set Up Repast in Eclipse.* URL: http://www.swarm.org/images/e/e6/How-to_use_Repast_in_Eclipse.pdf, Stand 02.05.2007, 2006.

[Mar07] Marrone, Paul: *The Complete Guide. All you need to know about Joone* URL: http://mesh.dl.sourceforge.net/sourceforge/joone/JooneCompleteGuide.pdf, Stand 21.05.2007, 2007.

[Mas07] MASON Developer Group: *MASON Multiagent Simulation Toolkit.* URL: http://cs.gmu.edu/~eclab/projects/mason/, Stand: 02.01.2007, 2007.

[Par06] Parker, M.T.; North, M.J.; Collier, N.T.; Howe, T.R.: *Agent-based Meta-Models.* In: Sallach, D.; Macal, C.M.; North, M.J. (Hrsg.): Proceedings of the Agent 2006 Conference on Social Agents: Results and Prospects, 2006.

[Rail06] Railsback, Steven F.; Lytinen, Steven L.; Jackson, Stephen K.: *Agent-based Simulation Platforms: Review and Development Recommendations.* Simulation 82, Seiten 609 – 623, 2006.

[Reu06] Reussner, Ralf; Hasselbring, Wilhelm (Hrsg.): *Handbuch der Software-Architektur.* Heidelberg: dpunkt.verlag, 2006.

[Rep07] Repast Developers Group: *RePast Agent Simulation Toolkit.* URL: http://repast.sourceforge.net/, Stand: 02.01.2007, 2007.

[Rock06] Rockenbach, Bettina; Milinski Manfred: *The efficient interaction of indirect reciprocity and costly punishment.* Nature 444, Seiten 718-723, 2006.

[Rupp04] Rupp, Chris: *Requirements-Engineering und -Management : professionelle, iterative Anforderungsanalyse für die Praxis.* München: Hanser, 2004.

[Schl06] Schlosser, Andreas; Voss, Marco; Brückner, Lars: *On the Simulation of Global Reputation Systems.* Journal of Artificial Societies and Social Simulation 9(1), URL: http://jasss.soc.surrey.ac.uk/9/1/4.html, 2006.

[Schw04] Schwenk, Gero: *Micro-Macro-Relations in the Kirk-Coleman-Model*. Giessen, Diplomarbeit, URL: http://geb.uni-giessen.de geb volltexte 2004 1726 . 2004.

[Sim08] Simmel, Georg: *Untersuchungen über die Formen der Vergesellschaftung*. 1. Auflage, Berlin: Dunckler & Humblot, 1908[41].

[Sne05] Sneed, Harry: *Software-Projektkalkulation: Praxiserprobte Methoden der Aufwandsschätzung für verschiedene Projektarten*. München: Hanser, 2005.

[Tob04] Tobias, Robert; Hofmann, Carole: *Evaluation of free Java-libraries for social-scientific agent based simulation*. Journal of Artificial Societies and Social Simulation 7(1), URL: http://jasss.soc.surrey.ac.uk/7/1/6.html, 2004.

[Troi99] Troitzsch, Klaus G.: *Dynamische Modelle komplexer sozialer Systeme: Was leisten Computersimulationen?*. In: Mainzer, Klaus (Hrsg.): Komplexe Systeme und Nichtlineare Dynamik in Natur und Gesellschaft. Berlin: Springer, 1999.

[Ull07] Ullenbohm, Christian: *Java ist auch eine Insel: Programmieren mit der Java Standard Edition Version 6*. Bonn: Galileo Computing, 6. aktualisierte und erweiterte Auflage, 2007[42].

[Wool95] Wooldrige, Michael; Jennings, Nicholas R.: *Intelligent agents: theory and practice*. Knowledge Engineering Review 10(2), Seiten 115–152, 1995.

[Wool02] Wooldridge, Michael: *An Introduction to Multiagent Systems*. Chichester: John Wiley and Sons Limited, 2002.

41 Dieses Buch ist als Online-Version unter der URL http://socio.ch/sim/soziologie/index.htm (Stand 06.02.2007) verfügbar.

42 Dieses Buch ist als Online-Version unter der URL http://www.galileocomputing.de/openbook/javainsel6/ (Stand 06.02.2007) verfügbar.